Compositional Specification of Embedded Systems with Statecharts *

Jan Philipps and Peter Scholz
{philipps,scholzp}@informatik.tu-muenchen.de

Technische Universität München, Institut für Informatik
D-80290 München, Germany

Abstract. During the last years, Statecharts have gained wide acceptance for the specification of reactive, embedded systems. However, most semantics suggested so far are either informal or hard to grasp. In this contribution, we present a Statecharts dialect that permits nondeterministic specifications, offers zero-delay broadcast communication, and handles negation in trigger expressions in a new way. We give a compositional formal semantics for this dialect, which is abstract enough for formal reasoning and yet easy to operationalize for simulators, model checking tools and code generation.

1 Introduction

Statecharts [6] are a visual specification language proposed for specifying reactive systems. They extend conventional state transition diagrams with structuring and communication mechanisms. Since there is also tool support through Statemate [11], Statecharts have become quite successful in industry.

However, the semantics of Statecharts used in Statemate [7] is based on a delayed broadcast (cause and action are separated in time), which leads to a very operational, implementation-level specification style. For a modeling language for abstract requirements specifications more abstract approaches are needed. Such approaches should contain the following concepts:

- Nondeterminism is needed to express underspecification of systems. With nondeterminism, detailed specifications can be abstracted to allow model checking; in the other direction, there is a natural concept of behavioral refinement through reduction of nondeterminism [3].
- For refinement, delayed broadcast as used in Statemate is not a suitable communication concept. When refining a subchart to a set of more concrete subcharts, additional delays are introduced. Thus, the I/O-behavior of the Statechart changes. Refinement rules would have to be more complex to compensate the additional delays. As observed in [10], this is not the case for instantaneous feedback.

* This work is partially funded by the German Federal Ministry of Education and Research (BMBF) as part of the compound project "KorSys".

Instantaneous feedback enjoys other nice properties for reactive systems; see [2] for a discussion. In this contribution, we introduce a dialect of Statecharts called μ-Charts; it features a formal semantics for nondeterministic Statecharts with instantaneous feedback. It is an extension of the Mini-Statecharts dialect presented in [15, 22]. As noted in previous works on the semantics of Statecharts [9, 19], or Statechart-like languages like Argos [13, 14], or imperative synchronous languages like Esterel [2], instantaneous feedback can lead to causality conflicts when trigger events with negation are allowed. Argos and Esterel require a static analysis to reject those programs where a conflict might occur. Both languages provide very elaborated but expensive analysis techniques. We handle these conflicts semantically through oracle variables and therefore do not have to apply such algorithms.

This paper is structured as follows. In Section 2 we introduce our Statecharts dialect and give an abstract syntax and a compositional step semantics for it. Section 3 shows how to extend the step semantics to a stream semantics, modeling the complete input/output behavior of a system. Finally, in Section 4 we give a brief conclusion and discuss future extensions.

Example

As running example we use a simplified specification of the central locking system for cars. The corresponding Statechart is pictured in Figure 1; it specifies the locking system of a two-door car. Table 1 shows the signals used for the specification.

The doors can be either unlocked, locked, or protected. Protected doors can only be opened with a key from outside the car, while locked doors can only be opened from inside the car by pushing a button. Locking and unlocking is specified in the subchart Normal. Most of the time, the controller is in state Ready. (Actually, this state has to be further decomposed. However, this is not important to understand our contribution and is therefore omitted for reasons of brevity.) When the driver locks the doors, the controller moves to state Lock, and signals the low-level controllers for the doors to lower the lock. When the doors are locked, the controller returns to Ready. The behavior for unlocking and protecting the doors is similar. The subcharts MotorLeft and MotorRight specify the behavior of the door locks themselves: they either raise or lower the lock buttons on the driver and passenger door. The state Crash is entered from either of the states in Normal, when the car's crash sensor is activated. Then the doors are automatically unlocked.

The specification need not store the current state of the doors; the locking mechanism is not damaged when it tries to lock an already locked door.

2 Abstract Syntax and Semantics

In this section, we formally define syntax and semantics of our μ-Charts. They are based on Mini-Statecharts, as first presented in [15] and later refined in [20,

