A Comparison of Statecharts Variants

Michael von der Beeck

Lehrstuhl für Informatik III, Aachen University of Technology
Ahornstr. 55, D-52074 Aachen, Germany
e-mail: beeck@i3.informatik.rwth-aachen.de

Abstract. The Statecharts formalism supports the development of intuitive graphical specifications for reactive systems. Nevertheless, some serious problems became apparent in the original Statecharts formalism so that many different Statecharts variants were proposed to overcome them. These problems are thoroughly described and approaches for solving them are evaluated. Furthermore, a set of distinctive features is elaborated which is used for a detailed comparison of the Statecharts variants. Finally, the feature set is used to characterize a new hypothetical Statecharts variant.

1 Introduction

The Statecharts formalism [8] has been developed in order to simplify the specification of reactive systems [9]. It is a state-based graphical formalism which can be considered as an enhancement of finite automata and their representation as state transition diagrams. In [10] Statechart's characteristic features are summarized by the following equation:

statecharts = *state-diagrams* + *depth* + *orthogonality* + *broadcast-communication*

The Statecharts formalism has become quite successful due to its hierarchical representation of state information and its expressive power in modelling concurrency by simultaneously active states and by parallel transition execution. Nevertheless, some problems concerning syntax and semantics diminish its usage. Therefore many Statecharts variants were developed in order to solve these problems.

Firstly, this paper describes the problems of the Statecharts formalism. Then a comprehensive set of distinctive features of Statecharts variants is elaborated. As its main contribution the paper provides a detailed comparison of 20 Statecharts variants.[1] Finally, an evaluation of the distinctive features is performed resulting in a characterization of a new Statecharts variant.

1. More precisely, two of these variants (Argos [20] and RSML [17]) are not called Statecharts. But in the following they are subsumed in the set of Statecharts variants due to their similarity to the Statecharts formalism.

2 The Statecharts Formalism

2.1 Introduction to Statecharts

The Statecharts formalism essentially consists of states and transitions like a finite automaton. In order to model depth complex states (AND- and OR-states) are used which contain substates and internal transitions. If the system specified by a Statechart resides in an OR-state, then it also resides in exactly one of its direct substates. Staying in an AND-state implies staying in all of its direct substates. If a complex state is left, each substate is also left. AND-states are provided in order to model concurrency: Their substates may contain transitions which may be executed simultaneously. These transitions can communicate by internal events which are broadcast all over the Statechart. Events can trigger transitions which can generate internal events. The transition labels determine triggering and event generation. Therefore, they consist of a trigger part and an action part. The trigger part at least provides a trigger expression which determines when the transition is triggered. The action part specifies which events are generated if the transition is executed. This mechanism offers the possibility that one event can cause a chain reaction of transition executions.

2.2 Problems

Shortly after the publication of Harel's paper [8] introducing the Statecharts formalism extensive research began in order to improve its syntax and semantics. On the one hand this work was motivated by Statechart's convincing concept in specifying hierarchical information and concurrency in a clear and intuitive way. On the other hand Statechart's incomplete and imprecise semantics as described in [8], its rich syntax and the synchronous approach led to several problems.

In the following we describe these problems and evaluate approaches intended to solve them. (Some problems are heavily intertwined so that they are treated in a common item.)

Problem list:

(1) Perfect Synchrony Hypothesis

(2) Self-Triggering, Causality

(3) Negated Trigger Event

(4) Effect of a Transition Execution is Contradictory to its Cause

(5) Inter-Level Transition

(6) State Reference

(7) Compositional Semantics, Self-Termination

(8) Operational Versus Denotational Semantics

(9) Instantaneous State

(10) Durability of Events

(11) Parallel Execution of Transitions

(12) Transition Refinement

(13) Multiple Entered or Exited Instantaneous State

(14) Infinite Sequence of Transition Executions at an Instant of Time

(15) Determinism

(16) Priorities for Transition Execution

(17) Preemptive Versus Non-Preemptive Interrupt

(18) Distinguishing Internal from External Events

(19) Time Specification, Timeout Event, Timed Transition

In the following these problems are dealt with in detail:

(1) Perfect Synchrony Hypothesis

The term perfect synchrony hypothesis was defined by the developers of Esterel [4]. It requires that a system will immediately react on external events i.e. input and corresponding output occur at the same time. According to [8] the hypothesis also applies to the Statecharts formalism. The question arises whether the hypothesis is appropriate, because in reality no reaction occurs simultaneously with the input that has caused it. But this hypothesis is justified, because it only represents an abstraction from the following: In reality a reaction will not take place instantaneously, but in time, i.e. before the next input occurs. In fact, many real-time systems exist which provide response times being much shorter than the time interval between the occurrences of two successive input events.

In [2] it is stated that a system to which the synchrony hypothesis applies can be decomposed in concurrent subsystems without any changes to its observable behaviour. Furthermore, it is shown that even asynchronous systems can be validated by synchronous models.[2]

(2) Self-Triggering, Causality

The term self-triggering describes the fact that a transition is executed without being caused by an external event. Fig. 1 provides an example. There a transition t_1 exists whose trigger expression requires an occurrence of the event a, but this event is also generated if the transition t_1 is executed, because t_1 generates the event b which triggers transition t_2 that generates the event a.

The synchrony hypothesis demands that input and output take place at the same instant of time. Therefore one can determine that a transition does not need an external event in order to be executed. This transition is triggered directly or indirectly by its own generated event.

2. Nevertheless, even the developers of Esterel introduced a paradigm called Communicating Reactive Processes (CRP) [5] which unifies a synchronous (Esterel) and an asynchronous (CSP) [11] programming language for the implementation of distributed systems.

