Statemate Course

W.-P. de Roever & R. v. Hanxleden
K. Baukus & J. Lukoschus

CAU Kiel

Semantics of Statecharts — Lesson III

We discuss the central concepts and decisions for various possible semantics for Statecharts (and the “real” implemented one).

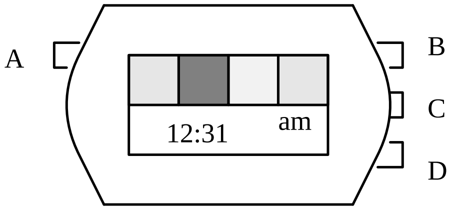
[HdR91] Introduction to Design Choices in the Semantics of Statecharts, C. Huizing, W.-P. de Roever, Information Processing Letters 37, p. 205-213, 1991

[HP98] Modeling Reactive Systems with Statecharts: The STATEMATE Approach, D. Harel, M, Politi. McGraw-Hill, 1998.

Another Example

Alarm watch

As an example of a statechart we use that of a simple digital watch with four buttons A, B, C and D like in the below picture:



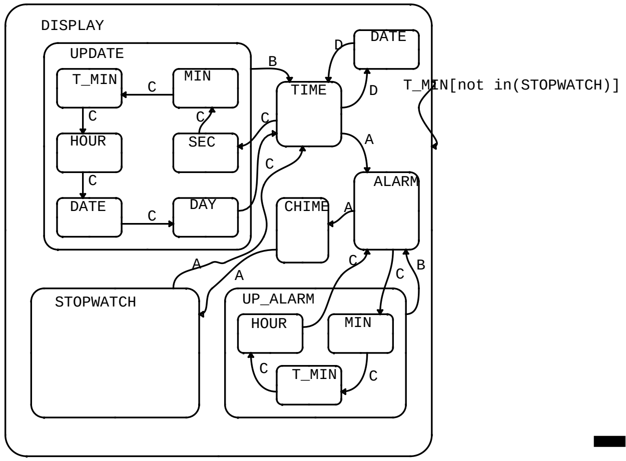
Events

The following events are considered as external:

- A, B, C and D describe the pushing of the four buttons and B_up the release of button B.
- The events Bt_In, Bt_Rm, Bt_Dy and Bt_Wk describe respectively the putting in, removal, drop dead and weakening of the battery.
- T_hits_Hr describes that the internal time has reached a whole hour and T_Hits_Tm describes that the internal time has reached the alarm time.
- T_Min describes that there are two minutes passed since for the last time a button has been pushed.

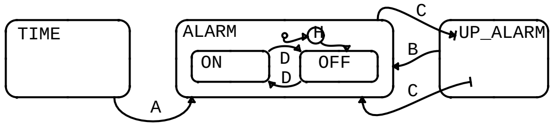
Display control

There is a special state $up - alarm$ for the changing of the internal state of the alarm. Note that $T_m in$ takes care of the resetting of a state.

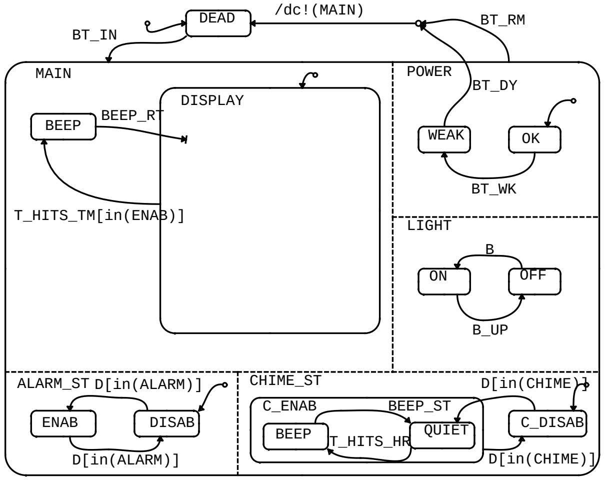


History of states

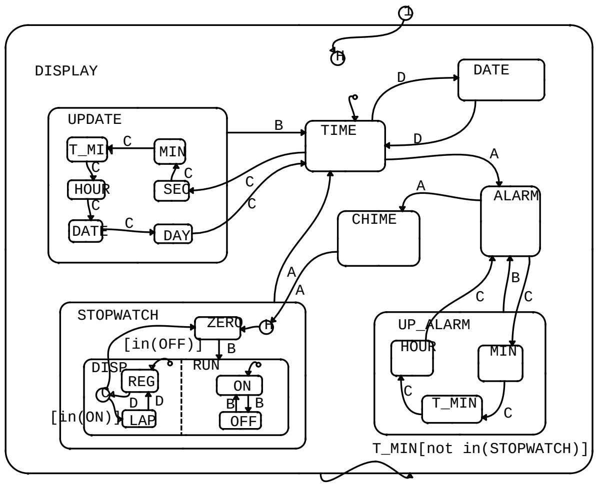
A frequently used way to enter a group of states is by the history of that group. The most simple example of this is the one where you enter the most recently visited state of a group. In the watch example this happens in the zoom-in of the $alarm$ state with two substates on and off. The problem is that the initial default is the off state but when we put on the alarm we want to get back the next time in state on. In the next statechart this described by the **H** connector.



