Statemate Course

Kai Baukus

Statemate/SDL

W.-P. de Roever	D. Hogrefe
K. Baukus	H. Neukirchen
CAU Kiel	MU Lübeck

Semantics of Statecharts

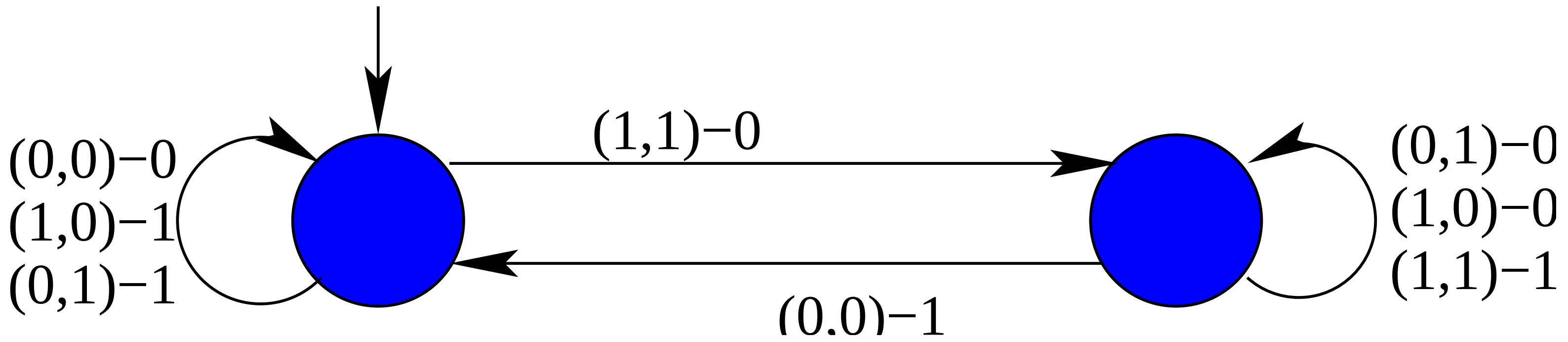
We discuss the central concepts and decisions for various possible semantics for Statecharts (and the “real” implemented one).

[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.

First Session

Serial Addition: Mealy Machine

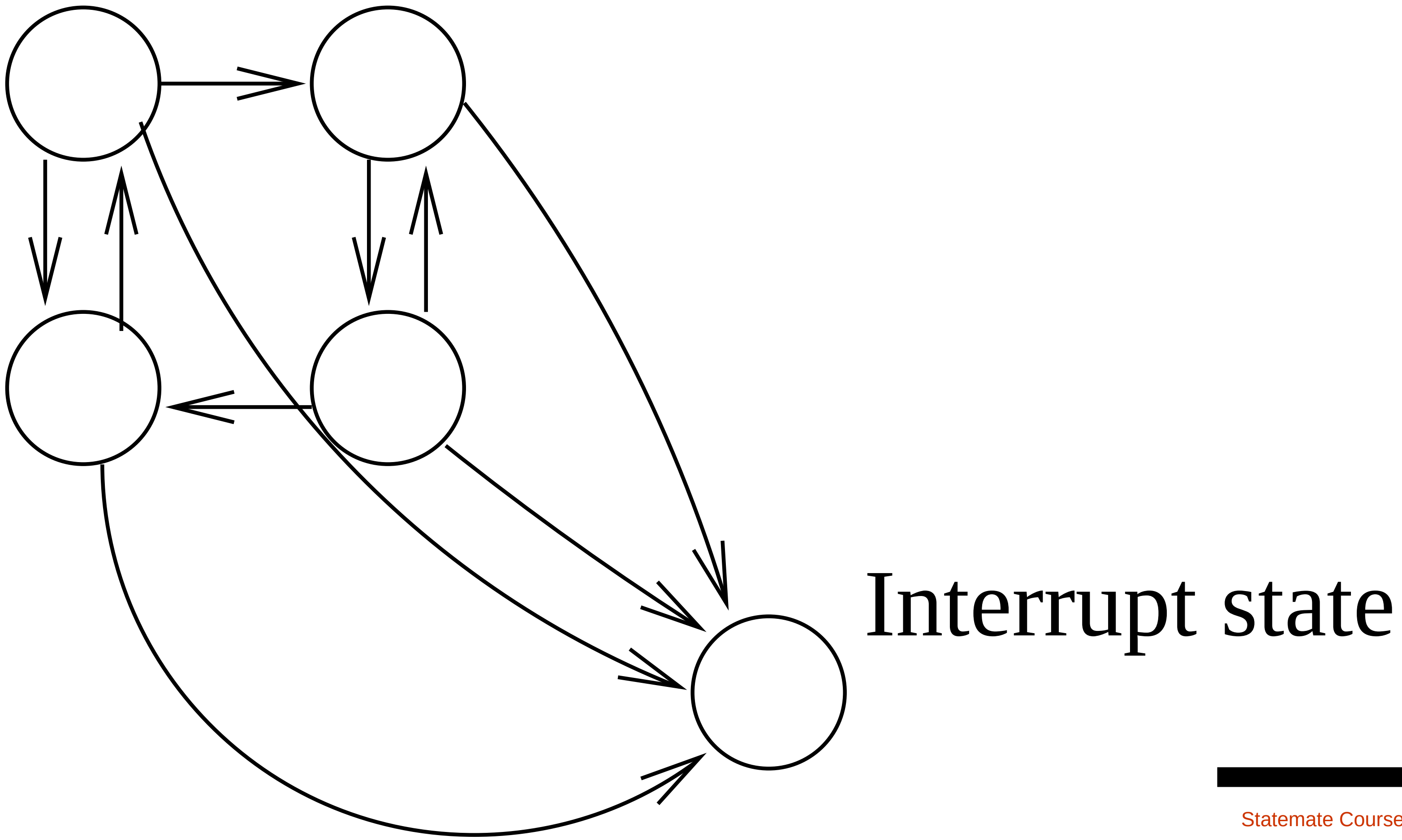


Disadvantages

- **They have no structure.** There is no strategy for their top-down or bottom-up development.
- state-transition diagrams are “flat”, i.e., without a natural notion of depth, hierarchy or modularity,
- state-transition diagrams are uneconomical concerning their transitions:think for instance of a high-level interrupt

Interrupt Transition

They are not economical w.r.t. transitions, when one event has all transitions as a starting point as in case of interrupts:



Disadvantages (cont'd)

- concerning the states state-transition diagrams are even very uneconomical: exponential blow-up
- They are not economical w.r.t. **parallel composition**: **Exponential growth** in the number of states when composed in parallel.
- the nature of state-transition diagrams is inherently sequential and so parallelism can't be represented in a natural way.

