Statemate Course

Statemate/SDL — Teleteaching Vorlesung

W.-P. de Roever
K. Baukus
CAU Kiel

D. Hogrefe
H. Neukirchen
MU Lübeck

Session VI

Semantics of Statecharts

Abstract: We discuss the central concepts and decisions for various possible semantics for Statecharts (and the "real" implemented one).

Literature: Dissertation Kees Huizing: "Semantics of reactive systems: comparison and full abstraction", Eindhoven University of Technology, 1991.

In particular the following pages are relevant:

- "Everything you always wanted to know about Statecharts", Huizing and de Roever.
- "On the semantics of reactive systems", Huizing and Gerth.

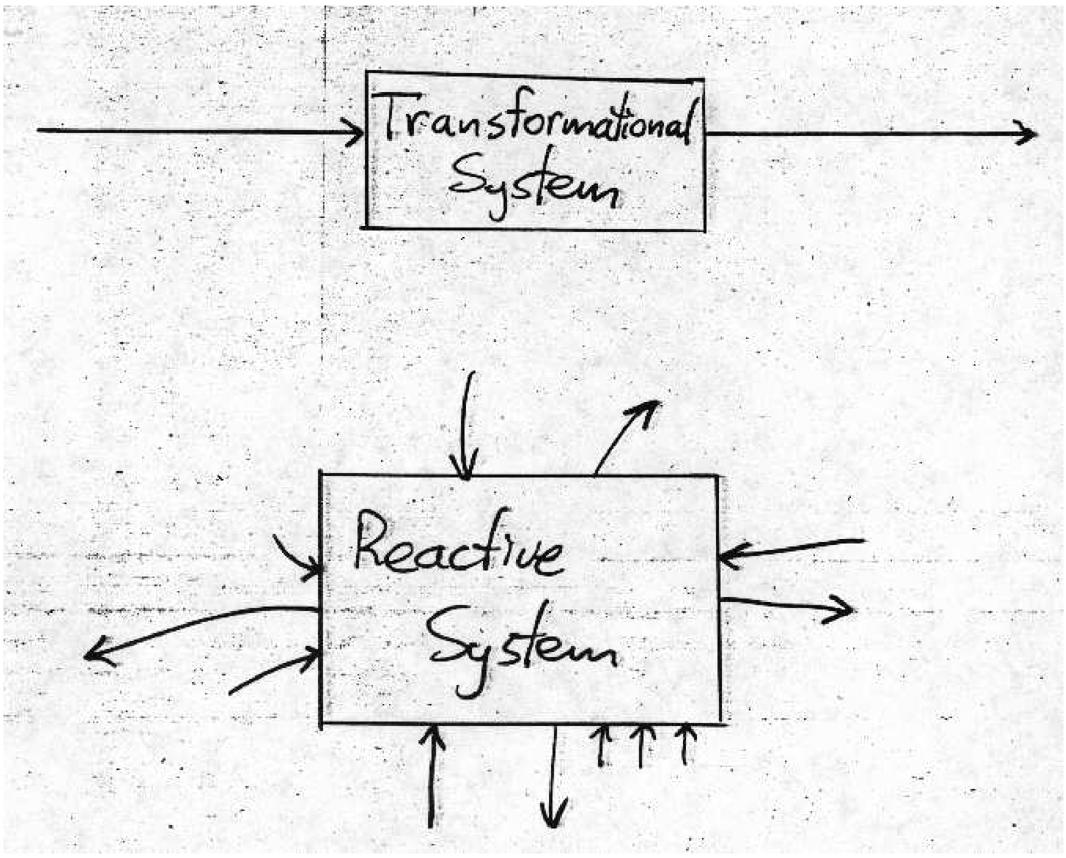
And:

Chapter 6 of "Modeling Reactive Systems with Statecharts", by David Harel and Michal Politi. McGraw-Hill, 1998.

