Modeling Reactive Systems with IAR visualSTATE Statecharts

Andrzej Wąsowski
(wasowski@itu.dk)

http://www.mini.pw.edu.pl/~wasowski/

27 November 2003

Outline

- Introductory Remarks
- Statecharts Syntactic and Semantic Basics.
- Modeling the Wrist Watch

[short break anticipated]

- Statecharts Odds&Ends.
- Statecharts as Formal Development Method
- Concluding Remarks

[short break anticipated]

- Project possibilities.
- Exercise (the air conditioner example)

Outline

- **Introductory Remarks**
- Statecharts Syntactic and Semantic Basics.
- Modeling the Wrist Watch

[short break anticipated]

- Statecharts Odds&Ends.
- Statecharts as Formal Development Method
- Concluding Remarks

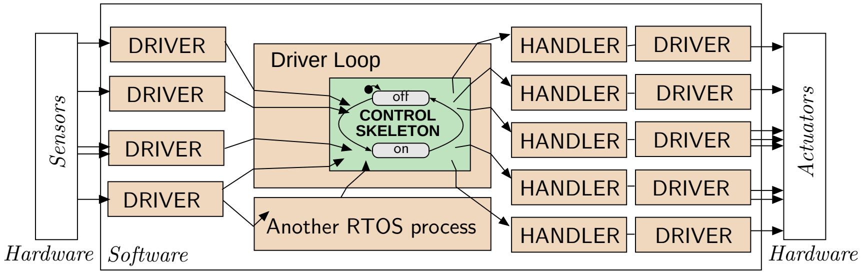
[short break anticipated]

- Project possibilities.
- Exercise (the air conditioner example)

Reactive Systems

- Transformational programs compute a result for the given input parameters (eg. compilers)
- Reactive programs (Pnueli,Harel,1985): constantly listen to incoming input and produce outputs in reaction to those (eg. embedded systems, user interfaces)
- Reactive system receives external stimuli from sensors and affects environment using actuators. Sensor may be a button, actuator may be a display.
- Discrete Reactive Systems ignore the continuity of time and work in lock steps.
- Synchrony Hypothesis is an assumption that reaction of a discrete system takes no time (outputs are available immediately). In practice it means that systems reaction is faster than frequency of environment events.

Architecture of Reactive System



- Control algorithm (skeleton).
- Brown parts are small and relatively easy.
- Sometimes multiple processes are avoided in favour of the loop.
- In some cases it is even possible to give up the RTOS entirely.

An Abridged History of Statecharts

- Statecharts: a visual modeling language mostly focused on discrete time systems.
- Proposed by David Harel in 1984 and implemented in STATEMATE.
- Accepted as one of the notations in UML (1996).
- BeoLogic uses a variant of statecharts as a specification language in their modeling tool visualSTATE.
- Presently visualSTATE is maintained by IAR Systems (Danish division).
- A Multitude of tools supports statecharts: visualSTATE, Rhapsody, ArgoUML, Rational Rose...

CREDITS: The vast part of this lecture is based on the example taken from the classic paper on statecharts:

"Statecharts: A Visual Formalism For Complex Systems"
David Harel, 1987

Outline

- Introductory Remarks
- **Statecharts: Syntactic and Semantic Basics**
- Modeling the Wrist Watch

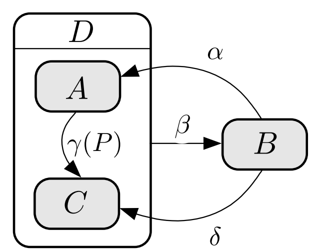
[short break anticipated]

- Statecharts Odds&Ends.
- Statecharts as Formal Development Method
- Concluding Remarks

[short break anticipated]

- Project possibilities.
- Exercise (the air conditioner example)