Introduction of Superstates

In state **on** the tv set can be in two sub-states: **normal** and **videotext**:

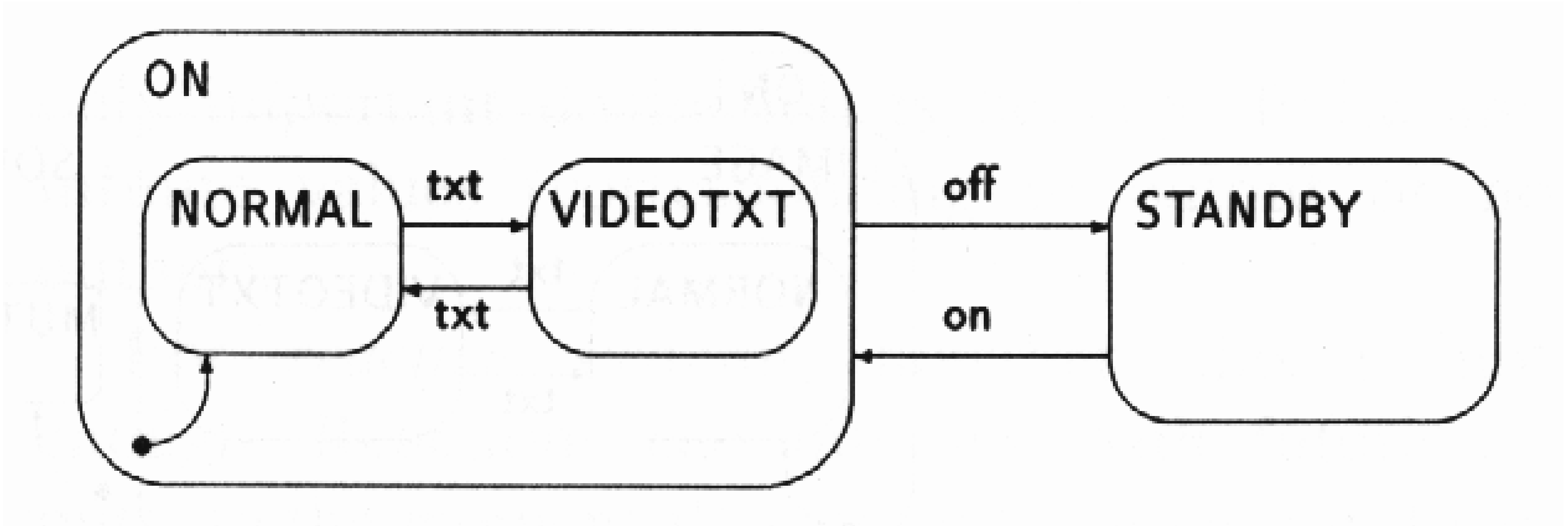


Fig. 3.

The $\longrightarrow$ arrow leading to **normal** specifies which sub-state should be entered when the higher level state**on** is entered, namely **normal**.

Orthogonality

- Two independent components can be put together into an **AND-state**, separated by a dotted line

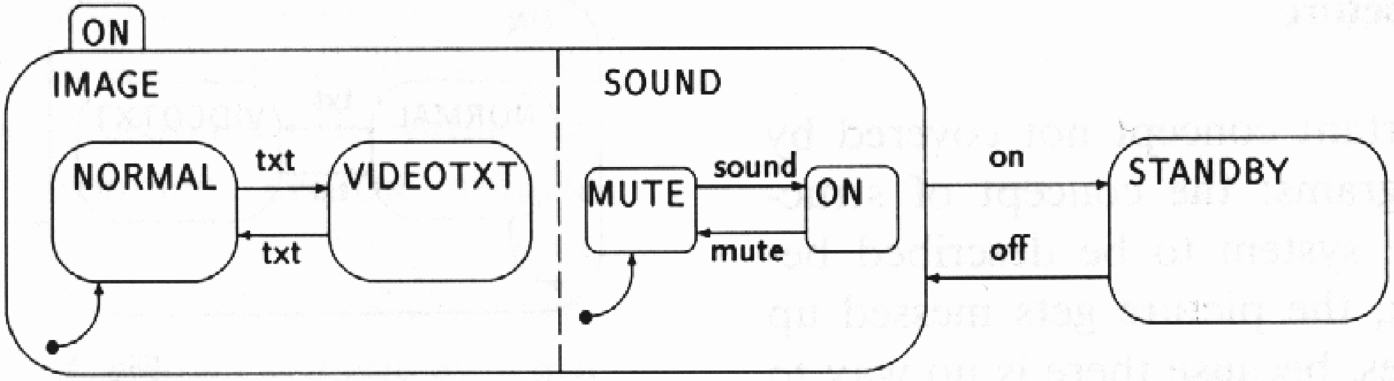


Fig. 4.

- Being in an AND-state means being in all of its immediate sub-states **at the same time.** This prevents the exponential blow-up familiar from composing FSMs in parallel.

Broadcast

In our case we split state **normal** in two orthogonal components **channel**, for selecting channels, and **sm** for switching to mute:

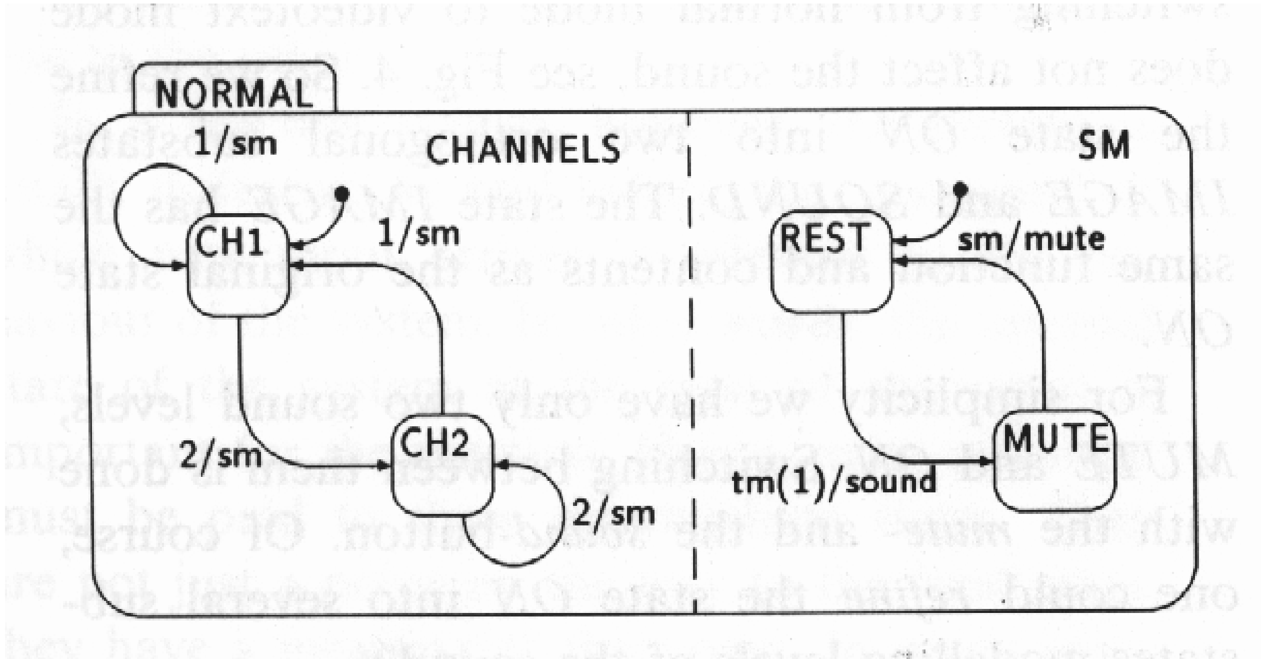


Fig. 5.