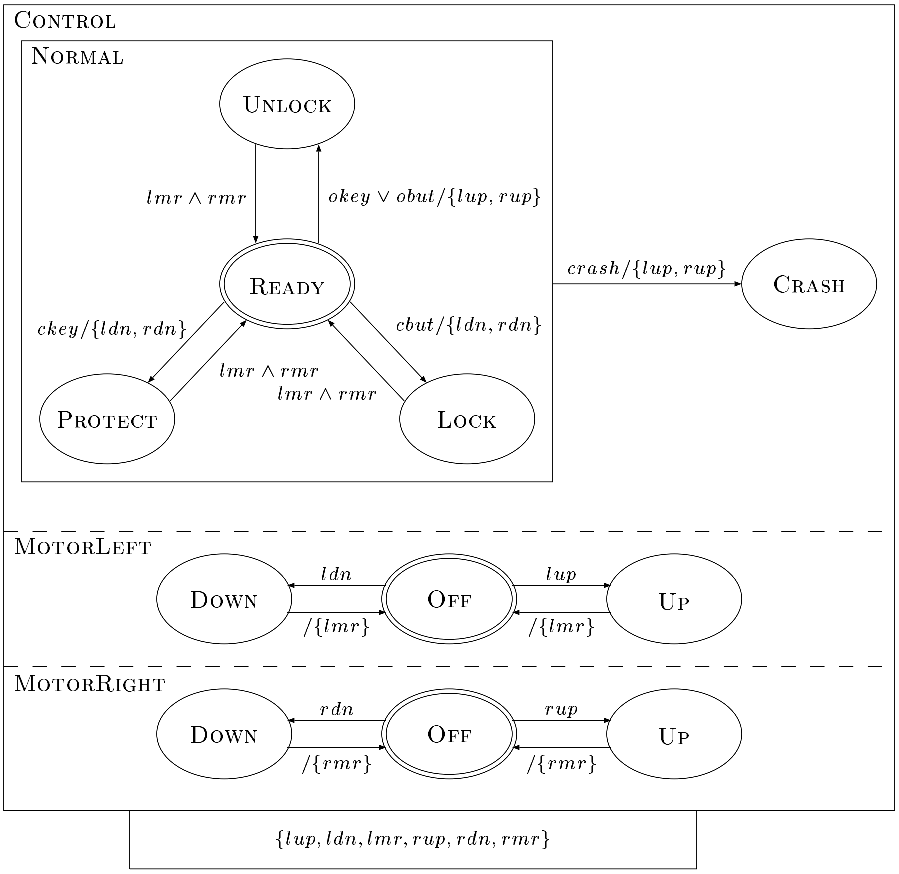


Figure 1. Central locking system

21, 22]. We only repeat those concepts that are a prerequisite for the extension to nondeterminism.

Throughout this paper, M denotes a set of signal names, *States* a set of state names, and *Ident* a set of identifier names for sequential automata. For any Statechart, only a finite number of signal, state, and automata names can be used; $\wp(X)$ denotes the set of finite subsets of some set X.

In our dialect, the set of μ-Charts $\mathcal{S}$ is defined inductively. A μ-Chart is either a sequential automaton, a parallel composition of two μ-Charts, the decomposition of a sequential automaton's state by another μ-Chart, or the result of a feedback construction. The inductive steps are motivated and defined in Sections 2.1 to 2.4. The semantics of a μ-Chart $S \in \mathcal{S}$ has the type

$$[\![S]\!] : \wp(M) \rightarrow \wp(\wp(M) \times \mathcal{S})$$

For each input signal set, the semantics determines a set of possible reactions.

Signal	*Meaning*	*Source*
crash	Crash sensor	External
okey	Opened with external key	External
ckey	Closed with external key	External
obut	Opened with internal locking button	External
cbut	Closed with internal locking button	External
lmr	Left motor ready	Internal
rmr	Right motor ready	Internal
lup	Left motor up	Internal
ldn	Left motor down	Internal
rup	Right motor up	Internal
rdn	Right motor down	Internal

Table 1. Signals used in the locking system

Each reaction is a pair consisting of an output signal set and the μ-Chart resulting from S after taking a step. The reaction set can be empty, if a chart cannot react to a given input. When we define the possible executions of a μ-Chart in Section 3, empty reaction sets are handled by letting the chart remain in its current configuration; the output will be empty.

2.1 Sequential Automata

Sequential automata are the basic elements of our Statecharts dialect. The construct

$$\mathsf{Seq}\ (N, \Sigma, \sigma_d, \sigma, \delta)$$

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

1. $N \in \mathit{Ident}$ is the unique identifier of the automaton.
2. $\Sigma \in \wp(\mathit{States})$ is a nonempty finite set of all states of the automaton.
3. $\sigma_d, \sigma \in \Sigma$ represent the default state and the current state, respectively.
4. $\delta : \Sigma \times \wp(M) \to \wp(\Sigma \times \wp(M))$ is the finite, total state transition function that takes a state and a finite set of signals and yields a set of next states paired with a finite set of output signals. If this set contains more than one pair, the automaton is nondeterministic; if the set is empty, the automaton cannot react to the current input when it is in state σ.