9 Semantics of Statecharts

9.1 Summary of previously discussed material (cfr. first lesson)

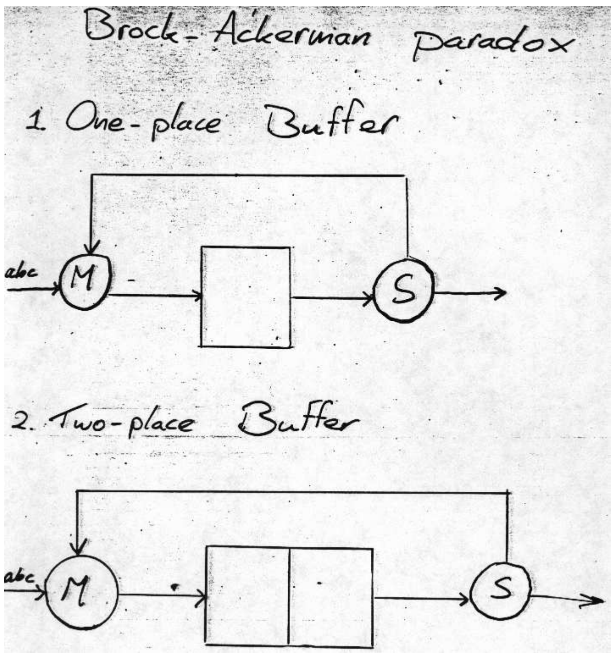
- There is a fundamental dichotomy between **transformational systems** described by the relation between initial and corresponding final states, i.e., their input/output behavior, and
- **Reactive systems**, whose only purpose is to maintain an ongoing relationship with their environment.



Brock-Ackermann Paradox

The Brock-Ackermann paradox explains why reactive systems cannot be characterized by a function mapping sequences of inputs to sequences of outputs.

- Consider two systems, a one-place buffer and a two-place buffer. If you consider these transformationally, they display the same initial-final state behavior.



in: $abc\ldots$
out: 1. e.g. $abac\ldots$
2. e.g. $abca\ldots$
not $abac\ldots$

But if the output of these systems is fed back, and merged with their input they behave differently. (See transparency)

- What's needed to characterize a reactive system is **recording the relative order of inputs and outputs**, i.e., the way they are interleaved.

Central Decisions for a Statecharts Semantics

- Semantics of reactive systems is state-based
- Observations are sequences of pairs of inputs I and corresponding outputs O, i.e., of pairs of the form (I, O).

  In practice a reactive system is therefore described by sequences of the following form:

$$S_1 \Longrightarrow_{I_1}^{O_1} S_2 \Longrightarrow_{I_2}^{O_2} S_3 \Longrightarrow_{I_3}^{O_3} \Longrightarrow \cdots$$

- Transitions don't take time, time is spent in states.

  This has a simple reason: the reaction of a reactive system to environmental inputs should be always well-defined. As a consequence, state-changes shouldn't take time.

Berry’s synchrony hypothesis

Reaction time between input (i.e., trigger) and corresponding output (i.e., response) is zero.

Why?

- Recall that individual reaction times are too complicated to handle, abstractly, on the high level of specification Statecharts are aiming at.
- a fixed non-zero reaction time wouldn’t allow transition refinement.
- Unspecified reaction times lead to chaos, and is not desired at a high level of abstraction.

$\Longrightarrow$ Only one reaction time satisfies all criteria: zero! For:

- Now transition can always be refined
- specific
- deterministic

Detailed Argumentation from Lesson I

Possibility 1 : Specify a concrete amount of time for each situation. This forces us to quantify time right from the beginning. **Clumsy**, and not appropriate at this stage of specification where one is only interested in the relative order and coincidence of events.

Possibility 2 : Fix reaction time between trigger **a** and corresponding action **a** within **e/a** (the label of a transition) upon 1 time unit.

Doesn't work: Upon refining **question/answer** to a **question/consult** and a **consult/answer** transition, there's a change of time, which may have far reaching effects (because of **tm(n)**-events, e.g.)

$\Longrightarrow$

A fixed execution time for syntactic entities (transitions, statements, etc.) is not **flexible** enough.