Compound Events

- In general one can label transitions by **compound events** s.a. $(\neg a \wedge b) \vee c, a \wedge b, c \vee d, \neg a,$ etc
- E.g., in:

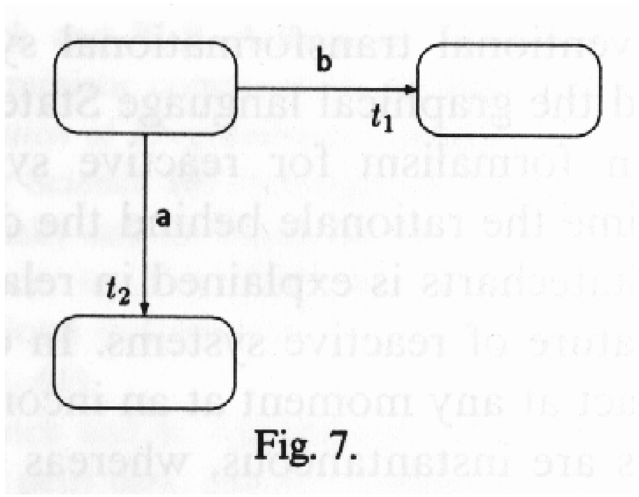


Fig. 7.

a can be replaced by $a \wedge \neg b$ to express priority of event b over event a.

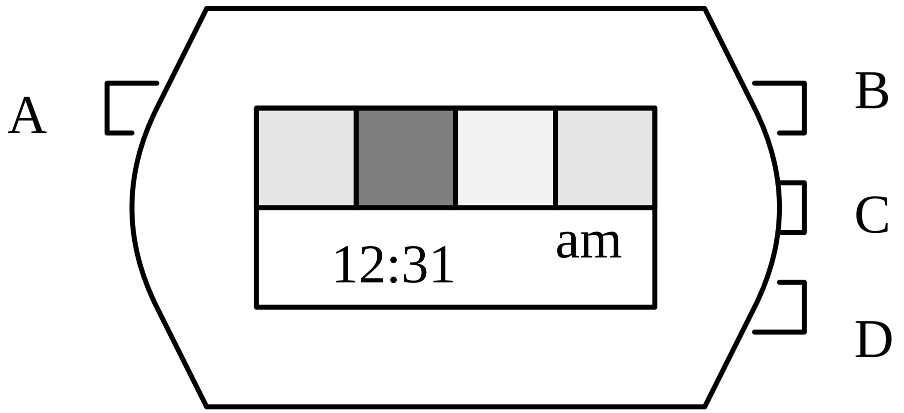
Statecharts

$$
\begin{aligned}
\text{Statecharts} \quad &= \quad \text{Mealy Machines} \\
&+ \quad \text{depth} \\
&+ \quad \text{orthogonality} \\
&+ \quad \text{broadcast} \\
&+ \quad \text{data}
\end{aligned}
$$

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:



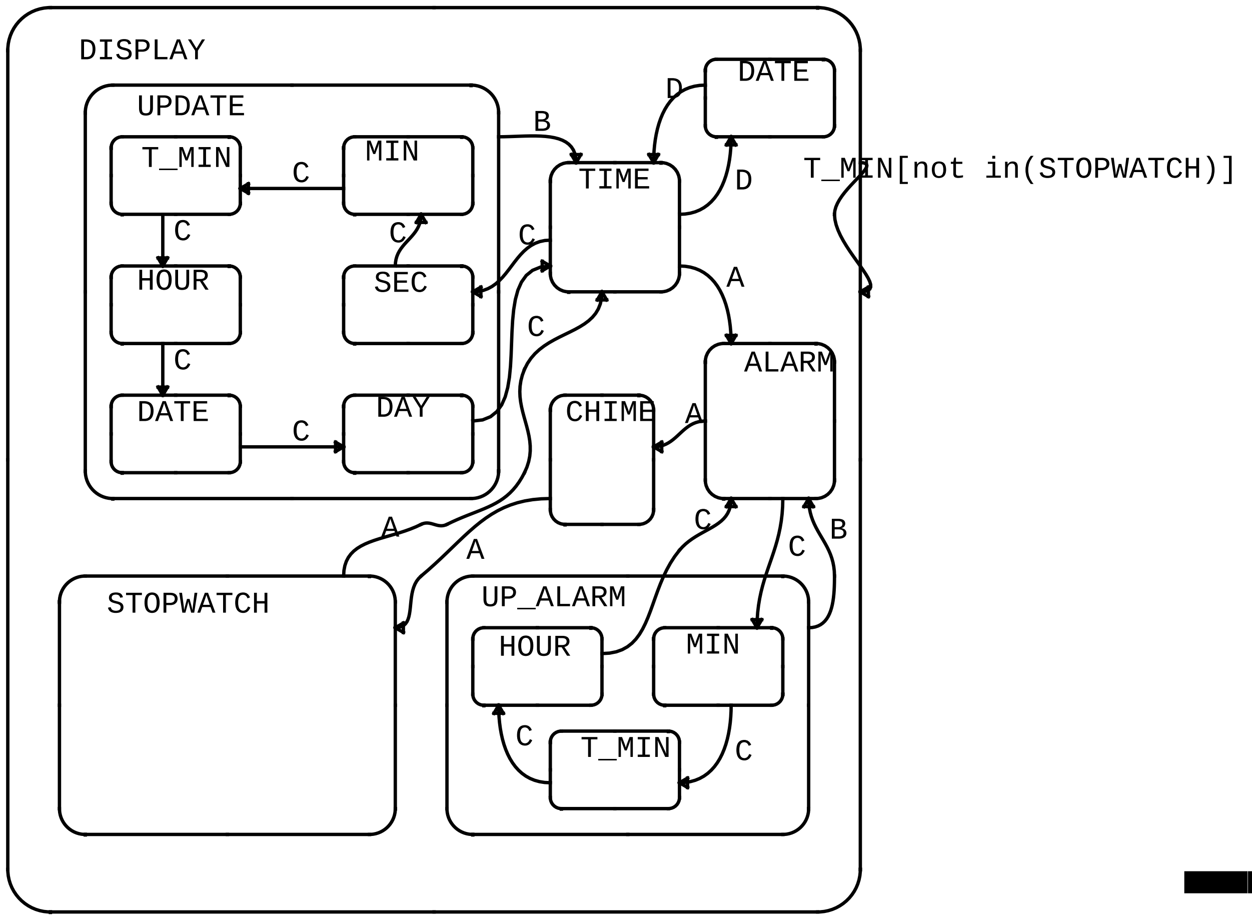
Events

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.