Syntactic Trivia



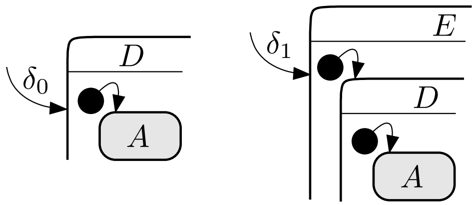
D superstate
A, B and C basic states
A,C: a xor-decomposition of D
β leaves all substates of D

Invariants:
1. $active(B)$ xor $active(D)$
2. $active(D) \equiv active(A)$ xor $active(C)$

- An extension of finite state machines and transition diagrams.
  - FSM: a single state active and a single transition taken at a time.
  - statecharts: multiple active states and concurrent transitions.
- Labels: Events and outputs possibly parameterized (value passing).
- Transition relation represented by arrows.
- Hierarchy relation represented by nesting of states.

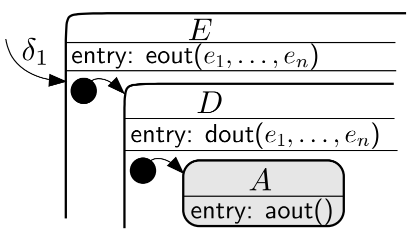
Harel87:Fig.2/p.234

Initial States



- Initial states and initial markers
  - D is the initial state of E
  - A is the initial state of D
  - ● is the initial marker or *connector*(in UML).
- δ_0,δ_1 enter initial configuration of D and E.

Entry Actions

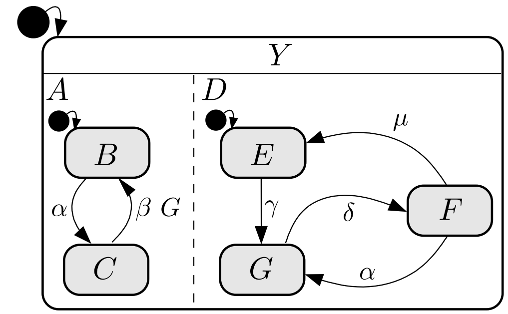


Transition's δ_1 own actions are executed first, followed by a sequence

eout(); dout(...); aout();

Outputs are provided as C functions available in seperate file. Outputs may be parameterized. See later on expressions.

Orthogonality



- All transitions are fired by events.
- Note one transition guarded on active substate of concurrent state.
- States A and D are called regions (or or-states).
- State Y is called a concurrent state (or and-state).
- Note the significant gain in succinctness (wrt to product automaton).

Harel87:Fig.19/p.242

Outline

- Introductory Remarks
- Statecharts: Syntactic and Semantic Basics
- **Modeling the Wrist Watch**

[short break anticipated]

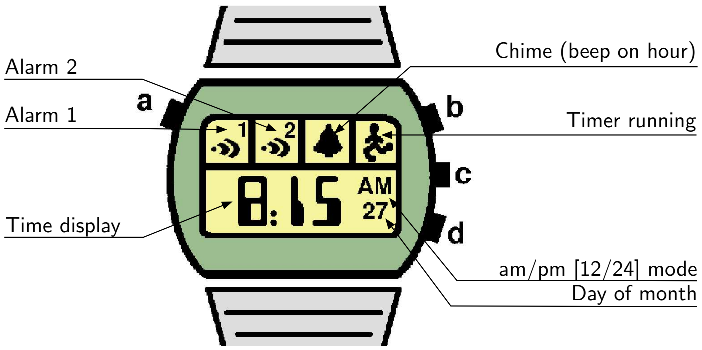
- Statecharts Odds&Ends.
- Statecharts as Formal Development Method
- Concluding Remarks

[short break anticipated]

- Project possibilities.
- Exercise (the air conditioner example)

The Running Example

A model of a wrist watch.