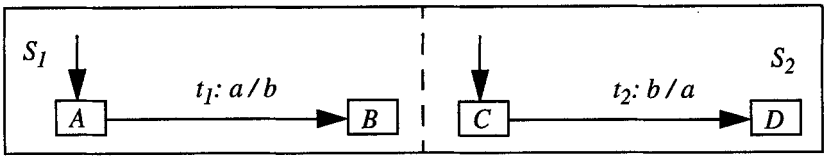

Fig. 1. A possible self-triggering Statechart

According to [14] a Statecharts semantics respects causality (is causal) if an external event has occurred for each executed transition so that the event directly or indirectly caused the transition's execution. This means that each execution of a transition t can be traced back to the occurrence of an external event which initiated a sequence of transition executions each one causing the next one, so that at last the execution of t is caused. Furthermore, cycles must not occur in this sequence of transitions. Therefore a semantics which allows the existence of self-triggering does not respect causality.

For example the "D-type" semantics of [14] is not causal, because it considers all events generated during a complete macro step as already existing at the very beginning of this macro step.

Conclusion: Causality represents a very fundamental property. If it does not exist, an intuitive Statecharts semantics is not achievable.

(3) Negated Trigger Event

The example of Fig. 2 reveals why it is important to be able to model non-occurrence of events.

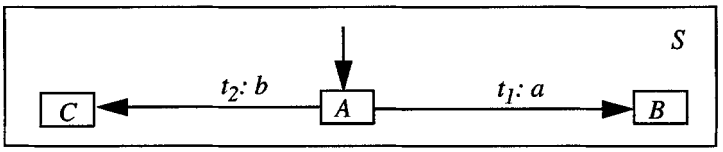

Fig. 2. A non-deterministic Statechart

Non-determinism exists if state A is the current state and both events a and b occur simultaneously. To achieve determinism the negated event $\neg b$ can be used in order to specify that transition t_1 shall only be executed if event a, but not event b has occurred. The corresponding transition label of t_1 is shown in Fig. 3. The transition label of t_2 remains unchanged. Therefore t_2 will be executed if event b occurs - solely or simultaneously with event a.

Nevertheless, in general the possibility to use negated events is insufficient to avoid non-determinism. (See items (15) and (16).)

Furthermore, the usage of negated trigger events imposes a problem which is treated in item (4).

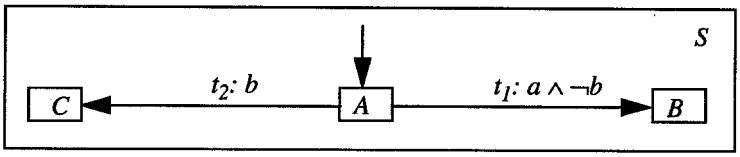

Fig. 3. A Statechart with negated trigger event to avoid non-determinism

(4) Effect of a Transition Execution is Contradictory to its Cause

This problem describes the fact that an internal event, i.e. an event generated by transition execution, is contradictory to the trigger expression of the transition whose execution - directly or indirectly - caused the generation of the event. This can be considered as an inconsistency between the trigger and the action part of the corresponding transition labels. More concretely, the problem states that the action part of a transition contains an event which occurs in negated form in the trigger expression of this or another transition whose execution causes the execution of the first one. A simple example is provided by Fig. 4.

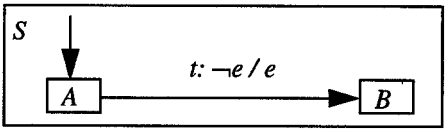

Fig. 4. Possible inconsistency between trigger and action part of transition *t*

If transition *t* is executed the event *e* will be generated according to the action part of *t*. But this event occurrence is contradictory to the trigger expression ¬*e* of *t*.

In order to decide whether this kind of inconsistency shall be permitted, a thorough consideration of the semantics of negated events occurring in the trigger part of a transition is required. Two appropriate possibilities exist:

- The effect of negation only depends on the past:

  In this case a transition containing a negated event in its trigger part is executed in a macro step if no transition executed in previously performed micro steps of this macro step has generated this event. This means that the execution does not depend on the fact, whether the event is generated in the same micro step or in one of the following micro steps of the same macro step. This condition is called *local consistency* [23].

  Therefore local consistency permits that transition *t* of Fig. 4 is executed.

- The effect of negation depends on the past and on the future:

  In this case the negation's effect does not only depend on the fact that the corresponding event is not generated in previous micro steps, but also that it is not generated in the same micro step or in subsequent micro steps of the current macro step.

More concretely: A transition containing a negated event in its trigger part may be executed in a macro step if no transition of any micro step of this macro step generates this event.

This condition is called *global consistency* [23].

Therefore global consistency prohibits that transition *t* of Fig. 4 is executed.

A semantics only requiring local consistency depends more heavily on the order of micro steps in a macro step than a semantics requiring global consistency. This is revealed by the following example which consists of two potential sequences of micro steps:

no.	sequence of micro steps	local consistency	global consistency
1	... *b/a*, ... ¬*a/*..., ...	-	-
2	... ¬*a/*..., *b/a*, ...	+	-

According to local consistency only sequence 2, but not sequence 1, represents a possible sequence of micro steps. In the case of global consistency it is of no concern whether the micro step with transition label *b/a* precedes or follows the micro step with transition label ¬*a/*... , since both sequences are prohibited.

A semantics M_{LC} only requiring local consistency is more intuitive than a semantics M_{GC} requiring global consistency, because the former one distinguishes a cause clearly from its effect - in contrast to the latter one. In M_{LC} the sequence of transition executions defines a (partial) order between the transitions which determines pairs (t_1, t_2) of transitions, so that t_1 causes the execution of t_2, though the complete sequence is executed in zero time. But since M_{LC} depends more on the order of micro steps in a macro step than M_{GC} (cf. last example), it is more difficult to define.

Conclusion: The clear distinction of cause and effect is more important than a semantics which is easy to define. Furthermore, global consistency contradicts to the motivation for demanding causality (item(2)). Therefore the property of global consistency should not be fulfilled.

