Reconciling Statechart Semantics

Rik Eshuis [a]

[a] *Eindhoven University of Technology, Department of Technology Management,*
P.O. Box 513, 5600 MB Eindhoven, The Netherlands. Tel.: +31 40 2472391.
Fax.: +31 40 2432612.

Abstract

Statecharts are a visual technique for modelling reactive behaviour. Over the years, a plethora of statechart semantics have been proposed. The three most widely used are the fixpoint, STATEMATE, and UML semantics. These three semantics differ considerably from each other. In general, they interpret the same statechart differently, which impedes the communication of statechart designs among both designers and tools. In this paper, we identify a set of constraints on statecharts that ensure that the fixpoint, STATEMATE and UML semantics coincide, if observations are restricted to linear, stuttering-closed, separable properties. Moreover, we show that for a subset of these constraints, a slight variation of the STATEMATE semantics coincides for linear stuttering-closed properties with the UML semantics.

1 Introduction

Background. Statecharts are a popular visual technique for modelling the behaviour of reactive systems [22,44]. Statecharts were introduced in the eighties by Harel [16,17] for use in the structured analysis approach STATEMATE [24]. Quickly after their introduction they were adopted in several object-oriented design methods as well, notably OMT [41] and ROOM [42]. The notations of these and a few other OO methods have been merged into UML [43], which is currently the de facto standard for modelling software systems. Thus, nowadays several variants of statecharts exist.

Over the years, many formal semantics have been proposed for each of these statechart variants. For example, Von der Beeck [2] counted in 1994 around twenty different semantics for statecharts, not including OO variants. The

Email address: `h.eshuis@tue.nl` (Rik Eshuis).

Table 1
Main differences between the three statechart semantics

	fixpoint	Statemate	UML
response to input events	immediate	immediate	delayed
event processing	parallel, instantaneous	parallel, instantaneous	single, non-instantaneous
generated events	sensed in same step	sensed in next step	sensed in some subsequent step

adoption of statecharts in OO approaches, especially UML, has led to a further increase of statechart formalisations.

Despite this great number of different formalisations, there is consensus about the ingredients that an actual semantics should have. All proposals use configurations, events, and steps to define the execution semantics of a statechart. A configuration is a valid global state of a statechart. While the system is in a configuration, events can occur. In response, the system leaves the current configuration by taking a set of transitions, called a step, and enters a new configuration. Moreover, by taking this step new events can be generated to which the system should respond in either the same or a next step.

All statechart formalisations share these features, but each formalisation uses its own assumptions in defining an actual execution semantics. Fortunately, the different proposals can be classified in three mainstream approaches (see Table 1). Proposals of the first approach are based on the fixpoint semantics for statecharts, initially proposed by Pnueli and Shalev [40]. Key feature of this semantics is that the system responds immediately to new input events and moreover responds infinitely fast. This feature is called the perfect synchrony hypothesis and was first introduced for the synchronous language Esterel [4]. Another peculiar feature of the fixpoint semantics is that events generated in a step are sensed and processed in the same step. Over the years, several extensions and refinements of the fixpoint semantics have been proposed (e.g. [31,32,34]). We are unaware of any commercial software tool implementing this semantics.

Proposals of the second approach focus on the semantics as implemented in the Statemate tool set [20,24]. Statemate supports two main semantics: the system can react in response to either a tick of the clock or to some new input events. In this paper, we only consider the latter semantics, which satisfies the perfect synchrony hypothesis. A peculiar feature of Statemate, distinguishing it from the fixpoint semantics, is that events generated in a step are sensed and processed only in the next step. The initial semantics of

STATEMATE statecharts was defined in prose by Harel and Naamad [21]. Subsequently, several researchers have presented formalisations of this semantics (e.g. [8,15,37]). The semantics of RSML statecharts [30] is a slight variation of the STATEMATE semantics [21,17].

The last group of formalisations focuses on statecharts for object-oriented systems. This group is expanding quickly due to the incorporation of statecharts in UML, the emerging de facto standard for modelling software systems. The main distinction between UML and the other two semantics is that UML does not use the perfect synchrony hypothesis [17,44]. In particular, taking a step takes time and during this time the next events can already arrive. To avoid that these are lost, they are stored in a queue. The system processes events from the queue one by one and responds to each event by taking a step. In contrast, in the fixpoint and STATEMATE semantics events are processed in parallel. Several UML tools like Rational Rose and Rhapsody implement the UML semantics or a slight variation thereof [18,19]. The official semantics is defined in prose in the UML standard text [43]. Several formalisations of the UML semantics have been proposed (e.g. [3,9,28]).

Problem. The existence of these different statechart formalisations can lead to a Babel-like confusion, because the same statechart can be interpreted completely differently under different semantics. This confusion impedes the communication of the meaning of statechart designs, since that meaning largely depends on the actual semantics the viewer (not the designer) is using. In addition, it hampers the exchange of statechart designs among different software tools. In the last years such exchanges are occurring frequently, leveraged by the development of XML-based languages. Consequently, it is for example possible to simulate a statechart design with one tool (because that tool has a nice animation facility), yet generate code with another tool (because that tool generates high quality code).

To illustrate this possible confusion, consider the simple statechart in Figure 1 (see Section 2 for definitions). If in the initial configuration events e and f occur, then

- under the fixpoint semantics, initially step $\{\mathsf{s1}\longrightarrow\mathsf{s2}, \mathsf{s3}\longrightarrow\mathsf{s4}, \mathsf{s5}\longrightarrow\mathsf{s6}\}$ is taken, since generated internal event i is sensed immediately,
- under the STATEMATE semantics, initially step $\{\mathsf{s1}\longrightarrow\mathsf{s2}, \mathsf{s5}\longrightarrow\mathsf{s6}\}$ is taken, since i is sensed in the next step, and
- under the UML semantics, initially either step $\{\mathsf{s1}\longrightarrow\mathsf{s2}\}$ is taken, if e is processed before f, or $\{\mathsf{s5}\longrightarrow\mathsf{s6}\}$, if f is processed before e.

Thus, under the three semantics different initial steps are taken in response to the same input events.

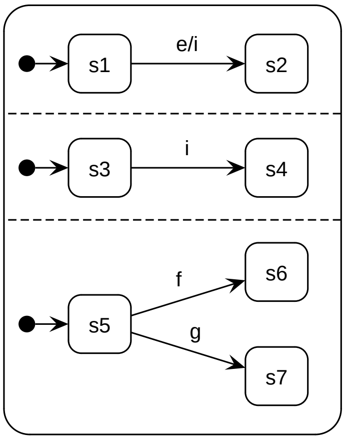

Fig. 1. Statechart for which the fixpoint, STATEMATE, and UML semantics exhibit different behaviour

Goal and approach. The goal of this paper is to identify a subset of statecharts for which the three mainstream semantics yield similar behaviour, that is, observations of statechart behaviour cannot distinguish between the three semantics. In this paper, observations are properties expressed in temporal logic. Naturally, observations cannot refer to events, since these are treated differently, as explained in Table 1. For example, in Figure 1, external event f and internal event i can occur simultaneously under the fixpoint and the UML semantics, but not under the STATEMATE semantics. Neither can observations refer to steps, since under the three semantics different steps can be taken in response to the same input events, even for simple examples like Figure 1. But for Figure 1, the end configuration eventually reached by taking these different steps is the same: {s2,s4,s6}. Thus, the net effect of the different reactions is the same under all three semantics, i.e. the same end configuration is reached eventually.

Nevertheless, observations cannot refer directly to end configurations, since these are reached through different steps under the three semantics. For example, in Figure 1, states s2 and s4 are entered simultaneously under the fixpoint semantics, but not under the other two semantics. Observations that refer to configurations can detect such differences. We therefore only consider observations that refer to states that belong to the same sequential component of a statechart. A sequential component identifies a maximal subset of the statechart that contains no parallelism. Figure 1 has three sequential components that act in parallel. Section 5.1 defines sequential components.

But even if observations refer to sequential components, only for a subset of statecharts the three semantics yield similar behaviour. For this subset of behaviour, which is identified by means of constraints on the statechart syntax (and one constraint on the UML semantics) in Section 4, we show that the three semantics are equivalent for linear, stuttering closed properties that are separable. Linear properties are expressed in past linear temporal logic (PLTL) [35]. A property is stuttering closed if the next time operator and its past time equivalent are not used [27]. A property is separable if it is equivalent to a separated property [39]. A property is separated[1] if it is a boolean combination of temporal formulas each of which only refers to a sequential component of the statechart (formal definitions can be found in Section 5). Regardless of which particular semantics is used, the outcome of verifying a linear, stuttering-closed, separable property is the same.

Unfortunately, the practical value of this result seems rather limited, since not every property is separable, and testing for separability requires finding for each property an equivalent separated one, which can be very hard. Moreover, quite a number of constraints on statecharts are used to prove the result.

However, we also show that for linear, stuttering-closed properties, the UML semantics is equivalent to a slightly modified version of the STATEMATE semantics, in which external events occur one by one. In particular, the same steps are taken under both semantics. For this much stronger result, much less constraints are needed. Its practical value is that the single-event STATEMATE semantics can be used to prove properties of the UML semantics. For the STATEMATE semantics, already efficient verification approaches exist [6,14]. The UML semantics uses a queue, which makes verification less efficient. In earlier work, we demonstrated this by comparing use of a STATEMATE-like semantics with that of a UML-like semantics for verifying UML activity diagrams [13].

The equivalence result does not extend to branching temporal logics like CTL, since under the UML semantics an event e that occurs is not immediately responded to. Consequently, a previous event that still awaits processing in the queue, may disable the effect of e. For example, if in Figure 1 event f occurs in state s5, then under the fixpoint and the STATEMATE semantics always s6 is reached next. But under the UML semantics, state s6 might not be reached, since g might have occurred before and still be in the queue; in that case, s7 is reached next, and the effect of f is disabled. At the end of Section 5.5, we discuss this topic in more detail.

To summarise, we list the restrictions used as well as the reason why they are needed:

- Constraints on statecharts, to rule out differences in behaviour for the three semantics.
- Sequential components and stuttering-closed, separable properties, because

[1] The notion of separability stems from partial order verification [38,39], but there a formula is separated if the components it refers to are orthogonal. However, sequential components can overlap, because a state can belong to multiple components.

Table 2
Omitted statechart constructs

construct	fixpoint	STATEMATE	UML
compound events	x	x	
negated events	x	x	
activities		x	x
synchronous calls			x
deferred events			x
dynamic choice points			x

the relation between the fixpoint, STATEMATE and the single-event STATEMATE semantics only holds for the end configuration of a reaction, not for the specific steps taken. To relate the single-event STATEMATE and UML semantics, sequential components are not needed.

- Linear properties, because under the UML semantics old events can disable the effects of current events, as explained above. Observations under the fixpoint, STATEMATE, and the single-event STATEMATE semantics can also refer to branching, stuttering-closed, separable properties, expressed in CTL [5].

Since we study statecharts that are meaningful under all three semantics, we only consider statechart constructs that are allowed by each of the three semantics. Table 2 shows the constructs we do not consider. We focus on the behaviour of a single statechart (as is done in the fixpoint and STATEMATE semantics[2]) whose transitions only contain single event triggers (as in UML). Since we consider a restricted set of statechart constructs, our formalisations of the different semantics are more simple than the existing statechart formalisations mentioned above. For example, for the fixpoint semantics an important problem is the treatment of negated events, and for the UML semantics the handling of synchronous calls between different statecharts, but we do not address these problems in this paper. Furthermore, to simplify the exposition, we consider for our main theorems statecharts without data and guard conditions. In Section 6, we discuss how the results can be extended to deal with statecharts with compound and negated events, data, guard conditions, history and deep history connectors.

[2] In STATEMATE, a system of multiple statecharts is similar to a global statechart in which all original statecharts act in parallel [21]. This precludes dynamic instantiation of statecharts, which is allowed in UML. Hence we focus on a single statechart only.

Related work. The relation between the different statechart semantics has received but little attention in the literature. Von der Beeck [2] gives an overview of twenty statechart semantics, including the fixpoint and STATEMATE semantics, but does not relate any of these formally. Crane and Dingel [7] give an informal overview of differences between UML and STATEMATE statecharts by means of examples. Maggiolo-Schettini et al. [34] compare different statechart step semantics, but these are all variants of the fixpoint semantics. Huizing and Gerth [26] compare high-level design choices made in different reactive semantics, among others the fixpoint, STATEMATE and Esterel [4] semantics. Next, there is related work [1,33] that studies the differences between the fixpoint semantics and Esterel semantics in a formal setting.

Compared to these other papers, the main contribution of this paper is the formal comparison of the three mainstream statechart semantics for statecharts that are meaningful under all three semantics, and in particular concrete example statecharts that illustrate the differences between the different semantics, a set of mostly syntactic constraints to rule out such differences, and theorems relating the different semantics to each other.

Structure of this paper. Section 2 recalls the syntax of statecharts and defines the notions of configuration and step, which are pivotal to any statechart semantics. Section 3 gives formalisations of the fixpoint, STATEMATE, and UML semantics of statecharts. Section 4 defines constraints on statecharts that are used in the next section to prove that a statechart exhibits similar behaviour under the three semantics. The applicability of the constraints is evaluated on a few example statechart designs taken from the literature. Section 5 shows that a statechart satisfying the constraints has stuttering similar runs under the different semantics. For the single-event STATEMATE and UML semantics, we even show that under both semantics the same steps are taken. Section 6 sketches how the results can be extended to deal with statecharts having guard conditions, compound and negated events, history and deep history connectors, and data, including assignment actions. We end with conclusions in Section 7.

2 Statecharts

We recollect some standard definitions of statecharts, mostly taken from Harel et al. [23] and Pnueli and Shalev [40], and introduce a few new ones for the semantics of transitions. For an introduction to the visual syntax of statecharts, we refer to Harel [16]. Figure 2 shows an example statechart, describing the behaviour of a controller for a turnstile that gets unblocked if the user en-

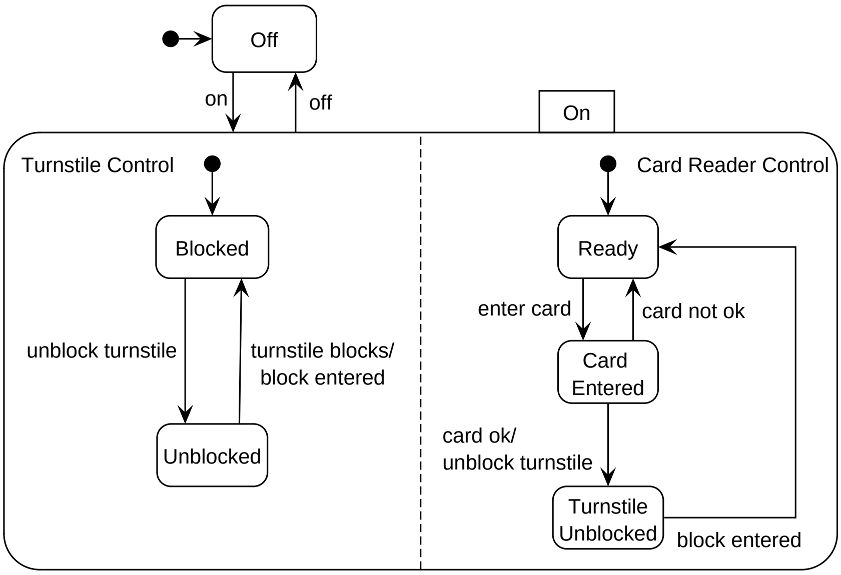

Fig. 2. Statechart of turnstile [44]

ters a valid card [44]. Details of this statechart are explained throughout the remainder of this section as illustration of the different statechart concepts.

Formally, a statechart SC is a tuple $(\mathcal{S}, \mathcal{T}, \mathcal{E})$, with $\mathcal{S}$ a set of states, $\mathcal{T}$ a set of transitions that connect the states, and $\mathcal{E}$ the set of events that transitions are triggered by. Set $\mathcal{E}$ is partitioned into sets $\mathcal{E}^{ext}$ and $\mathcal{E}^{int}$. Set $\mathcal{E}^{ext}$ contains all external events, which are generated by the environment of the system, while set $\mathcal{E}^{int}$ contains all internal events, which are generated by transitions in $\mathcal{T}$. For the example, $\mathcal{E}^{ext} = \{\mathsf{on}, \mathsf{off}, \mathsf{turnstile\ blocks}, \mathsf{enter\ card}, \mathsf{card\ not\ ok}, \mathsf{card\ ok}\}$ while $\mathcal{E}^{int} = \{\mathsf{unblock\ turnstile}, \mathsf{block\ entered}\}$.

In the next subsections, we discuss the syntax and semantics of states and transitions, respectively.

2.1 States

2.1.1 Syntax

Function $children : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{S})$ defines for each state s its immediate substates. If s is a child of s', we call s' the parent of s. By $children^*$ and $children^+$ we denote the reflexive-transitive and transitive closure of *children*, respectively. If $s \in children^*(s')$, we say that s is a descendant of s' and that s' is an ancestor of s. If s is ancestor or descendant of s', then s and s' are ancestrally related. In the example, Blocked is a child of Turnstile Control, which in turn is a child of On.

There are several types of state. If s has no children, so $\mathit{children}(s) = \emptyset$, then s is a BASIC state. Otherwise, s is composite. A composite state indicates either sequential (OR) or parallel (AND) behaviour. If the system is in an OR state, it is in exactly one of its children (so OR is actually XOR). If the system is in an AND state, it is also in every child of it. Function $\mathit{type} : \mathcal{S} \rightarrow \{\text{BASIC, AND, OR}\}$ assigns to each state its type. In the example, Blocked is BASIC, Turnstile Control is OR, while On is AND.

A special state is the root state of the statechart, denoted $\mathit{root} \in \mathcal{S}$, which has no parent. We require that *root* has type OR. Usually, *root* is not shown in the visual syntax.

Next, we require that every state $s \in \mathcal{S}$, except *root*, has a single parent, and that *root* is ancestor of every state in $\mathcal{S}$. These constraints ensure that states are structured in a rooted tree. Leaves of the tree are the basic states.

Function $\mathit{default} : \mathcal{S} \rightarrow \mathcal{S}$ identifies for each OR state s one of its children as the default state: $\mathit{default}(s) \in \mathit{children}(s)$. For example, the default state of Turnstile Control is Blocked. If a transition t enters s but does not explicitly enter any of its children, then t enters $\mathit{default}(s)$.

2.1.2 Semantics

For a set X of states, the least common ancestor (lca) of X, denoted $\mathit{lca}(X)$ is the state x such that:

- $X \subseteq \mathit{children}^*(x)$
- For every $y \in \mathcal{S}$ such that $X \subseteq \mathit{children}^*(y)$, we have that $x \in \mathit{children}^*(y)$.

Every set of states has a unique least common ancestor. For example, the lca of Blocked and Card Reader Control is On, while the lca of Blocked and Off is *root*.

Two states x, y, are orthogonal, written $x \perp y$, if x and y are not ancestrally related, and their lca is an AND state. In the example, Blocked and Card Reader Control are orthogonal.

A set X of states is consistent if for every x, $y \in X$, either x and y are ancestrally related or $x \perp y$. A consistent set X is maximal if for every state $s \in \mathcal{S} \backslash X$, $\{s\} \cup X$ is not consistent. A maximal consistent set of states is called a *configuration*. Configurations represent the valid global states of the statechart. In the example, {Blocked,Turnstile Control,Card Entered,Card Reader Control,On,*root*} is a configuration, but {Unblocked,Turnstile Control,Card Reader Control,On,*root*} is not, since no child of Card Reader Control is included.

Given a consistent set X of nodes, the default completion $dcomp(X)$ is the smallest set D such that:

- $X \subseteq D$
- if $s \in D$ and $type(s) = \text{AND}$ then $children(s) \subseteq D$
- if $s \in D$ and $type(s) = \text{OR}$ and $children(s) \cap X = \emptyset$ then $default(s) \in D$
- if $s \in D$ and $s \neq root$ then $parent(s) \in D$.

