Design of Embedded Systems: Models, Validation and Synthesis (EE 249)—Lecture 4a

Prof. Dr. Reinhard von Hanxleden

Christian-Albrechts Universität Kiel
Department of Computer Science
Real-Time Systems and Embedded Systems Group

13 September 2007

Last compiled: 4th October 2007, 18:45 hrs

Statecharts

Overview

Finite State Machines
- Finite Automata
- Moore Machines
- Mealy Machines

Statecharts

Stateflow

SyncCharts (Safe State Machines)

Finite Automata

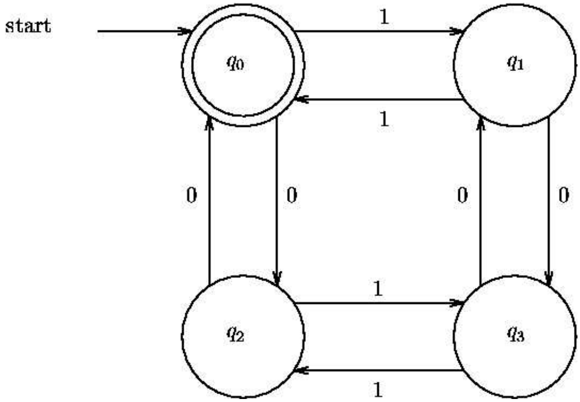
Formally a *finite automaton* is defined as a five tuple $(Q, \Sigma, \delta, q_0, F)$ where

Q	is a finite set of states,
Σ	is the input alphabet,
$q_0 \in Q$	is the begin state (initial state),
$F \subseteq Q$	is the set of final states,
$\delta : Q \times \Sigma \to Q$	is the transition function.

The transition function gives for every state q and every input symbol a the new state $\delta(q, a)$ that arises as reaction on the execution of a in state q.

Thanks to Willem-Paul de Roever and Kai Baukus for providing part of the following material

State Diagram



For each state, the possible reactions to input that arrives in that state is specified by a transition to other states.

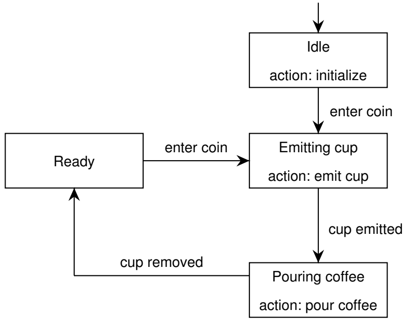
Extensions

- Restriction of these kind of automata as defined above: they have an input alphabet but not an output alphabet.
- There are two ways to extend the above model with output:
  1. Output can be associated with a state (a so called Moore machine)
  2. or with a transition (a so called Mealy machine).
- A Moore machine is a 6-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ where Q, Σ, δ, q_0 are the same as in the definition of the finite automaton,

Δ	is the output alphabet and
$\lambda : Q \rightarrow \Delta$	is the output function.

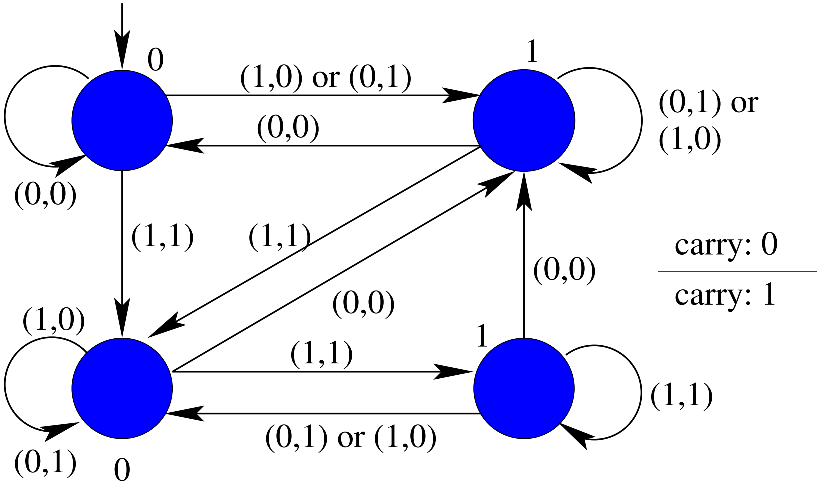
- A Mealy machine is also a 6-tuple $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$ but now λ is a function from $Q \times \Sigma$ to Δ.