Possibility 3 : Leave things open: say only that execution of a reaction takes some positive amount of time, and see at a later stage (closer to the actual implementation) how much time things take.

Clumsy, introduces far too much nondeterminism.

Reaction time of a system (2)

Summary : We want the execution time associated to reactions to have following properties:

- It should be accurate, but not depending on the actual implementation.
- It should be as short as possible, to avoid artificial delays.
- It should be abstract in the sense that the timing behavior must be orthogonal to the functional behavior.

$\Longrightarrow$

Only choice that meets all wishes is **zero reaction time**.

As a result all objections raised w.r.t. the possibilities mentioned on the previous page are met!

- Now, for instance, upon refining transition **question/answer** from previous page into two transitions, the reaction time of this refinement is the same as that of the original transition.
- Objection 3 on the previous transparency is resolved, too.
- Finally, also objection 1 (on previous transparency) is met, because $0 + 0 = 0$!

Berry's Synchrony Hypothesis

- Is Berry's synchrony hypothesis implementable?
  Yes, if the input frequency is low w.r.t. the time required for computing response.
- However, this hypothesis leads to a number of counterintuitive consequences, if carried through.
  Careful: the following example does not describe the Statecharts semantics as implemented in Statemate.

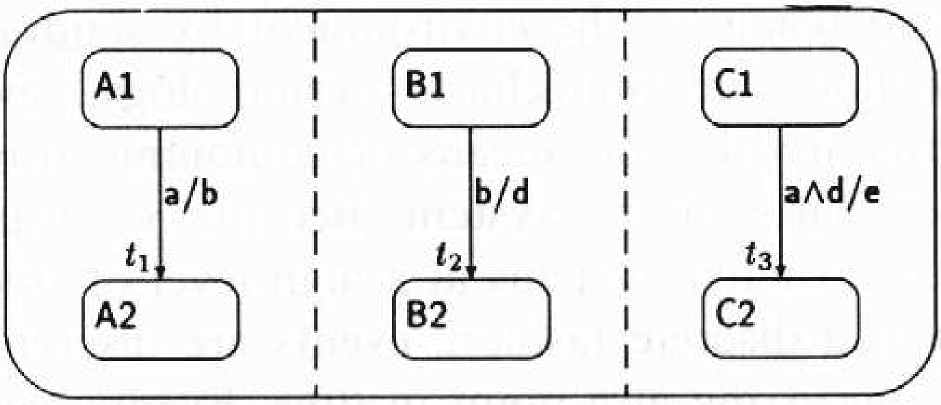


Fig. 6.

$\text{A} \wedge \text{D}$ is a generated trigger, since we assume the reaction time to be zero. A consequence is that transition t_3 is taken!!

Combination with Negation

The synchrony hypothesis leads to problems if combined with the possibility of checking the absence of signals (the latter is customary in the synchronous world, and a possibility not offered in the asynchronous world):