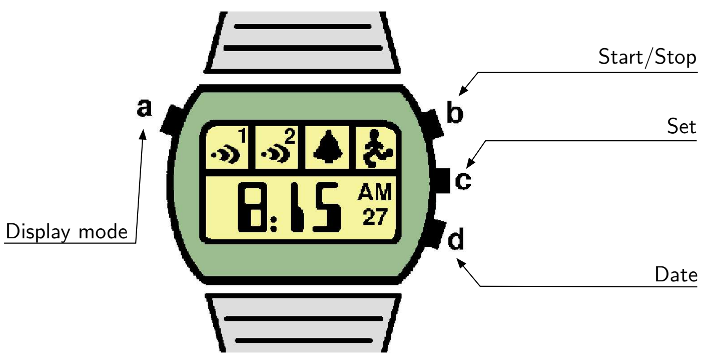


Display and beeper are the outputs of the watch device.

Harel87:Fig.7/p.236

The Running Example (II)

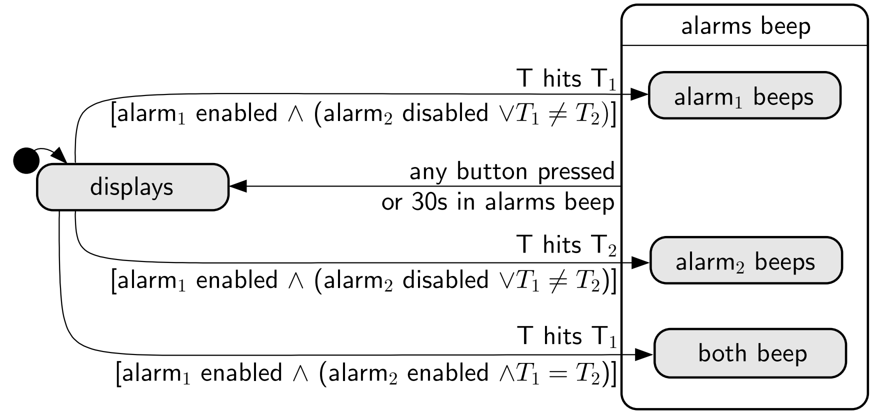
Buttons, events and behaviours.



Our goal is to assign exact meaning to button (inputs).

Harel87:Fig.7/p.236

Alarm Activation



Unfortunately not formal enough for implementation and verification.

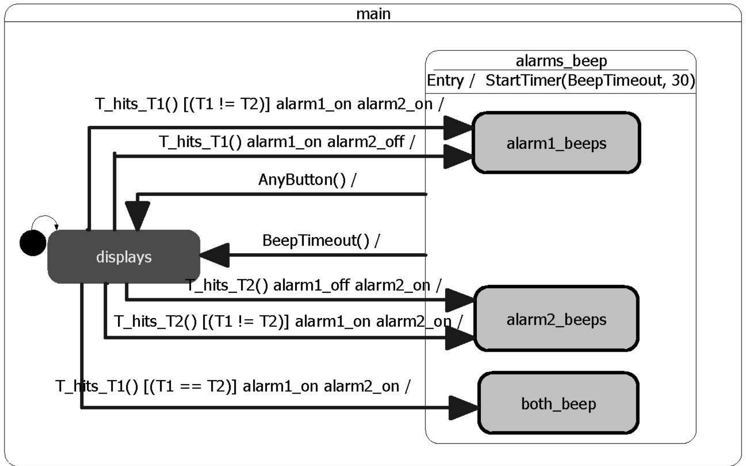
Harel87:Fig.8/p.237

Alarm Activation (II)

What shall we refine to move towards a formal model in visualSTATE?