(5) Inter-Level Transition

An inter-level transition crosses the borderline of one or more states, i.e. this transition connects states that are no direct substates of the lowest common parent state. A problem caused by inter-level transitions is mentioned in item (7).

(6) State Reference

Some Statecharts variants provide the possibility that the execution of a transition can be made dependent on the fact whether a certain state of a parallel component is active. This dependency can be modelled by the usage of special trigger conditions like *in(state)* in [8]. They are called state references. A problem induced by the usage of state references is described in item (7).

(7) Compositional Semantics, Self-Termination

The first formal Statecharts semantics [10] overcoming the informal semantics description from [8] was criticised, because it is not compositional.

A semantics of a language L is compositional if the semantics of a compound component of L is only defined by the semantics of its subcomponents, i.e. that no access to the internal syntactical structure of subcomponents is allowed.

A a benefit, a compositional semantics allows more efficient verification, since it provides the possibility to deduce properties of a complex component from already verified properties of its subcomponents without looking at their internal structure.

However, the property of compositionality should not be achieved by a semantics which exports all internal information of a component at its interface, because this semantics would only represent a coding of the syntax. This means that the semantics should be sufficiently abstract[3] in the sense that it does not distinguish too many syntactic components. If this condition is fulfilled at its best, the semantics is called fully abstract.

According to [6] and [19] inter-level transitions (cf. item (5)), state references (cf. item (6)) and the history-mechanism (cf. subchapter 3.1) impede the definition of a compositional Statecharts semantics.[4]

In some Statecharts variants the development of a compositional semantics was simplified by the prohibition of these syntactical components. Instead of the usage of inter-level transitions the concept of self-termination [20] is used. This means that two or more transitions on different hierarchical levels are executed simultaneously, so that a low-level transition triggers a high-level one. This means that the target state of the low-level transition will be left due to the execution of the high-level transition. One example of a Statechart with self-termination is given by Fig. 6, whereas Fig. 5 presents an equivalent Statechart using an inter-level transition (t).

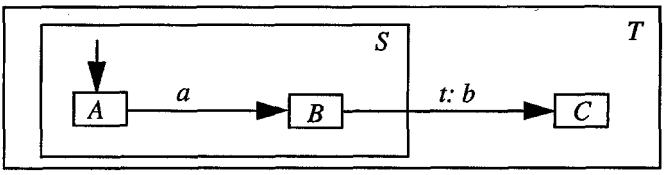

Fig. 5. Usage of an inter-level transition (t)

3. The most apparent mapping from syntax to semantics resulting in a compositional semantics not fulfilling the condition of abstractness at all is given by the identity function ([25]).
4. To avoid a potential misunderstanding: the existence of inter-level transitions or state references does not imply a non-compositional semantics. There exist compositional Statecharts semantics with inter-level transitions (e.g. in [12], [13], [15] and [22]).

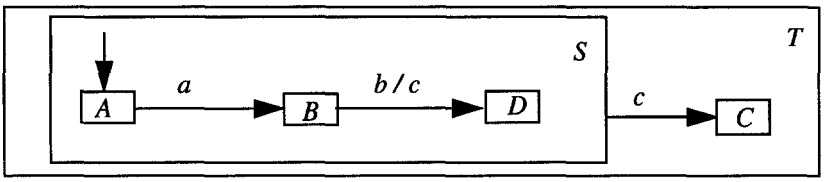

Fig. 6. Usage of self-termination instead of an inter-level transition

(8) Operational Versus Denotational Semantics

Often a denotational style [24] of semantics - implying compositionality - is used. In order to achieve compositionality, a denotational semantics assigns a meaning to each program phrase - not only to the complete program. The main aspect of denotational semantics is given by its purely mathematical basis: The behaviour of a program is not defined by determining its effect on a computer or an abstract computer model, as it is the case in operational semantics. Nevertheless, an operational semantics may be easier to understand than an equivalent denotational one, because the computational model can be more intuitive than the abstract mathematical components of a denotational semantics.

(9) Instantaneous State

An instantaneous state may be simultaneously entered and exited. The standard Statecharts paper [8] does not determine whether such states are allowed. In most subsequently published papers providing a formal Statecharts semantics (e.g. [10]) instantaneous states are prohibited. Fig. 7 provides an example. If transition t_1 is executed the internal event b is generated and state B will become active. In this case the question arises, whether transition t_2 will be executed at the same time. (Note that the trigger expression of t_2 is identical with the event being generated during the execution of transition t_1.)

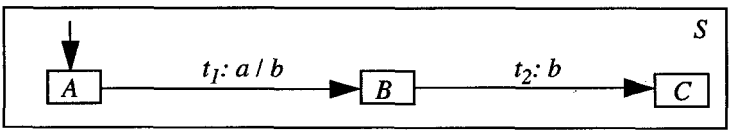

Fig. 7. A Statechart containing a (possibly) instantaneous state B

In the first formal semantics of Statecharts in [10] it was determined that no state may be entered and left during the same macro step, i.e. at the same instant of time. An essential implication of this is the restriction that at one instant of time several transitions may only be executed if they reside in parallel components. This implies in return the desirable property that only a finite number of transitions can be executed during one macro step, i.e. at one instant of time.

The example of Fig. 7 reveals that the existence of an instantaneous state (state B in this case) can lead to the fact that several substates of an OR-state are simultaneously active. This is contradictory to the fundamental requirement in

[8] that always at most one substate of an OR-state may be active. In [22] - describing a Statecharts variant providing instantaneous states - a less restrictive condition is presented: no pair of direct substates of an OR-state may collectively be active during a non-zero period of time.

If instantaneous states are explicitly provided a special case has to be considered separately. An example is given by Fig. 8.