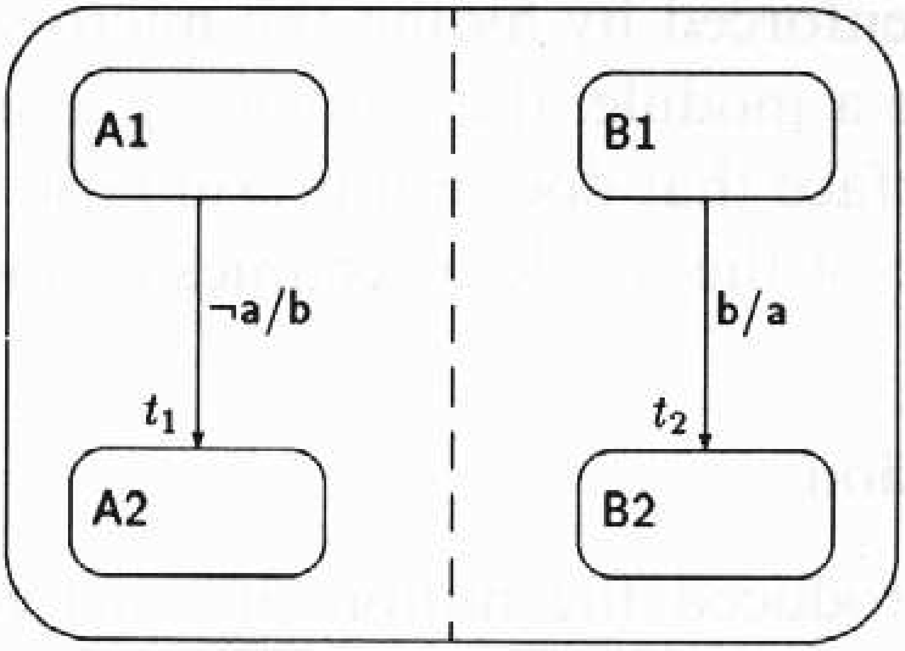


Fig. 8.

If **a** is absent, i.e., ¬ **a** holds as condition, transition t_1 is taken, i.e., **b** is generated, and hence t_2, i.e., **b/a** is taken, generating **a within the same time unit**, i.e., in zero time, hence transition t_1 should not be taken.

This is called the “Grandfather paradox”.

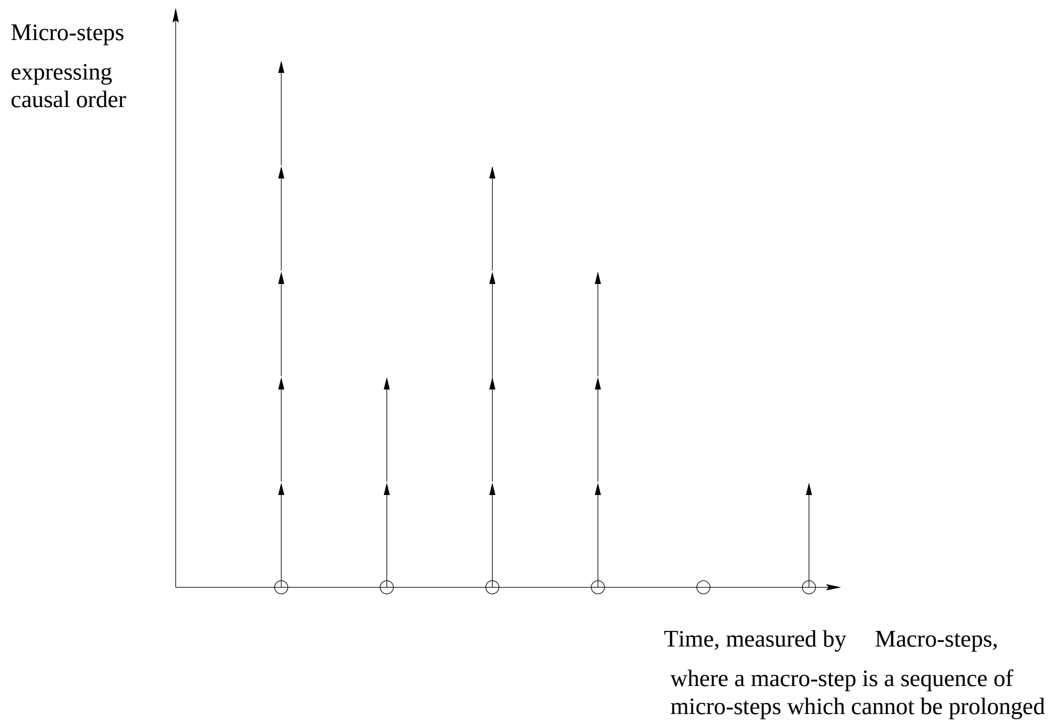
It’s solution is to **order** event occurrences causally, with **later events not influencing earlier events**:

$$\neg a \leq b \leq a$$

> Note here: this causal order has nothing to do with the passage of time; it merely refers to causal chains **within** one time step.