In our concrete syntax (see the example), we use a Boolean term t instead of a set of signals $x \in \wp(M)$ as trigger. It is straightforward to translate a partial transition function that deals with arbitrary Boolean terms as trigger condition into a set-valued total function (see for example [22]).

Example 2.1 (Sequential Automaton). Our running example contains four sequential automata: MotorLeft, MotorRight, Control, and the automaton Normal, which refines one of Control's states.

Each transition is annotated with a label such as "t/y", where t is a Boolean trigger condition and y the set of signals that are generated when the transition is taken. If y is empty, we simply write the transition label as "t"; if t equals true we omit it and just write "$/y$". Note that the two motor control automata allow nondeterministic behavior in the state OFF. For example, the left motor controller MOTORLEFT can follow any of the two transitions originating in OFF when both signals ldn and lup are present.

A transition takes place in exactly one time unit. In a specification with several automata working in parallel, more than one automaton can make a transition; all transitions taken in parallel automata are assumed to occur in the same time unit. The set of all system actions in one time unit is called a *step*.

We expect of sequential automata that:

– No two consecutive transitions in a sequential automaton are taken in a step.
– Only one branch of a nondeterministic choice is taken in a step.

To ensure these restrictions, we introduce additional signals. For each sequential automaton $\mathsf{Seq}\ (N, \Sigma, \sigma_d, \sigma, \delta)$ we introduce a signal $©_N$. Informally, this is a copyright on transitions of the automaton. When the signal is not present, the automaton may make a transition, whereupon it will generate $©_N$. If it is already present, the automaton has to stay in its current state.

The copyright signals are introduced in the following way. Each transition t/y of N is modified such that:

– The trigger condition c is strengthened by conjoining $\neg©_N$ to it.
– The action set y is extended by $©_N$.

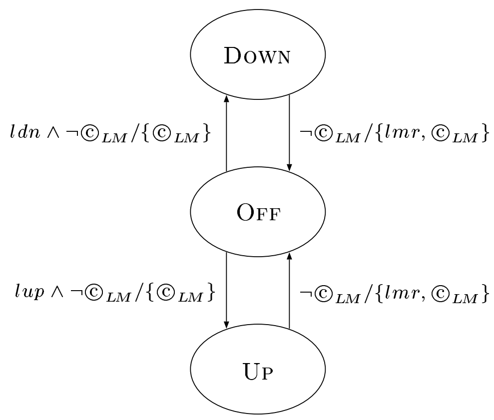


Figure 2. Motor control with copyrights

Example 2.2 (Sequential Automaton). Figure 2 shows the modified chart LEFT-MOTOR, where we abbreviated its name by *LM*.

Let C be the set of all possible copyright signals, $C := \{©_N \mid N \in \mathit{Ident}\}$. We write M_C to abbreviate $M \cup C$. The step semantics for a chart S then has the functionality:

$$[\![S]\!] : \wp(M_C) \to \wp(\wp(M_C) \times \mathcal{S})$$

Informally, a sequential, nondeterministic automaton $\mathsf{Seq}\ (N, \Sigma, \sigma_d, \sigma, \delta)$ takes a set of input signals, say x, produces a set of signals as output, say y, and then behaves like an automaton with modified actual state. This is formally denoted by:

$$[\![\mathsf{Seq}\ (N, \Sigma, \sigma_d, \sigma, \delta)]\!]x = \{(y, \mathsf{Seq}\ (N, \Sigma, \sigma_d, \sigma', \delta)) \mid (\sigma', y) \in \delta(\sigma, x)\}$$

Note that the reaction set may contain more than one pair. This reflects that the behavior of the automaton may be nondeterministic. Moreover, the reaction set may be empty, when the trigger condition of no transition from the current state is fulfilled. In this case, the automaton should remain in its current state without emitting any output signals. In Section 3, when the complete reactive behavior of a chart over time is introduced, empty reaction sets will indeed cause the chart to remain in its current state.

2.2 Parallel Composition

If S_1 and S_2 are elements of the set $\mathcal{S}$ 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. In the graphic notation parallel components are separated by splitting a box into components using dashed lines [6].

In our framework, parallel composition does not imply broadcast communication between the subcharts. Both subcharts operate independently; communication is introduced by an explicit feedback operator (see Section 2.4).

Example 2.3 (Parallel Composition). To specify the central locking system, we used three parallel composed charts: the controller and the two motors. One possible configuration of the overall system is that both motors are off and the controller is in its normal mode, while waiting for new input of the environment in its READY state. If no communication is specified, all parallel charts operate without any mutual interaction.

Informally, the parallel composition of μ-Charts behaves as S_1 and S_2 synchronously together. Generated signals of the parallel components are joined.