The Watch



Display with Stopwatch



Events

The basic events and condition are external, for example for the watch the pressing of a button is an external event and T_Hits_Tm is an external condition. Besides the external events the following internal events are allowed:

$entered(S)$,	abbreviation	$en(S)$,
$exit(S)$,	abbreviation	$ex(S)$,
$timeout(E, X)$,	abbreviation	$tm(E, X)$,
$true(C)$,	abbreviation	$tr(C)$,
$false(C)$,	abbreviation	$fs(C)$.

Action

An action can be an uninterpreted event symbol, called primitive event, and causes then other transitions in the statechart. Furthermore actions can turn on or off uninterpreted condition symbols. The following primitive actions are allowed:

$make\_true(C)$,	abbreviation	$tr!(C)$,
$make\_false(C)$,	abbreviation	$fs!(C)$,
$history\_clear(S)$,	abbreviation	$hc!(S)$,
$deep\_clear(S)$,	abbreviation	$dc!(S)$.

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**.

“Reasonable” Assumptions

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.

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)

Causality

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

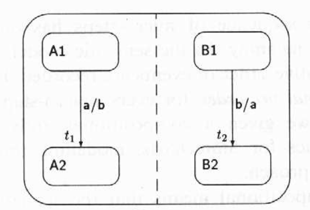


Fig. 9.

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

Impossibility Result

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 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.

Compositional Semantics

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!!

Classification of possible semantics for Statecharts

Several Classes of Semantics

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.
Anomaly: no simultaneity of action and reaction, i.e., no responsiveness. In semantics A the trigger $a \wedge d$ will not occur:

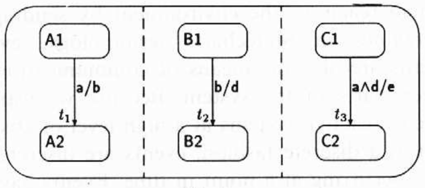


Fig. 6.

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 $b \wedge \neg d$ for the third transition; the transition is taken, because in the second micro-step, event d 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 $b \wedge \neg d$ transition is not taken.

Semantics C (cont'd)

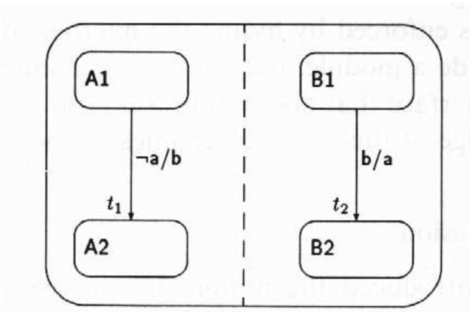


Fig. 8.

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.

Semantics C (cont'd)

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

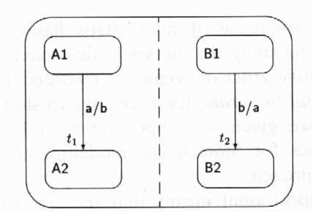


Fig. 9.

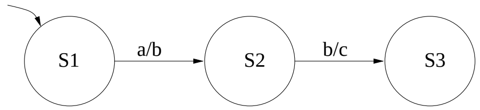
to be taken: reactions may trigger themselves. I.e., semantics D is not causal.
Note: In semantics D, the external world does not generate an a event!
Conclusion: This example should be rejected!

Semantics E

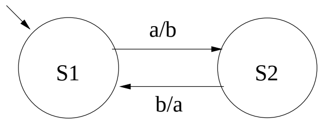
Events are generated at the next step, but no input from the environment is possible **before** the reaction of the system has completely died out.

> 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.

Semantics E (cont'd)



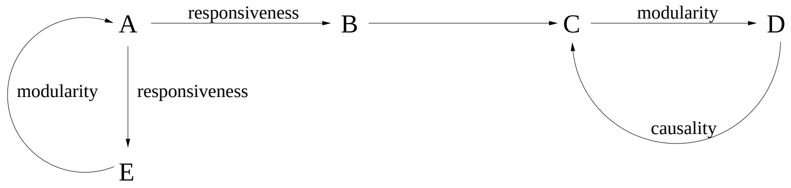
Generation of event a leads the system eventually to state S_3.



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

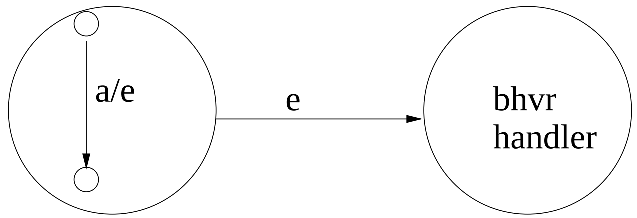
Situation

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:



Search for better Semantics

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**.

Argos Semantics

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.

> 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.

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.

Statecharts Status

- 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.

Input

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.

Step Algorithm: First Step

- 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;

Step Algorithm: Second Step

- 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

Step Algorithm: Third Step

- 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.