For example, $dcomp(\{\text{Unblocked}\}) = \{$Unblocked,Turnstile Control,Ready,Card Reader Control,On,$root\}$.

Note that each configuration is uniquely determined by its set of basic states. That is, if two configurations contain the same basic states, they are the same. Consequently, to denote a configuration, it suffices to list its BASIC states. In the sequel, we therefore only list the BASIC states of each configuration.

2.2 Transitions

2.2.1 Syntax

A transition connects source to target states. A transition can have multiple source and multiple target states. When a transition is taken, its source states are left and its target states are entered. For each transition $t \in \mathcal{T}$, $source(t)$ denotes the set of source states of t and $target(t)$ the set of target states:

$$source, target : \mathcal{T} \rightarrow \mathcal{P}(\mathcal{S}).$$

If $source(t) = \{x\}$ and $target(t) = \{y\}$, a convenient shorthand for t is $x \rightarrow y$. In the example, sample transitions are Off → On and Blocked → Unblocked.

To ensure that a transition can get enabled and enters a valid next configuration, we require that both $source(t)$ and $target(t)$ are consistent and non-empty.

The scope of a transition is the most nested OR state that contains both $source(t)$ and $target(t)$. Thus, it equals $l = lca(source(t) \cup target(t))$ only if l has type OR, which is usually the case. For example, the scope of transition Ready → Card Entered is Card Reader Control.

The event that triggers a transition t is denoted by $event(t)$. As discussed above, since UML only permits single events, we do not consider compound trigger events. If a transition has no trigger event, we use the special event null. Thus, $event(t) \in \mathcal{E} \cup \{\text{null}\}$.

We can classify transitions according to their trigger events:

- A transition t is *external* if $event(t) \in \mathcal{E}^{ext}$.
- A transition t is *internal* if $event(t) \in \mathcal{E}^{int}$.
- A transition t is a *completion* transition if $event(t) = \mathsf{null}$.

The set of events generated by a transition t is denoted $action(t)$. We require $action(t) \subseteq \mathcal{E}^{int}$. The set of events generated by a set T of transitions is denoted

$$generated(T) = \bigcup_{t \in T} action(t).$$

A transition t triggers transition t', written $t \gg t'$, if the trigger of t' is generated by t:

$$t \gg t' \;\Leftrightarrow\; event(t') \in action(t).$$

Note that a transition can trigger itself.

2.2.2 Semantics

Constructing a step. A statechart changes configuration by taking a set of transitions, called a step. To define steps, we need some auxiliary definitions. We assume that a configuration $C \subseteq \mathcal{S}$ and a set $I \subseteq \mathcal{E}$ of input events are given. For the UML semantics, I will be a singleton.

A transition is *relevant* if its sources are in C. The set of relevant transitions is defined as:

$$relevant(C) = \{\ t \in \mathcal{T} \mid source(t) \subseteq C\ \}.$$

A transition t is *enabled* in C for input events I if t is relevant in C and the trigger event of t is in I or null. The set of enabled transitions is defined:

$$enabled(C, I) = \{\ t \in \mathcal{T} \mid t \in relevant(C) \wedge event(t) \in I \cup \{\mathsf{null}\}\ \}.$$

Two transitions t_1 and t_2 are *consistent* if either they are equal or their scopes are orthogonal:

$$consistent(t_1, t_2) \Leftrightarrow t_1 = t_2 \quad \vee \quad scope(t_1) \perp scope(t_2).$$

A set T of transitions is consistent if every pair of transitions in the set is consistent:

$$consistent(T) \Leftrightarrow \forall t_1, t_2 \in T :\ consistent(t_1, t_2).$$

Two transitions t_1 and t_2 *conflict* if $t_1 \neq t_2$, their sources are consistent, and $scope(t_1)$ and $scope(t_2)$ are ancestrally related. In particular, t_1 and t_2 conflict

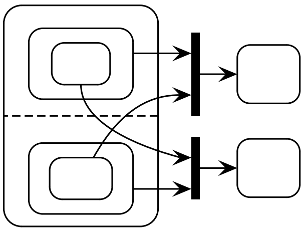

Fig. 3. Statechart for which it is unclear which transition has priority in UML

if $scope(t_1) = scope(t_2)$ and their sources are consistent.

$$\begin{aligned} conflict(t_1, t_2) \Leftrightarrow\ & t_1 \neq t_2 \\ & \wedge\ consistent(source(t_1) \cup source(t_2)) \\ & \wedge\ scope(t_1) \text{ and } scope(t_2) \text{ are ancestrally related.} \end{aligned}$$

Note that transitions t_1 and t_2 can be inconsistent yet not conflicting. For example, On → Off and Off → On are inconsistent, but not conflicting. However, On → Off and Unblocked → Blocked are conflicting.

Given a configuration C and set I of input events, a set T of transitions is *maximal* if adding an enabled transition to T would result in an inconsistent set:

$$maximal(T, C, I) \Leftrightarrow \forall t \in enabled(C, I) \setminus T : \neg consistent(T \cup \{t\}).$$

To choose between two enabled transitions t, t' that are conflicting, STATEMATE and UML use a priority rule $\prec$, where $t \prec t'$ if t has higher priority than t'. But STATEMATE and UML do not use the same priority rule. In STATEMATE, $t \prec^{SM} t'$ if the scope of t is a strict ancestor of the scope of t' [21]. In UML, $t \prec^{UML} t'$ if the sources of t are nested inside those of t' [43]. This definition is not precise, since it might be that the sources of the two transitions are nested inside each other (see Figure 3). A more precise definition, however, is lacking in the literature.

While STATEMATE and UML use a different priority rule, they do agree on how to use a priority rule to construct a step. Both approaches require that for each transition in the constructed step, there is no enabled transition outside the step with higher priority. Predicate *validPriority* captures this formally:

$$validPriority(St, C, I, \prec) \Leftrightarrow \forall t \in St\ \nexists t' \in enabled(C, I) \setminus St : t' \prec t.$$

Using these auxiliary definitions, we can now formally define a step. A set of transitions $St \subseteq \mathcal{T}$ is a *step* if and only if St is enabled, consistent, maximal, and satisfies the priority rule:

$$\begin{aligned} \mathit{isStep}(St, C, I, \prec) \Leftrightarrow\, & St \subseteq \mathit{enabled}(C, I) \\ & \wedge\ \mathit{consistent}(St) \\ & \wedge\ \mathit{maximal}(St, C, I) \\ & \wedge\ \mathit{validPriority}(St, C, I, \prec). \end{aligned}$$

For the example, in configuration {Blocked,Card Entered} with input events off and card ok, possible steps are {On → Off} and {Card Entered→Turnstile Unblocked}. If the last step is taken, event unblock turnstile is generated.

Taking a step. To define the effect of taking a step, we need some auxiliary definitions first.

First, observe that by taking a transition t, only states below $\mathit{scope}(t)$ are left and entered. The states entered by t, denoted $\mathit{enters}(t)$, are the states below $\mathit{scope}(t)$ that are in $\mathit{dcomp}(\mathit{target}(h))$:

$$\mathit{enters}(t) \;=\; \mathit{dcomp}(\mathit{target}(t)) \cap \mathit{children}^*(\mathit{scope}(t)).$$

In the example, enters(Off → On) ={On,Turnstile Control,Blocked,Card Reader Control,Ready}.

Given a configuration C and step St, function $\mathit{nextConfig}(C, St)$ defines the configuration reached by taking St:

$$\mathit{nextConfig}(C, St) = C \setminus \bigcup_{t \in St} \mathit{children}^*(\mathit{scope}(t)) \;\; \cup \;\; \bigcup_{t \in St} \mathit{enters}(t).$$

Thus, for each transition $t \in St$, the states in C that are below $\mathit{scope}(t)$ are left, and the states in $\mathit{enters}(t)$ are entered.

Finally, building on these definitions, we introduce some additional ones that are used in Section 4. A transition t *touches* another transition t' (or t' is touched by t) if t enters a state that is a source state of t':

$$\mathit{touches}(t, t') \Leftrightarrow \mathit{enters}(t) \cap \mathit{source}(t') \neq \emptyset.$$

For example, Off→On touches Blocked→Unblocked. However, Blocked→Unblocked does not touch On→Off, since On is not in enters(Blocked→Unblocked).

A sequence of $t_1, t_2, .., t_n$ of transitions is a *cycle* if and only if $\mathit{touches}(t_n, t_1)$ and for each pair t_i, t_{i+1}, where $0 < i < n$, $\mathit{touches}(t_i, t_{i+1})$.

3 Three execution semantics

This section formally defines the fixpoint, Statemate, and UML semantics for the syntax of statecharts defined in Section 2. As semantic model we use symbolic transition systems [9], proposed under the name synchronous transition systems in [35].

3.1 Symbolic transition systems

A symbolic transition system *STS* is a tuple $(V, \mathit{init}, \rightarrow)$, where

- V is a finite set of typed variables on some typed data domain $\mathcal{D}$. A valuation on V is type-preserving mapping $\sigma : V \rightarrow \mathcal{D}$. Denote by $\Sigma(V)$ the set of valuations on V.
- *init* is a first-order predicate over variables in V characterising the initial valuations.
- $\rightarrow$ is a transition predicate, a first-order predicate over variables in V, V' where unprimed variables refer to the current valuation and primed ones to the next valuation. For example the predicate $x = x' + 1$ relates a valuation σ to a next valuation σ' if and only if $\sigma'(x) = \sigma(x)+1$; we then write $\sigma \rightarrow \sigma'$.

A valuation is sometimes called a state or a snapshot [9,35]. However, to avoid confusion, in this paper the term 'state' refers only to a statechart, not to an STS.

A run of an STS is an infinite sequence of valuations:

$$\sigma_0\sigma_1\sigma_2..$$

such that σ_0 is initial, so σ_0 satisfies *init*, and for each pair σ_i, σ_{i+1} of valuations, $\sigma_i \rightarrow \sigma_{i+1}$, where $i \geq 0$.

3.2 Fixpoint semantics

In the fixpoint semantics, a statechart maps to a symbolic transition system STS^{FP}. Variables of STS^{FP} are

- $C : \mathcal{P}(\mathcal{S})$ the current configuration, which is a set of states.
- $I : \mathcal{P}(\mathcal{E})$ the current set of input events.

In the initial valuation, the configuration is the default completion of root and there are no input events:

$$init = dcomp(\{root\}) \wedge I = \emptyset.$$

In the fixpoint semantics, proposed by Pnueli and Shalev [40] to correct some inconsistencies in the first statechart semantics [23], the system waits in a stable valuation for events to occur and takes a single step in response. To formalise this, two kinds of transition predicates are needed. The first predicate, denoted $\rightarrow^{FP}_{event}$, models the occurrence of external events in a stable valuation:

$$\begin{aligned} \rightarrow^{FP}_{event} \Leftrightarrow\ & stable^{FP}(C, I) \\ & \wedge\ C = C' \\ & \wedge\ \emptyset \subset I' \subseteq \mathcal{E}. \end{aligned}$$

A valuation is defined to be stable if there are no input events to be processed:

$$stable^{FP}(C, I) \Leftrightarrow I = \emptyset.$$

Next, if events I have occurred, the system reacts by taking a step St, formalised by transition predicate $\rightarrow^{FP}_{step}$. A peculiar feature of the fixpoint semantics is that events generated in the current step are sensed immediately. That is, transitions triggered by generated events are enabled immediately and are taken in the same step. Thus, generated internal events are additional input events for the *isStep* predicate. Since for the fixpoint semantics, there is no existing priority rule, we use the STATEMATE definition.

$$\begin{aligned} \rightarrow^{FP}_{step} \Leftrightarrow\ & \neg stable^{FP}(C, I) \\ & \wedge\ \exists St \subseteq \mathcal{T} : isStep(St, C, I \cup generated(St), \prec^{SM}) \\ & \qquad \wedge\ C' = nextConfig(C, St) \\ & \qquad \wedge\ I' = \emptyset. \end{aligned}$$

Though this definition formalises the features of the fixpoint semantics listed in Table 1, it does not satisfy the causality principle, which is satisfied by the formalisation of Pnueli and Shavel [40]. The causality principe requires that each transition in a step must be (in)directly triggered by an external event. Our formalisation allows internal event generations that are not triggered by any external event, which violates causality. For example, in the initial configuration of the statechart in Figure 5 in Section 4, a possible step in response to $I = \{\mathsf{f}\}$ is $St = \{\mathsf{s3} \rightarrow \mathsf{s4}, \mathsf{s5} \rightarrow \mathsf{s6}\}$. But then i and j are generated spontaneously, violating causality, since f does not indirectly trigger any of the transitions in St. Instead, the step semantics of Pnueli and Shalev would define $St = \emptyset$. However, in Section 4 we define a syntactic constraint (C2) that rules out statecharts violating causality, rendering an additional semantic definition of causality superfluous.

Combining the two transition predicates, we have that a reaction in stable valuation σ_0 to a set of external input events is always a finite sequence consisting of two transitions

$$\sigma_0 \longrightarrow^{FP}_{event} \sigma_1 \longrightarrow^{FP}_{step} \sigma_2,$$

where the first transition models the receiving of input events and the second transition the reaction to these input event. It is impossible that a statechart diverges under the fixpoint semantics.

3.3 STATEMATE *semantics*

In the STATEMATE semantics, a statechart maps to a symbolic transition system STS^{SM}. As in the fixpoint semantics, variables of STS^{SM} are

- $C : \mathcal{P}(\mathcal{S})$ the current configuration, which is a set of states.
- $I : \mathcal{P}(\mathcal{E})$ the current set of input events.

The initial valuation is defined the same as for the previous semantics.

$$init = dcomp(\{root\}) \land I = \emptyset.$$

Like the fixpoint semantics, the STATEMATE semantics uses two transition predicates. On the surface, these are very similar to the ones defined for the fixpoint semantics, but as we will see, they differ subtly from them.

The first predicate, $\longrightarrow^{SM}_{event}$, models the occurrence of one or more external events in a stable valuation:

$$\begin{aligned} \longrightarrow^{SM}_{event} \Leftrightarrow\ & stable^{SM}(C, I) \\ & \land\ C = C' \\ & \land\ \emptyset \subset I' \subseteq \mathcal{E}. \end{aligned}$$

Note that this definition is identical to the one defined for the fixpoint semantics. However, the definition of stable valuation is somewhat different. In STATEMATE, a valuation is stable if there are no input events to be processed *and* there are no enabled transitions:

$$stable^{SM}(C, I) \Leftrightarrow I = \emptyset \land enabled(C, I) = \emptyset.$$

The second transition predicate, $\rightarrow^{SM}_{step}$, models the taking of a step. A step is only taken if the current valuation is not stable, i.e. there are some input events or some enabled transitions. The effect of taking a step is that a next configuration is reached and that some internal events (actions of the transitions in the step) are generated. These generated events are put in I'. Transition relation $\rightarrow^{SM}_{step}$ formalises this:

$$\begin{aligned}\rightarrow^{SM}_{step} \Leftrightarrow & \neg \mathit{stable}^{SM}(C, I) \\ & \quad \wedge \exists St \subseteq \mathcal{T} : \mathit{isStep}(St, C, I, \prec^{SM}) \\ & \qquad \wedge C' = \mathit{nextConfig}(C, St) \\ & \qquad \wedge I' = \mathit{generated}(St).\end{aligned}$$

Again, note that this definition is similar to its counterpart in the fixpoint semantics. The major difference is that internally generated events are sensed in the next step only, while these are sensed immediately in the fixpoint semantics.

Combining these transition predicates, we have that in STATEMATE a reaction to a set of external input events consists of a sequence of steps, called a superstep:

$$\sigma_0 \rightarrow^{SM}_{event} \sigma_1 \rightarrow^{SM}_{step} \sigma_2 .. \sigma_{n-1} \rightarrow^{SM}_{step} \sigma_n,$$

where $\sigma_0, \sigma_n \models \mathit{stable}^{SM}(C, I)$, and for every valuation σ_i, where $0 < i < n$, $\sigma_i \not\models \mathit{stable}^{SM}(C, I)$. The sequence might be infinite, in which case the statechart diverges. Then, for every valuation σ_i with $i > 0$, we have $\sigma_i \not\models \mathit{stable}^{SM}(C, I)$. Examples of diverging statecharts can be found in Section 4.

3.4 UML semantics

In the UML semantics, a statechart maps to a symbolic transition system STS^{UML}. Variables of STS^{UML} are

- $C : \mathcal{P}(\mathcal{S})$ the current configuration, which is a set of states, and
- $q : \mathcal{E}^*$ the current queue, which is a sequence of events.

For an event queue $q = e_1..e_n \in \mathcal{E}^*$, we introduce the following notation [9]:

- $\mathit{head}(q) = e_1$ denotes the first event of q if $q \neq \varepsilon$, i.e. the q is not empty.
- $\mathit{tail}(q) = e_2..e_n$, where $n \geq 2$, denotes q with the first element removed, and $\mathit{tail}(q) = \epsilon$ if $n < 2$.
- $\mathit{enqueue}(e, q) = qe$ denotes the result of appending event e to q, and $\mathit{enqueue}(E, q)$ denotes the result of appending all events in $E \subseteq \mathcal{E}$ in some arbitrary order to q.

In the initial valuation the system is in the default completion of *root* and the queue has no input events:

$$init = dcomp(\{root\}) \wedge q = \epsilon.$$

For the UML semantics, three transition predicates are needed. The first predicate, denoted $\rightarrow^{UML}_{event}$, models the occurrence of one or more external events, which are added to the queue. As in the other two semantics, such transitions do not change the current configuration:

$$\begin{aligned} \rightarrow^{UML}_{event} \Leftrightarrow\ & \exists E \subseteq \mathcal{E} : E \neq \emptyset \\ & \wedge\ C = C' \\ & \wedge\ q' = enqueue(E, q). \end{aligned}$$

Note that in this semantics, unlike in the other two, external events can occur in both stable and unstable valuations. In particular, they can occur while some other event is being processed.

Events in the queue are processed one by one. An event is processed if the current valuation is stable, i.e. there are no enabled completion transitions:

$$stable^{UML}(C, q) \Leftrightarrow enabled(C, \emptyset) = \emptyset.$$

In a stable valuation, the system processes the first event from the queue by taking a step. Note that an event can be either external or internal, since generated events are also inserted in the queue:

$$\begin{aligned} \rightarrow^{UML}_{step} \Leftrightarrow\ & q \neq \varepsilon \\ & \wedge\ stable^{UML}(C, q) \\ & \wedge\ \exists St \subseteq \mathcal{T} : isStep(St, C, \{head(q)\}, \prec^{UML}) \\ & \qquad \wedge\ C' = nextConfig(C, St) \\ & \qquad \wedge\ q' = enqueue(generated(St), tail(q)). \end{aligned}$$

After the step has been taken, the current valuation can be unstable: there are some enabled completion transitions. However, the next event can only be processed in a stable valuation. Therefore, the enabled completion transitions need to be taken first.

$$\begin{aligned}\rightarrow^{UML}_{completionstep} \Leftrightarrow\, &\neg stable^{UML}(C, q)\\ &\wedge\ \exists St \subseteq \mathcal{T} : isStep(St, C, \emptyset, \prec^{UML})\\ &\quad\wedge\ C' = nextConfig(C, St)\\ &\quad\wedge\ q' = enqueue(generated(St), q).\end{aligned}$$

Combining these transition predicates, we have that a reaction in configuration C to processing an event from the queue is typically a sequence

$$\sigma_0 \rightarrow^{UML}_{step} \sigma_1 \rightarrow^{UML}_{completionstep} \sigma_2 \rightarrow^{UML}_{completionstep} \sigma_3 .. \sigma_{n-1} \rightarrow^{UML}_{completionstep} \sigma_n,$$

where $\sigma_0, \sigma_n \models stable^{UML}(C, q)$. If there is a cycle of completion transitions, the sequence can be infinite: then for every valuation σ_i, where $i > 0$, $\sigma_i \not\models stable^{UML}(C, q)$. Thus, a statechart can diverge, as in the Statemate semantics.