Example: Moore Machine

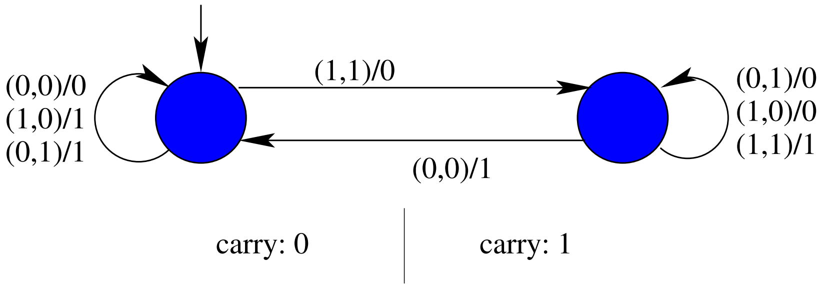


- Moore machine: output λ is associated with every state
- Mealy machine: $\lambda(q, a)$ gives output associated with the transition of state q on input a.

Serial Addition: Moore Machine



Serial Addition: Mealy Machine

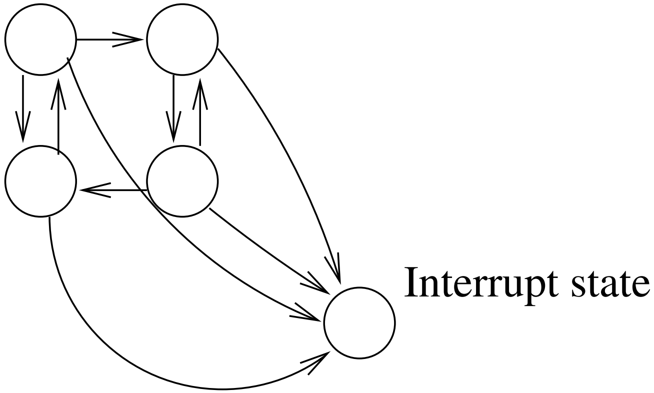


Disadvantages

- **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.

Overview

Finite State Machines

Statecharts
- Hierarchy
- Orthogonality
- Broadcast
- Time in Statecharts

Stateflow

SyncCharts (Safe State Machines)

Statecharts

- We need a formalism for the hierarchical development and refinement of Mealy machines.
- This is provided by Statecharts, invented by David Harel (1987)
- Statecharts display *hierarchy* and *structure*, and enable hierarchical development
- In the following, will illustrate this with the example of a television set with remote control
- The Statecharts used in this lecture follow the syntax of the original Statecharts, as invented by Harel, and as supported by the STATEMATE toolset

First Concept: Hierarchy

- Hierarchy or depth in states, and interrupts.
- This is achieved by drawing states as boxes that contain other boxes as sub-states.
- The television set can be in two states: **ON** and **STANDBY**. Switching between them is done by pushing the **on** and **off** buttons, generating the **on** and **off** events:

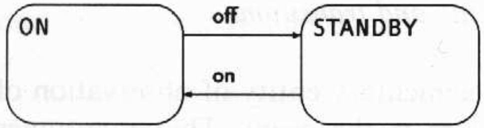


Fig. 2.

Zooming into **ON**

In state **ON**, the tv set can be in two sub-states: **NORMAL** and **VIDEOTEXT**:

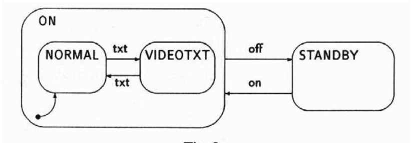


Fig. 3.

The $\longrightarrow$ arrow leading to **NORMAL** specifies which sub-state should be entered when the higher level state **ON** is entered, namely **NORMAL**; this state is also called the initial state (within **ON**)

Note: To be complete, one of the states **ON** and **OFF** would also need to be labeled as initial state (at top-level).

Superstates

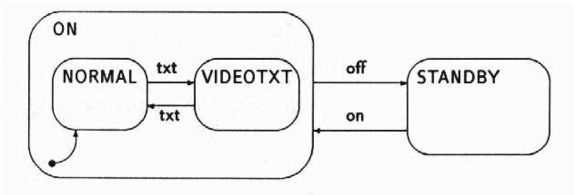


Fig. 3.

When in **ON**, and an event **off** is generated:

1. State **ON** (incl. all its sub-states) is left—so this acts like an interrupt
2. Control switches to state **STANDBY**.