The new semantics

- This leads to a semantics of the following form:


Micro-steps
expressing
causal order
Time, measured by Macro-steps,
where a macro-step is a sequence of
micro-steps which cannot be prolonged

- Macro-steps are observable steps $\overset{O}{\underset{I}{\Longrightarrow}}$
- Each macro-step is a sequence of micro-steps, that are ordered causally; one micro-step can never influence previous micro-steps.
- In Statecharts as implemented by Statemate causality is trivially obtained because in Statemate events generated in one step are only available in the next step, and only for that one. I.e., there is no causality within one step.

Problems with this new semantics

- The problem with macro-steps is that they lead to a **globally inconsistent** semantics, i.e., transitions are taken in one macro-step which aren't generated globally.

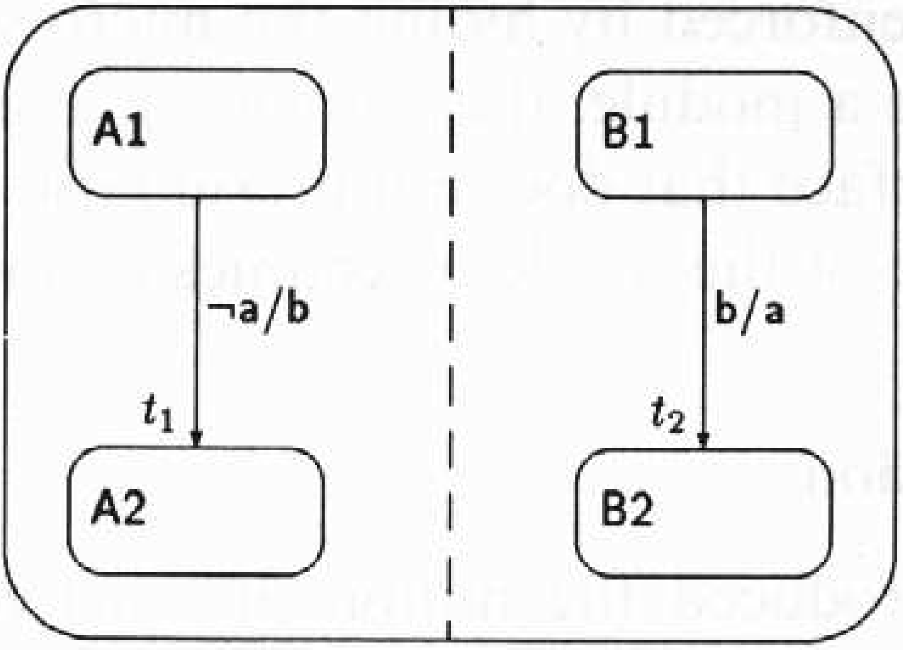


Fig. 8.

$$S_1 \Longrightarrow^{b}_{\emptyset} S_2 \Longrightarrow^{a}_{b} S_3$$

Here absence of triggers generates presence of triggers, which violates their absence within the same step (not globally consistent).

- These considerations lead to the fundamental question:

  Is a semantics for such languages possible which satisfies all "reasonable" assumptions? I.e., which is both good for program development and for program composition? The answer is **NO**.

- This is a serious problem. As it turns out, the semantics with macro-steps indicating passage of time, and refined by causally ordered micro-steps is a basis for a compositional semantics for Statecharts in which the semantics of a construct is a function of the semantics of its parts. But this semantics turns out to be too difficult to handle for the engineers of I-Logix, and of Israeli Aircraft Industries, its main customer for the Statemate system.

- Hence looking for a "best" semantics makes a lot of sense. What our theorem below says is that, in a certain sense, there is no best semantics. However, it does leave some room for the search for ever better semantics!!

There is no “best” semantics for Statecharts

Let’s list a couple of desirable properties of such a semantics:

Responsiveness: Reactions are simultaneous with their triggers — this facilitates refinement of transitions from a high to a lower level.