4 Constraints

We next define several constraints that are used in the theorems in Section 5. As mentioned in Section 1, in addition to the three semantics, we also consider single-event Statemate, or seStatemate for short, a variant of the Statemate semantics in which external events are constrained to occur one by one. We relate the fixpoint to the Statemate semantics, the Statemate to the seStatemate semantics, and finally the seStatemate to the UML semantics. Therefore, the constraints are grouped in three classes. As explained in Section 1, for the first two groups, we identify constraints that ensure that given a configuration, the effects of the system reactions under both semantics are similar, i.e., the same end configurations are eventually reached. For the last group, we define constraints that ensure that under both semantics the same steps are taken. This implies that the same end configurations are reached.

Each constraint is illustrated and motivated by presenting a counterexample statechart that violates it. Almost all of the constraints are structural, to ensure that they can be easily checked. The only semantical constraint (on the UML semantics) cannot be put structurally. In the definitions of the constraints, we use notions and concepts that were formally defined in Section 2. Formalisations are only provided if the informal definitions are ambiguous. We use the term 'stable configuration C' to denote a stable valuation with configuration C. As explained in Section 2, we only list the basic states of a configuration. In the example statecharts, events e and f are external whereas events i, j, k and l are internal.

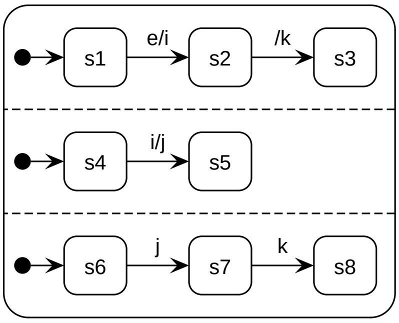

Fig. 4. Statechart to illustrate constraint C1

4.1 Fixpoint and STATEMATE *semantics*

Completion transitions. Under the fixpoint semantics, an enabled completion transition is only taken if some trigger event occurs, even though it does not need any trigger event to become enabled. While under the STATEMATE semantics a completion transition is taken as soon as it becomes enabled. This leads to a difference in behaviour, as illustrated by Figure 4. Under the fixpoint semantics, if in the initial configuration event e occurs, the next stable configuration will be {s2,s5,s7}. Next, to take the completion transition s2→s3 another trigger event must occur; then configuration {s3,s5,s8} is reached, so s8 is reachable under the fixpoint semantics. Under the STATEMATE semantics, the behaviour is quite different. If in the initial configuration e occurs, eventually stable configuration {s3,s5,s7} is reached, so s2→s3 is taken. But internal event k is processed while the system is in state s6. Consequently, state s8 is unreachable under the STATEMATE semantics.

We resolve this difference in behaviour by ruling out completion transitions.

C1 There are no completion transitions.

Divergence. Under the fixpoint semantics, a statechart cannot diverge since after each reaction a stable valuation is entered. While under the STATEMATE semantics divergence is possible, either due to a cycle of completion transitions (ruled out by C1) or to a cycle of internally generated trigger events. For an example of the latter, if in the initial configuration of Figure 5 event e occurs, the system diverges and will not respond if f occurs next.

We observe that in a diverging statechart a transition triggers itself indirectly. To define the constraint that rules out divergence by internally generated events, we need to define indirect triggering first. We write $t \gg^+ t'$ to denote a sequence of transitions $t_1, t_2, .., t_n$, with $t_1 = t$ and $t_n = t'$, such that for every t_i, t_{i+1}, where $0 < i < n$, $t_i \gg t_{i+1}$. If $t \gg^+ t$, then t triggers itself

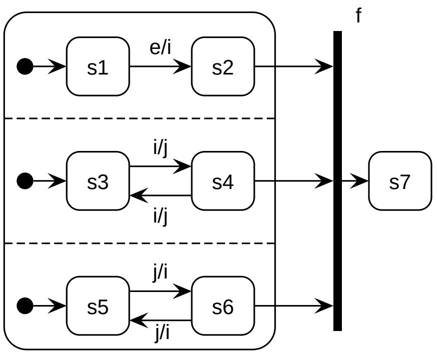

Fig. 5. Statechart to illustrate constraint C2

indirectly. For example, in Figure 5, transition s3→s4 indirectly triggers itself. Therefore, to guarantee absence of a cyclic chain of trigger events, we require that the triggers relation $\gg^+$ is acyclic:

C2 A transition does not indirectly trigger itself.

This constraint also rules out statecharts that violate causality under the fixpoint semantics (see Section 3.2). Such statecharts allow spontaneous event generations. However, a spontaneous event generation is only possible if there is a set of transitions triggering each other, which is ruled out by C2.

Event generation. Under the fixpoint semantics, events generated in a step are sensed in the same step, while under the Statemate semantics they are sensed in the next step. Three differences in behaviour are the result.

First, under the fixpoint semantics, external and internal transitions can be enabled in the same valuation. Under the Statemate semantics, this is impossible. Consequently, if an enabled external transition conflicts with an enabled internal one, the internal transition might disable the external one under the fixpoint semantics, but under the Statemate semantics the external transition is always chosen first. Figure 6 illustrates this issue. If in the initial configuration events e and f occur, then under the fixpoint semantics, the next stable configuration is either {s2,s4} or {s2,s5}. If the latter configuration is reached, the internal transition with trigger i has been taken, even though f occurs simultaneously with e. Under the Statemate semantics, the next stable configuration will always be {s2,s4}. Leveson et al. [30] first noted this issue, but attributed it mistakenly to the Statemate semantics [17].

Constraint C3 rules out statecharts like the ones shown in Figure 6 by forbidding conflicts between external and internal transitions:

C3 An external transition does not conflict with an internal transition.

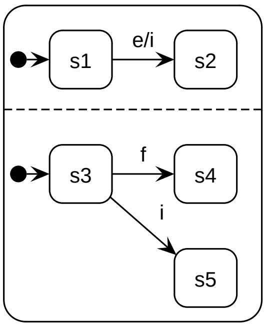

Fig. 6. Statechart to illustrate constraint C3

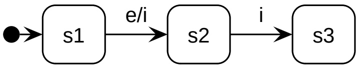

Fig. 7. Statechart to illustrate constraint C4

Second, under the STATEMATE semantics some internal transitions can be taken that cannot be taken under the fixpoint semantics. More precisely, under the STATEMATE semantics, an internal transition that becomes relevant in a reaction can be taken in that same reaction, whereas under the fixpoint semantics, such a transition can only be taken in the next reaction, when the next external events occur. For example, if e occurs in the initial configuration of Figure 7, under the STATEMATE semantics stable configuration {s3} is reached next. But under the fixpoint semantics, stable configuration {s2} is reached next, and the system stays in s2, since the only transition generating i has already been taken.

In Figure 7, a triggered transition is made relevant by its triggering transition, and thus is inconsistent with the triggering transition. But this is impossible under the fixpoint semantics, since that semantics only allows triggering between consistent transitions. To rule out statecharts like Figure 7, we therefore forbid event generations between inconsistent transitions:

C4 Each transition only triggers transitions that are consistent with it.

However, Constraint C4 is not sufficient to rule out the second difference, since an internal transition that becomes relevant in a reaction can also be triggered by a transition in a parallel branch. There are two cases: the internal transition is made relevant by an external transition, or by another internal transition. For the first case, consider Figure 8, where external transition s3 → s4 makes relevant internal transition s4 → s5. If in the initial configuration events e and f occur, then under the fixpoint semantics stable configuration {s2,s4} is entered, and s4 stays active, since the only transition generating i has been taken already. Under the STATEMATE semantics, however, stable configuration {s2,s5} is entered, because i is responded to while s4 is active.

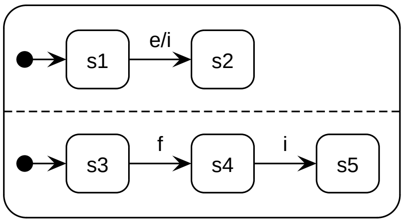

Fig. 8. Statechart to illustrate constraint C5

To rule out such statecharts, we require that if an internal transition t_i is touched by an external transition t_e, so t_e can make t_i relevant, then t_e is not consistent with the transitions triggering t_i. This ensures that t_i gets only triggered if t_e has been taken. Note that this constraint allows Figure 7, since s1→s2 is consistent with itself.

C5 If an internal transition is touched by an external transition, the external transition is not consistent with any transition triggering the internal transition:

$$\forall t_i, t_e, t \in \mathcal{T} : \mathit{internal}(t_i) \wedge \mathit{external}(t_e) \wedge \mathit{touches}(t_e, t_i) \wedge t \gg t_i \Rightarrow \neg \mathit{consistent}(t, t_e).$$

For the second case, consider Figure 9(a), where internal transition s5 → s6 makes relevant internal transition s6 → s7. If in the initial configuration event e occurs, under the fixpoint semantics stable configuration {s2,s4,s6} is reached, whereas under the Statemate semantics stable configuration {s2,s4,s7} is reached, since k is sensed and processed when s6 is active. The problem here is caused by two inconsistent internal transitions that are triggered by transitions that are consistent.

A third difference is that under the fixpoint semantics additional internal transitions can be taken that cannot be taken under the Statemate semantics. Under the fixpoint semantics, all internal transitions that are taken in a reaction, become simultaneously enabled. Under the Statemate semantics, a reaction has multiple steps and an internal transition taken in step $i+1$, where $i > 0$, gets only enabled after step i has been done. If consistent transitions generate events that trigger conflicting (and thus inconsistent) transitions, this can lead to a difference in behaviour, as illustrated by Figure 9(b). In the initial configuration, if e and f occur, then under the fixpoint semantics events j and k are sensed and processed while the system is in s7, and either stable configuration {s2,s4,s6,s8} or stable configuration {s2,s4,s6,s9} is reached next. But under the Statemate semantics, the next stable configuration is always {s2,s4,s6,s8}, because k is sensed and processed while the system is in s8.

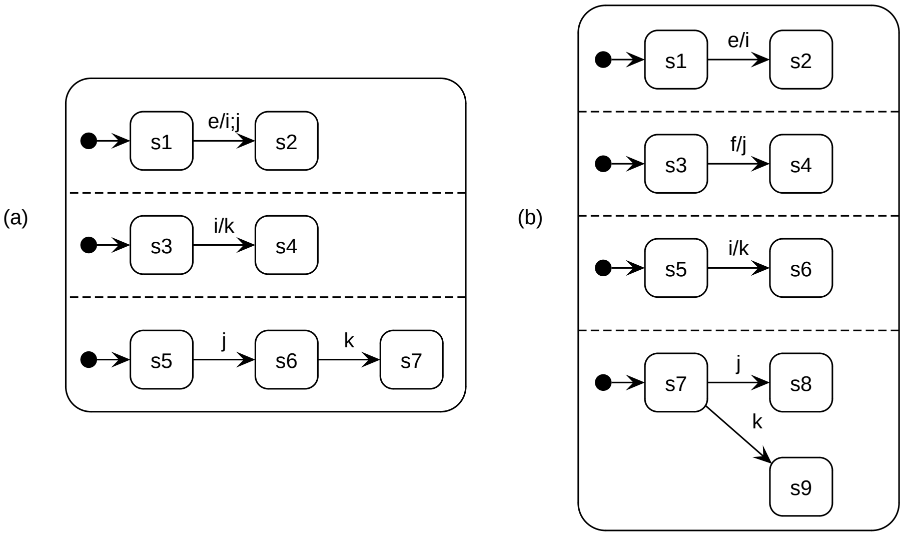

Fig. 9. Statecharts to illustrate constraint C6

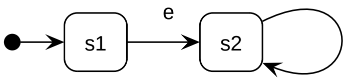

Fig. 10. Statechart to illustrate constraint C7

To rule out statecharts such as shown in Figure 9, constraint C6 states that the transitions triggered by two different consistent transitions should be consistent too:

C6 If two different transitions are consistent, then the transitions they trigger are consistent with each other.

4.2 Statemate *and se*Statemate *semantics*

Divergence. To relate the Statemate and single-event Statemate semantics, we do not need to drop completion transitions. Consequently, to rule out divergence, we need to impose a constraint, in addition to Constraint C2. Under the Statemate semantics, a statechart satisfying C2 can still diverge, because each step might result in a configuration in which a completion transition is enabled (see Figure 10).

The following constraint rules out this divergence:

C7 There is no cycle of completion transitions.

However, C7 is not sufficient to rule out all divergence, since a diverging cycle

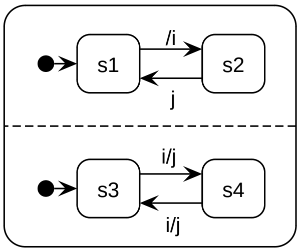

Fig. 11. Statechart to illustrate constraint C8

Table 3
Constraints for conflicting transitions

C3	An external transition does not conflict with an internal transition.
C9	An external transition does not conflict with a completion transition.
C10	A completion transition does not conflict with an internal transition.
C11	If two completion transitions are conflicting, they have the same sources.

may consist of a sequence of internal and completion transitions. For example, the statechart in Figure 11 diverges, but does not violate C7 (nor C2). To rule out such cycles, we put the following constraint:

C8 An internal transition is not touched by a completion transition.

This constraint is also needed in the sequel to rule out differences caused by event generation.

Conflicts. Conflicts between transitions might result in different behaviour under the Statemate and seStatemate semantics. Table 3 shows constraints that rule out such differences. Figure 12 shows for each constraint a statechart that violates it, if under the Statemate semantics events e and f occur simultaneously.

Constraint C3 was already introduced in the previous subsection. Violation of this constraint may also lead to differences in behaviour under the Statemate and seStatemate semantics, as shown in Figure 12(a). If in the initial configuration events e and f occur, then under the Statemate semantics the configuration will become {s2,s5}. Under the seStatemate semantics, if e occurs before f, stable configuration {s2,s4} is reached eventually. If f occurs before e, eventually stable configuration {s2,s6} is reached. Both configurations differ from the stable configuration reached under the Statemate semantics.

Constraint C9 rules out conflicts between external and completion transitions. To motivate it, consider the statechart in Figure 12(b), which violates

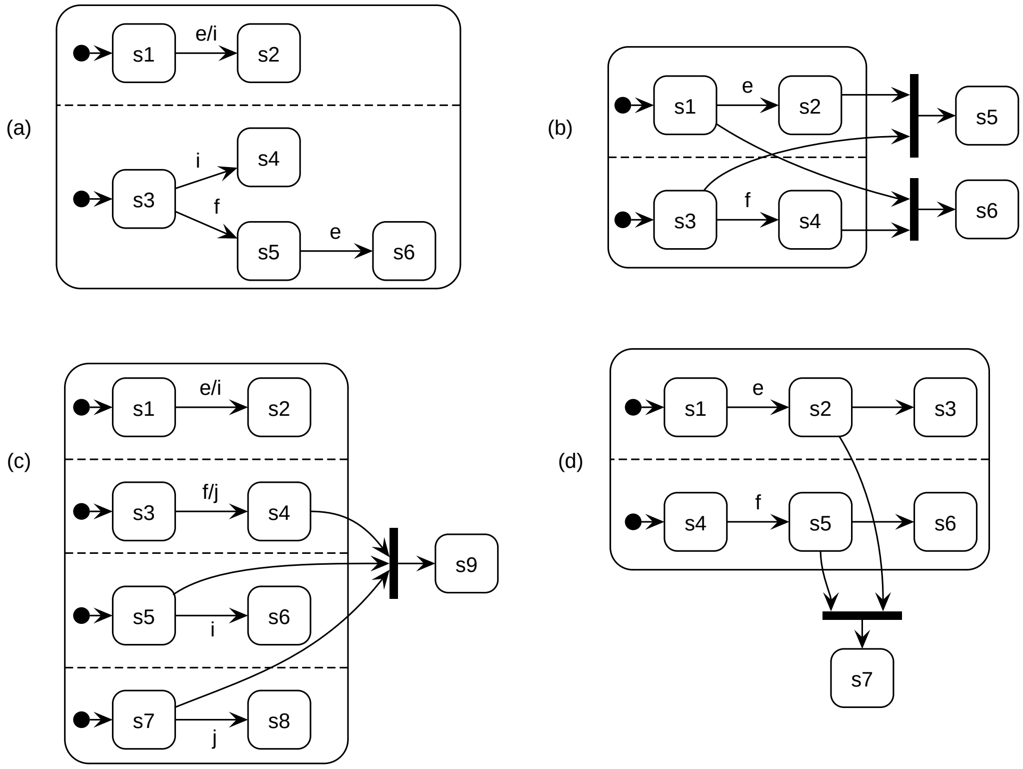

Fig. 12. Statecharts to illustrate the constraints in Table 3

the constraint. If in the initial configuration both e and f occur, then under the Statemate semantics the next stable configuration will be {s2,s4}. If e occurs before f, eventually stable configuration {s5} will be reached; if f occurs before e, eventually stable configuration {s6}. In both cases, a different stable configuration than under the Statemate semantics is reached.

Constraint C10 rules out conflicts between completion and internal transitions. An example of such a conflict is shown in the statechart in Figure 12(c). If in the initial configuration events e and f occur, then under the Statemate semantics the next stable configuration will be {s9}. Under the seStatemate semantics, whether e occurs before f or vice versa, always stable configuration {s2,s4,s6,s8} is reached eventually, which differs from the stable configuration reached under the Statemate semantics.

Constraint C11 requires that conflicting completion transitions have the same sources. The statechart in Figure 12(d) shows conflicting completion transitions with different sources. If in the initial configuration events e and f occur, then under the Statemate semantics the next stable configuration could be {s7}. Under the seStatemate semantics, this configuration is not reachable. Whether e occurs before f or vice versa, stable configuration {s3,s6} is always reached next.

Table 4
Constraints for event generation

C4	Each transition only triggers transitions that are consistent with it.
C5	If an internal transition is touched by an external transition, the external transition is not consistent with any transition triggering the internal transition.
C6	If two different transitions are consistent, then the transitions they trigger are consistent with each other.
C8	An internal transition is not touched by a completion transition.

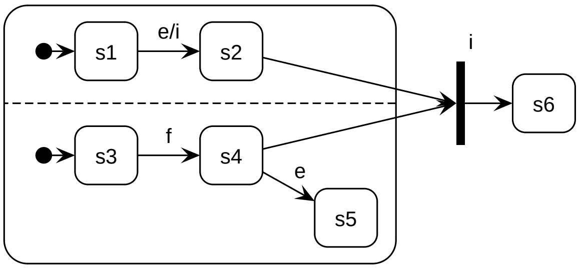

Fig. 13. Statechart to illustrate constraint C4 for Statemate/seStatemate

Event generation. Under the Statemate semantics, events are processed in parallel, and hence events are generated in parallel as well. Under the seStatemate semantics, events are processed one by one, and hence events are generated sequentially. The constraints defined to rule out the resulting differences in behaviour are listed in Table 4. All constraints have been defined already, but the motivating examples we present next are new and are all related to differences in behaviour due to event generation.

Constraint C4 is needed again, as illustrated by Figure 13. Under the Statemate semantics, if in the initial configuration events e and f occur, stable configuration {s6} is reached. Under the seStatemate semantics, if e occurs before f, then eventually stable configuration {s2,s4} is reached. If f occurs before e, then eventually stable configuration {s2,s5} is reached. Both configurations are different from the stable configuration reached under the Statemate semantics. Constraint C4 rules out the statechart in Figure 13.

Constraint C5 is needed again too, as shown by the example statechart in Figure 14, which violates the constraint. Under the Statemate semantics, if in the initial configuration e and f occur, then either stable configuration {s2,s7} or stable configuration {s3,s6} is reached. Whereas under the seStatemate semantics, if e occurs before f, eventually stable configuration {s2,s6} is reached, while if f occurs before e, then eventually stable configura-

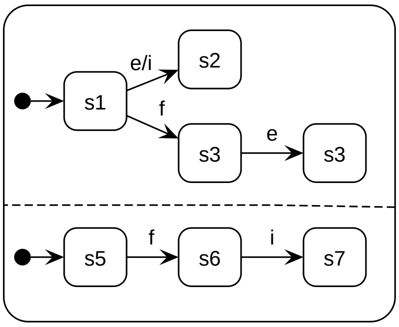

Fig. 14. Statechart to illustrate constraint C5 for Statemate/seStatemate

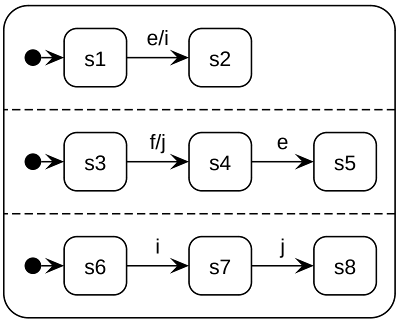

Fig. 15. Statechart to illustrate constraint C6 for Statemate/seStatemate

tion {s4,s6} is reached.

