Formal Verification of Statecharts with Instantaneous Chain Reactions *

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

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

Abstract. We present a method for symbolic model checking of μ-Charts, a Statecharts dialect with instantaneous broadcast communication. Due to this communication concept, μ-Charts satisfy the perfect synchrony hypothesis. The well-known causality conflicts that arise under instantaneous feedback from negative trigger conditions are resolved semantically through oracle signals. We have implemented a prototypical tool that translates μ-Charts specifications into μ-calculus formulae. These formulae are checked against temporal specifications using a μ-calculus verifier.

1 Introduction

Statecharts [4] are a visual specification language for reactive systems. They extend conventional state transition diagrams with structuring and communication mechanisms. Since there is also tool support through several providers like r-active or i-Logix (Statemate [7]), Statecharts have become quite successful in industry.

However, the semantics of Statecharts as used in Statemate is based on a delayed broadcast, which leads to a very operational, implementation-level specification style. For a modeling language for abstract requirements specifications more abstract approaches are desirable. These concepts are introduced in [12], where we present a dialect of Statecharts called μ-Charts. This dialect features a formal semantics for nondeterministic Statecharts with instantaneous feedback. It is an extension of Mini-Statecharts as presented in [11, 17]. Specification with instantaneous feedback fulfill the perfect synchrony hypothesis [1]. As noted in previous works on the semantics of Statecharts [5, 14], or Statechart-like languages like Argos [9, 10], instantaneous feedback can lead to causality conflicts when trigger events with negation are allowed.

Nevertheless, we prefer this kind of broadcasting, since delayed broadcast as used in Statemate is not a suitable communication concept for behavioral refinement. When refining a subchart to a set of more concrete subcharts, additional delays are introduced. Thus, the I/O-behavior of the Statechart changes.

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

Refinement rules would have to be more complex to compensate the additional delays. As observed in [6], this is not the case for instantaneous feedback.

If conflicts because of negated trigger conditions occur, they are handled semantically through oracle signals [12]. This is in contrast to Argos, which requires a static analysis to reject those Statecharts where a conflict might occur.

In this contribution, we demonstrate how to use the semantic model introduced in [12] as starting point for efficient formal verification, based on symbolic model checking techniques. We model chain reactions caused by instantaneous feedback as the least fixpoint of a transition relation. This chain of transitions is embedded in the outer transition relation that describes the observable behavior. As far as we know, this is apart from [13] the only work that deals with model checking of a specification formalism based on instantaneous broadcasting. The semantics presented in this paper is implemented in a prototypical tool that generates a set of μ-calculus formulae from a given specification. These relations can immediately be used as input for the model checking tool μ-cke [3].

This paper is organized as follows. In the sequel, we present our running example, a central locking system for cars. In Section 2 we introduce the Statecharts dialect used throughout this paper and give an abstract syntax for it. Section 3 shows how the formal semantics of μ-Charts can be described using μ-calculus formulae. In Section 4 we show some results of symbolic verification. Section 5 summarizes some experiences gained in this work.

Example: A Central Locking System

As running example we use a simplified specification of the central locking system for cars. This example was inspired by a case study from the local car industry. The corresponding μ-Chart is pictured in Figure 1; it specifies the locking system of a two-door car. Table 1 shows the signals used for the specification. We distinguish between signals that are input from the environment, so-called external signals, and signals that are generated by the system itself (internal signals). Notice that ellipses denote basic states of sequential automata while boxes denote states that are decomposed by other μ-Charts. Double frames denote default states.

Our central locking system consists essentially of three main parts: the CONTROL and the two door motors. These parts are composed in parallel. Locking and unlocking the doors leads to complex signal interactions. The default configuration of the system is that all doors are unlocked (UNLD) and both motors are OFF. Now the driver can lock the car either from outside by turning the key or from inside by pressing a button. Both actions generate the external signal c. The CONTROL generates the internal signals ldn and rdn and enters its locking state LOCKG, which is decomposed by the automaton in Figure 2.