- Undefined clocks (variables)
  - internal `int T1;` // second of the day to activate the alarm
- Imprecise events:
  - button events: a() b() c() d()
  - Any button pressed → AnyButton() = a ∨ b ∨ c ∨ d
  - 30s in alarms-beep → BeepTimeout()
  - T hits T$_1$ → external event T_hits_T1()
- Actions to set up timers for time related events:
  - T_hits_T1() is fired by an external RTOS process setup in initialization and controlled whenever setting are changed.
  - Set up BeepTimeout() timer whenever *alarms beep* is entered
- Missing states for enabledness of alarms (independent component)
- Eliminate disjunctions from guards (not allowed in visualSTATE)

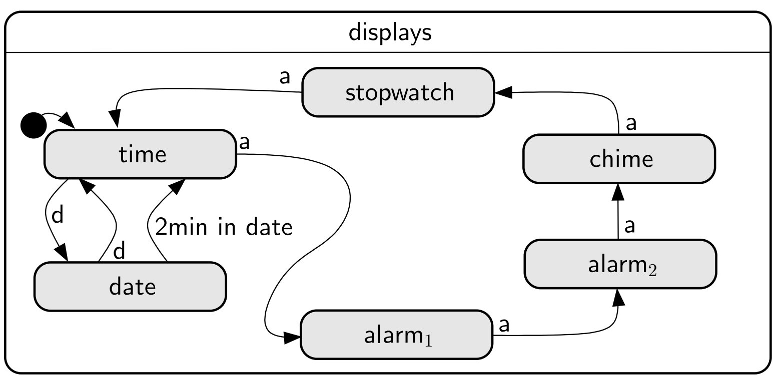
Alarm Activation (III)



Main fragment of visualSTATE implementation (tool printout).

Harel87:Fig.8/p.237

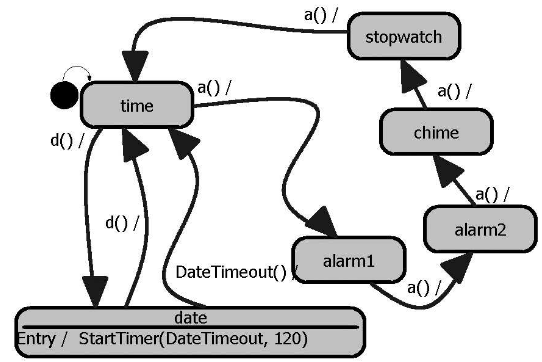
Refining Displays



Again timer event needs to be expressed with entry action.

Harel87:Fig.9/p.237

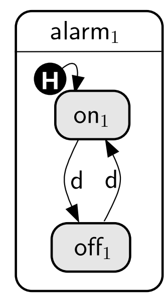
Refining Displays (II)



Note the *StartTimer(DateTimeout,120)* action, generating *DateTimout* after 120 units.

Harel87:Fig.9/p.237

History States



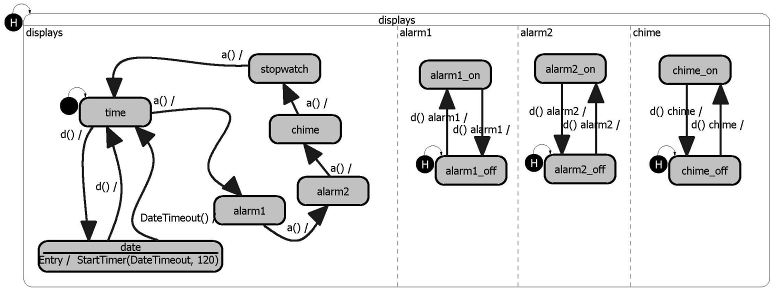
State $alarm_1$ will retain the information about its active substates across activations.

States $alarm_2$ and $chime$ are analogous.

Harel87:Fig.10b/p.239

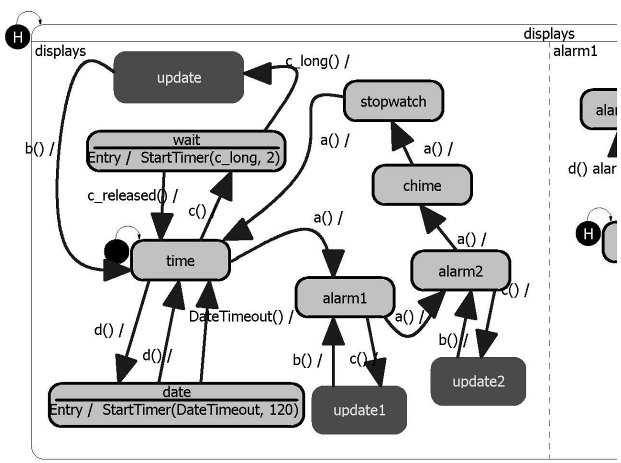
Alarms and Chime Setup

Alarms and chime can be activated and deactivated if display is in the proper mode.