Constraint C6 is also needed, as demonstrated by the statechart in Figure 15, which violates it. Under the Statemate semantics, if in the initial configuration e and f occur, then stable configuration {s2,s4,s7} is reached. Note that j is ignored, since it is processed while s6 is active. Under the seStatemate semantics, if e occurs before f, then eventually stable configuration {s2,s4,s8} is reached. If f occurs before e, then eventually stable configuration {s2,s5,s7} is reached. Both stable configurations differ from the one reached under the Statemate semantics.

Constraint C8 is needed again too, because under the seStatemate semantics some internal transitions can be taken in a reaction that cannot be taken under the Statemate semantics. To illustrate this, consider the statechart in Figure 16, which violates C8. If in the initial configuration events e and f occur, under the Statemate semantics the next stable configuration will be {s2, s6} because i is processed while the system is in s5. Under the seStatemate semantics, however, if e occurs before f then eventually stable configuration {s3,s6} is reached, while if f occurs before e, stable configuration {s2, s7} is reached eventually.

Extra effects. The same external event can trigger different transitions. Combined with the differences between single-event and parallel-event pro-

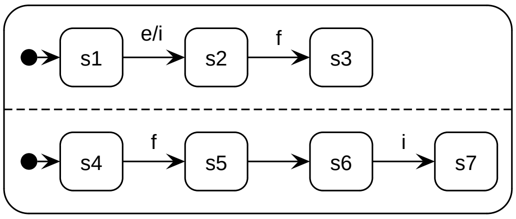

Fig. 16. Another statechart to illustrate constraint C8

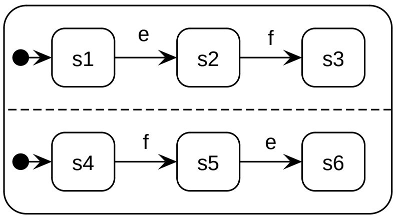

Fig. 17. Statechart to illustrate constraint C12

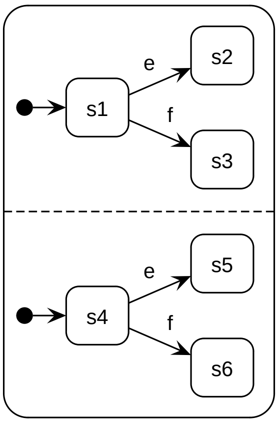

Fig. 18. Another statechart to illustrate constraint C12

cessing, this might cause that under the seStatemate semantics some additional transitions are taken when trying to simulate a Statemate reaction. To illustrate this, consider Figure 17. Under the Statemate semantics, if in the initial configuration events e and f occur, then the next stable configuration will be {s2,s5}. Under the seStatemate semantics, if e occurs before f, then stable configuration {s3,s5} is reached, whereas if f occurs before e, stable configuration {s2,s6} is reached. In both cases, a transition is taken that is not taken under the Statemate semantics, so under the seStatemate semantics events e and f have unavoidable extra effects. The effects are unavoidable in the sense that they cannot be avoided by changing the order of event processing.

Figure 18 gives another example. If in the initial configuration events e and f occur, then under the Statemate semantics a possible next configuration is {s2,s6}, so the step contains one transition triggered by e and one by f. But this configuration is not reachable under the seStatemate semantics,

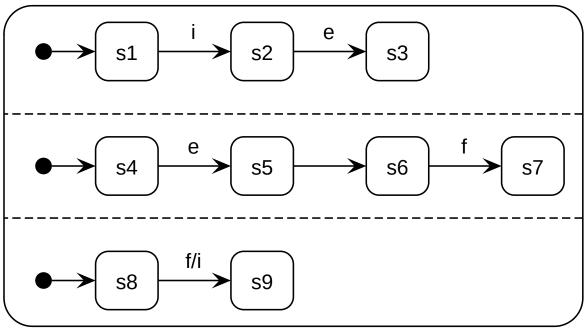

Fig. 19. Yet another statechart to illustrate constraint C12

since either both transitions triggered by e (leading to configuration {s2,s5}) or both transitions triggered by f (leading to {s3,s6}) are taken. Again, e and f have unavoidable extra effects under the seStatemate semantics.

To define a constraint that rules out these two statecharts and similar ones, we introduce a relation $\mathit{prec}(e, e')$ that is true if and only if e is to be processed before e' to avoid extra effects. The constraint, defined below, requires that *prec* is acyclic. Before we formally define *prec*, we illustrate two aspects of its definition by means of the two counterexamples.

First, if two external transitions t_1, t_2 touch each other, so $\mathit{touch}(t_1, t_2)$, and their trigger events occur simultaneously under the Statemate semantics, then the event of the touched transition t_2 should be processed first to avoid that t_2 gets taken extra. For example, in Figure 17, event f should be processed before e to avoid that s2→s3 is taken extra. However, e should be processed before f to avoid that s5→s6 is taken extra, so the *prec* relation is cyclic.

A more complicated case is shown in Figure 19. Here, transitions s4→s5 and s6→s7 do not touch each other directly, but indirectly through a completion transition. Still a similar problem occurs: f needs to be processed before e to avoid that s6→s7 is taken extra under the seStatemate semantics. But the example is even more complex. Though transition s8→s9 does not touch s2→s3, it does make that transition relevant, since it triggers s1→s2 which makes s2→s3 relevant. From this relation, we have that e should be processed before f to avoid extra effects, i.e. the taking of s2→s3. Combining these two precedence constraints, we again have that the *prec* relation is cyclic.

To cater for all this, we define a relation $\mathit{makesRelevant} \subseteq \mathcal{T} \times \mathcal{T}$. Let t, t' be two transitions. Then t makes t' relevant, written $\mathit{makesRelevant}(t, t')$ if there is a transition t'' touching t' and either

- t'' is external and $t = t''$;
- t'' is internal and t'' is consistent with t and t indirectly triggers t'';
- t'' is a completion transition and t makes t'' relevant.

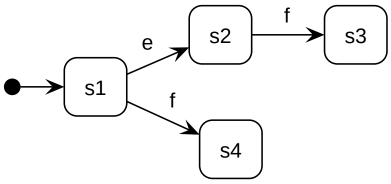

Fig. 20. Final statechart to illustrate constraint C12

Formally, *makesRelevant*(t, t') is defined to be the smallest predicate satisfying:

$$\begin{aligned}\exists t'' \in \mathcal{T} : \mathit{touches}(t'', t') \wedge (\ & (\mathit{external}(t'') \wedge t = t'') \\ & \vee(\mathit{internal}(t'') \wedge t \gg^{+} t'' \wedge \mathit{consistent}(t, t'')) \\ & \vee(\mathit{completion}(t'') \wedge \mathit{makesRelevant}(t, t''))\ ).\end{aligned}$$

If t makes t' relevant, then $\mathit{event}(t')$ should be processed before $\mathit{event}(t)$, so $\mathit{prec}(\mathit{event}(t'), \mathit{event}(t))$.

For the second aspect, consider again Figure 18. If two transitions t_1, t_2 are conflicting and there is a transition t_3 consistent with t_1 and with the same trigger event as t_1, then under the STATEMATE semantics a possible step contains both t_2 and t_3. To ensure that both these transitions are taken under the seSTATEMATE semantics, the event of t_2 should be processed before that of t_1/t_3. For example, in Figure 18, event f should be processed before e, to ensure that transition s1→s3 can be taken under the seSTATEMATE as well as STATEMATE semantics. But by similar reasoning e should be processed before f, so the *prec* relation is again cyclic.

This problem with conflicting external transitions also occurs if t_1 is making relevant another transition with the same trigger event as t_2. For example, for Figure 20 we have (first aspect) that f should be processed before e since s1→s2 makes s2→s3 relevant. But if under the STATEMATE semantics step {s1→s2} is taken, event e should be processed before f to simulate this step under the seSTATEMATE semantics. Again, the *prec* relation is cyclic.

Formalising these two aspects, we have the following definition:

$$\begin{aligned}\mathit{prec}(e, e') \Leftrightarrow e \neq e' \wedge \exists t, t' \in \mathcal{T} : \mathit{event}(t) = e \wedge \mathit{event}(t') = e' \wedge \\ (\ \mathit{makesRelevant}(t', t) \\ \vee(\mathit{conflict}(t, t') \wedge \exists t'' \in \mathcal{T} : \mathit{event}(t'') = e' \wedge \\ (\mathit{consistent}(t'', t) \vee \mathit{makesRelevant}(t, t''))\ ).\end{aligned}$$

Next, we require that *prec* is acyclic. Figures 17, 18, and 20 illustrate basic

Table 5
Different stable configurations reached from the initial configuration of the statecharts in Figure 21

	(a)	(b)	(c)	(d)
e and f simultaneously	$\{s2, s5\}$	$\{s6\}$	$\{s3, s6\}$	$\{s2, s4, s6\}$ or $\{s2, s4, s8\}$
e before f	$\{s2, s6\}$	$\{s2, s4\}$	$\{s3, s5\}$	$\{s2, s4, s7\}$
f before e	$\{s3, s5\}$	$\{s5\}$	$\{s2, s6\}$	$\{s2, s4, s9\}$

cases in which *prec* is cyclic. However, there are also more complex cases, for example Figure 19.

C12 The *prec* relation is acyclic.

Note that C12 does not rule out all conflicting transitions. Phrased differently, two conflicting transitions are not sufficient to get a cyclic *prec* relation between the two trigger events of the transitions. For example, Figure 2 is allowed by C12, even though there are several conflicting transitions in the statechart, because for example events card ok and card not ok each trigger only one transition, and thus do not have extra effects.

Final remarks. Most of the counterexamples presented in this subsection have some external event that triggers multiple transitions. Putting a constraint that states that each external event triggers at most one transition would rule out the presented counterexamples for C3, C4, C5, C6, C12 and C13. While such a constraint would indeed make constraints C12 and C13 superfluous, still C3, C4, C5, and C6 are needed then, as shown by Figure 21. Each of the alternative counterexamples in Figure 21 satisfy the new constraint, even for internal events, yet exhibit different behaviour for the STATEMATE and seSTATEMATE semantics. Table 5 shows the stable configurations reached from the initial configurations if external events e and f occur simultaneously or one by one. For each of the example statecharts, the reached stable configurations are different. This shows that it is not straightforward to find alternative constraints that rule out all differences in behaviour for the STATEMATE and seSTATEMATE semantics.

*4.3 se*STATEMATE *and UML semantics*

As explained in Section 1, we will relate the seSTATEMATE and UML semantics by showing that they can take the same steps in response to the same input events. To make this work, we have to identify several constraints.

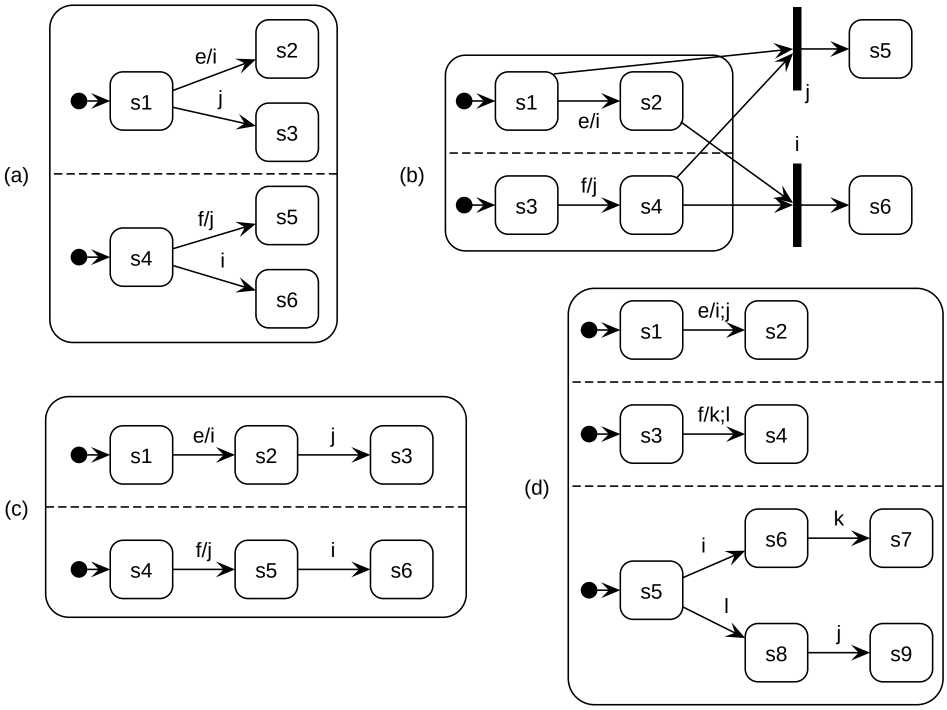

Fig. 21. Alternative statecharts that violate C3 (a), C4 (b), C5 (c), and C6 (d)

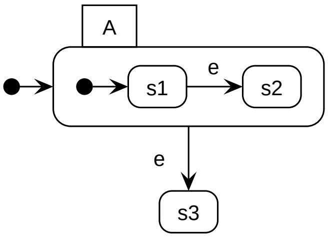

Fig. 22. Statechart to illustrate constraint C13

First, we have to ensure that transitions triggered by the same trigger event have the same priority under different semantics. Otherwise, both semantics construct different steps in response to some input event because they use different priority rules. For example, if in the initial configuration of Figure 22 event e occurs, then step {A→s3} would be constructed if the Statemate priority rule were used, whereas step {s1→s2} would be the result if the UML priority rule were used. To rule out such differences, we require that conflicting transitions have the same sources and the same scope. From the definitions of the two priority rules (see Section 2), it then follows that with this constraint two conflicting transitions with the same trigger event have equal priority under both semantics.

C13 Two conflicting transitions with the same trigger event have the same sources and the same scope.

Next, though under the seStatemate semantics external events occur one

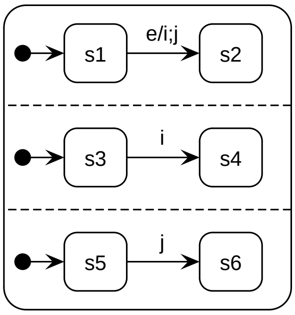

Fig. 23. Statechart to illustrate constraint C14

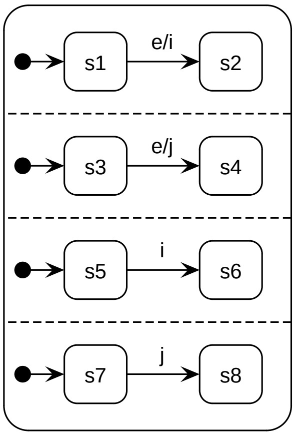

Fig. 24. Statechart to illustrate constraint C15

by one, internal events can still be processed in parallel, which is impossible under the UML semantics. Consequently, different steps can be taken under both semantics. For example, if in the initial configuration of Figure 23 event e occurs, then after step {s1→s2} step {s3→s4,s5→s6} is taken. This latter step cannot be taken under the UML semantics, since i and j are then processed one by one, not in parallel.

We therefore require that each transition generates at most one event (C14). However, that constraint still allows statecharts that generate more than one event in a step. If in the initial configuration of Figure 24 event e occurs, then two events are generated in response. We therefore also require that two consistent transitions with the same trigger event generate the same event (C15). So then either no event or one single event is generated.

C14 Each transition generates at most one event.

C15 Two consistent transitions having the same trigger event generate the same event.

Another difference in behaviour is due to completion transitions. Under the

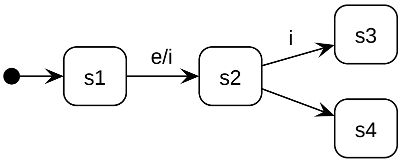

Fig. 25. Statechart to illustrate constraint C10 for seSTATEMATE/UML

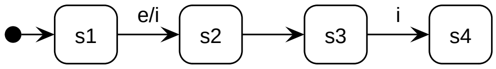

Fig. 26. Statechart to illustrate constraint C8 for seSTATEMATE/UML

UML semantics, events are only processed if no completion transitions are enabled, so completion transitions have priority over internal transitions. Consequently, if an internal transition conflicts with a completion one, the completion transition is always taken under the UML semantics, whereas under the seSTATEMATE semantics also the internal one can be taken. For example, if in the initial configuration of Figure 25 event e occurs, then under the UML semantics always {s4} is reached, so the internal transition is never taken. But under the seSTATEMATE semantics a possible configuration is {s3}. To rule out this difference, we require that completion and internal transitions do not conflict (constraint C10). Note that Figure 25 also violates C4, but this constraint we do not need here.

However, C8 is needed too, since an internal transition that is made relevant by a completion transition can be taken extra under the UML semantics. Figure 26 gives an example. If in the initial configuration e occurs, then under the seSTATEMATE semantics always s3 is reached, since i is processed while the system is in s2. Moreover, s4 is not reachable, since i has been generated already. Under the UML semantics, however, s4 is reached, since i is only processed after the completion transition s2 → s3 has been taken, so internal transition s3 → s4 is taken extra compared to the seSTATEMATE reaction.

Another consequence of the priority of completion transitions in UML is that under the UML semantics, a step does not contain both internal and completion transitions, which is possible under the seSTATEMATE semantics. For example, if in the initial configuration of Figure 27 event e occurs, under the seSTATEMATE semantics the second step taken is {s2→s3,s4→s5}. But under the UML semantics, the second step is {s2→s3} and s4→s5 is taken only in the third step. To rule out this difference, we require that internal and completion transitions are inconsistent.

C16 A completion transition is not consistent with an internal transition.

Finally, in STATEMATE internally generated events are processed immediately, i.e. before the next external events occur. In the UML, however, internal and

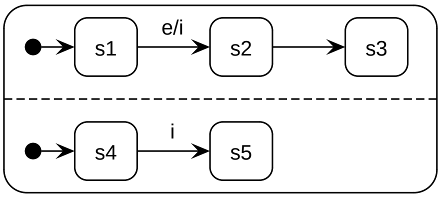

Fig. 27. Statechart to illustrate constraint C16

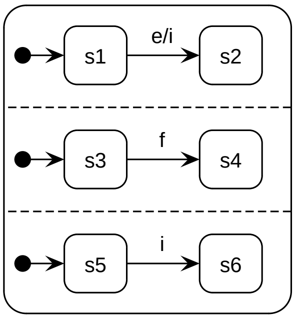

Fig. 28. Statechart to illustrate constraint C17

external events are processed interleaved. Thus, for the statechart in Figure 28, if in the initial configuration e occurs followed by f, under the UML semantics a possible sequence of steps is {s1→s2}, {s3→s4}, {s5→s6}, namely if f is processed before i. Under the seSTATEMATE semantics, however, such a sequence of steps is impossible.

To rule out this difference in behaviour, we require that under the UML semantics internal events have priority over external ones. This is the only constraint that is defined on the semantics, not on the syntax of statecharts. The only possible syntactical constraint in this case is to forbid internal transitions. However, that would rule out a whole range of statecharts that are still admissible with the current constraint.

C17 Under the UML semantics, internal events have priority over external events.