Instantaneously, influenced by ldn and rdn, respectively, both motors begin to lock the doors by entering their DOWN states. Those states are decomposed by the sequential automata pictured in Figure 3. Thus, the motors are additionally in their START states. As the speed of the motors depend on external influences like their temperature, each motor either needs one or two time units to finish

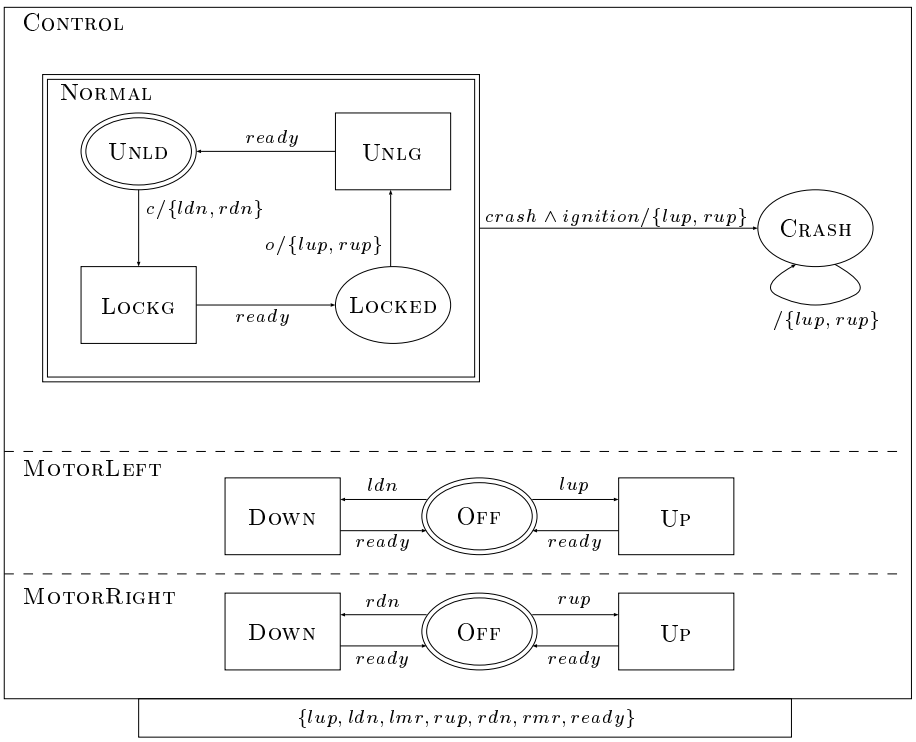

Figure 1. Central locking system

the lowering process. Only when both have sent their ready messages *lmr* and *rmr*, the Control enters the Both state and produces the signal *ready*. The effect of this signal is twofold: one the one hand the Control terminates itself immediately and enters the Locked state. On the other hand also both motors are triggered by this signal and are switched Off.

In our syntax communication is expressed by an explicit feedback operator. It is graphically indicated by the box sticked on the bottom of Figure 1.

Whenever the crash signal occurs and the ignition is on, the Control changes from the Normal mode in the Crash mode and generates the signals *lup* and *rup*. In Section 4 we will show that the crash signal indeed causes the doors to open.

2 Syntax

In this section, we formally define a textual syntax for μ-Charts. It corresponds to the graphical syntax used in the example. μ-Charts are based on Mini-Statecharts, as first presented in [11] and later refined in [15, 16, 17]. We only repeat those concepts that are a prerequisite for the extension to nondetermin-

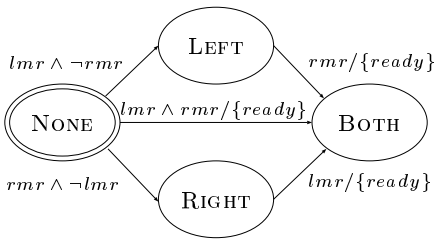

Figure 2. Decomposition of Lockg and Unlg

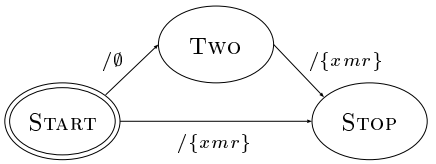

Figure 3. Decomposition of Down and Up, $x \in \{l, r\}$

ism and assume the reader to be familiar with the principles of hierarchical, interacting state machines.

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 chart, 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 for broadcasting. The inductive steps are motivated and defined in Sections 2.1 to 2.4.