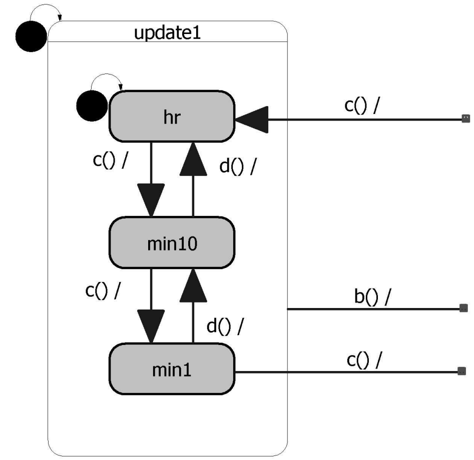
[Note a slight design change with respect to the original paper, to avoid relying on semantic subtleties.]

Update Modes for Time



Harel87:Fig.13/p.240

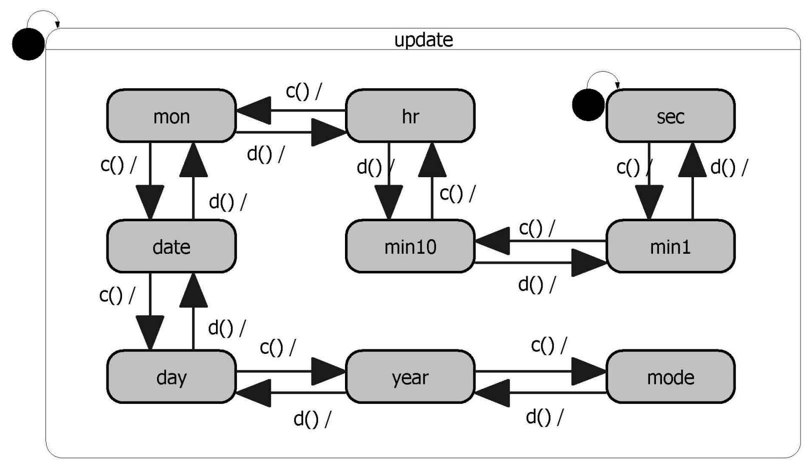
Update Modes for Time (II)



Note the transitions going out to and coming from upper level.

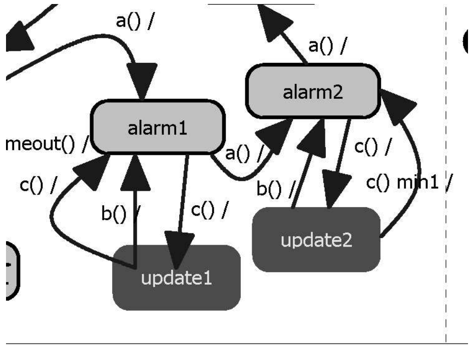
Harel87:Fig.15/p.241

Update Modes for Time (III)



Harel87:Fig.14/p.240

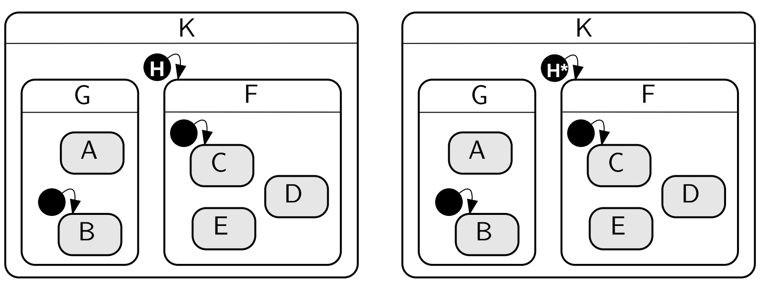
Update Modes for Time (IV)



State *update1* uses cross-level transitions, *update2* does not.