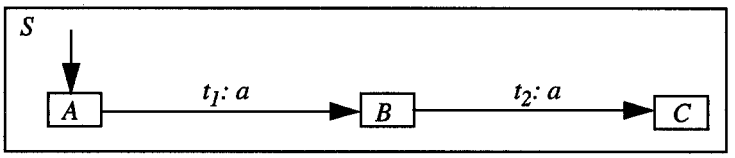

Fig. 8. How many occurrences of event *a* are necessary to reach state *C* from state *A*?

It has to be determined whether one or two occurrences of event *a* are necessary so that - starting in state *A* - both transitions t_1 and t_2 are executed. Of course two event occurrences are necessary if instantaneous states are forbidden. But both possibilities appear reasonable if instantaneous states are permitted:

- one occurrence of event *a* is sufficient

  In this case an event is not treated as an entity to be consumed after triggering. This possibility has the disadvantage that infinite sequences of transition executions are possible.

- two occurrences of event *a* are necessary

  In this case an event having triggered a transition is consumed, so that it cannot trigger another transition. This choice simplifies modelling of a counter which has to distinguish between different occurrences of the same event.

A third possibility combines both choices: For each event in the trigger expression it can be decided whether it will be consumed or not if it triggers the transition.[5] The events of the trigger expression have to be annotated accordingly.

The Statecharts formalism should be enhanced by this possibility, since it provides the most expressive solution.

(10) Durability of Events

In most Statecharts variants an event has an instantaneous occurrence. This means that an event occurs at one instant of time and exists only at this instant of time. This kind of event is often called *discrete event.*

5. A distinction between consumable and non-consumable events was for the first time proposed in [6] - not for modelling purposes, but in order to define a formal semantics for Statecharts.

In the case of discrete events the question how long they are available might appear strange, but the motivation of the question is as follows:

The statement that discrete events exist only for an instant of time is insufficient, if transitions are executed in zero time and if furthermore instantaneous states are allowed. (See item (9)). The first condition is fulfilled for all Statecharts variants and the second for some of them. Therefore it is possible that a sequence of transitions $t_1, t_2, ..., t_n$ $(n>1)$ with $t_i=(s_i, l_i, s_{i+1})$, $s_i \in$ *States*, $l_i \in$ *Labels*, $(1 \le i \le n)$ can be executed in zero time in an OR-state.

But according to item (9) it has to be determined whether one event can trigger more than one of these transitions. An example is provided by Fig. 8.

If the Statecharts semantics prohibits the execution of t_2 by the same occurrence of event a that has already triggered the transition t_1, then the availability of an event appears to be even more constrained than by the afore-mentioned statement saying that discrete events only exist for an instant of time.

In the case of Timed Statecharts [16] the discrete event approach is also taken. Furthermore, instantaneous states are allowed. Events persist as long as only untimed transitions are executed, i.e. as long as time does not progress. (See item (19).) Events are only consumed if a timed transition is taken.

The Statecharts variants of [22] do not use the discrete event approach. They use events which are allowed to have a duration.

(11) Parallel Execution of Transitions

As already described in item (9) the decision that a state must not be simultaneously entered and exited implies that only those transitions which reside in different parallel components of the Statechart may be simultaneously executed. Because of a second implication (finite number of transitions to be executed during one macro step) this decision is taken in most Statecharts variants.

(12) Transition Refinement

In item (11) an implication of the decision that a state must not be simultaneously entered and exited was mentioned: No sequence of transitions of an OR-state may be executed during one instant of time. This decision has the following disadvantage [22]: in the usual way of refining a state transition diagram or a finite automaton, a transition is replaced by a sequence of transitions. If this procedure is performed though the afore-mentioned decision was taken, the problem arises that the execution of a transition to be refined will occur in zero time whereas a sequence of transitions - composed of at least two transitions - will be executed in a non-zero time interval, because in each (non instantaneous) state time must progress, before it can be left.

This refinement problem does not exist if instantaneous states are allowed. In this case a transition can be replaced by a sequence of transitions connected by instantaneous states so that the complete sequence is executed in zero time.

(13) Multiple Entered or Exited Instantaneous State

A semantics allowing instantaneous states has the disadvantage that an infinite number of transition executions is possible at one instant of time. In order to avoid this possibility it can be defined that a state may be instantaneous, but that it must not be entered twice at the same time.

(14) Infinite Sequence of Transition Executions at an Instant of Time

In order to avoid the execution of an infinite number of transitions at one instant of time, two feasible possibilities exist: As already mentioned in item (9) instantaneous states can be prohibited. An alternative is that the less restrictive constraint can be applied requiring that a state must not be entered twice at the same time (as already mentioned in item (13)).

(15) Determinism

All Statecharts variants except Argos [20] allow the construction of non-deterministic Statecharts. Non-determinism arises if the trigger expressions of two transitions starting from a common state are simultaneously fulfilled. Furthermore, it is possible that one transition starts from a state S and a second from an ancestor state of S and both trigger expressions are again simultaneously fulfilled.

On the one hand a deterministic formalism like Argos[6] and Esterel [4] avoids inadvertent modelling of non-determinism. On the other hand non-determinism sometimes exists in the system to be modelled. The most intuitive and easiest way to specify this is to use a language providing non-determinism - e.g. Statecharts.

(16) Priorities for Transition Execution

In item (15) examples for the existence of non-determinism in Statecharts are mentioned. If priorities for the execution of transitions are introduced, some non-determinism will disappear. One example is presented in Fig. 9.

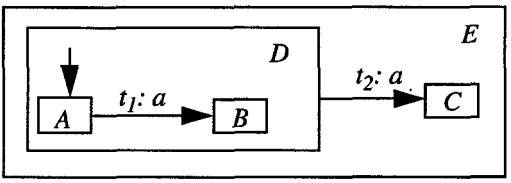

Fig. 9. A Statechart with possible non-determinism

According to the semantics of [10], it is not determined whether transition t_1 or transition t_2 has to be executed if the state A is active and the event a occurs.