2.1 Sequential Automata

Sequential automata Seq $(N, \Sigma, \sigma_d, \delta)$ are the basic elements of our Statecharts dialect. They consist of:

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. $\sigma_d \in \Sigma$ represents the default state.
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

Signal	*Meaning*	*Source*
crash	Crash sensor	External
o	Opened with external key	External
c	Closed with external key	External
ignition	Ignition on	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
ready	Un-/Locking process ready	Internal

Table 1. Signals used in the locking system

transition function that deals with arbitrary Boolean terms as trigger condition into a set-valued total function (see for example [17]).
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. Notice however that every single sequential automaton only is allowed to make one transition in one instant. The set of all system actions in one time unit is called a *step*.

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 [4].

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).

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 parallel composition 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, partial function ϱ, which is defined for those

states σ of the master that are further decomposed. The decomposition function ϱ yields the refining slave-chart. Suppose that $\mathsf{Seq}\ (N, \Sigma, \sigma_d, \delta)$ is a sequential automaton, then hierarchical decomposition is denoted by

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

where $\varrho : \Sigma \to \mathcal{S}$. Like other formal Statechart semantics [5, 9, 10], the approach presented here has no history states. It is possible to extend our semantics along the lines of [11]. 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 2.1 (Hierarchical Decomposition). As current system configuration, we assume that CONTROL is in the LOCKG state and that both motors are notifying the CONTROL that they have locked the doors. Thus, the current set of internal signals is $\{lmr, rmr\}$. Instantaneously, the *ready* signal is generated. We furthermore presume that exactly while all this happens, an external *crash* signal occurs. The overall signal set is then denoted by $\{lmr, rmr, ready, crash\}$. Hence, NORMAL changes its current state from LOCKG to LOCKED. In addition, the system moves from the NORMAL state to the CRASH state while generating the signal set $\{lup, rup\}$ if the ignition is on. Note that all actions come about instantaneously. Altogether, in the next instant, the CONTROL is in its 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 LOCKG to LOCKED, i.e. has not been immediately interrupted: we say that the *crash* signal has induced a *non-preemptive* interrupt. Notice however that though the NORMAL state still changed to LOCKED, finally both motors will open the doors. This property could be proven to be a theorem using the model checker. By strengthening the transitions in the slave chart with tests for the absence of signals, preemptive interrupts can be modeled as well.

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 [4], 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 [18]. For the deterministic version of our language [11, 15, 17], we provided different syntactic constructs with different communication timings. We believe that for nondeterministic, abstract specifications *instantaneous feedback* is the

proper concept, since it is better suited for behavioral refinement. Hence, we present only this operator here.

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

Example 2.2 (Feedback). When the chart is in the state UNLD, and the driver locks the door with the car key, then NORMAL moves to state LOCKG 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 c three transitions are taken, and at the same time the signals *ldn* and *rdn* are output.

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 $x \cup (z \cap L)$ is passed to S at the same instant.

3 Semantics

In this section we indroduce the transition relation for a μ-Chart. It is defined inductively following the syntactical structure of the language. The transition relations presented here are based on the semantics as presented in [12]. μ-Charts are synchronized by a global, discrete clock. Each transition relation formally denotes the relationship between two system configurations, i.e. the set of all currently valid control states of all sequential automata between two subsequent instants.

3.1 Preliminaries

Avoiding Multiple Transitions in one Step. As we deal with instantaneous feedback, more than one transition of different sequential automata can fire simultaneously. However, every single automaton only can make one step in one instant, i.e. no two consecutive transitions in a sequential automaton are taken in a step. This informal requirement has to be formalized in the automaton's transition relation. Furthermore, we have to ensure that only one branch of a nondeterministic choice in an automaton is taken in a step.

Both restrictions can be ensured using additional signals. For each sequential automaton Seq $(N, \Sigma, \sigma_d, \delta)$ we introduce a signal $©_N$. Informally, this is a copyright on transitions of the automaton signaling that N already made a step. When the signal is not present, the automaton may yet make a transition, whereupon it will generate $©_N$. If it is already present, the automaton has to

stay in its current state. The need for this signal will become clearer when we later introduce broadcast communication. The copyright signals are introduced in the following way. Each transition c/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 $©_N$.