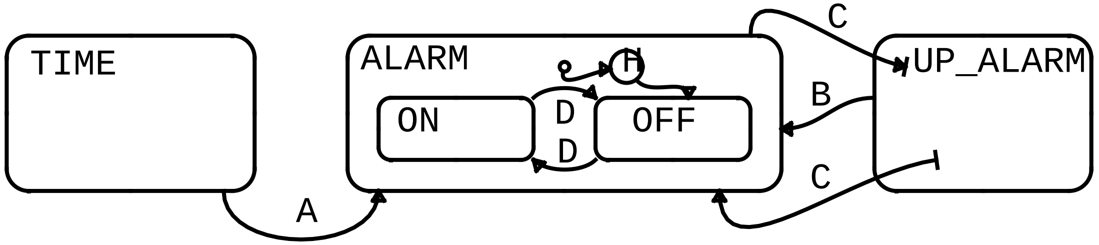
Display control

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

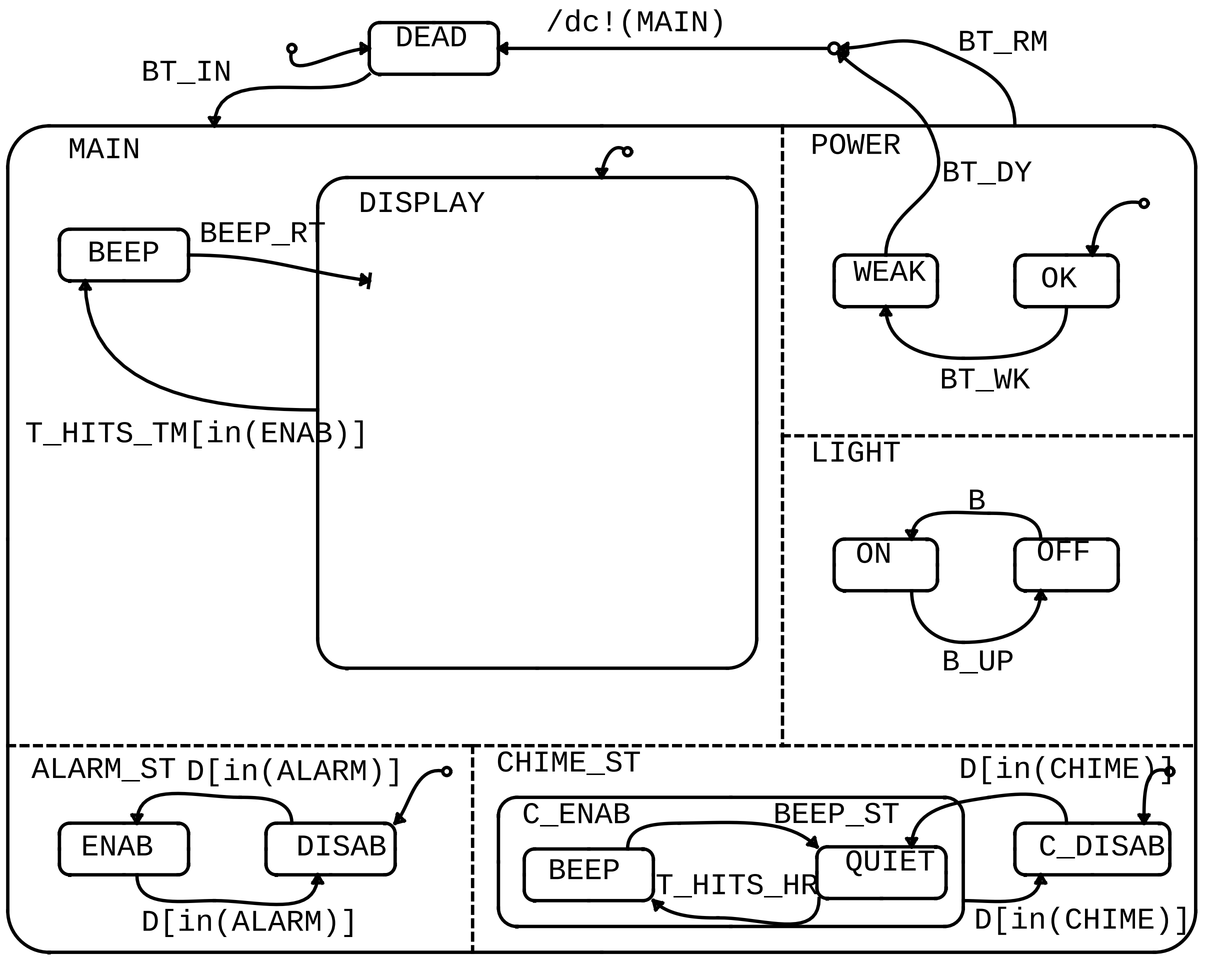


History of states

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.