4.4 Evaluation

To evaluate the applicability of the constraints, we have selected three example statechart designs from the literature, one representative for each semantics; see Figure 29, 30, 31, and 32. We tried to select statecharts that are based on real-world examples and that use orthogonal states and internal event broadcasting, since the presented counterexamples show that these features cause most differences in behaviour. Unfortunately, for the UML semantics such an example statechart design does not seem to exist. Though event-based com-

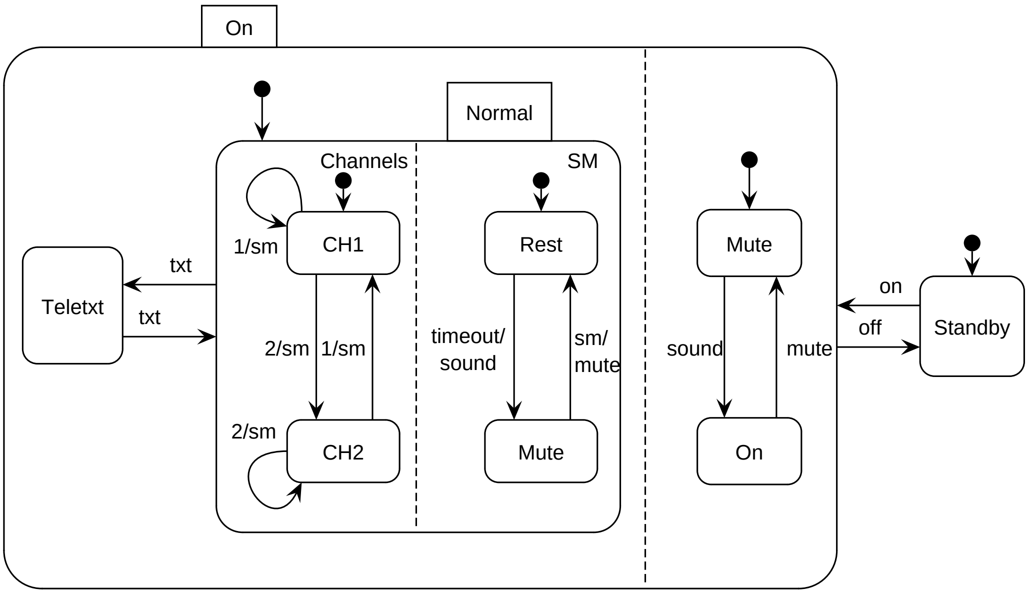

Fig. 29. Statechart for remove control of TV [25,37]

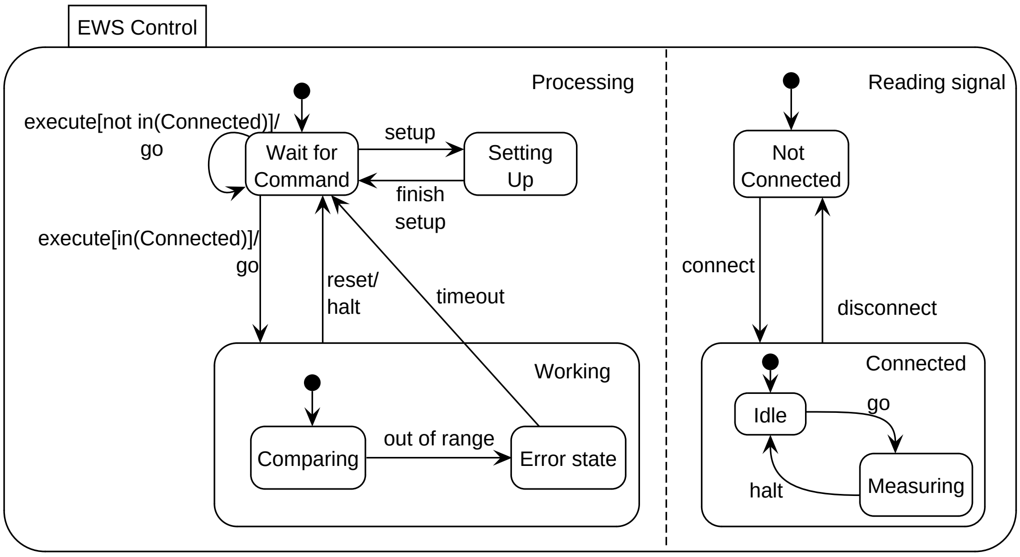

Fig. 30. Statechart of early warning system [20,24]

munication is used quite frequently in UML designs, such communication is between different statecharts (objects), not between orthogonal states of a single UML statechart. To illustrate this, we selected a few statecharts from a UML design of a cardiac pacemaker [10]; see Figure 31 and 32. The notation $O \rightarrow GEN(e)$ specifies that event e is generated and sent to object O. To ease the presentation, the UML statecharts have been simplified along the lines of an earlier version of the design [11], by aggregating a few BASIC nodes in which internal processing is done. Moreover, the non-send actions have been simplified, since these are not relevant here.

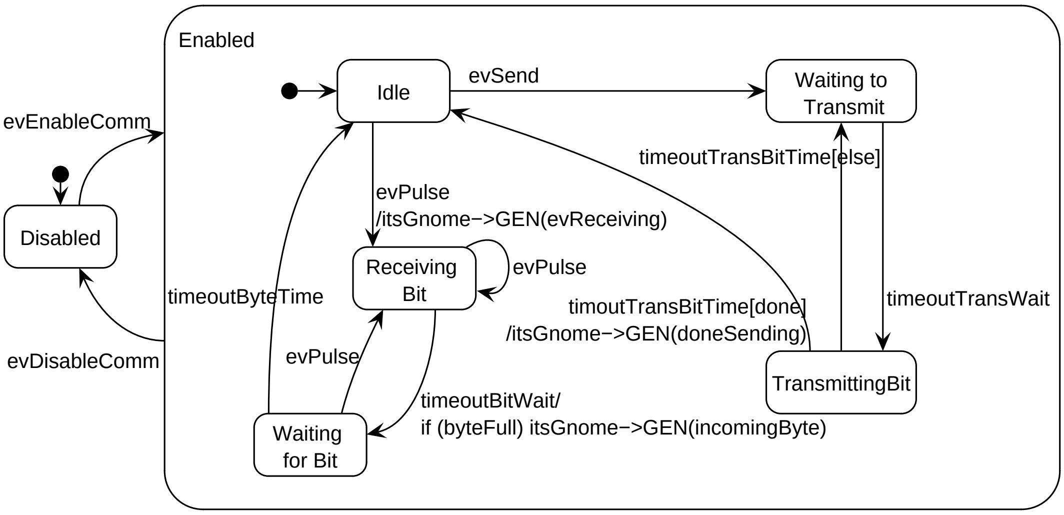

Fig. 31. Statechart of coil driver [10,11]

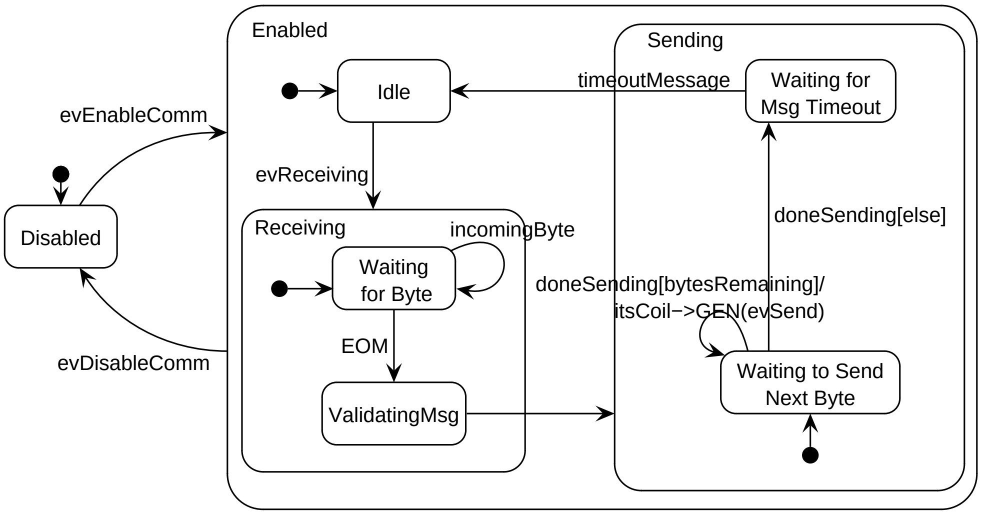

Fig. 32. Statechart of communication gnome [10,11]

Since both example UML statecharts are sequential and do not use internal events, they satisfy most constraints in a trivial way. Constraint C17 is not mentioned in the text [10,11], so it is not satisfied. To make the UML example more interesting, a single UML statechart could be constructed in which the two statecharts execute in parallel, and in which the sending of events to other objects is replaced by internal event broadcasting. However, under the UML semantics, the behaviour of such a statechart is not equivalent to the combined behaviour of the individual statecharts, so from a semantic point of view such an operation is not very meaningful. Moreover, the structure of such a statechart would resemble very much the examples shown in Figure 29 and 30.

We therefore only consider the examples in Figure 29 and 30 to test the constraints. Both examples violate constraints C3, C5, C12, and C17, but satisfy all other constraints. Note that most other constraints are trivially satisfied, since the statecharts do not use for example completion transitions. Constraint C17, which was defined on the UML semantics, is naturally not satisfied by the two examples, neither of which are UML-based. We now analyse the other constraint violations, to check whether the examples really exhibit different behaviour under the different semantics, or whether the constraints are too restrictive.

Constraint C3 (An external transition does not conflict with an internal transition) is violated in both cases by an external transition that conflicts with an internal transition with lower scope. For example, in Figure 30 external transition Connected → Not Connected conflicts with internal transition Idle → Measuring. According to the STATEMATE priority rule, which we also used for the fixpoint semantics, the external transition has priority over the internal transition if both are enabled. According to the UML priority rule, the internal transition has priority. Thus, both examples exhibit indeed different behaviour for these different semantics. However, for the fixpoint, STATEMATE and seSTATEMATE semantics, constraint C3 can be relaxed to: If an external transition conflicts with an internal transition, then the external transition has priority over the internal transition. While the relaxed version of C3 allows Figure 29 and 30, like C3 it rules out the counterexamples shown in Figures 6 and 12(a). But then relation *prec* needs to be extended, since the difference in behaviour needs to be reconciled by processing the trigger of the external transition (event disconnect for the example), before the trigger of the external transition (event execute for the example) that (in)directly triggers the internal transition.

Constraint C5 (If an internal transition is touched by an external transition, the external transition is not consistent with any transition triggering the internal transition) is violated in both cases, and indeed indicates a difference in behaviour for both statecharts under the fixpoint, STATEMATE and seSTATEMATE (and thus UML) semantics. For example, if in Figure 30 in configuration {Wait for Command,Not Connected} events execute and connect occur, then under the fixpoint semantics the next stable configuration is {Wait for Command,Idle} (since generated event go does not trigger any relevant transition), while under the STATEMATE semantics the next stable configuration is {Wait for Command,Measuring} (since go triggers a transition in the second step of the STATEMATE reaction). Under the seSTATEMATE semantics, if execute occurs before connect, then the next stable configuration is {Wait for Command,Idle}, but if connect occurs before execute, the next stable configuration is {Comparing,Measuring}. Thus, in the same configuration and with the same input events, under all three semantics different end configurations are reached.

Both examples also violate C12 (The *prec* relation is acyclic), because in both statecharts there is a cycle of external transitions. For example, in Figure 30 there is a cycle of two transitions that are triggered by connect and disconnect events. Since both transitions make each other relevant, the *prec* relation is cyclic. Nevertheless, for Figure 30 the behaviour under the Statemate and seStatemate semantics can be the same, if the event that triggers an irrelevant transition is processed before the event that triggers a relevant transition. However, such a precedence ordering on events uses the notion of relevant transition, which depends on the current configuration. In contrast, the formalisation of *prec* in Section 4.2 defines a static precedence ordering on events which is independent from any configuration. Extending *prec* to incorporate configurations will lead to a constraint that is much more difficult and expensive to check than C12.

Moreover, the statechart in Figure 29 would even violate such a relaxed version of C12. To see why, suppose the statechart is in CH1 and events 1 and 2 occur simultaneously. Then under the Statemate semantics, the next state is either CH1 or CH2. In both cases, only a single transition is taken and a single internal event sm is generated. Under the seStatemate semantics, either 1 occurs before 2 or vice versa. In both cases, also either CH1 or CH2 is reached next, but different transitions are taken compared to the Statemate reaction. For example, if CH1 is reached next, then in the Statemate reaction transition CH1 → CH1 has been taken, but in the seStatemate reaction transitions CH1 → CH2 and CH2 → CH1 (so 2 is then processed before 1). Consequently, two internal sm events are generated in the seStatemate reaction, rather than one. Thus, even though the same next states are coincidentally entered under both semantics, different transitions are taken and different events are generated. Therefore, the statechart in Figure 29 behaves differently under the Statemate and seStatemate (and thus UML) semantics.

In sum, constraints C3, C5, and C12 seem to be easily violated by existing statechart designs. Perhaps not entirely coincidental, constraints C5 and C12 also have the most complex definitions. However, the statechart designs that violate these constraints exhibit different behaviour for the different semantics. Thus, the constraints do not seem overly restrictive.

4.5 Conclusion

Table 6 and 7 summarise the constraints that we defined to rule out the presented counterexamples. Most constraints are syntactic, except the last one, which is defined on the semantics of UML. Thus, they can be easily checked. The constraints are used in the theorems of the next section. The constraints for the seStatemate vs. UML case ensure that under both semantics the

Table 6
Summary of constraints for semantics

		fixpoint vs Statemate	Statemate vs seStatemate	seStatemate vs UML
C1	There are no completion transitions	x		
C2	A transition does not indirectly trigger itself	x	x	
C3	An external transition does not conflict with an internal transition	x	x	
C4	Each transition only triggers transitions that are consistent with it	x	x	
C5	If an internal transition is touched by an external transition, the external transition is not consistent with any transition triggering the internal transition	x	x	
C6	If two different transitions are consistent, then the transitions they trigger are consistent with each other	x	x	
C7	There is no cycle of completion transitions		x	
C8	An internal transition is not touched by a completion transition		x	x
C9	An external transition does not conflict with a completion transition		x	
C10	A completion transition does not conflict with an internal transition		x	x
C11	If two completion transitions are conflicting, they have the same sources		x	
C12	The *prec* relation is acyclic		x	

same steps are taken, which implies that for each reaction the same end configurations are reached. For the weaker relation that the seStatemate and UML reactions lead to the same end configurations (without necessarily taking the same steps), only constraints C8, C10, C13 and C17 are needed.

Table 7
Summary of constraints (continued from Table 6)

		fixpoint vs Statemate	Statemate vs seStatemate	seStatemate vs UML
C13	Two conflicting transitions with the same trigger event have the same sources and same scope			x
C14	Each transition generates at most one event			x
C15	Two consistent transitions having the same trigger event generate the same event			x
C16	A completion transition is not consistent with an internal transition			x
C17	Under the UML semantics, internal events have priority over external events			x

Some constraints are arbitrary, in the sense that the presented counterexamples could be resolved in different ways. For example, instead of C7, C8, C9, C11, and C16, we could have adopted C1, which rules out completion transitions altogether, and thus is much more restrictive. However, our aim has been to define constraints that allow as many statechart designs as possible. Though in principle other sets of constraints can be defined, the discussion at the end of Section 4.2 shows that this is not straightforward. An alternative set of constraints may rule out the counterexamples presented in this paper, but alternative counterexamples may exist that it does not rule out. Moreover, most of the presented counterexamples, for instance for the most complex constraints C5 and C12, are rather simple. Therefore, it does not seem that straightforward to find alternative constraints that are more simple, yet less restrictive, than the ones we defined.

The evaluation of the constraints on a few existing, real-world statechart designs suggests that especially constraints C3, C5 and C12 are easily violated. However, the violating statechart designs do indeed exhibit different behaviour under the different statechart semantics, so the constraints do not seem overly restrictive. Moreover, this suggests that in practice, a statechart design is likely to exhibit different behaviour for the different semantics, which hampers a meaningful exchange of statechart designs among both designers and tools. However, to reach a final conclusion, the constraints need to be evaluated on a large set of industrial statechart designs, which is outside the scope of this paper.

To simplify the constraints, ruling out completion and internal transitions seems most fruitful. In that case, all constraints but C12 and C13 are automatically satisfied. Though such a simplification rules out a whole range of statecharts that are allowed by the current constraint set, there is empirical evidence in support of such a restriction. Our formalisation does not cover activities. If activities are considered, then in UML statechart designs, a completion transition is typically enabled if all internal activities in its source states have been completed, whereas in STATEMATE a completion transition is enabled as soon as its sources have been entered. STATEMATE expresses completion of an activity by a separate event, whereas UML uses completion (empty) events for this. Consequently, completion transitions can signify something different in STATEMATE and UML if activities are considered. Moreover, none of the fixpoint and STATEMATE statechart designs we found in the literature use completion (null) events. Regarding internal transitions, according to Leveson et al. [29] internal events are one of the key constructs that lead to errors in statechart designs. Instead, they propose to use data dependencies to determine the order in which transitions are taken, rather than relying on a specific statechart semantics. Independently, UML statecharts seem to follow this modelling style, since UML statecharts do not use internal event broadcasting very often.

5 Relation

We relate the different semantics pairwise to each other (fixpoint to STATEMATE, STATEMATE to seSTATEMATE and seSTATEMATE to UML), and show that the semantics are equivalent for linear, stuttering-closed, separable properties. As motivated in Section 1, we prove for the first two groups that given a configuration, the effects of the system reactions under the different semantics are similar, so the same end configurations are eventually reached. For the last group, we prove that the same steps are taken, which implies that the same end configurations are reached. The constraints defined in the previous section are used in the theorems and proofs.

5.1 Preliminaries

We introduce some concepts and notations for transition systems and statecharts that we use in the theorems and proofs.

Transition systems. Given two STSes $STS_1 = (V_1, init_1, \rightarrow_1)$ and $STS_2 = (V_2, init_2, \rightarrow_2)$, let $R \subseteq \Sigma_1 \times \Sigma_2$ be a relation on their valuations. Consider runs $\pi_1 = s_0, s_1, ..$ of STS_1 and $\pi_2 = t_0, t_1..$ of STS_2. Runs π_1 and π_2 are *stuttering R-equivalent* [5,12] if and only if there exists infinite sequences of natural numbers $i_0 = 0 < i_1 < i_2 \ldots$ and $k_0 = 0 < k_1 < k_2 \ldots$ such that for all $j \geq 0$, for all $i_j \leq l < i_{j+1}$ and $k_j \leq m < k_{j+1}$, $s_l \; R \; t_m$.

Next, we introduce notation for describing a sequence of transitions between stable valuations. Let σ, σ' be two stable valuations that are both either fixpoint, STATEMATE or UML valuations. If $\sigma \rightarrow \sigma_1 \rightarrow .. \rightarrow \sigma_n \rightarrow \sigma'$ such that each intermediary valuation σ_i is unstable, where $1 \leq i \leq n$, we write $\sigma \twoheadrightarrow \sigma'$.

Finally, given a state $s \in \mathcal{S}$, we write $\sigma \models in(s)$ if $s \in \sigma(C)$.

Statecharts. In the sequel, we sometimes use the concept of a sequential component of a statechart, which identifies a maximal subset of the statechart not containing any parallelism. Formally defined, a sequential component of a statechart is a maximal set $X \subseteq \mathcal{S}$ of states such that for any $x, y \in X$:

- x and y are inconsistent, or
- x and y are ancestrally related.

Thus, if a configuration contains two states of a sequential component, they are hierarchically related. For example, Figure 2 has two sequential components: $\{$*root*,Off,On,Turnstile Control,Blocked,Unblocked$\}$ and $\{$*root*,Off,On,Card Reader Control,Ready,Card Entered,Turnstile Unblocked$\}$.

5.2 From fixpoint semantics to STATEMATE *and back*

Before we prove that fixpoint runs are stuttering equivalent to STATEMATE runs with respect to sequential components, we introduce a general lemma and theorem that we use in the sequel. The general lemma states that the activation of a state s in a valuation σ' can be computed from some previous valuation σ and the steps taken to reach σ' from σ.