We assume all signals $©_N$ to be disjoint from signals in M and define $M_©$ by $M \cup \{©_N \mid N \in Ident\}$.

Negation in Trigger Expressions. 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 a static analysis of the chart, which might reject charts that do not really have causality conflicts. This is for instance the approach taken by Argos [9] 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 *oracle signals* that predict the presence or absence of a given signal in a step. For each signal a that occurs negatively in the trigger of a transition, we introduce a new signal $\widetilde{a}$ that replaces a in the trigger part of a transition label. We define $\widetilde{M}$ to be $M \cup \{\widetilde{a} \mid a \in M\}$. However, oracle signals can cause the following two inconsistencies:

- A signal a is generated by the system or input from the environment, although the oracle forecasts its absence. In other words, a is in the signal 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 signal set, but not a.

The requirement to avoid these inconsistencies is formally expressed by:

$$Consistence(x, y, o) \equiv (\bigwedge_{s \in x \cup y} s \in o) \wedge (\bigwedge_{s \in o} s \in x \cup y)$$

where x, y, and o denote the sets of input, output, and oracle signals respectively. This technique is similar to that used in the bottom-up evaluation of logic programs with negation as presented in [8]. For a detailed discussion of this topic the interested reader is referred to [12].

3.2 Configurations

Configurations $c \in \mathcal{C}$ are defined inductively. The configuration of a sequential automaton is simply its current state. To denote an And-chart's $\mathsf{And}\ (S_1, S_2)$ configuration we use a tuple (c_1, c_2), where c_1 and c_2 are the configurations of the parallel components S_1 and S_2, respectively. The configuration of $\mathsf{Feedback}\ (S, L)$ is simply the configuration of S.

For hierarchical decomposition we need a slightly more subtle notation. The master is decomposed in $n =_{def} |\,\mathsf{dom}\ \varrho|$ slaves, where $\mathsf{dom}\ \varrho$ denotes the domain

of the partial function ϱ. The configurations of these slaves are denoted by $c_1, \ldots, c_n$, whereas the configuration of the master is denoted by c_m. The overall configuration of Dec $(N, \Sigma, \sigma_d, \delta)$ by ϱ is then the $(n+1)$-tuple $(c_m, c_1, \ldots, c_n)$.

In the sequel, we will formulate the transition relations for every single syntactic construct of the μ-Charts language. We have two different categories of predicates: one for initialization and one for the transition step from one configuration to the following. These predicates have the type:

$$Init_S : \mathcal{C} \to Bool$$
$$Trans_S : \mathcal{C} \times \wp(M_{©}) \times \mathcal{C} \times \wp(M_{©}) \times \widetilde{M} \to Bool$$

for every μ-Chart S. A predicate $Trans_S(c, x, c', y, o)$ is true whenever the current configuration of S is c and S can, stimulated by the set of input signal set x, reach the subsequent configuration c' in exactly one instant while producing the output signal set y. The set o includes those oracles that are needed for the treatment of negative signals in S.

3.3 Sequential Automata

Initially, a sequential automaton $S =_{def}$ Seq $(N, \Sigma, \sigma_d, \delta)$ is in its default state σ_d. For a set of input signals x coming from the environment, S generates a set of output signals y and changes its configuration, i.e. its current state from c to c':

$$Init_S(c) \equiv (c = \sigma_d)$$
$$Trans_S(c, x, c', y, o) \equiv (c', y) \in \delta(c, x \cup o)$$

3.4 Parallel Composition

The tuple (c_1, c_2) is the initial configuration of chart $S =_{def}$ And (S_1, S_2) whenever c_1, c_2 are the initial configurations of charts S_1, S_2, respectively:

$$Init_S((c_1, c_2)) \equiv Init_{S_1}(c_1) \wedge Init_{S_2}(c_2)$$

The formal semantics is defined by the following case distinction, which yields three mutually exclusive 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 even both may not react at all.