6. In [21] the determinism of Argos is weakened as follows: It is stated that the semantics of Argos can deal with non-deterministic components, but that the Argos operators do not introduce non-determinism i.e. they preserve determinism.

But the semantics from [23] provides a priority concept requiring that transition t_2 has to be executed in this case, because the scope of t_2 is on a higher level than the scope of transition t_1. (The scope of a transition is the OR-state which is an ancestor state of both the source and target state of this transition and which is no ancestor of another OR-state with these properties, i.e. the scope is the lowest OR-state with these properties.)

Fig. 10 shows a Statechart which is non-deterministic even if the afore-mentioned priority concept is used, since t_1 and t_2 have the same scope E.

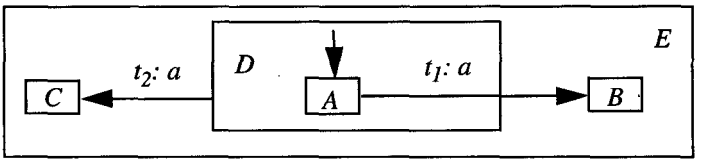

Fig. 10. A Statechart with possible non-determinism

But according to the semantics from [7], the Statechart of Fig. 10 is deterministic. This semantics also uses the state hierarchy in order to define priorities between transitions. The higher the source state of a transition, the higher the priority. In this semantics transition t_2 will be executed if state A is active and event a occurs.

Finally, Fig. 11 presents an example of a Statechart which is non-deterministic in all the afore-mentioned semantics.

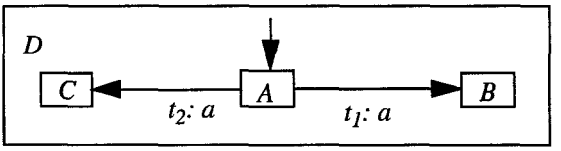

Fig. 11. A non-deterministic Statechart

(17) Preemptive Versus Non-Preemptive Interrupt[7]

The usage of OR- and AND-states leads to a reduction of transitions compared with conventional state transition diagrams. In the case of an OR-state s which contains the substates s_i ($1 \le i \le n$) this is achieved if a set t_1, ..., t_n of transitions with t_i starting from state s_i ($1 \le i \le n$) and ending in a common state s' is replaced by only one transition. This new transition starts from the OR-state s and ends in state s'. Such a transition starting from an OR-state represents an interrupt-mechanism: Independent from the substate being active, the OR-state (and the substate) are left if the transition is triggered. This kind of transition is very useful, but its behaviour has to be considered more precisely.

7. Instead of the term *interrupt* used here and in [14], the term *interruption* is used in [20] and the term *abortion* is used in [3].

There is the problem to determine the behaviour of a Statechart like that of Fig. 12 - providing the interrupt-mechanism by transition t_2 - if the state A is active and the events a and c occur simultaneously.

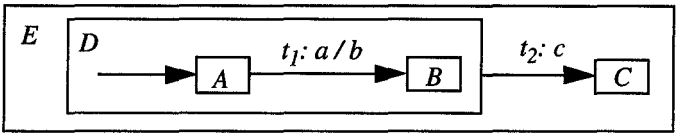

Fig. 12. A Statechart with preemptive or non-preemptive interrupt by transition t_2

In general, an interrupt might be preemptive or non-preemptive. These notions will be described using the Statechart of Fig. 12:

- The interrupt is called *preemptive* if the fact that transition t_2 is executed prevents the execution of t_1.
  This case can be further subdivided depending on the fact whether priorities for the execution of transitions were introduced as described in item (16):
  - t_2 will be executed exclusively if such a priority concept exists.
    Though the trigger expression of transition t_1 is also fulfilled, the source state D of t_2 is left without executing transition t_1, because t_1 resides on a lower level than t_2. The execution of t_1 is prevented by the execution of t_2. This implies that the action of transition t_1 - generation and broadcasting of the event b - does not occur.
  - One of the transitions t_1 and t_2 will be non-deterministically selected and executed if the priority concept does not exist.
- The interrupt is called *non-preemptive* if the execution of t_2 does not prevent the execution of t_1.
  This possibility also depends on the existence of a priority concept.
  - t_2 will be executed if the priority concept exists. (No statement about the execution of t_1 is implied in this case.)
  - t_1 and t_2 will be simultaneously executed if no priority concept exists. During execution of t_1 its action - generation and broadcasting of the event b - will also be completely performed, though the state change from the parent state D of t_1 to the source state C of t_2 occurs.

A simultaneous execution of several transitions does not imply that the state B will become active for a non-zero period of time, because both transitions t_1 and t_2 will be executed in zero time. Therefore time cannot progress until the state C has become active. In particular, the simultaneous execution of several transitions does not imply that two or more states being substates of a common OR-state are simultaneously active for a non-zero amount of time. (See also item (9).)

A summary of these possibilities w.r.t. Fig. 12 is provided by Table 1:

preemptive / non-preemptive	priority concept	transition execution
n	-	$t_1 \wedge t_2$
p	-	$t_1 \mid t_2$ [a]
n	+	t_2 [b]
p	+	$(\neg t_1) \wedge t_2$

Table 1: Interrupt types and their effects

a. The symbol "|"denotes exclusive-or.
b. No statement exists about the execution of transition t_1.

Conclusion: With respect to experiences known from Esterel applications [3], the Statecharts formalism should provide both interrupt types.

(18) Distinguishing Internal from External Events

During the execution of micro steps within a macro step the occurrence of external events is not sensed. Therefore, the synchrony hypothesis can be fulfilled which requires that a reaction on an input event will be performed before the next input event occurs.

On the contrary internal events, i.e. events generated by transition execution, can influence the actual macro step by triggering transitions which constitute micro steps of the actual macro step.