Lemma 1 *Let σ be a valuation, s a state, and let St_1, St_2, .., St_k be a sequence of steps that is taken such that a valuation σ' is reached.*

$$\begin{aligned}\sigma' \models in(s) \Leftrightarrow count(s,\sigma) &- |\{t \in \textstyle\bigcup_{i:1..k} St_i \mid s \in children^*(scope(t))\}| \\ &+ \ \ |\{t \in \textstyle\bigcup_{i:1..k} St_i \mid s \in dcomp(scope(t) \cup target(t))\}| = 1\end{aligned}$$

where

$$count(s,\sigma) = \begin{cases} 1 \text{ , if } \sigma \models in(s) \\ 0 \text{ , otherwise.} \end{cases}$$

Proof. By definition, in each step St_i, there is at most one transition t entering or leaving s. By filtering the transitions leaving or entering s from the sequence of steps, we can derive a sequence of transitions $t_1, t_2, .., t_l$. From the definition of *nextConfig*, it follows that in this sequence, if some transition t_i, where $0 < i < l$, enters (leaves) s, the next transition leaves (enters) s. Using this observation, the claim can be easily proven. □

Observe that each stable STATEMATE valuation is a stable fixpoint valuation by definition, and that by C1, each stable fixpoint valuation is also stable in STATEMATE. We use Lemma 1 to prove the next theorem, which states that if from a stable valuation σ another stable valuation σ' can be directly reached under the fixpoint semantics, so $\sigma \twoheadrightarrow^{FP} \sigma'$, then under the STATEMATE semantics valuation σ' is also reachable from σ, but through some additional intermediary stable valuations. This theorem shows that the valuation resulting from a set of concurrent external events equals the valuation that results when the events occur sequentially in arbitrary order.

Theorem 1 *Let SC be a statechart satisfying C1, C2, C3, C4, C5, and C6. Let σ, σ' be some valuations that are stable under both the fixpoint and* STATEMATE *semantics.*

i If $\sigma \twoheadrightarrow^{FP} \sigma'$, then $\sigma \twoheadrightarrow^{SM} \sigma'$.
ii If $\sigma \twoheadrightarrow^{SM} \sigma'$, then $\sigma \twoheadrightarrow^{FP} \sigma'$.

Proof. (i) Under the fixpoint semantics, a reaction to a set of input events consists of one step only, while under the STATEMATE semantics, a reaction consists of a sequence of steps. Denote by St^{FP} the single step taken under the fixpoint semantics to reach σ' from σ. By C1 and C2, under the STATEMATE semantics, the system does not diverge. Therefore, a reaction consists of a finite sequence of steps $St_1^{SM}, St_2^{SM}, .., St_k^{SM}$; denote the valuation reached by σ''.

We will prove $St^{FP} = \bigcup_{i:1..k} St_i^{SM}$. From Lemma 1 then follows that $\sigma' = \sigma''$.

$\subseteq$ **direction:** (Sketch.) The claim can be easily proven by induction on the causal chain of transitions in St^{FP}. Given step St^{FP}, its causal chain is a sequence $CC_1, CC_2, .., CC_n$, of sets of transitions, where

$$CC_1 = \{\ t \in St^{FP} \mid \mathit{external}(t)\ \}$$

and

$$CC_{j+1} = \{\ t \in St^{FP} \mid \exists t' \in CC_j : t' \gg t\ \}$$

where $1 \leq j < n$.

For the induction case, C3 is needed.

$\supseteq$ **direction:**

We prove by induction on the sequence of steps taken under the STATEMATE semantics that each transition taken in some step St_i^{SM}, where $0 < i \leq k$, is taken in St^{FP}.

Base case: step St_1^{SM}.

Take an arbitrary transition $t \in St_1^{SM}$. By definition of the STATEMATE semantics, since St_1^{SM} is the first step, transition t must be triggered by an external event e. Event e can also occur in σ under the fixpoint semantics, so $t \in St^{FP}$.

Induction step: step St_{i+1}^{SM}, where $i > 0$.

By definition of the STATEMATE semantics and by C1, St_{i+1}^{SM} only contains internal transitions (all external transitions have been taken in step St_1^{SM}).

By the induction hypothesis all transitions in previous steps $St_1^{SM}, .., St_i^{SM}$ can be taken, i.e., they are in St^{FP}.

Take an arbitrary transition $t \in St_i^{SM}$. We now show that t can be taken under the fixpoint semantics, by showing that t can become relevant and enabled.

- t is relevant: By C4 and C5, $\mathit{source}(t) \subseteq \sigma(C)$.
- t can be enabled: By C3, t can only conflict with another internal transition, say t_{conflict}. Let t_{trigger} be the transition triggering t, so $t_{\mathit{trigger}} \gg t$. By definition of the STATEMATE semantics, $t_{\mathit{trigger}} \in St_i^{SM}$. By the induction hypothesis, $t_{\mathit{trigger}} \in St^{FP}$. By C6, under the fixpoint semantics the trigger event of t_{conflict} is not generated. Therefore, $t \in St^{FP}$.

(ii) By similar reasoning as (i). □

Next, we show that every fixpoint semantics run has a stuttering equivalent Statemate run and vice versa, provided observations are restricted to sequential components of SC. Given a sequential component X of statechart SC, define a relation R_X such that

$$\begin{aligned} \sigma \; R_X \; \sigma' \Leftrightarrow \quad & \forall x \in X : \sigma \models in(x) \Leftrightarrow \sigma' \models in(x) \\ & \wedge \; \sigma \models stable^{FP}(C, I) \Leftrightarrow \sigma' \models stable^{SM}(C, I). \end{aligned}$$

Theorem 2 *Let SC be a statechart satisfying C1, C2, C3, C4, C5, and C6. Let X be some sequential component of SC.*

i *For each fixpoint semantics run* $\pi = \sigma^{FP}_{init} \rightarrow \sigma^{FP}_1 \rightarrow \sigma^{FP}_2 \rightarrow ..$, *there exists a* Statemate *semantics run* π' *that is* R_X*-stuttering equivalent.*
ii *For every* Statemate *semantics run* $\pi' = \sigma^{SM}_{init} \rightarrow \sigma^{SM}_1 \rightarrow \sigma^{SM}_2 \rightarrow ..$, *there exists a fixpoint semantics run* π *that is* R_X*-stuttering equivalent.*

Proof. (Sketch.) We only prove (i); (ii) can be proven by similar reasoning.

By definition of the fixpoint semantics, run π can equivalently be written as $\sigma^{FP}_0 \twoheadrightarrow^{FP} \sigma^{FP}_2 \twoheadrightarrow^{FP} \sigma^{FP}_4 ...$ For each σ^{FP}_i, σ^{FP}_{i+2}, where $i \geq 0$ and i is even, from Theorem 1(i) it follows that $\sigma^{FP}_i \twoheadrightarrow^{SM} \sigma^{FP}_{i+2}$. So run
$\pi' = \sigma^{FP}_0 \twoheadrightarrow^{SM} \sigma^{FP}_2 \twoheadrightarrow^{SM} \sigma^{FP}_4 ..$ exists under the Statemate semantics.

Let σ^{FP}_i, σ^{FP}_{i+2} be valuations from π (and thus from π'). Denote by T^{FP} the transitions taken under the fixpoint semantics to reach σ^{FP}_{i+2} from σ^{FP}_i and denote by T^{SM} the transitions taken under the Statemate semantics to reach σ^{FP}_{i+2} from σ^{FP}_i. From the proof of Theorem 1, it follows that $T^{FP} = T^{SM}$, where T^{FP} is the single step taken in the fixpoint reaction $\sigma^{FP}_i \twoheadrightarrow^{FP} \sigma^{FP}_{i+2}$.

We now argue that for each pair σ^{FP}_i, σ^{FP}_{i+2} and each sequential component X, at most one transition in $T^{FP} (= T^{SM})$ affects the states in X. Observe that by definition of X, the value of states in X can only change by taking a transition of which a source state and a target state are in X. However, this implies that the scope of t is in X too. Since X does not contain consistent (parallel) states, for each step St taken under fixpoint or Statemate semantics, at most one transition in St can affect states in X, i.e. at most one transition $t \in St$ has $scope(t) \in X$. Thus, if St does not contain a transition whose scope is in X, no states belonging to X are left or entered by taking St. Since T^{FP} is the single step taken in the fixpoint reaction, at most one transition in $T^{FP} (= T^{SM})$ can affect states in X. Using this observation and since $T^{FP} = T^{SM}$, it is easy to prove that π and π' are stuttering R_X-equivalent. □

We use Lemma 1 to prove a theorem that is similar to Theorem 1. The theorem states that if from a stable valuation σ another stable valuation σ' can be directly reached under the STATEMATE semantics, so $\sigma \twoheadrightarrow^{SM} \sigma'$, then under the seSTATEMATE semantics valuation σ' is also reachable from σ, but through some additional intermediary stable valuations. (Note that every stable STATEMATE valuation is also a stable seSTATEMATE valuation and vice versa.) Though the theorem is proven using the STATEMATE priority rule, it can be proven for any priority rule that induces an acyclic relation on transitions.

Theorem 3 *Let SC be a statechart satisfying C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12. Let σ, σ' be valuations that are stable under both the* STATEMATE *and se*STATEMATE *semantics. If $\sigma \twoheadrightarrow^{SM} \sigma'$,*
then $\sigma \twoheadrightarrow^{seSM} \sigma_1 \twoheadrightarrow^{seSM} \sigma_2 .. \twoheadrightarrow^{seSM} \sigma'$.

Proof. Denote by $St_1^{SM}, St_2^{SM}, .., St_k^{SM}$ the sequence of steps that are taken under the STATEMATE semantics to reach σ' from σ. Let $E = \{e_1, .., e_n\}$ be the set of external events that occur under the STATEMATE semantics in the successor valuation of σ.

Under the seSTATEMATE semantics, these external events can occur sequentially in any order, say e_1, e_2, .., e_n, such that for some σ'',
$\sigma \twoheadrightarrow_1^{seSM} \sigma_1 \twoheadrightarrow_2^{seSM} \sigma_2 .. \twoheadrightarrow_n^{seSM} \sigma''$. Denote by $St_1^{seSM}, St_2^{seSM}, .., St_l^{seSM}$ the sequence of all steps taken under the seSTATEMATE semantics to reach σ''. By definition of $\twoheadrightarrow$, between two pair of valuations σ_i, σ_{i+1}, where $i \geq 0$, such that $\sigma_i \twoheadrightarrow^{seSM} \sigma_{i+1}$, one or more steps are taken. Therefore, $l \geq n$.

We show $\bigcup_{i:1..k} St_i^{SM} = \bigcup_{j:1..l} St_j^{seSM}$. From Lemma 1, it then follows that $\sigma' = \sigma''$.

$\subseteq$ direction:

We prove by induction on the sequence of steps taken under the STATEMATE semantics that each transition taken in some step St_i^{SM}, where $0 < i \leq k$, can also be taken under the seSTATEMATE semantics.

To simulate the STATEMATE semantics, events in the seSTATEMATE semantics need to be processed in a certain order. Assume without any loss of generality that every event $e \in E$ triggers some transition in *relevant*(C). (Irrelevant events in E can be processed before the relevant ones.)

Given two transitions t, $t' \in$ *relevant*(C), such that $\{$*event*(t), *event*$(t')\} \subseteq E$ and $t \in St_1^{SM}$ and $t' \notin St_1^{SM}$. Then *event*(t) beats *event*(t'). Formally, the

$\mathit{beats}(E, C, \mathit{St}) \subseteq \mathcal{E} \times \mathcal{E}$ relation is defined as:

$$\begin{aligned}(e, e') \in \mathit{beats}(E, C, \mathit{St}) \Leftrightarrow\ & \{e, e'\} \subseteq E \\ & \wedge\ \exists t, t' \in \mathcal{T} : \mathit{event}(t) = e \wedge \mathit{event}(t') = e' \\ & \wedge\ (\ (\mathit{conflict}(t, t') \wedge t \in \mathit{St} \wedge t' \notin \mathit{St}) \\ & \qquad\qquad \vee(\mathit{makesRelevant}(t', t)\ ).\end{aligned}$$

By C12 and the fact that the Statemate priority relation between transitions is acyclic, the *beats* relation between events in E is acyclic.

Now, process events in E in such an order that if an event e is processed, all events that beat e have been processed already.

Base case: step St^{SM}_1.

Take an arbitrary transition $t \in \mathit{St}^{SM}_1$. Since σ is stable and St^{SM}_1 is the first step, transition t must be triggered by an external event. If t's trigger event is e_1, t is taken in St^{seSM}_1. Otherwise, if t's trigger event is e_j, for some $1 < j \leq n$, then by C2, C7 and C8, the statechart does not diverge, so e_j is eventually processed.

We now show that the sources of t are still active when e_j is processed. By C3 and C9, t's sources can only be left because of another external transition with trigger event e. By definition of *beats*, we have that e_j beats e. Hence, e is not processed before e_j, and thus, the sources of t stay active until e_j has been processed. And when e_j is processed, t becomes enabled and can be taken.

Induction step: step St^{SM}_{i+1}, where $i > 0$.

By definition, St^{SM}_{i+1} only contains completion and internal transitions (all external transitions have been taken in step St^{SM}_1). Take an arbitrary transition $t \in \mathit{St}^{SM}_{i+1}$. We now show that t can be taken in some step under the seStatemate semantics.

By the induction hypothesis, all transitions in the previous steps $\mathit{St}^{SM}_1, .., \mathit{St}^{SM}_i$ can be taken (but not necessarily in the same order). So by the induction hypothesis, the sources of t are entered, but not necessarily all simultaneously at the same time.

- t is a completion transition. By C9 and C10, t only conflicts with completion transitions. Then, by C11, t is only conflicting with completion transitions that have the same sources. Thus, none of t's source states is left before all of t's sources have become active. So, t's sources become active under the seStatemate semantics. Thus, t becomes enabled and can be taken.

- t is an internal transition. Let t_{trigger} be the transition triggering t, so t_{trigger} generates $\mathit{event}(t)$. By C4, t_{trigger} is consistent with t.

  First, we show $\mathit{event}(t)$ is processed only when t is relevant, so all sources of t are active. This is obviously true if t is in $\mathit{relevant}(C)$. So assume t is not relevant in C. Let t' be a transition touching t. Obviously, t' is taken in some earlier step than St_{i+1}^{SM}. By C8, t' is not a completion transition. Next, by definition of the Statemate semantics, $t_{\mathit{trigger}} \in St_i^{SM}$. Since both t' and t_{trigger} are taken in some earlier step than St_{i+1}^{SM}, both t' and t_{trigger} are by the induction hypothesis also taken under the seStatemate semantics. We have to show that under the seStatemate semantics, $\mathit{event}(t)$ is processed only after t' has been taken, so t' is taken before or simultaneously with t_{trigger}, which generates $\mathit{event}(t)$.

  · t' is external. Then $t' \in St_1^{SM}$. Then, since t_{trigger} is not consistent with t' by C5, $t_{\mathit{trigger}} \notin St_1^{SM}$ by definition of step. Since $t_{\mathit{trigger}} \in St_i^{SM}$, step St_1^{SM} is before St_i^{SM}, for $i > 1$. Since $t' \in St_1^{SM}$, $t_{\mathit{trigger}} \in St_i^{SM}$, for $i > 1$, and by C5 t_{trigger} is inconsistent with t', there is a path from t' to t_{trigger}. Each transition $t_{\mathit{intermediate}}$ on this path is taken in the steps between St_1^{SM} and St_i^{SM}. By the induction hypothesis, $t_{\mathit{intermediate}}$ is also taken under the seStatemate semantics.

  Thus, t' is taken before t_{trigger} under the seStatemate semantics.

  · t' is internal. Denote by t'_{trigger} the transition triggering t'. By definition of the Statemate semantics, $t' \in St_k^{SM}$, where $1 < k < i+1$, and $t'_{\mathit{trigger}} \in St_{k-1}^{SM}$. Since $k < i+1$, transition t'_{trigger} is taken in an earlier step St_{k-1}^{SM} than the step St_i^{SM} in which t_{trigger} is taken. Moreover, since t' and t are inconsistent by C6, t'_{trigger} is inconsistent with t_{trigger}. Thus, there is a path from t'_{trigger} to t_{trigger}. Each transition $t_{\mathit{intermediate}}$ in this path is taken in the steps between St_{k-1}^{SM} and St_i^{SM}. By the induction hypothesis, $t_{\mathit{intermediate}}$ is also taken under the seStatemate semantics.

  Thus, t'_{trigger} is taken before t_{trigger} under the seStatemate semantics, hence t' is taken before or simultaneously with t_{trigger} under the seStatemate semantics.

  In both cases, the transition t' touching t is taken before or simultaneously with the transition t_{trigger} that triggers t, so $\mathit{event}(t)$ is only processed after t' has been taken. By C2, C7 and C8, the statechart does not diverge, so $\mathit{event}(t)$ is eventually processed.

  Next, we show that t can become relevant under the seStatemate semantics. By C3 and C10, t only conflicts with other internal transitions. Suppose under the seStatemate semantics a source of t is left by some conflicting internal transition t_{internal} before t has become relevant. We show that then the source is re-entered under the seStatemate semantics before t becomes relevant. First, by the $\supseteq$ direction, t_{internal} is taken under the Statemate semantics as well. Since t and t_{internal} are inconsistent but are both taken under the Statemate semantics, there is a path connecting t and t_{internal}. Since under the seStatemate semantics t_{internal} is relevant before t is relevant, there is a path from t_{internal} to t. Therefore, t_{internal} is taken before t under the Statemate semantics, in some step St_p^{SM}, where $1 < p < i+1$. Each transition $t_{intermediate}$ in the path from $t_{internal}$ to t is taken in the steps between St_p^{SM} and St_{i+1}^{SM}. By the induction hypothesis, $t_{intermediate}$ is also taken under the seStatemate semantics. This implies the source of t is entered again before t becomes relevant. Thus, all the sources of t can become active, i.e. t can become relevant under the seStatemate semantics.

Since t can become relevant and $event(t)$ is processed only when t is relevant, t can become enabled and be taken under the seStatemate semantics.

$\supseteq$ direction:

We show by contradiction that under the seStatemate semantics no extra transitions are taken. Consider a transition t taken in some arbitrary seStatemate step St_j^{seSM}, where $0 < j \leq l$, such that all transitions taken in previous seStatemate steps, are taken under the Statemate semantics. Suppose t is not taken in some step St_i^{SM} where $0 < i \leq k$. Then after the Statemate reaction has finished, all of t's sources are active.

- t is an external transition. Then $event(t)$ is processed because it triggers some other external transition $t' \in St_1^{SM}$. Since t is not in St_1^{SM}, there must be some transition $t'' \in St_1^{SM}$ that makes t relevant, and $event(t'')$ is processed before $event(t)$, so $event(t'')$ precedes $event(t)$ (otherwise t would not be taken). But since t'' makes t relevant, also $event(t)$ precedes $event(t'')$. So C12 is violated.
- t is a completion transition. Then t becomes enabled as soon as it has become relevant. But then it can be taken under the Statemate semantics as well, which leads to a contradiction.
- t is an internal transition. By C4, t is triggered by a transition $t_{trigger}$ consistent with t. By assumption, $t_{trigger}$ has been taken already under the Statemate semantics.

  Let t' be a transition touching t. By C8, t' is external or internal.

  · t' is external. Then by C5, $event(t')$ is processed before $event(t_{trigger})$. So, if $t_{trigger}$ is taken, t' has been taken already under the Statemate semantics.

  · t' is internal. Then by C6, the trigger transition t_x of t' is inconsistent with $t_{trigger}$. Since t_x is taken before $t_{trigger}$ under the seStatemate semantics (since t' is taken before t), there is a path from t_x to $t_{trigger}$. Then t_x must have been taken already under the Statemate semantics, otherwise it could not trigger t'. So t' is taken before or simultaneously with $t_{trigger}$ under the Statemate semantics. So the event generated by $t_{trigger}$ is processed only after t' has been taken under the Statemate semantics.

  In both cases, t' can also taken under the Statemate semantics, leading to a contradiction.

□

Using this theorem, we now show that STATEMATE runs are stuttering equivalent to seSTATEMATE runs w.r.t. sequential components. (The reverse direction is trivial, since by definition each seSTATEMATE run is also a STATEMATE run.)