Deep History

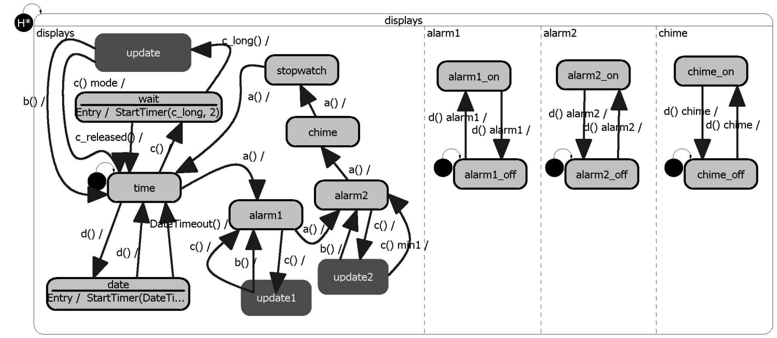
History (H) affects only the level at which it is placed.



Deep history (H*) affects the state of entire subhierarchy.

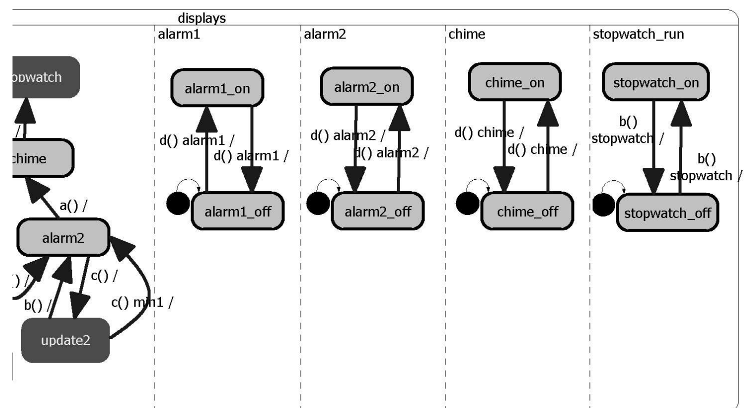
Harel87:Fig.11/p.239

Displays are Deep History



Whenever alarm starts to beep and is cancelled, control returns to the previous configuration.

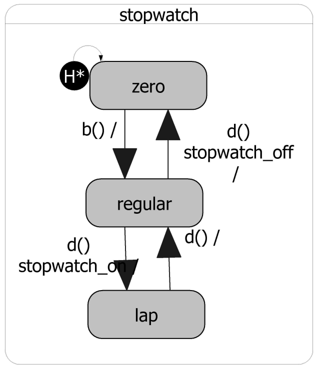
Stopwatch



Stopwatch is runing (or not) independently of the operation of display controls. It may only be started or stopped when in *stopwatch* state (note the guard on transitions in *stopwatch_run*).

Harel87:Fig.25/p.246

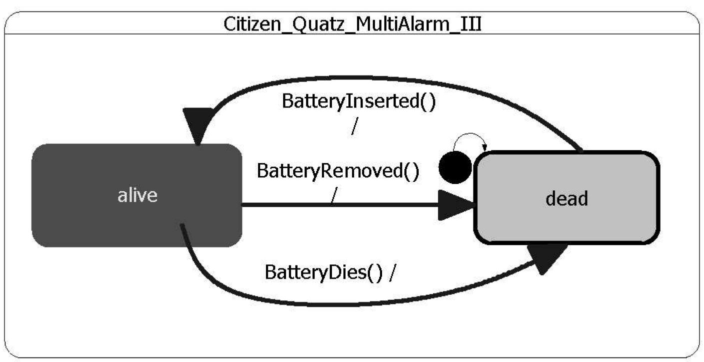
Stopwatch (II)



- Event b starts the stopwatch
- Event d toggles between the lap and regular mode.
- Lap mode only makes sense while the stopwatch is running.
- Resetting only works when stopwatch is off.