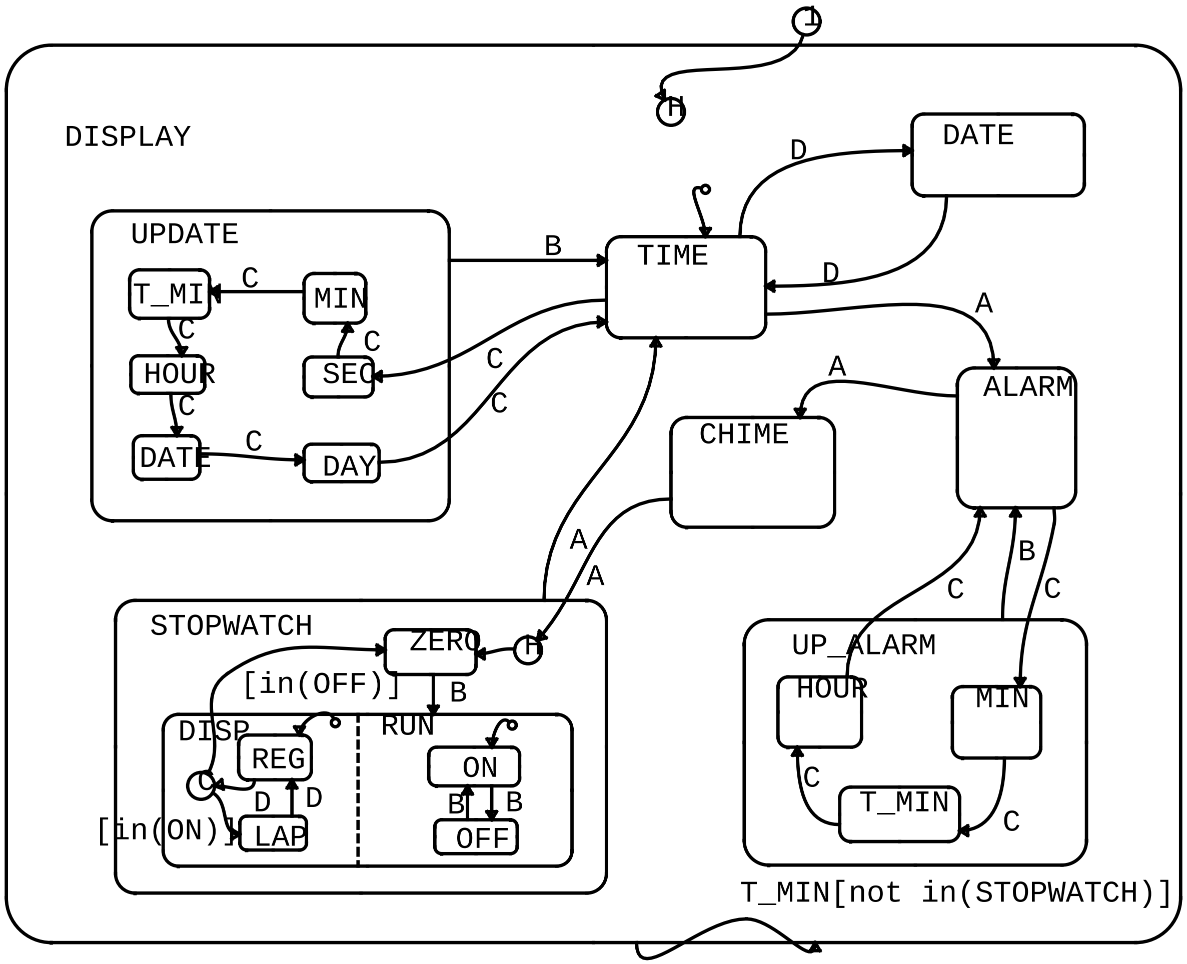


The Watch



DEAD
/dc!(MAIN)
BT_RM
BT_IN
MAIN
DISPLAY
POWER
BT_DY
BEEP
BEEP_RT
WEAK
OK
BT_WK
T_HITS_TM[in(ENAB)]
LIGHT
B
ON
OFF
B_UP
ALARM_ST
D[in(ALARM)]
ENAB
DISAB
D[in(ALARM)]
CHIME_ST
D[in(CHIME)]
C_ENAB
BEEP_ST
BEEP
QUIET
T_HITS_HR
C_DISAB
D[in(CHIME)]

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	$dc!(S)$.

Real-Time Systems

Real-Time Systems

- the environment can deliver data continuously, for example via temperature sensor.
- data can be delivered from different sources simultaneously and must therefore be processed in parallel.
- the time scale is fast by human standard (milli seconds instead of seconds),
- the system must react in time and accurately on input from the environment.

Characteristics

- the environment of a real-time systems contains often equipment that act as sensors of the systems (a typical example are optical scanners, that collect a continuous stream of relative unstructured data) and also equipment that can change the environment physical (like ‘actuators’ that continuously change the position of a tap).
- real-time systems must often simultaneously process input from different sources (in an industrial process control system, for example, the values of temperature, pressure and liquid concentration must be correlated simultaneously to execute adjustments of heating-apparatus and taps)

Characteristics

- the time scale of much real-time systems is fast considered to human standards (in an automated systems for the control of the speed of a car, for instance, the real speed must be measured many times a second to insure a comfortable ride; although this is fast for human standards, it is slow for real-time standards where the speed of operation is near the speed of the available implementation technology)
- real-time systems must react punctual and accurate on the input from the environment (a valid value on the wrong moment is often more worse that no value at all)
- the data in a real-time system has often a limited period of validity (think of for instance a green traffic light for an automated car control system)

Characteristics

- the observation of the environment by the real-time system is always behind so that there is an inconsistency between the perception and the real state of that environment
- a real-time system must be prepared for every possible situation that can happen in the environment and must be safe with respect to appearance of critical fault situations
- a real-time system must satisfy high reliability requirements: a guaranteed MTBF (Mean Time Between Failure) and MTTR (Mean Time To Repair), for instance, telephone exchanges are considered not to be out of order for more than 2 hours in 40 years.

Time in Statecharts

Time

- The elementary unit of observation in a reactive system is the **event**

Time

- The elementary unit of observation in a reactive system is the **event**
- The **environment** sends events to the system to **trigger computations**, the system reacts to the environment by sending, or **generating**, events.

Time

- The elementary unit of observation in a reactive system is the **event**
- The **environment** sends events to the system to **trigger computations**, the system reacts to the environment by sending, or **generating**, events.
- Events are also means of **communication** between parts of a system.

Time (cont'd)

- Because one wants to specify reactive systems at the **highest level of abstraction** in a discrete fashion, events are **discrete signals**, occurring at a **point in time.**
- Events have **no duration**; they are generated from one state to another. Hence, transitions have a discrete uninterruptable nature and **all time is spent in states.**

Reason

In a reason system new inputs may arrive at any moment. Therefore the current state it is in should be always clear. Since transitions have no duration, there are no “transient” periods in between states.
Therefore, the reaction on a possible input is always well defined.

> Of course this is an abstraction from reality. (At deep levels of electronic implementations, one encounters levels where discrete reasoning makes no sense anymore)

Statecharts is meant to be a high level specification language, where this abstraction can be maintained and is appropriate.

Reaction Time

- We know that transitions have no duration, but **when** do they take place, relative to the trigger? And:

  HOW LONG DOES IT TAKE THE SYSTEM TO COMPUTE A REACTION UPON AN EXTERNAL EVENT?

- For transformational systems this is easy — the only important distinction is between finite and infinite values (corresponding to a final state or no final state)
- For reactive systems this is not enough:
  We have to know when an output occurs relative to the events in the input sequence (see Brock-Ackermann paradox) $\Longrightarrow$
  One has to determine **the reaction time of a sequence.**

What's the reaction time of a reactive systems upon an external event in the high level Specification Language Statecharts?

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.

Solution

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.**

Problems Disappear

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

This choice, that the reaction time between a trigger and its event is zero, is called **Berry’s synchrony hypothesis.**
Is this implementable? No, a real computation takes time.
But in actual implementation this means:

The reaction comes before the next input arrives,

or, so to say,

Reactions are not infinitely fast but fast enough.