(19) Time Specification, Timeout Event, Timed Transition

Progress of time can only be modelled while staying in states, since the transitions of all Statecharts variants are executed in zero time.

Some variants use timeout events. A timeout event *tm(e, n)* is generated *n* time steps after the last occurrence of the event *e* and can be used for triggering transitions like conventional events.

The most expressive possibility for explicitly modelling time is provided by those Statecharts variants offering timed transitions. Two ways exist how timed transitions can be triggered:

- The source state of the transition must continuously be active and a condition given in the transition label must continuously be fulfilled for a time interval specified by a lower and an upper bound. (This means that no discrete event can cause the transition execution.)
- The source state of the transition must continuously be active until a (discrete) event specified in the transition label occurs in the time interval specified by a lower and an upper bound.

In general, timeout events and timed transitions could be used with discrete or continuous time domains. Nevertheless, in the Statecharts formalism timeout events are always used with a discrete time domain.

3 Comparison of Statecharts Variants

In this chapter 20 Statecharts variants are compared. At first, all features are listed which are later on used for the comparison. Then the Statecharts variants are enumerated. Finally, these variants are compared with respect to the distinctive features.

3.1 Survey of Distinctive Features

The following list presents all essential features in which Statecharts variants differ from each other. The list is classified in syntactical and semantic features. Some of these features were already explained in the problem list, the other ones are described in the feature list.

Feature list:

- Syntax:
  - graphical / textual
    The graphical notation is one of Statechart's most important features for achieving clear specifications. Nevertheless, there also exist some textual Statecharts variants. But they were only developed to perform semantic investigations. They are not intended to be used for real specification purposes.
  - negated trigger event (item (3) of problem list)
  - timeout event (item (19) of problem list)
  - timed transition (item (19) of problem list)
  - disjunction of trigger events
    The trigger expression provides the possibility to connect events by the logical AND-operator. Furthermore, in some Statecharts variants the logical OR-operator may also be used, so that disjunctions of events are possible.
  - trigger condition
    Besides the trigger expressions some Statecharts variants also offer additional trigger conditions. In contrast to the trigger expression which checks whether an (instantaneous) event has occurred, the trigger condition checks whether a variable contains the right value or whether a state in another parallel component of the Statechart is active. (See also problem list, item (6)).
  - state reference (item (6) of problem list)
  - assignment to variable
    All Statecharts variants allow the generation of events during transition execution. This possibility is modelled by an action in the transition label. Another kind of action exists if variables and assignments to variables are allowed. All Statecharts variants providing this facility use global variables i.e. variables to be accessed from all over the Statechart. The variable access is performed according to "multiple read - single write". Variables may also be used in trigger conditions in order to check whether a variable contains a certain value.

- inter-level transition (item (5) of problem list)
- history mechanism
  The history mechanism provides a state change to that substate of an OR-state which has been the last one active before the OR-state has been left. This mechanism realizes a sort of suspend/resume formalism.

- Semantics:
  - operational / denotational (item (8) of problem list)
  - compositional (item (7) of problem list)
  - synchrony hypothesis fulfilled (item (1) of problem list)
  - deterministic (item (15) of problem list)
  - concurrency modelled by interleaving / true concurrency
    Almost every Statecharts variant models concurrency by interleaving. Only in two Statecharts variants transitions may be executed actually in parallel. This type of parallelism is called *true concurrency.*
  - discrete / continuous time
    Most Statecharts variants are based on a discrete time domain (usually the natural numbers). Only those providing timed transitions use continuous time (e.g. the non-negative real numbers).
  - globally consistent (item (4) of problem list)
  - causal (item (2) of problem list)
  - instantaneous state (item (9) of problem list)
  - finite number of transition executions at one instant (item (14) of problem list)
  - priorities (item (16) of problem list)
  - non-preemptive interrupt (item (17) of problem list)
  - preemptive interrupt (item (17) of problem list)
  - distinction between internal and external events (item (18) of problem list)
  - local event
    Most Statecharts variants only provide global events, i.e. that they can be sensed all over a Statechart. Motivated by semantic considerations, local events were introduced in several Statecharts variants. Their scope is determined by the state hierarchy.
  - discrete / continuous event
    Discrete events only exist at an instant of time, whereas continuous events can exist during a finite period of time.

3.2 Statecharts Variants

Many different Statecharts variants exist. Though substantial syntactical and semantic differences exist between them, most of them are still referred to as Statecharts.

Table 2 provides a short overview. Each variant is associated with the variant number (No.) which is later on used in the comparison (Table 3). Furthermore the name, its developer and the reference (Ref.) of each variant are given. Since some variants were published in the same article, their entries are provided with additional information, so that they can be identified unambiguously. The table entry with number 21 designates a hypothetical Statecharts variant. It is characterized by the set of features it shall provide and represents a symbiosis of existing Statecharts variants.

No.	Name	Developer	Ref.	Additional Information
1	Statecharts	Harel	[8]	
2	Statecharts	Harel, Pnueli et al.	[10]	
3	Statecharts	Huizing, Gerth	[14]	causal, locally consistent
4	Statecharts	Huizing, Gerth	[14]	causal, globally consistent
5	Statecharts	Huizing, Gerth	[14]	not causal, globally consistent
6	Statecharts	Pnueli, Shalev	[23]	operational semantics
7	Statecharts	Pnueli, Shalev	[23]	declarative semantics
8	Statecharts	Huizing, Gerth, de Roever	[15]	
9	Statecharts	Hooman, Ramesh, de Roever	[12]	
10	RSML	Leveson et al.	[17]	
11	Argos	Maraninchi	[20]	
12	Modular Statecharts	Classen	[6]	
13	Statecharts	Maggioli-Schettini, Peron	[18]	
14	Statecharts	Day	[7]	
15	Statecharts	Peron	[22]	non-instantaneous, no priorities
16	Statecharts	Peron	[22]	non-instantaneous, priorities
17	Statecharts	Peron	[22]	instantaneous, no priorities
18	Statecharts	Peron	[22]	instantaneous, priorities
19	Timed Statecharts	Kesten, Pnueli	[16]	
20	Hybrid Statecharts	Kesten, Pnueli	[16]	
21	Statecharts	von der Beeck	here	