$$\begin{aligned} &Trans_S((c_1, c_2), x, (c_1', c_2'), y, o) \equiv \\ &\quad (\exists y_1, y_2. Trans_{S_1}(c_1, x, c_1', y_1, o) \wedge Trans_{S_2}(c_2, x, c_2', y_2, o) \wedge y = y_1 \cup y_2) \vee \\ &\quad ((\nexists y_2, c. Trans_{S_2}(c_2, x, c, y_2, o)) \wedge Trans_{S_1}(c_1, x, c_1', y, o) \wedge c_2' = c_2) \vee \\ &\quad ((\nexists y_1, c. Trans_{S_1}(c_1, x, c, y_1, o)) \wedge Trans_{S_2}(c_2, x, c_2', y, o) \wedge c_1' = c_1) \end{aligned}$$

The first conjunction represents the case when both charts S_1 and S_2 can react in their current configurations c_1 and c_2 on the current signals x. In this case the

overall reaction is simply denoted by the logical conjunction of both transition predicates $Trans_{S_1}$ and $Trans_{S_2}$. The other two conjunctions are true whenever only one of S_1 or S_2 can react on the current stimuli in its current configuration. Should none of the three terms be true, the overall transition predicate $Trans_S$ is false, i.e. S cannot react at all.

3.5 Hierarchical Decomposition

A decomposed chart $S =_{def} \mathsf{Dec}\ (N, \Sigma, \sigma_d, \delta)\ \mathsf{by}\ \varrho$ is in its initial configuration iff the master $A =_{def} \mathsf{Seq}\ (N, \Sigma, \sigma_d, \delta)$ and all existing slaves $\{S_1, \ldots, S_n\} =_{def} \mathsf{dom}\ \varrho$ are in their initial configurations:

$$Init_S((c_m, c_1, \ldots, c_n)) \equiv Init_A(c_m) \wedge \bigwedge_{i=1}^{n} Init_{S_i}(c_i)$$

To define the step relation for the decomposition, we distinguish four mutually exclusive cases. The first case occurs whenever the current state c_m of the master is refined by a slave S_i (in this case $\varrho(c_m)$ is defined, i.e. $c_m \in \mathsf{dom}\ \varrho$ and $\varrho(c_m) = S_i$), and both A and S_i can react. All other, currently not active slaves keep their current configuration $\bigwedge_{j \neq i} c_j = c'_j$. Generated signals of both master and active slave are collected: $y = y_s \cup y_m$. Notice that whenever the transition predicate $Trans_A$ of the master is true, the slave is initialized through the predicate $Init_{S_i}(c'_i)$. This first case is formally denoted by:

$$\begin{aligned} &Trans^1_S((c_m, c_1, \ldots, c_n), x, (c'_m, c'_1, \ldots, c'_n), y, o) \equiv \\ &\quad \exists y_m, y_s, c.Trans_A(c_m, x, c'_m, y_m, o) \wedge \\ &\qquad c_m \in \mathsf{dom}\ \varrho \wedge S_i = \varrho(c_m) \wedge Trans_{S_i}(c_i, x, c, y_2, o) \wedge \\ &\qquad y = y_s \cup y_m \wedge Init_{S_i}(c'_i) \wedge \textstyle\bigwedge_{j \neq i} c_j = c'_j \end{aligned}$$

Here both master and slave can react on the current set of input stimuli. In this case, the master *interrupts* the slave's reaction. Remember that our semantics deals with non-preemptive interruption: so the slave still can terminate its current action, i.e. generate all output signals y_s. However, even then it will be re-initialized.

Whenever the master's current state c_m is not decomposed ($c_m \notin \mathsf{dom}\ \varrho$), all slaves stay in their current configurations ($\bigwedge_{i=1}^{n} c_i = c'_i$) and only the master itself reacts:

$$\begin{aligned} &Trans^2_S((c_m, c_1, \ldots, c_n), x, (c'_m, c'_1, \ldots, c'_n), y, o) \equiv \\ &\quad Trans_A(c_m, x, c'_m, y, o) \wedge c_m \notin \mathsf{dom}\ \varrho \wedge \textstyle\bigwedge_{i=1}^{n} c_i = c'_i \end{aligned}$$