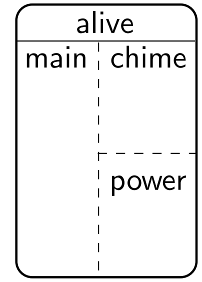
Harel87:Fig.25/p.246

Top Level



Harel87:Fig.26/p.246

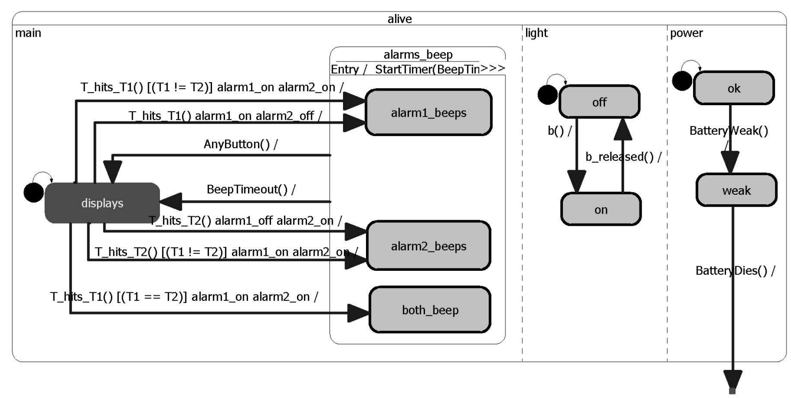
Zooming into Alive



- All we have done so far is in *main*.
- We have less regions than original paper due to some simplifications in the model.
- Fortunately remaining parts are small and easy.

Harel87:Fig.27/p.247

Zooming into Alive (II)



- Note that b always activates light (despite all its other features).
- Region *power* has a transition leaving the level

Harel87:Fig.28/p.248

Outline

- Introductory Remarks
- Statecharts: Syntactic and Semantic Basics
- Modeling the Wrist Watch

[short break anticipated]

- **Statecharts Odds&Ends**
- Statecharts as Formal Development Method
- Concluding Remarks

[short break anticipated]

- Project possibilities.
- Exercise (the air conditioner example)

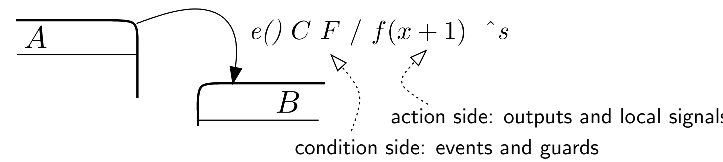
Signal Communication

- Signals are means of asynchronous communication.
- Signals are similar to events but are not visible for environments.
- Signals may be triggered in any action (on transitons, on entry and exit to states).
- Signals are placed in the condition of a transition in the same way as events.
- In visualSTATE signals are global (i.e. directed to the entire system).
- System with signals works in two stage steps:
  - Microstep: apply one event or signal to the model put all signals produced in a signal queue
  - Macrostep pop a signal from a queue and run a microstep. Iterate until the queue is empty.
- Only macrosteps are observable from external perspective.

Transitions Revisited

Let us summarize the syntax of transitions.