The formal semantics is defined by three cases. An And-chart can perform a step when at least one of the subcharts makes a step (notice that in our setting also a self-loop is a step); one or more of the charts may not react at all. This is the case, when the reaction set of such a chart returns an empty set. The reaction set of a parallel composition is the union of these cases:

$$\begin{aligned}
[\![\mathsf{And}\ (S_1, S_2)]\!]x = \{&(y_1 \cup y_2, \mathsf{And}(S_1', S_2')) \mid (y_1, S_1') \in [\![S_1]\!]x \wedge (y_2, S_2') \in [\![S_2]\!]x\} \\
\cup \{&(y_1, \mathsf{And}(S_1', S_2)) \mid (y_1, S_1') \in [\![S_1]\!]x \wedge [\![S_2]\!]x = \emptyset\} \\
\cup \{&(y_2, \mathsf{And}(S_1, S_2')) \mid [\![S_1]\!]x = \emptyset \wedge (y_2, S_2') \in [\![S_2]\!]x\}
\end{aligned}$$

Thus, when neither Statechart makes a transition, the semantics of the parallel composition yields an empty reaction set, too.

Obviously, $\mathsf{And}\ (S_1, S_2)$ is commutative and associative. We therefore write $\mathsf{And}\ (S_1, \ldots, S_n)$ to denote $n \in \mathbb{N}$ nested parallel μ-Charts.

2.3 Hierarchical Decomposition

The concept of hierarchically structuring the state space is essential for Statecharts. In our Statecharts dialect, hierarchy is introduced by replacing states of a sequential automaton (the *master*) with arbitrary charts (the *slaves*). This replacement is expressed by a finite function ϱ, which for any state σ of the master yields either the corresponding slave-Statechart, or NoDec, if the state is not replaced by a slave.

Suppose that $\mathsf{Seq}\ (N, \Sigma, \sigma_d, \sigma, \delta)$ is a sequential automaton, then hierarchical decomposition is denoted by

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

where $\varrho : \Sigma \to \mathcal{S} \cup \{\mathsf{NoDec}\}$.

Like other formal Statechart semantics [9, 13, 14], the semantics presented here has no history states. It is possible to extend our semantics along the lines of [15]. Due to space limitations we omit this extension here. Throughout this paper, we assume that the slave is always re-initialized when leaving it.

Example 2.4 (Hierarchical Decomposition). In our example, the NORMAL state of the CONTROL is replaced by another sequential automaton, also called NORMAL, which describes the current action of the locking system. Here CONTROL and NORMAL represent master and slave, respectively. As current system configuration, we assume that CONTROL is in the LOCK state and both motors are notifying the CONTROL that they have finished the lowering process. Thus, the current set of internal signals is $\{lmr, rmr\}$. We furthermore presume that exactly while the motors are sending lmr and rmr, respectively, an external $crash$ signal occurs. The overall signal set is then denoted by $\{lmr, rmr, crash\}$. Hence, NORMAL changes its current state from LOCK to READY. In addition, the system moves from the NORMAL state to the CRASH state while generating the signal set $\{lup, rup\}$. Note that all actions come about instantaneously.

Altogether, in the next instant of time, NORMAL is in its READY state, the CONTROL in the CRASH mode and both motors are in their OFF states. The automaton NORMAL is "frozen" until it is re-entered. Thus, we say that it has been *interrupted*. However, NORMAL still was able to change its current state from LOCK to READY, i.e., has not been immediately interrupted: we say that the *crash* signal has induced a *non-preemptive* interrupt. By strengthening the transitions in the slave chart with tests for the absence of signals, preemptive interrupt can be modeled as well.

To define the formal semantics for the decomposition, we distinguish four mutually exclusive cases. The first case occurs whenever the current state σ of the master $A =_{def} \mathsf{Seq}\ (N, \Sigma, \sigma_d, \sigma, \delta)$ is refined by a slave ($\varrho(\sigma) \neq \mathsf{NoDec}$), and both master and slave produce a non-empty reaction set: $[\![A]\!]x \neq \emptyset \neq [\![\varrho(\sigma)]\!]x$. The reaction set of the hierarchical decomposition is then

$$\begin{aligned}[\![\mathsf{Dec}\ A\ \mathsf{by}\ \varrho]\!]x = \{(y_m \cup y_s, \mathsf{Dec}\ A'\ \mathsf{by}\ \varrho') \mid \exists S' \in \mathcal{S} :\\ (y_m, A') \in [\![A]\!]x \wedge (y_s, S') \in [\![\varrho(\sigma)]\!]x \wedge \varrho' = \varrho[init(S')/\sigma]\}\end{aligned}$$

Here $init(S')$ initializes all sequential automata contained in S' according to their default states.