Example

See the following figure:

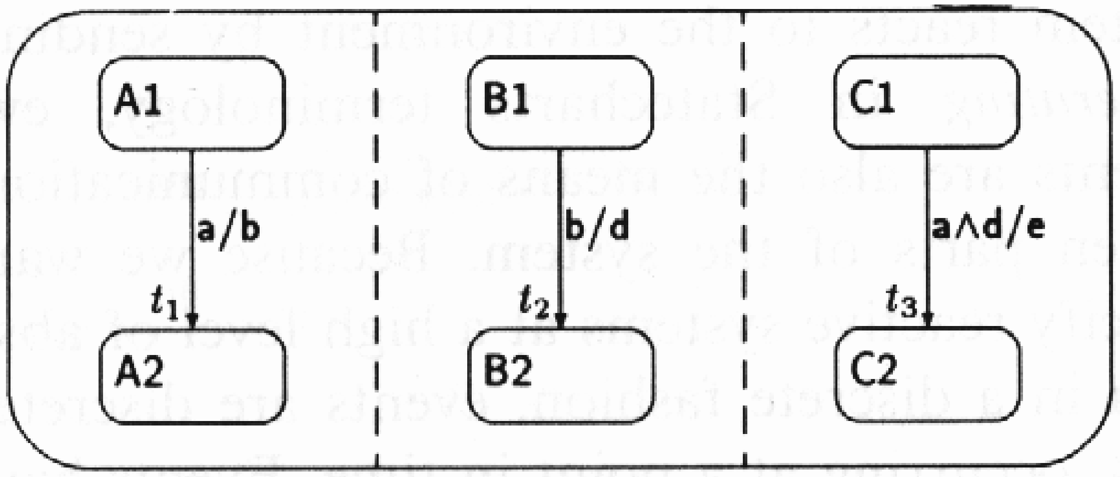


Fig. 6.

A consequence is that transition t_3 is taken!!

Negations and paradoxes

- Idea of immediate reaction works fine as long as transitions only triggered by primitive events, or or conjunctions and disjunctions of them.
- However, one also needs **negations of events** to trigger a transition. E.g.: to specify priority:

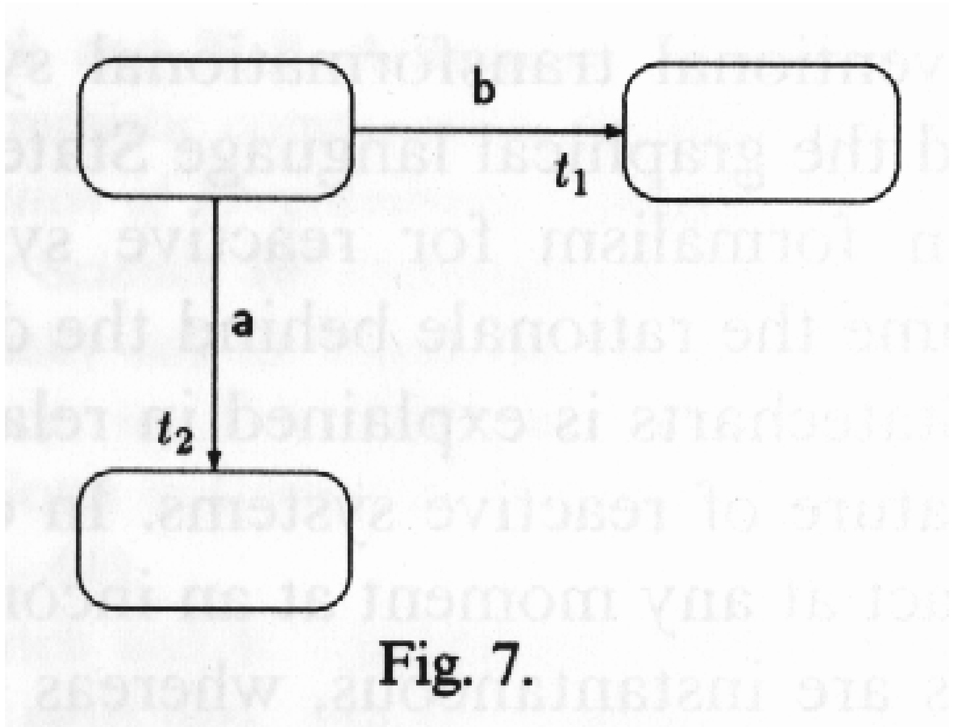


Fig. 7.

Problem

What semantics to give to Statecharts in the next figure?

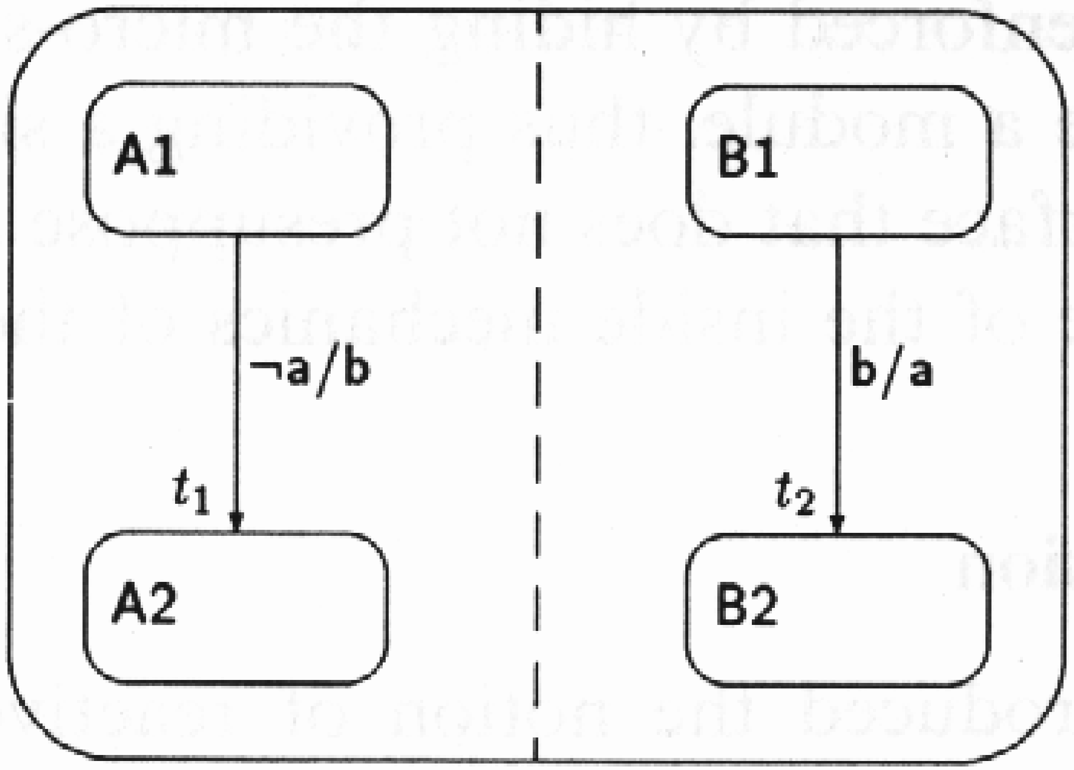


Fig. 8.