Causality: Without a causal order of the micro-steps inside a macro-step, charts s.a.:

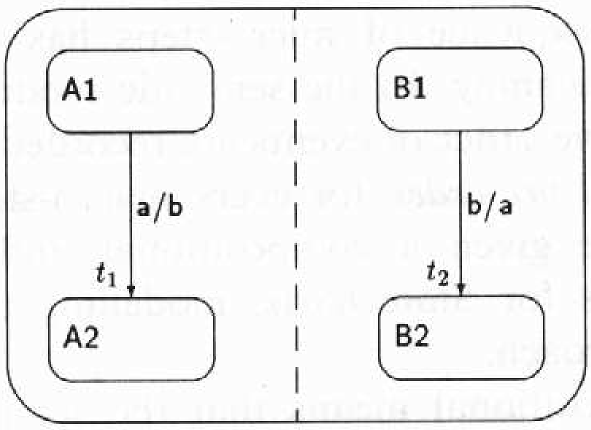


Fig. 9.

would trigger each other, which makes no causal sense. Such charts are excluded imposing causality.

Modularity: Modules can be composed on the basics of their macro-steps, i.e., the external interface of a (parallel) composition of modules is of the same nature as their mutual interface w.r.t. each other. (This is inspired by a paper by Pnueli and Shalev)

Impossibility of a Semantics being Causal, modular, and responsive

Modularity, causality, and responsiveness can be mathematically expressed; the **impossibility** of all three being satisfied simultaneously becomes a theorem, proved in the paper by Huizing and Gerth.

However, also intuitively this is clear:

- Causality and responsiveness leads to

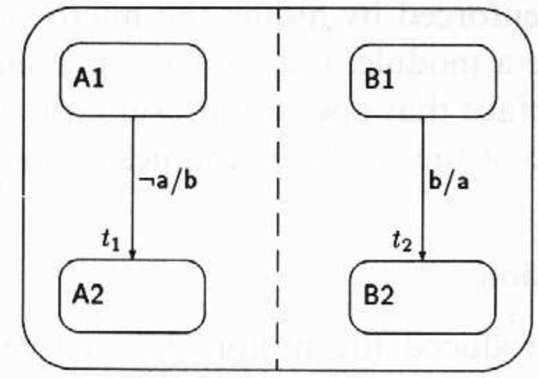


Fig. 8.

  examples in which both a and its absence $\neg a$ occur within the same macro-step $\Longrightarrow$ no global consistency $\Longrightarrow$ no modularity

- Modularity and responsiveness imply there exists no satisfactory semantics for the example above. This choice is made in the synchronous language ESTEREL, in which examples as the one above are excluded on syntactic grounds by a compiler.

9.2 Classification of possible semantics for Statecharts

Next we list a few possible semantics for Statecharts, semantics A – E, of which E is closest to the one actually implemented in Statemate, and discuss the anomalies allowed by them (including those of the implemented semantics of Statecharts).

Semantics A

Events generated as a reaction to some input can only be sensed in the step following that input. (This is a choice made in the implemented semantics of Statecharts.)

Anomaly: no simultaneity of action and reaction, i.e., no responsiveness.

In semantics A the trigger $a \wedge c$ will not occur:

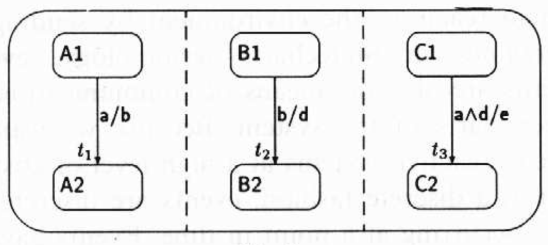


Fig. 6.

This example makes clear that in semantics A the moment of generation of an event is too important — a too detailed analysis of charts is required for adopting it.

Semantics B

In order to overcome the problem with semantics A, absence of responsiveness, micro-steps are introduced, with events sensed in the next micro-step.

Then, in the previous example the third transition is taken.

Consider now the trigger $b \wedge \neg c$ for the third transition; the transition is taken, because in the second micro-step, event c is not yet sensed. This example also works for semantics A.