In this way interrupts are handled without cluttering the picture with arrows

Second Concept: Orthogonality

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

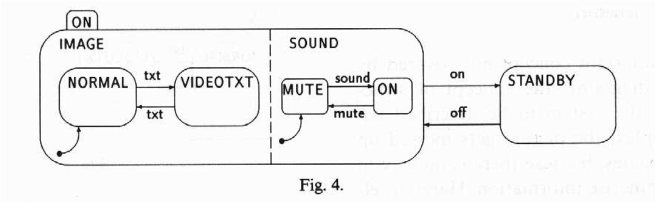


Fig. 4.

- 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.**

Third Concept: Broadcast

In our case we split state **NORMAL** in two orthogonal components **CHANNEL**, for selecting channels, and **SM** for switching to mute:

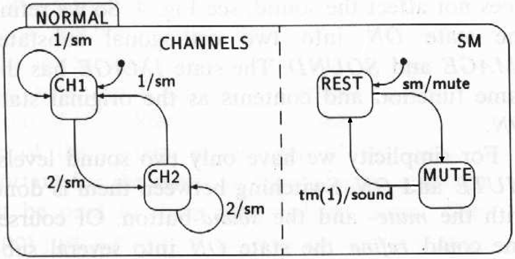


Fig. 5.

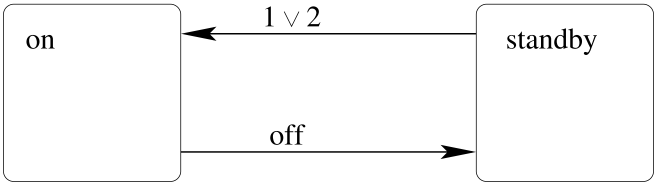
Note special time-out event **tm(1)**

Actions and Transitions

- Orthogonal components can communicate by generating events which are broadcast
- This can be done in a time-dependent manner: introducing the generation of events $e/a_1; \dots; a_n$ and time-out events tm(1), ...
- In general, the label of a transition consists of two parts:
  1. Trigger, determininig if and when a transition will be taken
  2. Action, performed when a transition is taken.
- This action is the generation of a set of events.

Fourth concept: Compound events

When in state **STANDBY**, dependent on whether one presses button 1 or 2, one makes sure to switch to states **CH1** or **CH2** in **ON**. This is indicated as follows:



Transition Labels

- In general, one can label transitions by compound events such as $(\neg a \wedge b) \vee c$, $a \wedge b$, $c \vee d$, $\neg a$, etc.

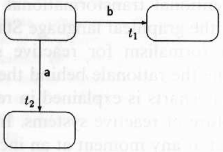


Fig. 7.

- To express priority of event b over event a in this statechart: can replace a by $a \wedge \neg b$

Summa Summarum

In a nutshell, one may say with David Harel:

Statecharts	=	Mealy Machines
	+	depth
	+	orthogonality
	+	broadcast
	+	data

Overview

Finite State Machines

Statecharts
- Hierarchy
- Orthogonality
- Broadcast
- Time in Statecharts

Stateflow

SyncCharts (Safe State Machines)

Characteristics of 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.

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 reactive 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 $\Longrightarrow$
  One has to determine *the reaction time of a sequence*.

Reaction Time in Statecharts

Question: What's the reaction time of a reactive system upon an external event in Statecharts?

Possibility 1

Specify a concrete amount of time for each situation.

- This forces us to quantify time right from the beginning.
- This is clumsy, and not appropriate at this stage of specification where one is only interested in the relative order and coincidence of events.

Reaction Time in Statecharts

Question: What's the reaction time of a reactive system upon an external event in Statecharts?

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 consequences (*e. g.*, because of tm(n)-events)
- $\Longrightarrow$ A fixed execution time for syntactic entities (transitions, statements, etc.) is not flexible enough.

Reaction Time in Statecharts

Question: What's the reaction time of a reactive system upon an external event in Statecharts?

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:

1. It should be accurate, but not depending on the actual implementation.
2. It should be as short as possible, to avoid artificial delays.
3. 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.
- However, in actual implementation this means:

  The reaction comes before the next input arrives,

  or, put another way,

  Reactions are not infinitely fast, but fast enough.