Grandfather Paradoxon

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.
But that means that event **b** is not generated, and hence event **a** is not generated, so transition t_1 should be taken, etc.
$\Longrightarrow$ PARADOXON!
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

Solution

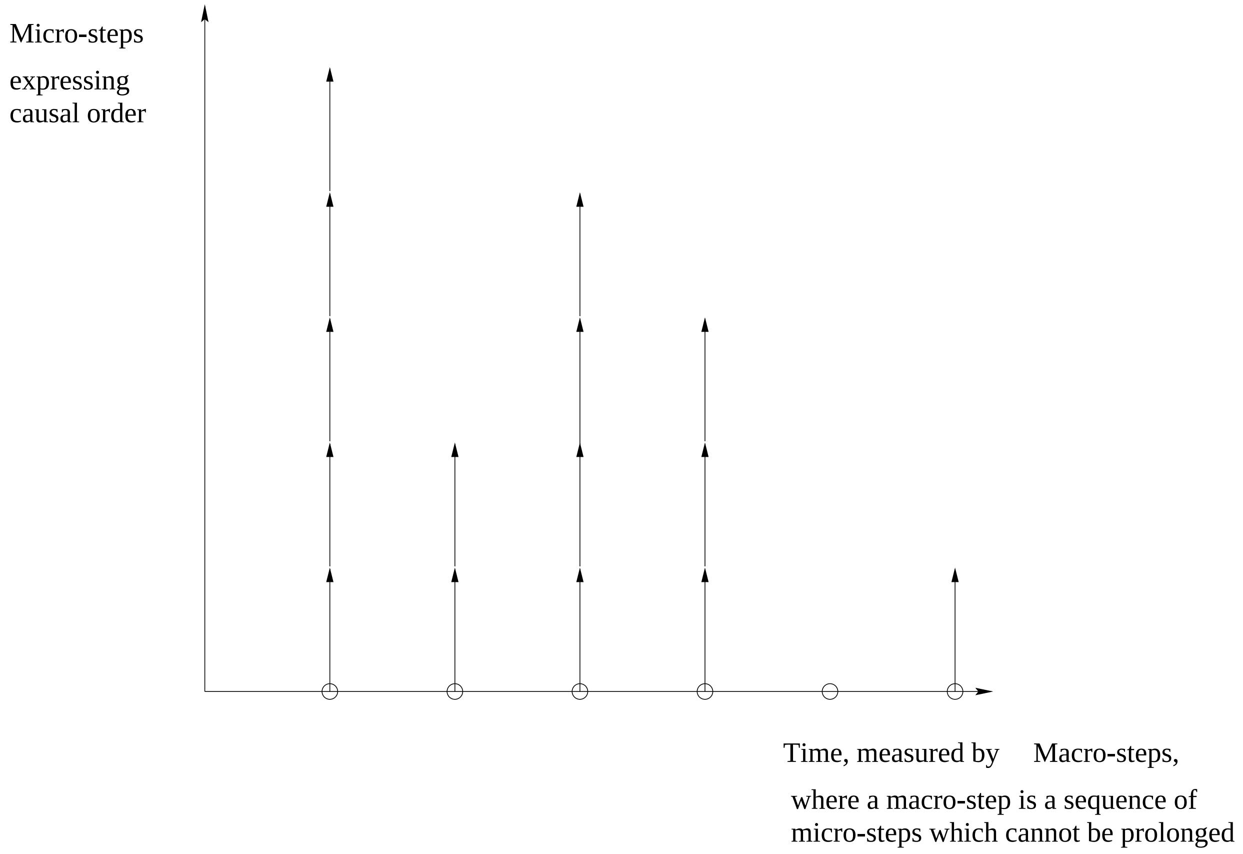
Introduce two levels of time

- **Macro steps**, for counting time, (these are observable) time steps, and
- **Micro steps**, which describe the causal chain within reactions.
  Every macro-step is then divided in an arbitrary but finite number of micro-steps.

This sequence of micro-steps has only an operational meaning.

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

The New Semantics (cont’d)

- Macro-steps are observable steps $\Longrightarrow^{O}_{I}$
- 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 New Semantics

The problem with macro-steps is that they lead to a **globally inconsistent** semantics.

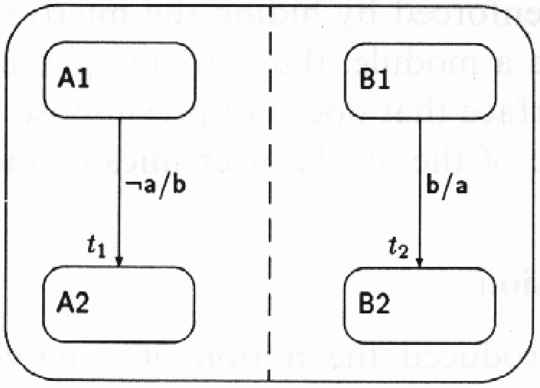


Fig. 8.

$$S_1 \overset{b}{\underset{\emptyset}{\Longrightarrow}} S_2 \overset{a}{\underset{b}{\Longrightarrow}} S_3$$

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

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.:

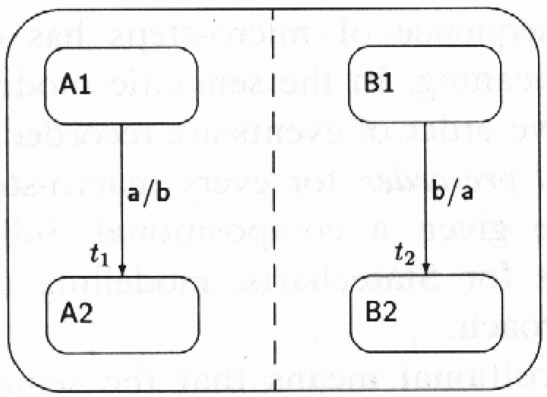


Fig. 9.

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:

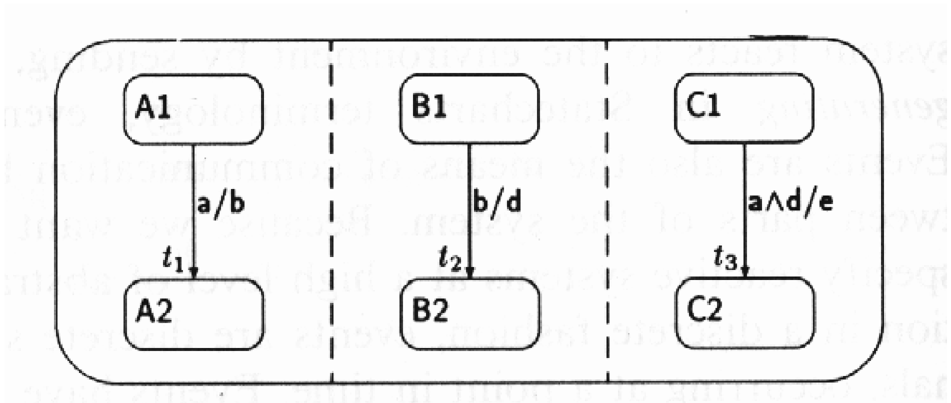


Fig. 6.