If however a slave exists but is not able to make a step, again only the master reacts but now the current slave S_i is initialized and all other slaves do not change their configuration:

$$\begin{aligned} &Trans^3_S((c_m, c_1, \ldots, c_n), x, (c'_m, c'_1, \ldots, c'_n), y, o) \equiv \\ &\quad Trans_A(c_m, x, c'_m, y, o) \wedge c_m \in \mathsf{dom}\ \varrho \wedge S_i = \varrho(c_m) \wedge \\ &\qquad \nexists y_s, c'_s.Trans_{S_i}(c_i, x, c'_s, y_s, o) \wedge Init_{S_i}(c'_i) \wedge \textstyle\bigwedge_{j \neq i} c_j = c'_j \end{aligned}$$

Finally, if the master cannot react, but the current slave S_i can, we have:

$$\begin{aligned}
&Trans^4_S((c_m, c_1, \ldots, c_n), x, (c'_m, c'_1, \ldots, c'_n), y, o) \equiv \\
&\quad \nexists y_m, c'_m . Trans_A(c_m, x, c'_m, y_m, o) \wedge \\
&\quad\quad c_m \in \mathsf{dom}\ \varrho \wedge S_i = \varrho(c_m) \wedge \textstyle\bigwedge_{j \neq i} c_j = c'_j \wedge Trans_{S_i}(c_i, x, c'_i, y, o)
\end{aligned}$$

Overall, the complete transition relation is the disjunction of these cases:

$$\begin{aligned}
&Trans_S((c_m, c_1, \ldots, c_n), x, (c'_m, c'_1, \ldots, c'_n), y, o) \equiv \\
&\quad Trans^1_S((c_m, c_1, \ldots, c_n), x, (c'_m, c'_1, \ldots, c'_n), y, o) \vee \\
&\quad Trans^2_S((c_m, c_1, \ldots, c_n), x, (c'_m, c'_1, \ldots, c'_n), y, o) \vee \\
&\quad Trans^3_S((c_m, c_1, \ldots, c_n), x, (c'_m, c'_1, \ldots, c'_n), y, o) \vee \\
&\quad Trans^4_S((c_m, c_1, \ldots, c_n), x, (c'_m, c'_1, \ldots, c'_n), y, o)
\end{aligned}$$

The predicate $Trans_S$ is false iff neither master nor slave can react to the current input.

3.6 Broadcast Communication

The initialization predicate for $S = \mathsf{Feedback}(R, L)$ is defined as:

$$Init_S(c) \equiv Init_R(c)$$

The transition relation $Trans_S$ is built up from a number of auxiliary predicates. As we deal with a chain reaction when defining the semantics of the instantaneous feedback, we first have to fix the termination of this reaction. It terminates when in the current configuration c the chart S cannot react any more on the current input stimuli x:

$$Term_S(c, x, o) \equiv \nexists y, c' . Trans_R(c, x, c', y, o)$$

The predicate $Cone_S$ constructs the set of all intermediate points in the chain reaction by the μ-calculus formula:

$$\begin{aligned}
&Cone_S(c, x, c', y, o) \equiv \\
&\quad \mu\Psi.(Trans_R(c, x, c', y, o) \vee \\
&\quad\quad \exists x', y', y'', c'' . \Psi(c, x, c'', y'', o) \wedge Trans_R(c'', x', c', y', o) \wedge \\
&\quad\quad x' = x \cup (y'' \cap L) \wedge y = y' \cup y'')
\end{aligned}$$

In order to verify whether $Cone_S(c, x, c', y, o)$ yields true we have to verify whether either of the two following possibilities is true. The first alternative is that c and c' represent two subsequent configurations, i.e. are reachable in one step: $Trans_R(c, x, c', y, o)$. Otherwise, it has to be verified whether c and c' can be reached via an intermediate configuration c''. All reachable configurations from c are computed by applying the least fixpoint operator μ on predicate Ψ. Notice that the external stimuli x are available during the whole chain reaction and that only those internal signals which occur in L can be fed back: $x' = x \cup (y \cap L)$. The overall transition relation of S is then defined as:

$$\begin{aligned}
&Trans_S(c, x, c', y, o) \equiv \\
&\quad Cone_S(c, x, c', y, o) \wedge Term_S(c', x \cup (y'' \cap L), o)
\end{aligned}$$