If the master is not further decomposed in the current state σ ($\varrho(\sigma) = \mathsf{NoDec}$), but by itself may react ($[\![A]\!]x \neq \emptyset$), we get

$$[\![\mathsf{Dec}\ A\ \mathsf{by}\ \varrho]\!]x = \{(y, \mathsf{Dec}\ A'\ \mathsf{by}\ \varrho) \mid (y, A') \in [\![A]\!]x\}$$

Whenever the master can react and the current state σ is decomposed by a slave which however cannot react in its current state:

$$[\![\mathsf{Dec}\ A\ \mathsf{by}\ \varrho]\!]x = \{(y, \mathsf{Dec}\ A'\ \mathsf{by}\ \varrho') \mid (y, A') \in [\![A]\!]x \wedge \varrho' = \varrho[init(\varrho(\sigma))/\sigma]\}$$

Although the slave cannot react it is re-initialized because we follow the convention that whenever the master makes a step the slave has to be initialized. The next case occurs if, although the master cannot react in the current step, the slave can react:

$$[\![\mathsf{Dec}\ A\ \mathsf{by}\ \varrho]\!]x = \{(y, \mathsf{Dec}\ A\ \mathsf{by}\ \varrho') \mid \exists S' \in \mathcal{S} : (y, S') \in [\![\varrho(\sigma)]\!]x \wedge \varrho' = \varrho[S'/\sigma]\}$$

In this case, the function ϱ is changed to ϱ' to reflect the slave's change.

Finally, if none of the above-mentioned cases is true, the overall reaction of the hierarchical decomposition is simply the empty set.

2.4 Broadcast Communication

Parallel composition is used to construct independent, concurrent components. To allow interaction of such components, our language provides a broadcast communication mechanism. In [6], for example, this mechanism already is integrated in the parallel 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 [23]. For the deterministic version of our language [15, 20, 22], we provided different syntactic constructs with different communication timings. We believe that for nondeterministic, abstract specifications, *instantaneous feedback* is the proper concept, and present here only this operator.

Suppose that $S \in \mathcal{S}$ is in an arbitrary μ-Chart and $L \in \wp(M)$ is the set of signals which should be fed back, then the construct

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

is also in $\mathcal{S}$. Graphically, the feedback construction is denoted with a box below the μ-Chart S. The box contains the signals L that are fed back.

Example 2.5 (Feedback). When the chart is in the state READY, and the driver locks the door with the car key, then NORMAL moves to state PROTECT, and emits the signals ldn and rdn. Without feedback, these signals would not be sent to the motor control subcharts. But since both signals are fed back, they are added to the input of the specification. Thus, both motors move to their DOWN states. This feedback is *instantaneous*, i.e. upon input of the signal $ckey$ the three state changes and the output of ldn and rdn occur at the same time.

Instantaneous feedback follows the perfect synchrony hypothesis of Berry [1]; it demands that an action and the event causing this action occur at the same instant of time. Therefore, the signals in z generated by chart S are instantaneously intersected with the signals L to be fed back and then joined with the external signals x. This signal set is passed to S at the same instant of time.

We first define the semantics of $\mathsf{Feedback}\ (S, L)$ for the case that no transition trigger refers negatively to signals. In Section 2.5 we extend the semantics to handle negation as well.

In the unnegated case we have to find a solution for the following equation:

$$Z = \{z \cup y \mid z \in Z \ \wedge \ y \in \pi_1(\llbracket S \rrbracket(x \cup (z \cap L)))\}$$

where π_1 filters the first component of the output set:

$$\pi_1(\{(y, S) \mid y \in \wp(M_C) \ \wedge \ S \in \mathcal{S}\}) =_{def} \{y \mid y \in \wp(M_C)\}$$

This solution can be found by computing the least fixpoint for the first projection of the subsequent function:

$$f^S_{x,L}(Z) =_{def} \{(z \cup y, S') \mid z \in Z \ \wedge \ (y, S') \in \llbracket S \rrbracket(x \cup (z \cap L)))\}$$

with respect to the following reflexive and transitive standard ordering on $\wp(\wp(M_C))$. For all $X, Y \in \wp(\wp(M_C))$ we define:

$$X \sqsubseteq Y =_{def} \forall x \in X \ \exists y \in Y : x \subseteq y$$

Formally, the semantics of the instantaneous feedback is defined by:

$$[\![\mathsf{Feedback}\ (S,L)]\!]x = \{(y, \mathsf{Feedback}(S',L)) \mid (y,S') \in \mathit{lfp}(f^S_{x,L}, \{\emptyset\})\}$$

lfp computes the least fixed point for the first projection of the above function with respect to the subset ordering. The computation starts with an empty set of signal sets, since at the beginning of the communication no signals are generated yet. *lfp* is defined as follows:

$$\mathit{lfp}(f^S_{x,L}, Y) = \text{if } \pi_1(f^S_{x,L}(Y)) = Y \text{ then } f^S_{x,L}(Y) \text{ else } \mathit{lfp}(f^S_{x,L}, \pi_1(f^S_{x,L}(Y))).$$

Notice that in general the first projection of $f^S_{x,L}$ is not monotonic w.r.t. $\sqsubseteq$. But since for each set of signal sets Z it holds that $Z \sqsubseteq \pi_1(f^S_{x,L}(Z))$, and since there are only finitely many signals — hence, finitely many sets of signal sets — the existence of least fixpoints is ensured.