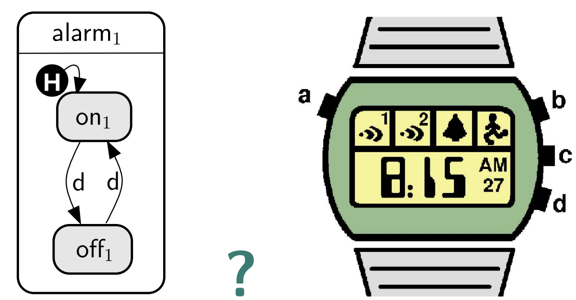
- Transitions are labeled with conditions and actions:



- Condition part: event/signal/event-group, positive/negative condition, guard (C expression)
- Action part: function calls, assignments to variables and triggering signals

Abstract vs Physical States

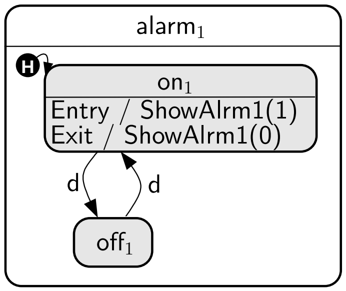
- Our model has been constructed in terms of states and transitions.
- These states were <u>abstract</u>. For instance we have not specified any relation between the state $alarm_1.on_1$ and the fact that the alarm indicator on display is visible.



- Such abstract models are useful for analysis of systems but not for development of real programs!

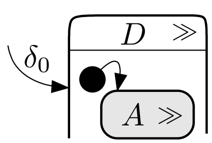
Abstract vs Physical States (II)

- Abstract state is the state in the model.
- Physical state is the state of the device or environment.
- In models used for synthesis of systems abstract states need to be related to physical states.
- One typical way to achieve this is by use of entry and exit actions.



visualSTATE Odds&Ends

- Model constants
- Model variables with restricted domains
- External vs Internal variables
- Internal rules
- Do reactions
- Parameterized events may pass the value of sensor readout.
- Entry/exit actions can be hidden (prevents cluttering of diagrams)



Outline

- Introductory Remarks
- Statecharts: Syntactic and Semantic Basics
- Modeling the Wrist Watch

[short break anticipated]

- Statecharts Odds&Ends
- **Statecharts as Formal Development Method**
- Concluding Remarks

[short break anticipated]

- Project possibilities.
- Exercise (the air conditioner example)

Formal Development

- Abstract modeling.
- Automatic model verification.
- Tool-supported debugging (simulation and monitored execution).
- Tool-supported program synthesis (code generation).
- Systematic test of implementation [in progress].

visualSTATE model checker

Model checker automatically verifies if following hold in the model:

- No unused components [states, variables]
- No unreachable guards. It must be possible to enable all of the guards in the system. This means that there must exist a reachable state for each guard g that enables this guard. Unreachable guards mean dead code (dead transitions).
- No conflicting transitions.
- No deadlocks.
- No illegal operations. Arithmetic operations should be checked for overflow and illegal operations such as division by zero.
- No divergent behavior. If the signal queue is used then the macrostep should always be finite.
- No overflow of the signal queue.

visualSTATE Code Generator

- visualSTATE contains a translator of models into C programs.
- Program implementing the control algorithm is generated automatically.
- Programmer should provide
  - Code for external C functions, drivers and handlers
  - Main loop feeding external events to the system
  - and a RTOS (if needed).
- The generated code has very modest memory requirements. An order of 50 words of RAM is sufficient for execution. ROM usage for a model of 200 transitions (rather complex) is in order of 10kb.

Outline

Reactive Programming Agora

- Synchronous Languages — a family of languages based on the strong synchrony hypothesis, namely that outputs are produced instantenously with inputs. They present a somewhat unsual programming style, but exhibit clean and compact mathematical semantics and are easier to model check. Esterel is the main imperative dialect. Lustre and Signal follow the functional programming style, while Argos is the visual incarnation of the semantics, rather similar to stateacharts.
- Timed Triggered Languages (eg. Giotto) — based on periodic tasks and data-flow like networks.
- Timed Automata — specifications of systems with continuous time.
- Hybrid Automata — specifications of systems with continuous control.
- So far it seems that statecharts are the only language becoming popular in mainstream development tools.

Thank you for Your attention.