4 Symbolic Verification

The transition relations defined in the previous section are partial. When a chart cannot react to its current input, the relation is undefined. Intuitively, in this case however the chart should stay in its current configuration. The execution of a chart S is therefore defined over the following, total, step relation:

$$\begin{aligned} &Step_S(c, x, c', y) \equiv \\ &\quad (\exists o. Trans_S(c, x, c', y, o)) \vee \\ &\quad (\forall c'', y'', o. \neg Trans_S(c, x, c'', y'', o) \wedge c = c' \wedge y = \emptyset \wedge \\ &\quad Consistence(x, y, o)) \end{aligned}$$

The oracle signals in o nondeterministically predict the absence or presence of signals in a step. This prediction is needed for the proper treatment of negative trigger expression in sequential automata. Of course, such a guess might lead to inconsistencies, if in fact a signal predicted to be present is neither input from the environment, nor generated by the system, or vice versa. Such inconsistencies are detected with the predicate *Consistence* defined in Section 3.1. They can only occur with instantaneous feedback of a signal that can be generated in one subchart, and whose absence is checked in another subchart. If there is no consistent oracle guess, the chart will remain in its current configuration.

Experimental Results

Since all sets occurring in the formulae of the previous section are finite, it is straightforward to translate them into propositional μ-calculus. We have developed a prototypical compiler that translates a given textual μ-Charts specification into a set of μ-calculus formulae following the above mentioned semantical definitions. This first version of the compiler has been written in the language Perl 5.0 [19].

The μ-calculus formulae generated by the compiler are the input for the μ-calculus verifier μ-cke [3].

We have tried to prove that whenever the central locking system is not yet in its CRASH mode and a crash signal occurs while the ignition is on, both motors will open their doors:

$$\begin{aligned} &\mathsf{AG}(\neg in(\textsc{Crash}) \wedge crash \wedge ignition \wedge \phi \Rightarrow \\ &\quad \mathsf{AF}(in(\textsc{MotorLeft.Up})) \wedge \mathsf{AF}(in(\textsc{MotorRight.Up}))) \end{aligned}$$

where ϕ is a predicate that is true whenever all internal signals (see Table 1) are absent.

However, this property turned out to be false for the specification in Figure 1. The reason is the following: Remember that each motor can nondeterministically need either one or two steps to terminate whenever it is in its DOWN or UP state.

Assume that CONTROL is in its LOCKG state, both motors are in their DOWN, START configuration, and the *crash* signal occurs. If for instance the right motor needs two steps to lock the door and the left motor one step, the LOCKG changes

from NONE to LEFT. The overall signal set (including internal communication) now is $\{crash, ignition, lmr\}$. Although we allow non-preemptive interrupt, *ready* was not generated yet. The CONTROL changes to its CRASH mode and *ready* cannot be produced anymore. As a consequence, both motors will "starve" in their DOWN states and never will be triggered by lup or rup.

This problem can be avoided by substituting *ready* by xmr and xmr (with $x \in \{l, r\}$), respectively (see Figure 4). In this case the termination of the motors' lowering process does not any longer depend on the existence of the signal *ready* but the motors can terminate themselves.

Figure 4. Motor version 2, $x \in \{l, r\}$

In case of an accident, unlocking the doors is a time critical task. However, notice that though the motor version in Figure 4 *eventually* allows to open the central locking, it still may need four steps to do that. This time is caused by is the nondeterministic behavior of the motors. When in OFF the left motor for instance can follow either edge because, if the current signal set is $\{lup, ldn\}$, both trigger conditions are valid. Hence, before following the opening command lup it first can decide to act upon the command ldn. In order to avoid this delay, we must give the signals lup and rup a higher priority than the signals ldn and rdn by additional negative triggers as shown in Figure 5.

Figure 5. Motor version 3, $x \in \{l, r\}$

In this case, each motor needs at most three steps to enter its UP state if it is in its DOWN state and exactly one step, if it is OFF. It may be interesting for the reader that we started to prove at first the above mentioned property with the system specification as pictured in Figure 1 and only discovered later applying the model checking techniques presented in this paper that our original specification could not keep this restriction.