Unfortunately, this property does not hold when trigger expression with negation are handled in the standard way. Instead, we make use of oracle variables.

2.5 Negation in Trigger Expressions

So far we only considered μ-Charts where each event expression occurs positively in a transition trigger. It is desirable, however, to be able to test for the *absence* of signals as well as for their presence. For example, negative signal expressions allow us to introduce priorities between transitions. As an example, we examine our locking system again. The two motor control charts in Figure 1 suffer from the following problem: when a crash occurs in the same instant the driver wants to lock the door, pressing the locking button, the motor controllers can choose nondeterministically between raising or lowering the locks. This is a safety-critical problem that must be avoided. We therefore modify the charts as in Figure 3 by conjoining the trigger condition on the transition originating from OFF and ending in DOWN with $\neg crash$. Now the controller can only lock the door, when there is no signal from the crash sensor.

Negation in trigger expressions can lead to some tricky causality problems. For example, what would be the semantics of a transition labeled $\neg a/a$? Some Statecharts semantics simply disallow Statecharts with causality problems. They require either a static analysis of the chart, which might reject charts that do not really have causality conflicts, or a thorough state exploration, which even with today's advanced model checking techniques is untractable for larger charts. This is for instance the approach taken by Argos [13] or the reactive programming language Esterel [2].

We handle these conflicts semantically. In case of a causal conflict, the transition is simply not taken. We accomplish this through *oracles* that predict which signals will be input from the environment or generated by the system in each step.

For each signal a that occurs negatively in the trigger of a transition, we introduce a new *oracle signal* $\widetilde{a}$ that replaces a in the trigger part of a transition

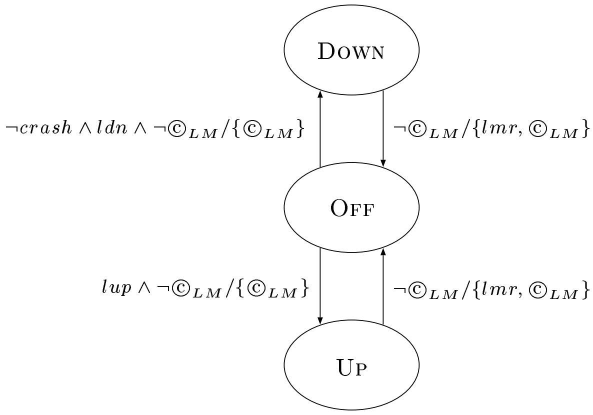


Figure 3. Motor control with priorities

label. For example, the transition label

$$\neg crash \wedge ldn$$

is transformed into

$$\widetilde{\neg crash} \wedge ldn$$

Oracle signals are never generated by transitions. At the beginning of each step in the execution of a chart, the system makes a guess about the input or generation of signals, and thus determines the value of the oracle signals. This guess introduces additional nondeterminism; for n oracle signals, there are 2^n possible oracle guesses. For those signals a that are predicted to become present, the oracle signal $\widetilde{a}$ is added to the input from the environment. Then, the step construction is similar to the unnegated case. In particular, the existence of fixpoints is guaranteed: since all negatively occurring signals are converted to oracles, and oracle signals can never be generated by the system, a choice made by the system can never be invalidated. Whereas in the unnegated case there always is a *least* fixpoint, we now get a set of *minimal* fixpoints. As we will see later, this introduces additional nondeterminism into a specification.

However, some fixpoints may be *inconsistent* in the following sense:

- A signal a is generated by the system, although the oracle forecasts its absence. In other words, a is in the event set, but not $\widetilde{a}$.
- A signal a that is predicted to be present, is neither input nor generated by the system. In other words, $\widetilde{a}$ is in the event set, but not a.