For a sequential component X of statechart SC, define a relation R_X such that

$$\begin{aligned}\sigma\ R_X\ \sigma' \Leftrightarrow \forall x \in X : \sigma \models \mathit{in}(x) \Leftrightarrow \sigma' \models \mathit{in}(x) \\ \wedge\ \sigma \models \mathit{stable}^{SM}(C,I) \Leftrightarrow \sigma' \models \mathit{stable}^{SM}(C,I).\end{aligned}$$

For the seSTATEMATE semantics, we do not use a separate predicate for $\mathit{stable}(C,I)$, but reuse the definition from the STATEMATE semantics.

Theorem 4 *Let SC be a statechart that satisfies C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12. Let X be some sequential component of SC. For each* STATEMATE *run* $\pi = \sigma_0^{SM} \rightarrow \sigma_1^{SM} \rightarrow \sigma_2^{SM} \rightarrow ..$, *there exists a se*STATEMATE *run* π' *that is stuttering* R_X*-equivalent.*

Proof. Similar to the proof of Theorem 2, using C7 instead of C1 and Theorem 3 instead of Theorem 1. □

*5.4 From se*STATEMATE *to UML and back*

We first show that every seSTATEMATE run has an R-related UML run and vice versa, where R relates valuations of seSTATEMATE with valuations of UML. Let σ^{seSM} be an arbitrary seSTATEMATE valuation and let σ^{UML} be an arbitrary UML valuation. Then R is defined by:

$$\sigma^{seSM}\ R\ \sigma^{UML} \Leftrightarrow \forall s \in \mathcal{S} : \sigma^{seSM} \models \mathit{in}(s) \Leftrightarrow \sigma^{UML} \models \mathit{in}(s).$$

Theorem 5 *Let SC be a statechart satisfying Constraints C8, C10, C13, C14, C15, C16. For every se*STATEMATE *run* $\pi = \sigma_{init}^{seSM} \rightarrow \sigma_1^{seSM} \rightarrow \sigma_2^{seSM} \rightarrow ..$, *there exists a UML run* π' *that is R-stuttering equivalent.*

Proof. (Sketch.) Construct a run π' by mapping each stable seSTATEMATE valuation σ^{seSM} in π to a stable UML valuation σ^{UML} in which the queue is empty. Clearly, σ^{seSM} and σ^{UML} are R-related. Let St_1^{seSM}, St_2^{seSM}, .. be the sequence of steps taken under the seSTATEMATE semantics from σ^{seSM} such that either a stable valuation is reached after taking the (finite) sequence, or the system diverges, so then the sequence is infinite.

By C16, each step in the sequence contains either only completion transitions or only internal transitions. Suppose in the sequence a step St_i^{seSM} only containing completion transitions is followed by a step St_{i+1}^{seSM} only containing internal transitions. Then by definition of the seSTATEMATE semantics, an internal transition from St_i^{seSM} is either consistent with or touches a completion transition from St_{i+1}. However, then C16 and C8 are violated, respectively. Therefore, the sequence consists of either only steps containing completion transitions, or only steps containing internal transitions, or steps containing internal transitions followed by steps containing completion transitions.

We only consider the last case, since the other cases can be proven by similar reasoning. We show by induction on the sequence of steps that under the UML semantics the same sequence of steps can be taken, but augmented with some additional empty steps at the end. Since both (se)STATEMATE and UML use the same semantics for taking a step, this implies the same subsequent configurations are reached by taking the steps in the sequence. Consequently, since σ^{seSM} and σ^{UML} are R-related, the resulting valuations reached by taking the steps in the sequence, are R-related too.

Base case: step St_1^{seSM}.

If in σ^{seSM} some external event e occurs that causes step St_1^{seSM} to be taken, e can also occur in σ^{UML}. Under the UML semantics, event e is put in the queue and a step St_1^{UML} is taken. Since by definition of the mapping σ^{seSM} and σ^{UML} have the same configuration, the same transitions are enabled by e. Next, by C13, St_1^{seSM} equals St_1^{UML}.

Induction step: step St_{i+1}^{seSM}.

By the induction hypothesis, the previous steps $St_1^{SM}..St_i^{SM}$ have been taken, so the same configurations are reached under the seSTATEMATE and UML semantics. Denote by σ_{i+1}^{seSM} and σ_{i+1}^{UML} the R-related valuations reached. By C16, St_{i+1}^{SM} contains either (i) only completion transitions or (ii) only internal transitions.

For (i), since σ_{i+1}^{seSM} and σ_{i+1}^{UML} have the same configuration, the same completion transitions are enabled in σ_{i+1}^{seSM} and σ_{i+1}^{UML}. Therefore, by C13, the step St_{i+1}^{UML} taken in σ_{i+1}^{UML} equals step St_{i+1}^{seSM}. By C14 and C15, at most one event i is generated in St_i^{seSM}. If an event i is generated, I contains only i in σ_{i+1}^{seSM}, while i has been added to queue q in σ_{i+1}^{UML}. By C16, i does not trigger any consistent transitions. By C8, i does not trigger any transition touched by the (completion) transitions in St_{i+1}^{seSM}. Therefore, if i is subsequently processed, an empty step is taken. However, i is only processed if all completion steps have been taken and the reaction has finished. By C8, then no internal steps can be taken further, so all internal events in the queue can be processed one by one, and a sequence of empty steps is taken as result.

For (ii), by C14 and C15, there is a single internal event i that triggers the transitions in St_{i+1}^{seSM}. By definition of the seSTATEMATE semantics, event i is generated in St_i^{seSM}. By the induction hypothesis, i is generated too in St_i^{UML}. Since by C8 no completion transitions have been taken in the previous steps, all previously generated internal events have been immediately processed under the UML semantics. So the queue was empty when i was generated. Therefore i is head of the queue and next is processed immediately. Since by the induction hypothesis the same configurations are reached after taking St_i^{seSM} and St_i^{UML}, i triggers the same transitions in σ_{i+1}^{UML} as in σ_{i+1}^{seSM}. Therefore, by C13, step St_{i+1}^{UML} equals step St_{i+1}^{seSM}.

□

Next, we show that every UML run has a stuttering R-equivalent seSTATEMATE run.

Theorem 6 *Let SC be a statechart satisfying Constraints C8, C13, C14, C15, C16, C17. For every UML run $\pi' = \sigma_{init}^{UML} \rightarrow \sigma_1^{UML} \rightarrow \sigma_2^{UML} \rightarrow ..$, there exists a se*STATEMATE *run π that is R-stuttering equivalent.*

Proof. (Sketch.) Construct a run π by mapping each stable valuation σ_j^{UML} in which the first event e in the queue is external, to an R-similar stable seSTATEMATE valuation σ_i^{seSM}. By C17, under the UML semantics, internal events generated in the processing of e are processed before the next external event from the queue is processed. Using similar reasoning as in the proof of Theorem 5, one can prove that the successor valuations of σ_i^{seSM} and σ_j^{UML} are R-related as well. However, C10 is not needed, since in an unstable UML valuation an internal and a completion transition cannot be both enabled. □

Theorem 5 and 6 can be weakened by referring to sequential components only, similar to Theorem 2 and 5. In that case, only C8, C10, C13 and C17 are needed to ensure that the same end configurations are reached under both semantics.

5.5 Temporal logic

Having formally related the different semantics by means of stuttering relations, we now use this result to specify a set of properties that cannot distinguish between the different semantics, i.e. each property is true under one semantics iff it is true under the other. As property specification language, we use propositional linear temporal logic with both past and future operators (PLTL) [35].

Since properties need to be invariant under the semantics used, they are restricted in two ways. First, we do not use the next time operator and its past time equivalent. A property containing such operators is not stuttering closed. Such a property could detect the number of times a transition system stutters, which distinguishes the semantics from each other. We denote the subset of PLTL we use by PLTL-X.

Second, properties can only refer to variables common to all semantics, in this case state variables and events. However, events are processed differently by the different semantics. Since properties referring to events could detect this difference, we only allow state variables to be referenced. Given a statechart SC, the set $AP(SC)$ of atomic propositions is defined by:

$$AP(SC) = \{ \ in(s) \mid s \in \mathcal{S}_{SC} \ \}.$$

Given set $AP(SC)$, the set of past linear temporal logic formulas without next and its past time equivalent (PLTL-X), is defined by:

$$\varphi ::= p \mid \varphi \wedge \varphi \mid \neg\varphi \mid \varphi \,\mathsf{U}\, \varphi \mid \varphi \,\mathsf{S}\, \varphi$$

where $p \in AP(SC)$, U stands for Until, and S for Since. We use the usual abbreviations for $\vee$, $\Rightarrow$, and so on.

The semantics of PLT-X is defined in terms of paths of an STS. A path is an infinite sequence of valuations, $\pi = \sigma_0\sigma_1\sigma_2..$, such that for every $i \geq 0$, $\sigma_i \rightarrow \sigma_{i+1}$. Let π^j denote the suffix of π starting at σ_j. The satisfaction relation $\models$ for formulas is defined inductively as follows:

$$\begin{array}{rcl}
\pi^j \models in(s) & \Leftrightarrow & s \in \sigma_j(C) \\
\pi^j \models \neg\varphi & \Leftrightarrow & \pi^j \not\models \varphi \\
\pi^j \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow & \pi^j \models \varphi_1 \text{ and } \pi^j \models \varphi_2 \\
\pi^j \models \varphi_1 \,\mathsf{U}\, \varphi_2 & \Leftrightarrow & \text{there exists } k \geq j \text{ such that } \pi^k \models \varphi_2, \\
 & & \text{and } \pi^i \models \varphi_1 \text{ for every } j \leq i < k \\
\pi^j \models \varphi_1 \,\mathsf{S}\, \varphi_2 & \Leftrightarrow & \text{there exists } 0 \leq k \leq j \text{ such that } \pi^k \models \varphi_2, \\
 & & \text{and } \pi^i \models \varphi_1 \text{ for every } k < i \leq j.
\end{array}$$

A formula φ is true in a valuation iff it is true for all paths starting in the valuation. A formula φ is true of a symbolic transition system STS, written $STS \models \varphi$, iff it is true for all paths starting in the initial valuation of STS.

The following lemma states that stuttering R-equivalent paths satisfy the same PLTL-X properties. In the theorem, $\varphi(SC)$ denotes that the atomic propositions contained in PLTL-X formula φ are elements of $AP(SC)$. The proof is straightforward and therefore omitted. The lemma is a slight modification of the folklore theorem stating that stuttering equivalent paths satisfy the same PLTL-X formulas, first observed by Lamport [27].

Lemma 2 *Let $\varphi(SC)$ be a PLTL-X property of a statechart SC. Let π and π' be two stuttering R-equivalent paths where $R = \{(\sigma, \sigma') \mid \forall p \in AP(SC) : \sigma \models p \Leftrightarrow \sigma' \models p\}$. Then $\pi \models \varphi(SC) \Leftrightarrow \pi' \models \varphi(SC)$.*

The main theorem states that PLTL-X properties referring to states of a single sequential component are invariant under the semantics used.

Theorem 7 *Let SC be a statechart satisfying Constraints C1, C2, C3, C4, C5, and C6. Let SC' be a sequential component of SC, and let $\varphi(SC')$ be a PLTL-X property whose atomic propositions are in $\{in(s) | s \in \mathcal{S}_{SC'}\}$. Denote by $FP(SC)$ the fixpoint STS and by $SM(SC)$ the* Statemate *STS. Then $FP(SC) \models \varphi(SC') \Leftrightarrow SM(SC) \models \varphi(SC')$.*

Proof. $\Rightarrow$. Suppose $FP(SC) \models \varphi(SC')$. We show that for every Statemate run π, $\pi \models \varphi(SC')$. By Theorem 2(ii), for π a stuttering R-equivalent FP run π' exists. So $\pi' \models \varphi(SC')$. Then by Lemma 2, $\pi \models \varphi(SC')$.

$\Leftarrow$. Suppose $SM(SC) \models \varphi(SC')$. We show that for every FP run π, $\pi \models \varphi(SC')$. By Theorem 2(i), for π a stuttering R-equivalent Statemate run π' exists. Thus $\pi' \models \varphi(SC')$, and by Lemma 2, $\pi \models \varphi(SC')$. □

Theorem 8 *Let SC be a statechart satisfying Constraints C2, C3, C4, C5, C7, C8, C9, C10, C11, C12, C13, C14, C15, C16 and C17. Let SC' be a sequential component of SC, and let $\varphi(SC')$ be a PLTL-X property whose atomic propositions are in $\{in(s) | s \in \mathcal{S}_{SC'}\}$. Denote by $SM(SC)$ the* Statemate *STS and $UML(SC)$ the UML STS. Then $SM(SC) \models \varphi(SC') \Leftrightarrow UML(SC) \models \varphi(SC')$.*

Proof. $\Rightarrow$. Suppose $SM(SC) \models \varphi(SC')$. We show that for every UML run π, $\pi \models \varphi(SC')$. By Theorem 6, for π a stuttering $R_{SC'}$-equivalent seStatemate run π' exists. By definition, π' is also a Statemate run. So $\pi' \models \varphi(SC')$. Then by Lemma 2, $\pi \models \varphi(SC')$.

$\Leftarrow$. Suppose $UML(SC) \models \varphi(SC')$. We show that for every Statemate run π, $\pi \models \varphi(SC')$. By Theorem 4, for π a stuttering $R_{SC'}$-equivalent seStatemate run π' exists. By Theorem 5, for π' a stuttering $R_{SC'}$-equivalent UML run π'' exists. Since π'' is an UML run, $\pi'' \models \varphi(SC')$. Thus, by Lemma 2, $\pi' \models \varphi(SC')$, and again by Lemma 2, $\pi \models \varphi(SC')$. □

For a statechart SC, a PLTL-X formula $\varphi(SC)$ is separated if and only $\varphi(SC)$ is a boolean combination of s PLTL-X formulas, such that for each PLTL-X formula $\varphi(SC_i)$, where $i : 1..s$, SC_i is a sequential component of SC. A PLTL-X formula $\varphi(SC)$ that is logically equivalent to a separated formula is called separable [39]. The following corollary follows immediately from Theorems 7 and 8.

Corollary 3 *Let SC be a statechart satisfying Constraints C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C15, C16 and C17. Denote by $FP(SC)$ its fixpoint STS, by $SM(SC)$ its* Statemate *STS, and by $UML(SC)$ its UML STS. Let $\varphi(SC)$ be a PLTL-X property of SC. If $\varphi(SC)$ is separable, then*

$$FP(SC) \models \varphi(SC) \Leftrightarrow SM(SC) \models \varphi(SC), \text{ and}$$
$$SM(SC) \models \varphi(SC) \Leftrightarrow UML(SC) \models \varphi(SC).$$

The following corollary, which states that stuttering-closed properties are equivalent for seStatemate and UML STSes, follows immediately from Lemma 2 and Theorems 5 and 6.

Corollary 4 *Let SC be a statechart and let $\varphi(SC)$ be a PLTL-X property of SC. Denote by $seSM(SC)$ the single-event* Statemate *STS of SC and by $UML(SC)$ the UML STS of SC. Then*

$$seSM(SC) \models \varphi(SC) \Leftrightarrow UML(SC) \models \varphi(SC).$$

As indicated in Section 1 by means of a counterexample, Corollary 3 does not extend to branching-time logics like CTL. The counterexample in Figure 1 shows that there is no stuttering bisimulation relating stable valuations under the (se)Statemate and UML semantics, since under the UML semantics a stable valuation may have events in the queue, in which case the events in the queue are processed before the events that occur next, whereas the (se)Statemate semantics does not use a queue. But for linear, stuttering-closed properties, we only need to construct for each run under the seStatemate semantics a stuttering-equivalent run under the UML semantics, and vice versa (cf. Theorems 5 and 6).

To construct the runs, we use mappings between seStatemateand UML valuations. These mappings are not bijective, which we illustrate next using the counterexample (see Figure 1). An unstable seStatemate valuation in which

C is the initial configuration and external event f occurs, maps according to Theorem 5 to an unstable UML valuation in which the queue only contains f. But according to Theorem 6, an unstable UML valuation in which C is the initial configuration and f occurs but g is the external event in the queue to be processed next, maps to an unstable seSTATEMATE valuation in which g occurs. If in a subsequent UML valuation f is to be processed next, this valuation maps to a seSTATEMATE valuation in which f occurs. Since properties do not refer to events, the mapping from UML to seSTATEMATE valuations is correct. As explained in the introduction, properties cannot refer to events, since these are treated differently by the three semantics (cf. Table 1).

6 Advanced constructs

In this section, we look at some advanced constructs and discuss how the theorems of Section 5 can be extended to deal with them. We only consider the fixpoint-STATEMATE and the STATEMATE-seSTATEMATE cases. For the seSTATEMATE-UML case, adding new constructs like the in predicate does not cause differences in behaviour, since the theorems in Section 5.4 show that under both semantics exactly the same steps are taken and thus the same next configurations are reached.

6.1 The in*-predicate*

A transition t can test the current configuration in its guard condition by using the predicate $\mathsf{in}(x)$, where $x \in \mathcal{S}$ is a state.

Using the in predicate can lead to differences in behaviour between both fixpoint and STATEMATE semantics, and STATEMATE and seSTATEMATE. For example, suppose in the initial configuration of Figure 33 events e and f occur. Under the fixpoint semantics, the next configuration will be {s2,s3,s6} because the system is not in s6 when testing the guard condition of s3→s4. Under the STATEMATE semantics, however, configuration {s2,s4,s6} is reached, since s6 is entered before i is processed. Finally, under the seSTATEMATE semantics, if e occurs before f, configuration {s2,s3,s5} is reached, while if f occurs before e, configuration {s2,s3,s7} is reached. All these configurations are different from each other.

Such differences can be ruled out by restricting usage of the in predicate to external transitions only. That would rule out the statechart in Figure 33, since an internal transition uses the in-predicate. But even with this restriction, we still have that the order of processing events in seSTATEMATE influences the

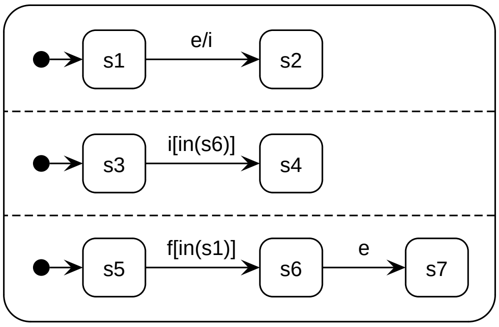

Fig. 33. Statechart with in predicate

outcome of the test. For example, if s3 → s4 is removed from Figure 33, the resulting statechart exhibits only the same behaviour under the STATEMATE and seSTATEMATE semantics if f is processed before e, since otherwise s1 has been left while f is processed, and s5 → s6 is disabled. Thus, to simulate the STATEMATE semantics with seSTATEMATE, events have to be processed in a certain order. This order should not conflict with the order defined by the *prec* relation (see Section 4.2).

6.2 History

Both STATEMATE and UML have the constructs of history and deep history connector. These constructs replace the concept of default state. If an OR state *o* with a history connector is entered, the child state of *o* that was active last is entered again. If *o* is entered for the first time, the default state of *o* is entered. With a deep history connector, all descendants of *o* that were active last are entered again upon entry of *o*. An elaborate introduction to history and deep history connectors can be found in the original statechart paper by Harel [16].

Theorems 1 and 3 can be easily extended to deal with history and deep history connectors without any additional constraints, since transitions in sequential components are taken in the same order (Theorems 2, 5, and 6). Thus, OR states are left and entered by the same transitions in the different semantics.

STATEMATE also has a clear history action, which can be used to erase the history or deep history of an OR state. UML does not appear to have this action. Usage of this action can lead to differences in behaviour, since transitions with the clear history action are not necessarily taken in exactly the same order under the different semantics.