Table 2: Statecharts variants

3.3 Comparison of Statecharts Variants According to Distinctive Features

Statecharts Feature	Variant Number 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
graphical / textual	g	g	g	g	g	g	g	g/t	g/t	g	g	g/t	g	g	g	g	g	g	g	g	g
negated trigger event	+	+	+	+	+	+	+	+	+	-	+	-	+	+	-	-	-	-	+	+	+
timeout event	+	-	-	-	-	-	-	+	+	+[9]	-	-	-	+	+[4]	+[4]	+[4]	+[4]	+[4]	+[4]	+[4]
timed transition	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+[2]	+[2]	+
disjunction of trigger events	-	+	+	+	+	-[13]	-[13]	+	+	-	-	-	+	+	-	-	-	-	-	-	+
trigger condition	+	+	-	-	-	-[13]	-[13]	+[3]	-	+	-	-	-	+	-	-	-	-	+	+	-
state reference	+	+	-	-	-	-[13]	-[13]	+[3]	-	+	-	-	-	+	-	-	-	-	-	-	-
assignment to variable	+	+	-	-	-	-[13]	-[13]	-	-	+	-	-	-	+	-	-	-	-	+	+	-
inter-level transition	+	+	-[14]	-[14]	-[14]	+	+	+	+	+	-	-	+	+	+	+	+	+	-	-	-
history mechanism	+	+	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-	-
operational/denotatio.	-	o	o	o	o	o	[11]	d	d	o	o	d	o	o	d	d	d	d	o	o	?
compositional	-	-	-	-	-	-	-	+	+	-	+	+		-	+	+	+	+	+	+	+
synchrony hypothesis	+	+	+	+	+	+	+	+	+	+[7]	+	+	+	-	-	-	-	-	-[8]	-[8]	-
deterministic	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-
interleav./true concurr.	i	i	i	i	i	i	i	i	i	i	i	i	i	i	t	t	t	t	15	15	?
discrete/contin. time	d	d	d	d	d	d	d	d	d	d	d	d	d	d	c	c	c	c	c	c	c
globally consistent	-	-	-	+	+	+	+	-[13]	+	+[5]	-	+[5]	-	-	+[5]	+[5]	+[5]	+[5]	-	-	-
causal	+	+	+	+	-	+	+	+	+	+[1]	-[12]	+	+	+	+	+	+	+	+	+	+
instantaneous state	?	-	-	-	-	-	-	-	-	+	-	+	-	-	-	-	+	+	+	+	+
finite transition no.	?	+	+	+	+	+	+	+	+	-	+	-	+	+	+	+	+	+	-	-	+
priorities	-	-	-[14]	-[14]	-[14]	-[10]	-[10]	-	-	-	+	-	-	+	-	+	-	+	-	-	+
non-preempt. interrupt	?	-	-[14]	-[14]	-[14]	-	-	-	-	?	+	-	-	-	-	-	+	+	?	?	+
preemptive interrupt	?	+	-[14]	-[14]	-[14]	+	+	+	+	+	+	-	+	+	+	+	-	-	+	+	+
distinc. int./ext. event	+	+	+	+	+	+	+	+	+	+[6]	+	+	+	-	+	+	+	+	+	+	+
local event	-	-	-	-	-	-	-	+	+	+	+	+	-	-	-	-	-	-	-	-	+
discrete/contin. event	d	d	d	d	d	d	d	d	d	d	d	d	d	d	c	c	c	c	d	d	d

Table 3: Comparison of Statecharts variants according to distinctive features

Legend of Table 3:

"+" The Statecharts variant provides this feature.

"-" The Statecharts variant does not provide this feature.

"?" It is not known whether the Statecharts variant provides this feature.

"1" This variant requires a more rigorous form of causality than the other causal variants: At first all transitions triggered by an external event are executed before transitions triggered by an internal event are executed.

"2" The timed transitions of this Statecharts variant cannot be triggered by discrete events, but only by a boolean expression over shared variables.

"3" The trigger condition must be a state reference.

"4" A timeout can be modelled in this variant as a special case of a more general time condition.

"5" This Statecharts variant is trivially globally consistent, because it prohibits negated events in the trigger part of transition labels.

"6" Events generated by transition execution may be internal or external events: Internal events are broadcast, whereas external events are sent to precisely one explicitly specified receiver.

"7" The synchrony hypothesis is only valid inside a component state machine of the RSML formalism. Communication between component state machines can occur asynchronously.

"8" This Statecharts variant provides timed and untimed transitions. Therefore the synchrony hypothesis is not fulfilled, if timed transitions are used.

"9" A timeout event has to be defined in the trigger condition.

"10" In [23] it is stated that this variant can express priorities. However, this only means that negated events can be used in order to avoid simultaneously valid trigger expressions of a pair of transitions.

"11" This variant has a declarative, non-compositional semantics.

"12" According to [20] Argos is causal, but there the term causal is used with another semantics. There it just means reactive and deterministic. This notion is weaker than that used here.

"13" The introduction of this property is only sketched as an enhancement.

"14" This feature is not provided in this variant, because the usual hierarchical states (AND- and OR-states) of the Statecharts formalism are not considered in the description of this variant.

"15" This variant provides timed and untimed transitions. Timed transitions are executed according to true concurrency, whereas untimed transitions are executed in an interleaved way.

In the following some very specific features of individual Statecharts variants are described which were not considered in Table 3.