Example

See the following figure:

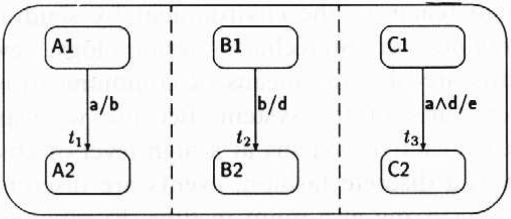


Fig. 6.

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 conjunctions and disjunctions of them.
- However, one also needs **negations of events** to trigger a transition.
- Example: To specify priority between (reacting on) event **a** and event **b**

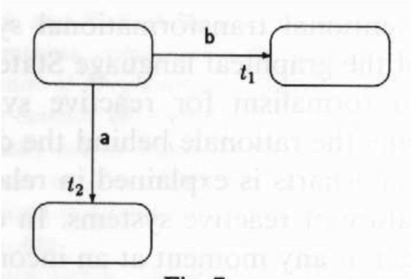


Fig. 7

Problem: Grandfather Paradoxon

What semantics to give to this Statechart?

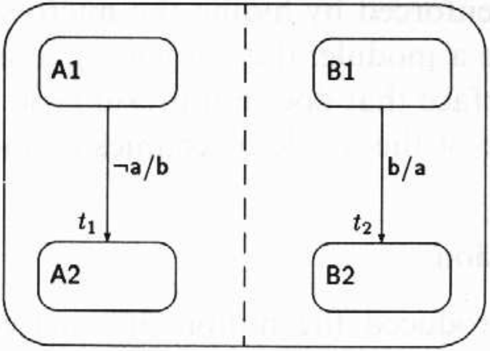


Fig. 8.

Grandfather Paradoxon

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.*

Solution

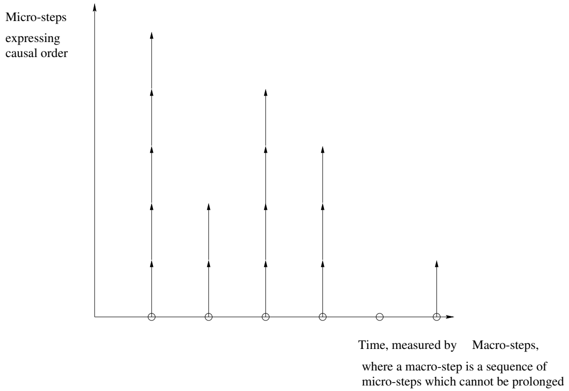
Introduce two levels of time

- 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 STATEMATE 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 STATEMATE Semantics (cont'd)

- Macro-steps are observable steps $\Longrightarrow_I^O$
- Each macro-step is a sequence of micro-steps, that is ordered causally; one micro-step can never influence previous micro-steps.
- In Statecharts as implemented by STATEMATE ("Harel-Statecharts"):
  - 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 STATEMATE Semantics

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

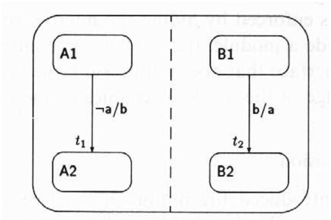


Fig. 8.

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

Here absence of triggers generates presence of triggers, which violates their absence within the same step.

Overview

Finite State Machines

Statecharts

Stateflow
  Simulink Interface
  Semantics

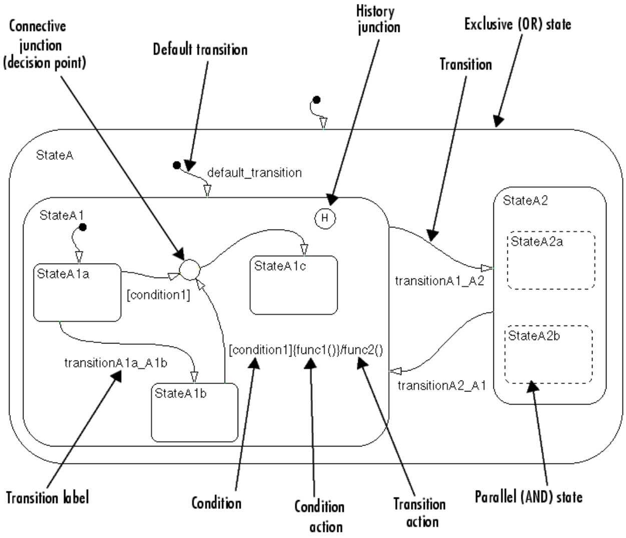
SyncCharts (Safe State Machines)