Disadvantage: Semantics B is too subtle to be of any practical use; same objection as to semantics A.

Semantics C

Requires global consistency of **every** micro-step. The reaction of the system to an input should

- not only be enabled by events generated in previous micro-steps
- but also by events generated in the full macro-step.

As a consequence, the $b \wedge \neg c$ transition is not taken.

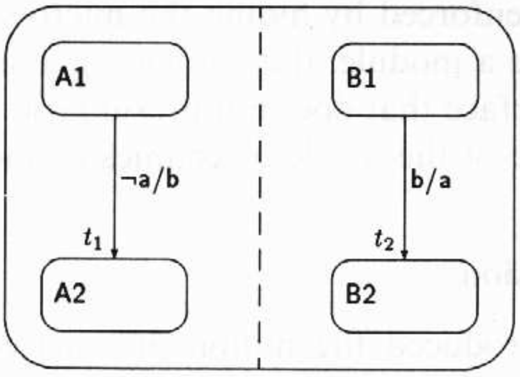


Fig. 8.

This example is excluded in semantics C, leads to contradiction. I.e., syntactical means must be found to exclude it, as done in ESTEREL by a compiler.

This makes a lot of sense, as evidenced by the considerable success of ESTEREL of Gérard Berry.

However, this semantics is not modular. This implies that a modular development of the system is cumbersome, since every developer has to know the detailed micro-behavior of the other processes. Hence, this semantics is appropriate for top-level guys only, and that's what Gérard Berry's crowd consists of.

Semantics D

All events generated during some macro-step considered as if they were present right from the beginning of the macro-step.

Semantics D allows

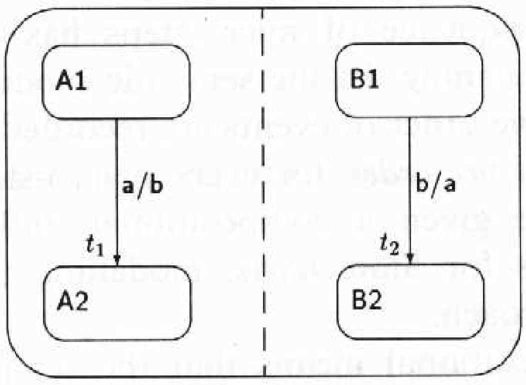


Fig. 9.

to be taken: reactions may trigger themselves. I.e., semantics D is not causal.

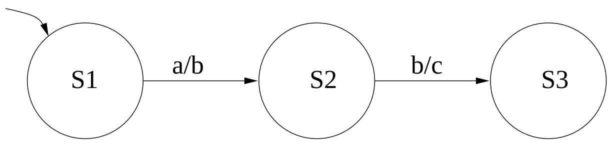
Note: In semantics D, the external world does not generate an a event!

Conclusion: This example should be rejected!

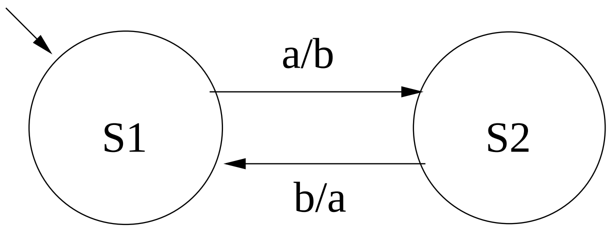
Semantics E

Events are generated at the next step, but no input from the environment is possible **before** the reaction of the system has completely died out.

This semantics is heavily non-modular, since one macro-step may contain several steps of the A semantics. Events remain active only for the duration of such a step, hence, in one macro-step an event can be activated and deactivated several times, thus leading to a much more complex interface between subsystems, than between the system and its environment.



Generation of event a leads the system eventually to state S_3.