Thus, we must ensure that neither of these cases holds. The first condition can be checked locally when a transition is taken. We only have to extend the step function f from the unnegated case to:

$$g^S_{x,L}(Z) =_{def} \{z \in f^S_{x,L}(Z) | \; \forall a \in \pi_1(z) : \tilde{a} \in \pi_1(z)\}$$

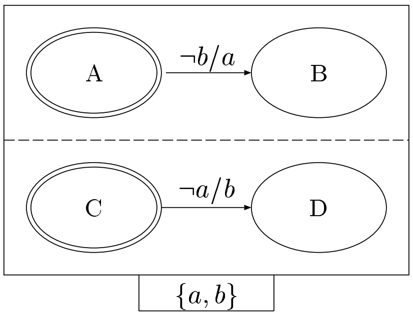
The second consistency condition, however, can only be checked once a fixpoint is reached. We therefore define the *self-fulfilling* fixpoints as those signal sets $SF \subseteq M$ where

$$\widetilde{s} \in SF \implies s \in SF$$

Note that while there are always fixpoints, the existence of *consistent* fixpoints is not guaranteed. An example is a μ-Chart with two states connected by the single transition $\neg a/a$. The modified transition label reads

$$\neg\widetilde{a}/a$$

Assume now that a is not input by the environment. When the oracle guesses a to become generated, i.e. $\widetilde{a}$ is added to the input set, a will not be generated, hence the fixpoint reached is not self-fulfilling. If, otherwise, the oracle guesses a to not be generated, then $\widetilde{a}$ is not added to the input set, and a will be generated, violating the local consistency condition. Since there is no consistent fixpoint, the system must remain in its current state. When a is input by the environment, the system will also remain in its current state. This time, however, there is a consistent fixpoint $\{a, \widetilde{a}\}$. In other words, the transition will never be taken.



Oracles	*Signal set*
$\neg\widetilde{a}, \neg\widetilde{b}$	a, b
$\widetilde{a}, \neg\widetilde{b}$	a
$\neg\widetilde{a}, \widetilde{b}$	b
$\widetilde{a}, \widetilde{b}$	—

Figure 4. Pathological case

Figure 4 shows another example. When no external input is provided, what should be the reaction of this chart? Our construction introduces two oracle signals, $\widetilde{a}$ and $\widetilde{b}$. The transition labels are then translated to $\neg\widetilde{a}/b$ and $\neg\widetilde{b}/a$, respectively. When neither a nor b is provided from the environment, the fixpoint construction results in the output signal sets shown to the right of the specification. For each possible oracle guess there is one solution. The solution in the first row violates local consistency, and must therefore be rejected. The solution in the last row is not self-fulfilling, and must be rejected, too. Thus, there are only two solutions: either only the upper transition is taken, resulting in the output signal set $\{b\}$, or only the lower transition is taken, resulting in output signal set $\{a\}$. Intuitively, there is a race between the two transitions; whichever transition is taken first, determines the reaction of the composed chart.

Thus, negation can introduce nondeterminism into a μ-Chart. In the older deterministic version of our dialect, [22], this chart would have to be rejected. The same holds for other deterministic dialects, like for instance Argos. Since pathological cases such as this one can be handled semantically in our dialect, we do not need to perform a static analysis of specifications to determine whether they must be rejected.

3 Reactive Behavior

In the previous section we have introduced a formal step semantics, which expresses the behavior of μ-Charts in one single instant of time. Reactive systems however have continuously to interact with the environment. Hence, their complete input/output behavior has to be described by the aid of communication histories.

We model the communication history of μ-Charts by streams carrying sets of signals. Mathematically, we describe the behavior of μ-Charts by stream processing functions. Hence, we briefly discuss the notion of streams and stream processing functions. For a detailed description we refer for example to [3].

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. In our setting, a stream processing function is a function with type $X^\omega \to X^\omega$.

To describe the complete input/output behavior, the semantic model associates with every chart S a set of stream processing functions:

$$[\![S]\!]_{io} : \wp(\wp(M)^\omega \to \wp(M)^\omega).$$

A function $f : \wp(M)^\omega \to \wp(M)^\omega$ is in $[\![S]\!]_{io}$, iff:

$$f(x\&s) = y\&g(s)$$

where

$$((y, S') \in [\![S]\!]x \vee y = \emptyset \wedge S' = S \wedge [\![S]\!]x = \emptyset) \wedge g \in [\![S']\!]_{io}$$

Note that at this point empty reactions of μ-Charts are resolved: the charts then remains in the current configuration, and the set of output signals is empty.

4 Conclusion and Future Work

The Statecharts dialect presented in this paper offers instantaneous feedback and nondeterminism. Both concepts are under discussion: [23] for example, argues that specifications with instantaneous feedback are unintuitive and difficult to understand. While this is certainly true for Statecharts with causality conflicts, where as default behavior the Statechart remains in its current state, it remains to be seen how often these cases occur in practice. Also, the delayed step semantics, as implemented for instance in Statemate, forces the designer to use a low-level, operative specification style with variable assignments and artificial sequentializations of component behaviors.