Stateflow

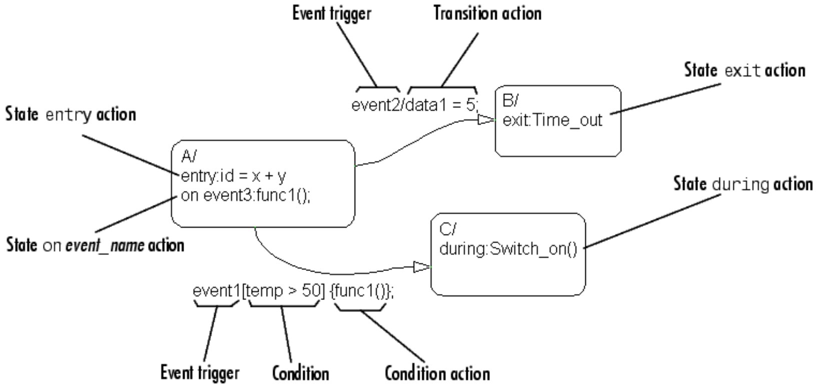
Stateflow is a special module embedded in Matlab/Simulink (The MathWorks). It is used to model state based control and supervisory logic inside Simulink.

- Different modelling environment
- Control is state based
- Syntax is a Statechart dialect
- Semantics differ from synchronous SyncCharts (E-Studio)
- Interfaces to Simulink environment
- Features states, transitions, events, trigger, conditions, actions, parallelism, hierarchy

Stateflow: States

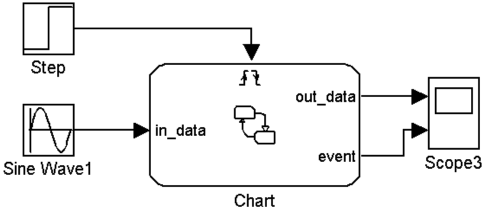


Stateflow: Transitions and Actions



Simulink Interface

Stateflow Charts can be embedded into Simulink models.



Stateflow distinguishes: Data (only values) and Events (only boolean signals). Stateflow charts get evaluated only either at

- events: Data changes are only detected if some other event happened (if no external event occurs, the chart is not evaluated)
- predefined sample time: Chart is evaluated regularly: Also data changes are detected. No external event inputs allowed.

Stateflow Semantics

- Stateflow semantics are not formally specified, only informally by Stateflow manual.
- Single-Event run-to-completion semantics
- Exactly one event is evaluated when it occurs

⇒ Triggers with multiple concurrent events (*e. g.* "A and B") are not possible (only disjunction is possible, using the OR operator: "A || B")

⇒ Negation of events (*e. g.* "not A") is not possible

- Parallel states are evaluated by some predefined execution order, depending on graphical layout/user input (see next slide)

Events and Actions

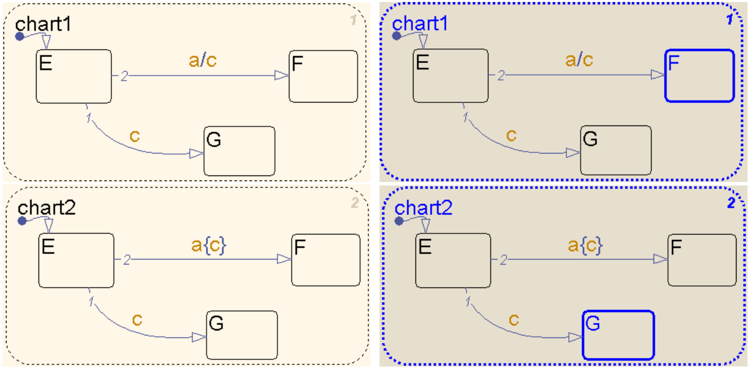
Event
: When event occurs
- it is processed from the top or root of the diagram down through the hierarchy.
- At each level in the hierarchy a check for the existence of a valid explicit or implicit transition among the children of the state is conducted.