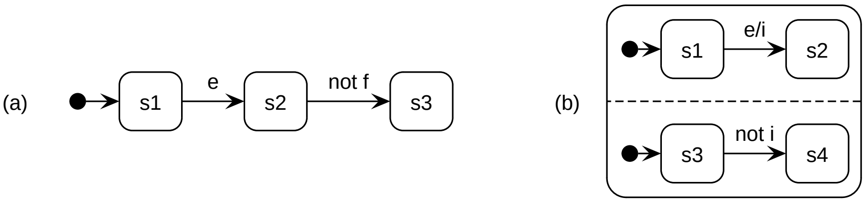

Fig. 34. Statecharts with negated event triggers

6.3 Compound and negated events

As explained in Section 1, both the fixpoint and STATEMATE semantics allow statecharts that use compound and negated event triggers, whereas the UML semantics only allows statecharts having single event triggers. A compound event is a conjunction of literals, where each literal is either an event or a negated event. A negated event tests the absence of an event, and therefore resembles more a guard condition than an actual trigger event. We show that using compound and negated events can lead to differences in behaviour under the fixpoint and STATEMATE semantics, and define an additional constraint that is needed to rule out such differences.

The first difference is due to negated event triggers. Figure 34(a) shows a statechart with a negated external trigger event. If external event e occurs in the initial configuration, then under the STATEMATE semantics stable configuration {s3} is reached, whereas under the fixpoint semantics stable configuration {s2} is reached, since an additional external event, different from f, needs to occur under the fixpoint semantics to enable transition s2 → s3. This issue resembles the difference in the enabling of a relevant completion transition under the fixpoint and STATEMATE semantics, which has been excluded by constraint C1. Also a statechart with a negated internal trigger event can behave differently under both semantics, as illustrated by Figure 34(b). If event e occurs in the initial configuration, then under the fixpoint semantics configuration {s2,s3} is reached, since the generation of i disables s3 → s4. However, under the STATEMATE semantics {s2,s4} is reached, since s3 → s4 is taken in the same step as s1 → s2.

A similar difference in behaviour is due to compound events only referencing negated events; Figure 35 gives an example. By similar reason as for Figure 34, it can be shown that if e occurs in the initial configuration, for (a), under the fixpoint semantics stable configuration {s2} is reached, whereas under the STATEMATE semantics stable configuration {s3} is reached, while for (b), under the fixpoint semantics stable configuration {s2,s3} is reached, whereas under the STATEMATE semantics {s2,s4} is reached.

Another difference in behaviour occurs if a compound event references (negated) internal events, since the two semantics sense internal events differently (cf.

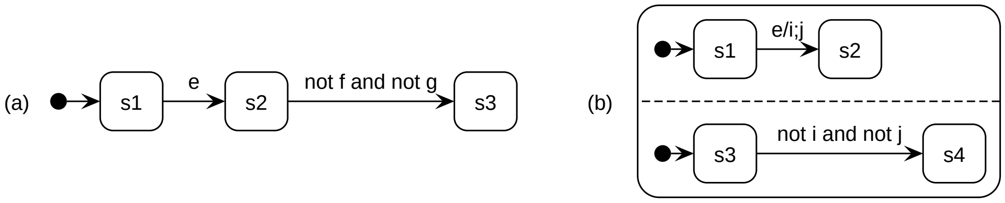

Fig. 35. Statecharts with compound events only referencing negated events

Table 8
Different stable configurations reached if e and f occur in the initial configurations of the statecharts in Figure 36

	(a)	(b)	(c)	(d)
fixpoint semantics	$\{s2, s4\}$	$\{s2, s3\}$	$\{s2, s4, s6, s8\}$	$\{s2, s4, s6, s7\}$
STATEMATE semantics	$\{s2, s3\}$	$\{s2, s4\}$	$\{s2, s4, s6, s7\}$	$\{s2, s4, s6, s8\}$

Table 1). Figure 36 shows several example statecharts to illustrate this; the reached stable configurations are listed in Table 8. Figure 36(a) and (b) show statecharts with compound events that reference both an internal and an external event, and a negated internal and external event, respectively. If external events e and f occur in the initial configuration, then for (a), under the fixpoint semantics s4 is entered, since f and internal event i are sensed in the same step, whereas under the STATEMATE semantics the system stays in s3, since i is sensed in a later step than f. Whereas for (b), under the fixpoint semantics the system stays in s3 and under the STATEMATE semantics s4 is entered. Figure 36(c) and (d) show statecharts with compound events that reference multiple internal events and internal and negated internal events, respectively. If external events e and f occur in the initial configuration, then for (c), under the fixpoint semantics s8 is entered, since internal events j and k are sensed in the same step, whereas under the STATEMATE semantics the system stays in s7, since k is sensed in a later step than j. While for (d), under the fixpoint semantics the system stays in s7, whereas under the STATEMATE semantics s8 is entered.

These differences in behaviour can be ruled out by requiring that:

- each negated event is part of a compound event (cf. Figure 34),
- each compound event does not reference any internal and negated internal events (cf. Figure 36), and
- each compound event references at least one non-negated (positive) event (cf. Figure 35), which is external (cf. Figure 36).

In other words, an internal event can only be used as a single event trigger, negation is only allowed for external events, and a negated external event must be part of a compound event referencing at least one non-negated external event. Using this additional constraint, Theorem 1 and 2 can be extended for

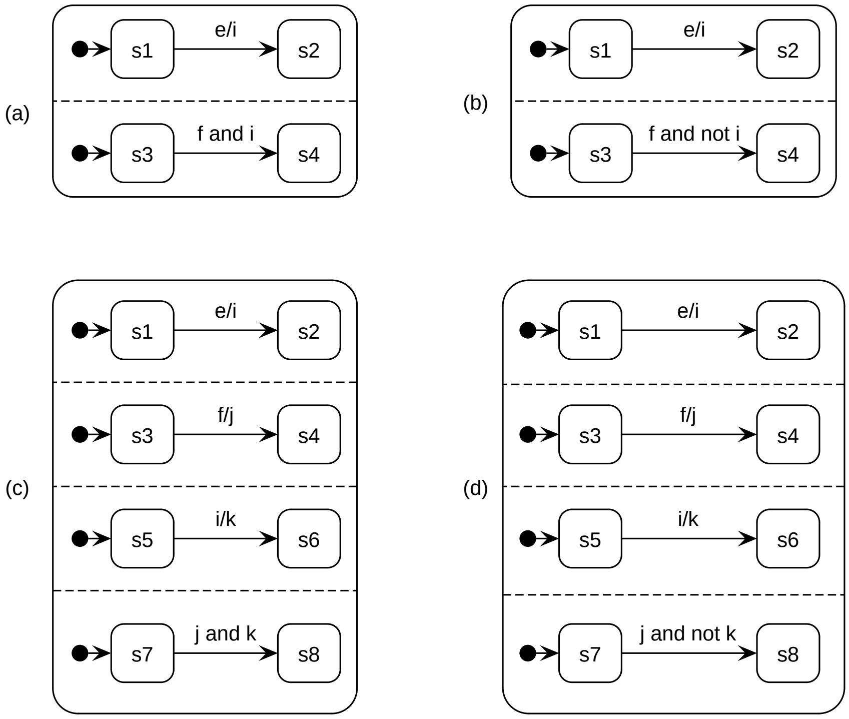

Fig. 36. Statecharts with compound events that reference (negated) internal events

negated and compound events.

According to Harel and Naamad [21], for the STATEMATE code generator it was decided, based on user experience, that the default mode only allows compound events that reference at least one non-negated event. Thus, the constraint is partly enforced in some tools in practice.

6.4 Data

Statecharts can contain local variables, which are updated in actions and tested in guard conditions and actions. Different transitions might access the same local variable v, either for testing or for updating. If these transitions are consistent, this can easily lead to race conditions: depending upon the particular order in which the transitions are taken, variable v can get assigned a different value, or a test of v might yield different results [21]. A simple example is shown in Figure 37. Assuming the initial value of x is zero, the transition s1→s2 can only be taken if the other transition has already been taken, making the guard $[x = 2]$ true.

To avoid race conditions, it suffices to require that if two transitions t_1 and t_2 access the same variable and either t_1 or t_2 updates the variable, then either (i) t_1 and t_2 are inconsistent, or (ii) t_1 indirectly triggers t_2, so $t_1 \gg^+ t_2$. Such

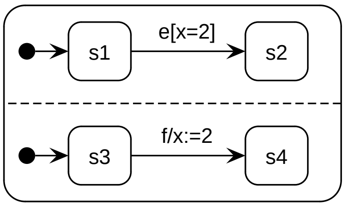

Fig. 37. Statechart having a race condition

a constraint would rule out Figure 37.

7 Conclusion

This paper makes several contributions. First, we have presented the three mainstream statechart semantics in a coherent framework. This shows the similarities and subtle differences among the semantics. For example, all semantics use the concept of a stable valuation, but define stability in slightly different ways. Consequently, the behaviour of the three semantics is quite different. The statecharts presented to motivate the constraints give concrete examples of these differences. However, since we studied statecharts that are meaningful under all three semantics, we did not consider constructs like negated and compound events, and synchronous calls.

Second, we have defined several constraints that highlight the differences between the different semantics. The constraints can act as sanity checks for statechart designs in general. If a constraint is violated, this may indicate that the statechart is ambiguous, in the sense that different semantics may attach completely different behaviours to it. We evaluated the constraints on some real-world example statechart designs taken from the literature. A few constraints are violated by the examples, but the examples indeed exhibit different behaviour for the three semantics. Thus, the constraints do not seem overly restrictive. However, this also suggests that in practice, a statechart design is likely to exhibit different behaviour for different semantics, which hampers a meaningful exchange of statechart designs among both designers and tools. A further evaluation of the constraints on a large set of industrial statechart designs is needed to reach a final conclusion regarding the exchangeability of statechart designs in practice.

Third, we have formally related the three semantics and shown which properties are preserved. We are unaware of other approaches in which these three completely different statechart semantics are formally compared. We have shown that the fixpoint, STATEMATE, and UML semantics resemble each other only in a weak sense, since properties must be separable. However, seSTATEMATE and UML are much more similar. In particular, we have shown that the main difference between the STATEMATE and the UML semantics is not that STATEMATE uses the perfect synchrony or ‘zero time’ assumption, whereas UML does not [18,17]. Instead, the main difference is that STATEMATE allows events to be processed in parallel, whereas UML only supports single-event processing. The main problem of simulating parallel-event processing with single-event processing is that in single-event processing events can have extra effects, i.e. trigger extra transitions, that are not present with parallel-event processing. Most constraints relating STATEMATE to its single-event variant dealt with this problem.

There are several directions for further work. First, the constraints can be implemented in a tool to support the checking of statechart designs. The constraints can also be useful to define design rules for statecharts, which are to a large extent still lacking in the literature. Moreover, the analysis can serve as starting point to implement a meaningful exchange of statechart designs among different commercial tools. Next, the analysis can be extended to deal with other statechart variants as well, for example StateFlow [36]. Also, the discussion in Section 6 can be elaborated in a formal setting for the omitted statechart constructs listed in Table 2. Another challenging extension would be to study under what conditions a set of asynchronously communicating UML statecharts exhibits similar behaviour as a single STATEMATE statechart.

Acknowledgements

I would like to thank the referees for their constructive comments.

Glossary

C	a configuration, subset of $\mathcal{S}$
ε	the empty queue
$\mathcal{E}$	the set of events of a statechart
$\mathcal{E}^{ext}$	the set of external events of a statechart
$\mathcal{E}^{int}$	the set of internal events of a statechart
I	a set of input events, subset of $\mathcal{E}$
π	a run of an STS
q	a queue of input events
$\mathcal{S}$	the set of states of a statechart
σ	a valuation
$\sigma \rightarrow \sigma'$	STS transition from σ to σ'
$\sigma \twoheadrightarrow \sigma'$	sequence of STS transitions between stable valuations σ and σ'
$\Sigma(V)$	set of valuations on set of variables V
St	a step, subset of $\mathcal{T}$
$\mathcal{T}$	the set of transitions of a statechart
T	a set of statechart transitions, subset of $\mathcal{T}$
$t \gg t'$	statechart transition t triggers t'
$t \prec t'$	statechart transition t has priority over t'
$x \perp y$	states x and y are orthogonal
$x \rightarrow y$	statechart transition with source set $\{x\}$ and target set $\{y\}$

References

[1] J. Aguado and M. Mendler. Constructive semantics for instantaneous reactions. In D.R. Ghica and G. McCusker, editors, *Proc. Games for Logic and Programming Languages (GALOP 2005)*, pages 16–31, 2005.

[2] 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, Lecture Notes in Computer Science 863, pages 128–148. Springer, 1994.

[3] M. von der Beeck. A structured operational semantics for UML-statecharts. *Software and System Modeling*, 1(2):130–141, 2002.

[4] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[5] M.C. Browne, E.M. Clarke, and O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59(1-2):115–131, 1988.

[6] W. Chan, R. Anderson, P. Beame, S. Burns, F. Modugno, D. Notkin, and J. Reese. Model checking large software specifications. *IEEE Transactions on Software Engineering*, 24(7):498–520, 1998.

[7] M.L. Crane and J. Dingel. UML vs. classical vs. Rhapsody statecharts: Not all models are created equal. In L.C. Briand and C. Williams, editors, *Proc. 8th International Conference on Model Driven Engineering Languages and Systems Conference (MoDELS 2005)*, Lecture Notes in Computer Science 3713, pages 97–112. Springer, 2005.

[8] W. Damm, B. Josko, H. Hungar, and A. Pnueli. A compositional real-time semantics of STATEMATE designs. In W.-P. de Roever, H. Langmaack, and A. Pnueli, editors, *Proc. Compositionality: The Significant Difference (COMPOS '97)*, Lecture Notes in Computer Science 1536, pages 186–238. Springer, 1998.

[9] W. Damm, B. Josko, A. Pnueli, and A. Votintseva. A discrete-time UML semantics for concurrency and communication in safety-critical applications. *Science of Computer Programming*, 55(1-3):81–115, 2005.

[10] B.P. Douglass. *Real Time UML: Advances in the UML for Real-Time Systems.* Addison-Wesley Professional, 3rd edition, 2004.

[11] B.P. Douglass, D. Harel, and M.B. Trakhtenbrot. Statecharts in use: Structured analysis and object-orientation. In G. Rozenberg and F.W. Vaandrager, editors, *Lectures on Embedded Systems*, Lecture Notes in Computer Science 1494, pages 368–394. Springer, 1998.

[12] E.A. Emerson, S. Jha, and D. Peled. Combining partial order and symmetry reduction. In E. Brinksma, editor, *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'97)*, Lecture Notes in Computer Science 1217, pages 19–34. Springer, 1997.

[13] R. Eshuis. Symbolic model checking of UML activity diagrams. *ACM Transactions on Software Engineering Methodology*, 15(1):1–38, 2006.

[14] R. Eshuis, D.N. Jansen, and R. Wieringa. Requirements-level semantics and model checking of object-oriented statecharts. *Requirements Engineering Journal*, 7(4):243–263, 2002.

[15] M. Fränzle, J. Niehaus, A. Metzner, and W. Damm. A semantics for distributed execution of Statemate. *Formal Aspects of Computing*, 15(4):390–405, 2003.

[16] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[17] D. Harel. Statecharts in the making: a personal account. In *HOPL III: Proc. of the 3rd ACM SIGPLAN conference on History of programming languages*, pages 5–1 – 5–43. ACM Press, 2007.

[18] D. Harel and E. Gery. Executable object modeling with statecharts. *IEEE Computer*, 30(7):31–42, 1997.

[19] D. Harel and H. Kugler. The Rhapsody semantics of statecharts (or, on the executable core of the UML) - preliminary version. In H. Ehrig, W. Damm, J. Desel, M. Große-Rhode, W. Reif, E. Schnieder, and E. Westkämper, editors, *Integration of Software Specification Techniques for Applications in Engineering*, Lecture Notes in Computer Science 3147, pages 325–354. Springer, 2004.

[20] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M.B. Trakhtenbrot. Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, 1990.

[21] D. Harel and A. Naamad. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[22] D. Harel and A. Pnueli. On the development of reactive systems. In K.R. Apt, editor, *Logics and Models of Concurrent Systems*, volume 13 of *NATO/ASI*, pages 447–498. Springer, 1985.

[23] D. Harel, A. Pnueli, J. P. Schmidt, and S. Sherman. On the formal semantics of statecharts. In *Proceedings of the Second IEEE Symposium on Logic in Computation*, pages 54–64. IEEE, 1987.

[24] D. Harel and M. Politi. *Modeling Reactive Systems with Statecharts: the STATEMATE approach.* McGraw-Hill, 1998.

[25] C. Huizing and W. P. de Roever. Introduction to design choices in the semantics of Statecharts. *Information Processing Letters*, 37(4):205–213, 1991.

[26] C. Huizing and R. Gerth. Semantics of reactive systems in abstract time. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proc. REX Workshop (Real-Time: Theory in Practice)*, Lecture Notes in Computer Science 600, pages 291–314. Springer, 1992.

[27] L. Lamport. What good is temporal logic? In R.E.A. Mason, editor, *Proc. of the IFIP Congress on Information Processing*, pages 657–667. North-Holland, 1983.

[28] D. Latella, I. Majzik, and M. Massink. Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model-checker. *Formal Aspects of Computing*, 11(6):637–664, 1999.

[29] N.G. Leveson, M.P.E. Heimdahl, and J.D. Reese. Designing specification languages for process control systems: Lessons learned and steps to the future. In O. Nierstrasz and M. Lemoine, editors, *Proc. ESEC/FSE '99*, Lecture Notes in Computer Science 1687, pages 127–145. Springer, 1999. Also ACM SIGSOFT *Software Engineering Notes*, volume 24, number 6.

[30] N.L. Leveson, M.P.E. Heimdahl, H. Hildreth, and J.D. Reese. Requirements specification for process-control systems. *IEEE Transactions on Software Engineering*, 20(9):684–707, 1994.

[31] F. Levi. A compositional μ-calculus proof system for statecharts processes. *Theoretical Computer Science*, 216(1-2):271–310, 1999.

[32] G. Lüttgen and M. Mendler. The intuitionism behind Statecharts steps. *ACM Transactions on Computational Logic*, 3(1):1–41, 2002.

[33] G. Lüttgen and M. Mendler. Towards a model-theory for Esterel. In F. Maraninchi, A. Girault, and E. Rutten, editors, *Proc. Int. Workshop on Synchronous Languages, Applications, and Programming (SLAP 2002)*, volume 65(5) of *Electronic Notes in Theoretical Computer Science*, pages 95–109, 2002.

[34] A. Maggiolo-Schettini, A. Peron, and S. Tini. A comparison of statecharts step semantics. *Theoretical Computer Science*, 290(1):465–498, 2003.

[35] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer, 1992.

[36] The Mathworks. Stateflow and Stateflow coder users guide, 2005. Available at `http://www.mathworks.com`.

[37] E. Mikk, Y. Lakhnech, and M. Siegel. Hierarchical automata as model for statecharts. In R.K. Shyamasundar and K. Ueda, editors, *Proc. Third Asian Computing Science Conference (ASIAN '97)*, Lecture Notes in Computer Science 1345, pages 181–196. Springer, 1997.

[38] D. Peled. All from one, one from all: on model checking using representatives. In *Proc. International Conference on Computer Aided Verification (CAV'93)*, Lexture Notes in Computer Science 697, pages 409–423. Springer, 1993.

[39] D. Peled. On projective and separable properties. *Theoretical Computer Science*, 186(1-2):135–156, 1997.

[40] A. Pnueli and M. Shalev. What is in a step: On the semantics of statecharts. In T. Ito and A.R. Meyer, editors, *Theoretical Aspects of Computer Software*, Lecture Notes in Computer Science 526, pages 244–265. Springer, 1991.

[41] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modelling and Design.* Prentice-Hall, 1991.

[42] B. Selic, G. Gullekson, and P. Ward. *Real-Time Object Oriented Modeling.* John Wiley & Sons, 1994.

[43] UML Revision Taskforce. *UML 2.0 Superstructure Specification.* Object Management Group, 2003. OMG Document Number ptc/03-07-06. Available at `http://www.uml.org`.

[44] R.J. Wieringa. *Design Methods for Reactive Systems: Yourdon, Statemate and the UML.* Morgan Kaufmann, 2003.