The step relation $Steps$ for the overall specification requires 3877 BDD nodes; the initialization predicate requires another 21 nodes. We expect that the number of nodes can be further reduced with a different encoding of the configurations. For hierarchically decomposed charts, the configuration is the algebraic product of the configurations of the master and all slaves. This is redundant,

since only one slave can be active at a time. The μ-calculus verifier μ-cke does not yet support the encoding of algebraic sums. In any case, however, the product encoding is necessary when slaves shall remember their configuration instead of being re-initialized when entered.

In order to test the scalability of our approach we have recently verified a more complex version of the locking system. Its specification contains about three times as many states and transition as the one in Figure 1. However, it turned out that this larger example is already the limit for our prototypical tool. Since the critical factor in the verification turned out to be the size of the intermediate BDDs, we believe that this is because of the very general nature of the verifier μ-cke. A dedicated, optimized implementation of the needed verification algorithms could alleviate this problem.

5 Conclusion

The Statecharts dialect presented in this paper offers instantaneous feedback and nondeterminism. We have shown how to deal with both concepts formally and demonstrated that model checking for specifications with instantaneous chain reactions is possible. We demonstrated our approach by an example. Once more it turned out that formal verification is of great help in debugging specifications for time- and safety-critical reactive systems. The results presented in Section 4 show that it is difficult to trust in a formal specification without proving central system properties.

We expect that in our framework larger specifications can be verified than in approaches without instantaneous feedback. The reason is that in the feedback definition intermediate configurations that occur only during chain reactions are hidden through the fixpoint construction. With other communication mechanisms, these intermediate configurations remain visible. Moreover, we believe that specifications with instantaneous broadcasting are more concise than those written in e.g. the Statemate dialect. Future work will focus upon the treatment of larger case studies to examine whether these conjectures hold.

Finally, it remains to be seen whether BDD-based symbolic verification techniques are indeed the best approach for model checking μ-Charts. For instance, in our example only 22 configurations are reachable. It is possible that non-symbolic techniques are more efficient for μ-Chart specifications. However, the high-level input language of μ-cke turned out to be very helpful for rapid prototyping of our language definition, semantics, and verification approach.

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. Technical Report 842, INRIA, 1988.

3. A. Biere. Eine Methode zur μ-Kalkül-Modellprüfung. Slides for the AKFM from 23.05.96, GI/ITG-Fachgespräch "Formale Beschreibungstechniken für verteilte Systeme" (in German), 1996.
4. D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8:231 – 274, 1987.
5. J.J.M. Hooman, S. Ramesh, and W.P. de Roever. A Compositional Axiomatization of Statecharts. *Theoretical Computer Science*, 101:289 – 335, 1992.
6. C. Huizing and W.-P. de Roever. Introduction to Design Choices in the Semantics of Statecharts. *Information Processing Letters*, 37, 1991.
7. i-Logix Inc., 22 Third Avenue, Burlington, Mass. 01803, U.S.A. *Languages of Statemate*, 1990.
8. 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.
9. F. Maraninchi. Operational and Compositional Semantics of Synchronous Automaton Compositions. volume 630 of *Lecture Notes in Computer Science*, pages 550 – 564. Springer-Verlag, 1992.
10. F. Maraninchi and N. Halbwachs. Compositional Semantics of Non-deterministic Synchronous Languages. ESOP'96, 1996.
11. 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.
12. J. Philipps and P. Scholz. Compositional Specification of Embedded Systems with Statecharts. 1997. TAPSOFT/FASE'97.
13. J. Philipps and T. Yoneda. Symbolic Model Checking of Statecharts. Technical Report FTS-95-37, IEICE, 1995.
14. 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.
15. P. Scholz. An Extended Version of Mini-Statecharts. Technical Report TUM-I9628, Technische Universität München, D-80290 München, 1996.
16. P. Scholz. A Light-Weight Formalism for the Specification of Reactive Systems. 1996. SOFSEM'96.
17. P. Scholz, D. Nazareth, and F. Regensburger. Mini-Statecharts: A Compositional Way to Model Parallel Systems. 1996. PDCS'96.
18. M. von der Beeck. A Comparison of Statecharts Variants. volume 863 of *Lecture Notes in Computer Science*, pages 128 – 148. Springer, 1994.
19. L. Wall and R.L. Schwartz. *Programming in perl*. Carl Hanser, 1993.