Condition Action
- Executed as soon as the condition is evaluated as true,
- but before the transition destination has been determined to be valid.
- No condition $\Longrightarrow$ an implied condition evaluates to true and the condition action is executed

Transition Actions
- Executed when the transition is actually taken:
- Executed after the transition destination has been determined to be valid
- and the condition, if specified, is true
- Consists of multiple segments $\Longrightarrow$ only executed when the entire transition path to the final destination is valid

Stateflow Example: Early Return Logic

Event a is triggered from outside



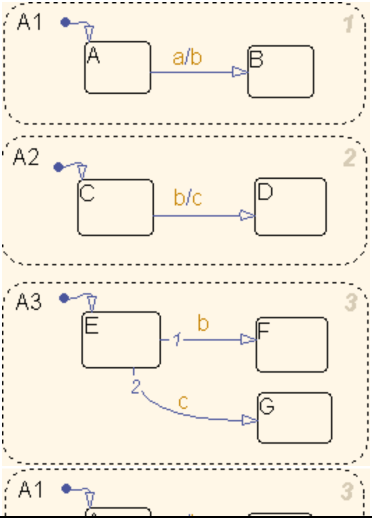
- chart1 takes the normal transition and executes its transition action after F was decided to be the next active state
- When a is triggered, chart2 evaluates its outgoing transition from E to F. Condition is true $\Longrightarrow$ its condition action is

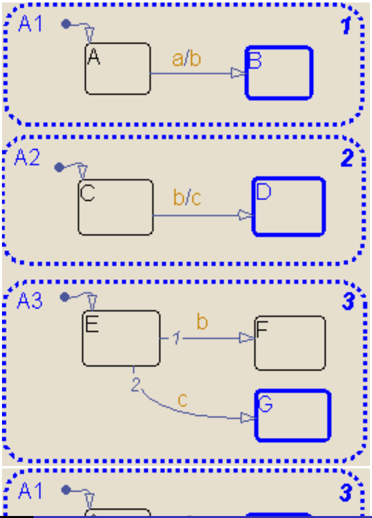
Stateflow Run-To-Completion Semantics

- If an event is broadcast, the active transitions triggered by this event are evaluated successively according to the execution order of their parent states (*e. g.* in parallel states)
- In each transition evaluation, new signals might be emitted by transition or condition actions
- Each new event emission calls this interpretation algorithm immediately recursively and runs to completion
- and only then resumes with the processing of the next transition for the original event.

Stateflow Example 1

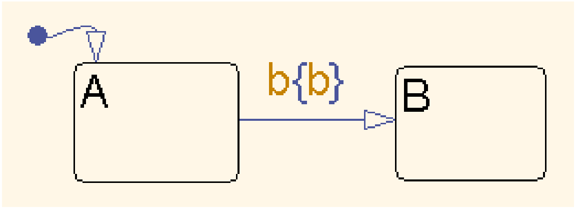
Event a is triggered from outside





Stateflow Example 3

What happens here when signal b is emitted?



Stack Overflow (runtime exception in newer Simulink versions)

Overview

Finite State Machines

Statecharts

Stateflow

SyncCharts (Safe State Machines)
- States
- Transitions
- Connectors
- Esterel Studio

Similarities

SyncCharts are made up of elements common to most Statecharts dialects:

- States
- Initial/terminal states
- Transitions
- Signals/Events
- Hierarchy
- Modularity
- Parallelism

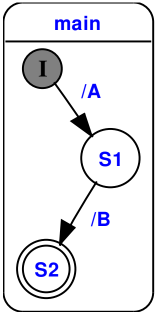
Differences

SyncCharts differ from other implementations of Statecharts:

- Synchronous framework
- Determinism
- Compilation into backend language Esterel
- No interpretation for simulations
- No hidden behaviour
- Multiple events
- Negation of events
- No inter-level transitions

Simple Sequential Automaton

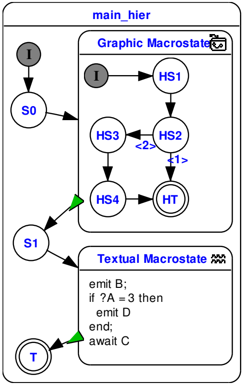
SyncChart:



Elements:

- States:
  - Regular state (circle)
  - Terminal state (doubled circle)
  - Hierarchic state (box with rounded edges)
- Transitions:
  - Arrows with labels
- Connectors:
  - Colored circles with single letters

Hierarchic States



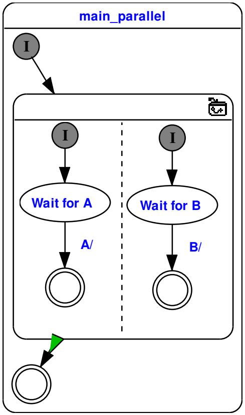
```
emit B;
if ?A = 3 then
  emit D
end;
await C
```

SyncCharts know four types of states:

- Simple States: Carry just a label.
- Graphic Macrostates: Encapsulates a hierarchy of other states, including further graphic states.
- Textual Macrostates: Contain statements of the Esterel language. They are executed on entry of the state.
- Run Modules: Include other modules.