Leveson [8] rejects nondeterminism on the ground that the behavior of safety-critical systems should not allow arbitrary choices. While this may be true for specifications that are close to an implementation, we believe that in the early design phases nondeterminism is essential to avoid overspecification. Nondeterminism can also be used to model the system's environment.

Our language, while offering the main concepts of Statechart, does not yet cover the whole spectrum of practical applications. Current work is focused on extending the language to deal with integer-valued signals in the style of [20, 21], and with constructs for the abstract specification of real-time properties.

Further research is also necessary in the areas of code generation, compilation into hardware, and model checking techniques. In [17] we outline how deterministic μ-Chart specifications can be implemented in hardware. First steps towards model checking of our language are described in [16].

The obvious problem for these operational applications of our semantics is the handling of the oracle variables, since fixpoints can be reached that are not self-fulfilling. Simple interpreters would need backtracking to implement a full step semantics; a more sophisticated approach would be to use symbolic techniques like BDDs [5] and a μ-calculus formalization similar to the one in [18]. For interpreters without BDDs the combinatorial explosion resulting from the oracle variables can be reduced through lazy oracle guesses, as introduced in [12].

Nevertheless, for time critical industrial applications it will be necessary to reduce the nondeterminism caused by the oracle guesses. A medium-term goal is therefore the development of a refinement rule system in the tradition of the Focus rule system [4], where a refinement step reduces nondeterminism.

Acknowledgments

We would like to thank Herbert Ehler, Christian Prehofer, and the anonymous reviewers for many constructive remarks.

References

1. G. Berry. Real Time Programming: Special Purpose or General Purpose Languages. *Information Processing 89*, 1989.
2. G. Berry and G. Gonthier. The Esterel Synchronous Programming Language: Design, Semantics, Implementation. *scp*, 19(2):87–152, nov 1992.
3. M. Broy. Interaction Refinement - The Easy Way. In *Program Design Calculi*, volume 118 of *NATO ASI Series F: Computer and System Sciences*. Springer, 1993.

4. M. Broy and K. Stølen. Specification and Refinement of Finite Dataflow Networks – a Relational Approach. volume 863 of *Lecture Notes in Computer Science*, pages 247–267, 1994.
5. R. E. Bryant. Graph Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, 8(C-35):677–691, 1986.
6. D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231 – 274, 1987.
7. D. Harel and A. Naamad. The Statemate Semantics of Statecharts. *IEEE Transactions on Software Engineering Method*, 1996.
8. 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.
9. J.J.M. Hooman, S. Ramesh, and W.P. de Roever. A Compositional Axiomatization of Statecharts. *Theoretical Computer Science*, 101:289 – 335, 1992.
10. C. Huizing and W.-P. de Roever. Introduction to Design Choices in the Semantics of Statecharts. *Information Processing Letters*, 37, 1991.
11. i-Logix Inc., 22 Third Avenue, Burlington, Mass. 01803, U.S.A. *Languages of Statemate*, 1990.
12. K. Inoue, M. Koshimura, and R. Hasegawa. Embedding Negation as Failure into a Model Generation Theorem Prover. In D. Kapur, editor, *CADE-11*, number 607 in Lecture Notes in Artificial Intelligence, pages 400–415, 1992.
13. F. Maraninchi. Operational and Compositional Semantics of Synchronous Automaton Compositions. volume 630 of *Lecture Notes in Computer Science*, pages 550 – 564. Springer-Verlag, 1992.
14. F. Maraninchi and N. Halbwachs. Compositional Semantics of Non-deterministic Synchronous Languages. ESOP'96, 1996.
15. D. Nazareth, F. Regensburger, and P. Scholz. Mini-Statecharts: A Lean Version of Statecharts. Technical Report TUM-I9610, Technische Universität München, D-80290 München, 1996.
16. J. Philipps and P. Scholz. Formal Verification of Statecharts with Instantaneous Chain Reactions. 1997. TACAS'97.
17. J. Philipps and P. Scholz. System-Level Hardware Design with μ-Charts. 1997. CHDL'97.
18. J. Philipps and T. Yoneda. Symbolic Model Checking of Statecharts. Technical Report FTS-95-37, IEICE, 1995.
19. 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.
20. P. Scholz. An Extended Version of Mini-Statecharts. Technical Report TUM-I9628, Technische Universität München, D-80290 München, 1996.
21. P. Scholz. A Light-Weight Formalism for the Specification of Reactive Systems. 1996. SOFSEM'96.
22. P. Scholz, D. Nazareth, and F. Regensburger. Mini-Statecharts: A Compositional Way to Model Parallel Systems. 1996. PDCS'96.
23. M. von der Beeck. A Comparison of Statecharts Variants. volume 863 of *Lecture Notes in Computer Science*, pages 128 – 148. Springer, 1994.