- RSML (variant number 10) provides
  - asynchronous one-to-one communication between complex component state machines whereas the usual synchronous broadcasting mechanism is used inside these components
  - precisely defined interface descriptions at the component state machines specifying the communication with other component state machines, i.e. the mapping from received external information onto internal information and from internal generated information onto external information to be sent
  - transition labels which are more intuitive (e.g. usage of AND/OR-tables) and noted separately (not at the arrows between the states representing the transition)
  - AND/OR-tables providing a two-dimensional representation of the disjunctive normal form in order to clearly arrange trigger conditions
  - transition buses providing a representation being more clearly than state transition diagrams by reducing the number of arrows
- Hybrid Statecharts (variant number 20) are an enhancement of Timed Statecharts (variant 19) supporting the specification of continuous processes. For this purpose a basic state can be associated with a differential equation describing a continuous change which lasts as long as this state is active.

4 Conclusion

A detailed comparison of existing Statecharts variants was presented. As a prerequisite, problems of the Statecharts formalism were described, approaches for solving these problems were evaluated, and distinctive features with respect to the variants' syntactical and semantic aspects were elaborated. As an outcome of the comparison a characterization of a new hypothetical Statecharts variant was presented. This variant still has to be precisely defined and will be used in an integration with Structured Analysis similar to the approach presented in [1].

5 References

1. M. von der Beeck: *Integration of Structured Analysis and Timed Statecharts for Real-Time and Concurrency Specification*, Proc. of ESEC '93, LNCS, vol. 717, Springer, pp. 313-328, 1993
2. A. Benveniste, G. Berry: *The Synchronous Approach to Reactive and Real-Time Systems*, Proc. of the IEEE, vol. 79, no. 9, pp. 1270-1282, 1991
3. G. Berry: *Preemption in Concurrent Systems*, Proc. of FSTTCS 93, LNCS, vol. 761, Springer, pp. 72-93, 1993

4. G. Berry, G. Gonthier: *The ESTEREL synchronous programming language: design, semantics, implementation*, Science of Computer Programming, vol. 19, pp. 87-152, 1992

5. G. Berry, S.Ramesh, R. Shyamasundar: *Communicating Reactive Processes*, Proc. of ACM Symp. on Principles of Programming Languages, Charleston, 1993

6. A. Classen: *Modulare Statecharts: Ein formaler Rahmen zur hierarchischen Prozeßspezifikation*, Master Thesis, (in German), Lehrstuhl für Informatik II, Aachen University of Technology, Germany, 1993

7. N. Day: *A Model Checker for Statecharts (Linking CASE tools with Formal Methods)*, Technical Report 93-35, University of British Columbia, Vancouver, Canada, 1993

8. D. Harel: *Statecharts: A visual formalism for complex systems*, Science of Computer Programming, vol. 8, pp. 231-274, 1987

9. D. Harel, A. Pnueli: *On the development of reactive systems*, in: Logics and Models of Concurrent Systems, NATO ASI Series, vol. 13, ed. K. Apt, Springer, pp. 477-498, 1985

10. D. Harel, A. Pnueli, J. Schmidt, R. Sherman: *On the Formal Semantics of Statecharts*, Proc. of 2nd IEEE Symp. on Logic in Computer Science, Ithaca, NY, pp. 54-64, 1987

11. C. Hoare: *Communicating Sequential Processes*, Prentice Hall, 1987

12. J. Hooman, S. Ramesh, W. de Roever: *A compositional axiomatization of Statecharts*, Theoretical Computer Science, vol. 101, no. 2, Elsevier, pp. 289-335, 1992

13. C. Huizing: *Semantics of Reactive Systems: Comparison and Full Abstraction*, Ph.D. thesis, Technical University Eindhoven, The Netherlands, 1991

14. C. Huizing, R. Gerth: *Semantics of Reactive Systems in Abstract Time*, LNCS, vol. 600, Springer, pp. 291-314, 1992

15. C. Huizing, R. Gerth, W. P. de Roever: *Modelling Statecharts behaviour in a fully abstract way*, LNCS, vol. 299, Springer, pp. 271-294, 1988

16. Y. Kesten, A. Pnueli: *Timed and Hybrid Statecharts and their Textual Representation*, LNCS, vol. 571, Springer, pp. 591-620, 1992

17. N. Leveson, M. Heimdahl, H. Hildreth, J. Reese: *Requirements Specification for Process-Control Systems*, Technical Report 92-106, University of California, USA, 1992

18. A. Maggioli-Schettini, A. Peron: *Semantics of Full Statecharts Based on Graph Rewriting*, Dipartimento di Informatica, Universita di Pisa, Italy, 1993

19. F. Maraninchi: *Argos: a Graphical Synchronous Language for the Description of Reactive Systems*, RT-C29, LGI-IMAG Grenoble, France, 1991

20. F. Maraninchi: *Operational and Compositional Semantics of Synchronous Automaton Compositions*, LNCS, vol. 630, CONCUR '92, Springer, pp. 550-564, 1992

21. F. Maraninchi: *Languages for reactive systems: a common framework for comparing Statecharts and Argos*, Technical Report, Spectre Report C34, LGI-IMAG, Grenoble, France, 1992

22. A. Peron: *Synchronous and Asynchronous Models for Statecharts*, Technical Report TD-21/93, Dipartimento di Informatica, Universita di Pisa, Italy, 1993

23. A. Pnueli, M. Shalev: *What is in a Step: On the Semantics of Statecharts*, LNCS, vol. 526, Springer, pp. 244-264, 1991

24. D. Scott, C. Strachey: *Towards a Mathematical Semantics for Computer Languages*, Proc. of Symposium on Computers and Automata, (ed. J. Fox), Polytechnic Institute of Brooklyn Press, New York, USA, pp. 19-46, 1971

25. B. Steffen: *Hierarchische Spezifikationen*, GI/ITG-Fachgespräch für Verteilte Systeme, Kiel, 1994