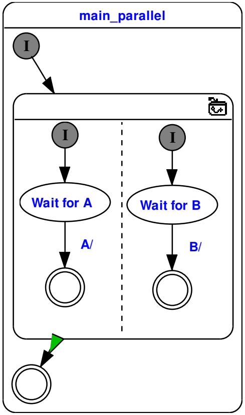
Transitions are **not** allowed to cross the boundaries of graphic macrostates. This is in contrast to other modelling tools.

Parallel States



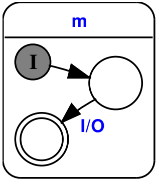
- Dashed lines (horizontal or vertical) separate parallel executed states inside a graphic macrostate.
- Each segment may be segmented into further parallel segments, but iterative segmentation does not introduce additional hierarchy. All parallel segments in a graphic macrostate are at the same level.

Parallel States



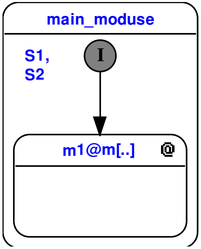
- A transition outside the graphic macrostate with normal termination is activated, when all parallel segments have reached their terminal state.
- If just one segment does not have one or if it is not reached, then the normal termination transition will never be activated.

Modules



A module like this with an interface:

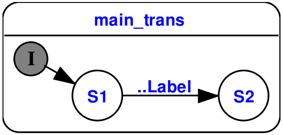
```
input I;
output O;
```



... can be used as a Run Module with these signal bindings:

```
  signal S1 / I;
  signal S2 / O
```

Syntax of Transition Labels



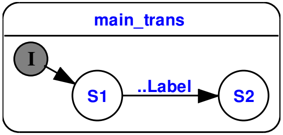
Informal syntax of a transition label between states S1 and S2, all elements are optional:

```
# factor trigger {condition} / effect
```

Basic activation and action:

- `trigger` is an expression of signal presence like "`A or B`"
- Enclosed in braces is the `condition`. It is a data expression over signal values or variables like "`?A=42`"
- Behind a single "/" follows the "`effect`" as a list of emitted signals if the transition is executed. Multiple signal names are separated with ",".

Syntax of Transition Labels



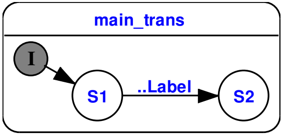
Informal syntax of a transition label between states S1 and S2, all elements are optional:

```
# factor trigger {condition} / effect
```

Extensions:

- "#" is the flag for an immediate transition
- "`factor`" is the (natural) number of instants a transition must be active before it is executed. These active instants does not need to be consecutive, but `S1` must be active all the time.

Transition Labels: Examples



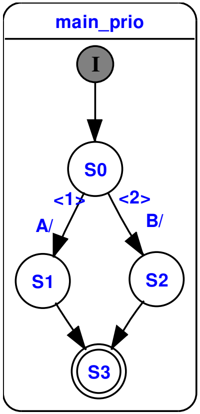
The following label examples belong to the transition originating at S1 and leading to S2:

- `A/B`
  After entering `S1` the signal `A` is tested from the next instant on. If `A` is present, then `B` is emitted in the same instant and state `S2` is entered.
- `/B`
  After enabling `S1`, `B` is emitted in the next instant and `S2` is entered.
- `3 A/`
  The transition is executed, if `S1` is active consecutively and signal `A` is present for 3 times.

Transition Labels: Examples

- `#A/`
  If `S1` is entered, signal `A` is tested from the same instant on. If `A` is present in the instant `S1` is entered then state `S2` is entered in the same instant.
- `{?A=42}/`
  The transition is executed, if the (valued) signal `A` carries the value 42. `A` does not need to be present for this test.
- `A {?A=42}/`
  This test succeeds if `A` is present and carries the value 42.
- `A and (B or C)/`
  Logical combination of signal presence.
- `{?A=10 and (?B<3 or ?C=1)}/`
  Logical combination of value tests.
- `/A(2), B(4)`
  Emission of multiple valued signals.