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 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 $b \wedge \neg d$ transition is not taken.

Semantics C (cont'd)

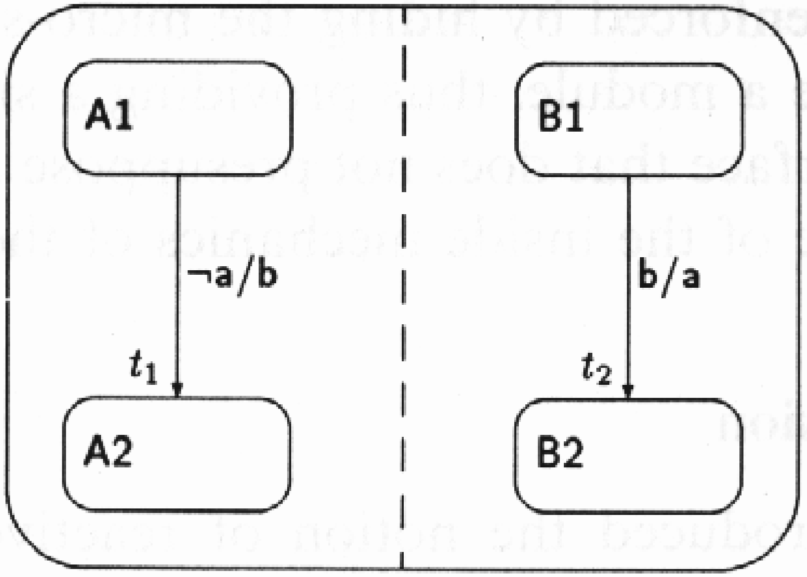


Fig. 8.

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

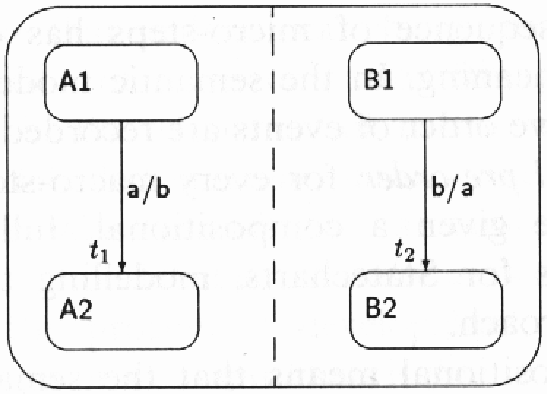


Fig. 9.

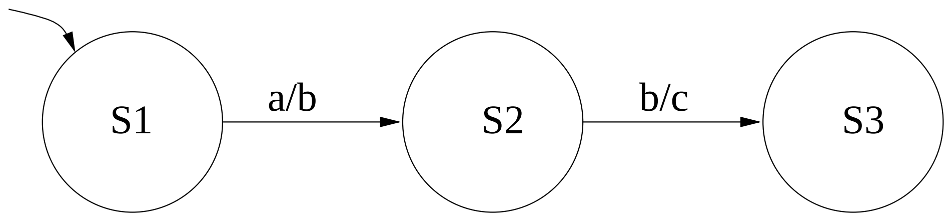
to be taken: reactions may trigger themselves. I.e., semantics D is not causal.
Note: In semantics D, the external world does not generate an a event!
Conclusion: This example should be rejected!

Semantics E

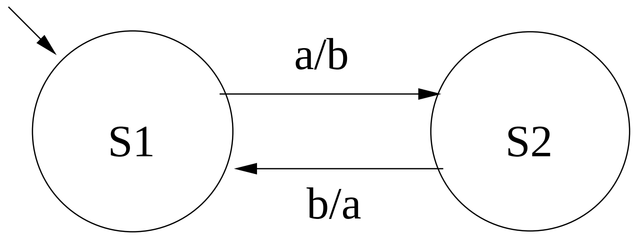
Events are generated at the next step, but no input from the environment is possible **before** the reaction of the system has completely died out.

> 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 Statecharts, this example leads to an infinite loop (the so-called: "go repeat" mode): try it out yourself!

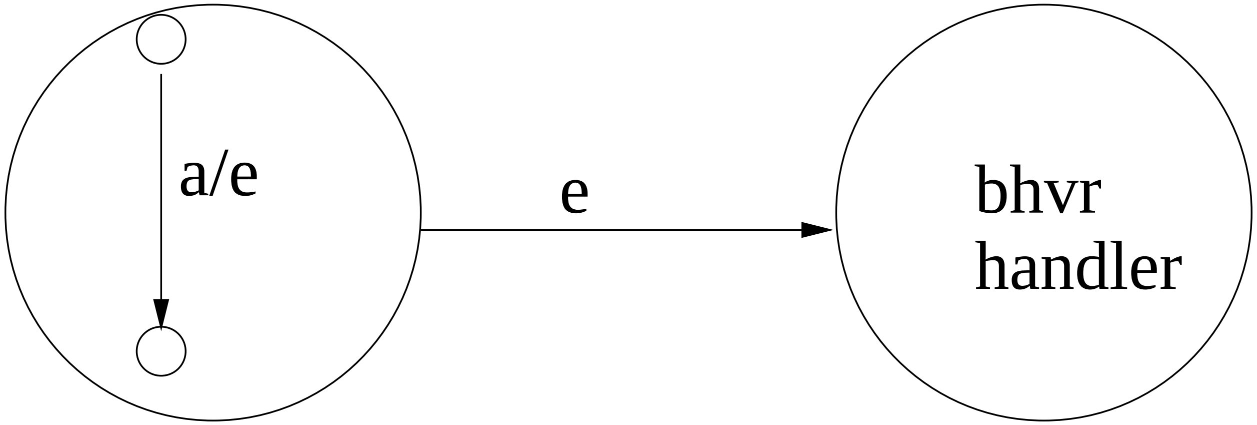
Situation

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:

$$\begin{tikzcd}
A \arrow[r, "\text{responsiveness}"] \arrow[d, "\text{responsiveness}"] & B \arrow[r] & C \arrow[r, "\text{modularity}"] & D \arrow[l, bend left=90, "\text{causality}"] \\
E \arrow[u, bend left=90, "\text{modularity}"] & & &
\end{tikzcd}$$

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;

Statecharts Status

- a list of states in which the system currently resides;
- a list of activities that are currently active;

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;

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;

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;

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;

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.