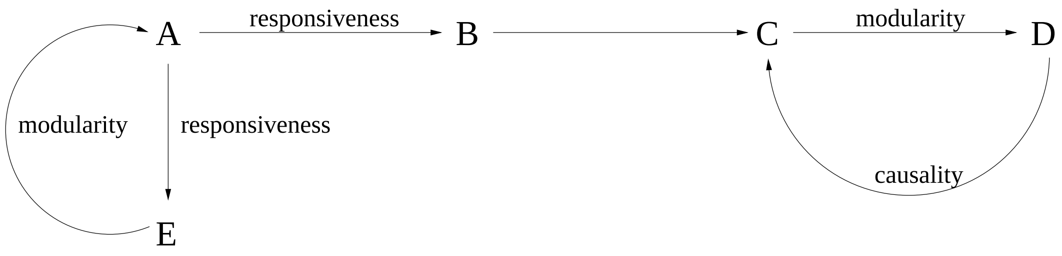


In semantics E, as in the implemented semantics of Statecharts, this example leads to an infinite loop (the so-called: “go repeat” mode): try it out yourself!

Situation

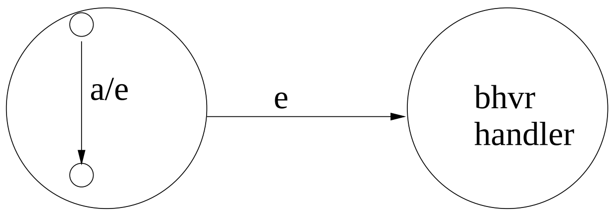
No “best” semantics $\Longrightarrow$ still room for better ones

The situation is summarized in the following figure, showing how each semantics is an attempt to improve on the other one:



What to do? The search is now on for better semantics

1. Several cleaner semantics have been proposed, notably by Florence Maraninchi. She opts for semantics D, in which both charts such as example C and D are excluded, resulting in Argos semantics:



Generation of event a leads to exit transition e being taken. This is called **non-preemptive interrupts**.

The Argos semantics leads to a cleaner concept of state-hierarchy in which **inter-level transitions are not allowed**. Probably a too heavy investment in their “old” semantics, manyear-wise, prevented I-Logix from adopting the cleaner Argos semantics of Maraninchi in Statemate.

2. Huizing and Gerth propose a compositional semantics in which the causal chains inside a module are hidden from its external behavior. This proposal has not yet caught on.

9.3 Statecharts as Implemented

This leaves us with the semantics of Statecharts as it is implemented in Statemate. Computing that semantics is a fairly involved algorithm, only recently (1996) published in a paper by David Harel and A. Naamad.

Operational semantics

We describe the contents of the system status, and the algorithm for executing a step.

The **status** includes:

- a list of states in which the system currently resides;
- a list of activities that are currently active;
- current values of conditions and data-items;
- a list of regular and derived events that were generated internally in the previous step;
- a list of timeout events and their time for occurrence;
- a list of scheduled actions and their time for execution;
- relevant information on the history of states.

The **input** to the algorithm consists of:

- the current system status;
- a set of external changes that occurred since the last step;
- the current time

The **step** execution algorithm works in three main phases:

1. - calculate the events derived from the external changes and add them to the list of events;
   - perform the scheduled actions whose scheduled time has been exceeded, and calculate their derived events;
   - update the occurrence time of timeout events if their triggering events have occurred;
   - generate the timeout events whose occurrence time has been exceeded;
2. - evaluate the triggers of all relevant transition reactions;
   - prepare a list of all states that will be exited and entered;
   - evaluate the triggers of all static reactions
3. - update the history of states;
   - carry out all computations prescribed by the actions in the list produced in the second phase;
   - carry out all updates called for by the actions
   - update the list of current states.

Synchronous/Asynchronous Semantics

Synchronous Semantics: Environment interacts with the system after each step and time advances. This is conceptually quiet easy and appropriate for synchronous hardware. But, the system's reaction on the external input has to be simple (compare with semantics A).

Asynchronous Semantics: Synchrony Hypothesis: system may react with a chain reaction. External input only in **stable** states. Easier to model complex systems, abstraction from real-time. But, the implementation has to be shown to satisfy the assumptions of zero reaction time.