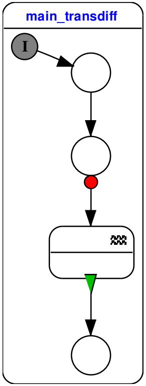
Transition Priorities



- When more than one transition departs a state, an automatic (but editable) priority ordering is established.
- The transition labels are evaluated according to their priority.
- The first label that succeeds activates its transition.
- Low numbers mean high priority.

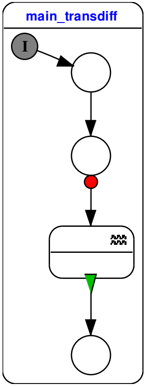
Transition Types

SyncCharts feature four different types of transitions: They are differentiated by a symbol at the arrow root:



- Initial connector: Initial arc
  Initial arcs connect the initial connectors of the chart with the other states.
- No symbol: Weak abort
  When the trigger/condition of the transition is enabled, then the actions of the originating state in the current instant are executed for a last time, then the transition action, and the entry action of the new state.
  In other words:
  The old state can "express it's last will".

Transition Types

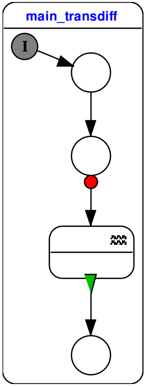


- Red bullet: Strong abort
  The action for the current instant of the old state is not executed. Only the transition action and the entry action of the new state is executed.
- Green triangle: Normal termination
  This transition can be used to exit macro states. It is activated when the macro state terminates.

All these transition types must not be confused with "immediate" or "delayed" evaluation of the transition label (label prefix "#").

Transition Types and Labels

Some transition types have restrictions on their labels:



- Initial arc:
  These are always "immediate," therefore the additional flag "#" is not needed.
- Weak abort: No restrictions.
- Strong abort: No restrictions.
- Normal termination:
  They support no triggers or conditions because they are activated by the termination of the originating state. The immediate flag is not used either.

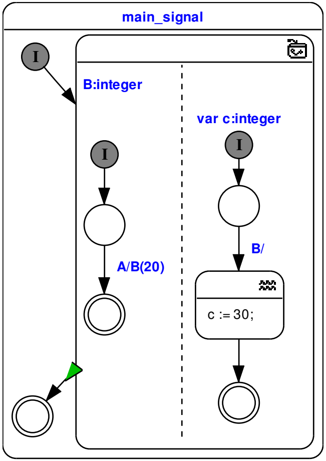
Transition Types and Priorities

The type of a transition interacts with it's priority:

- Strong abort: Highest priority
- Weak abort: Middle priority
- Normal termination: Lowest priority

Esterel Studio enforces these rules by changing the numerical priorities of the transitions.

Local Signals and Variables



Local signals:

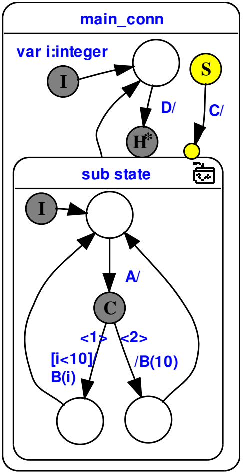
- Defined in the body of a graphical macrostate
- Shared between parallel threads

Local variables:

- Attached to the initial connector
- Not shared

Connectors

This (artificial) SyncChart demonstrates all four connector states:



I Initial connector:

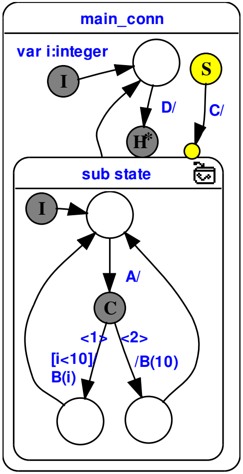
- Activated at activation of the macrostate
- Only departing transitions permitted
- All connected transitions are “immediate”

C Conditional connector:

- All departing transitions are “immediate”
- One departing “default” transition without condition must be present.

Connectors

This (artificial) SyncChart demonstrates all four connector states:

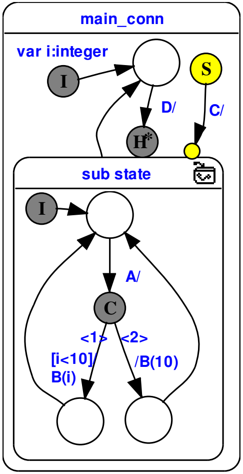


S Suspend connector:

- The suspend state is always active.
- Only one departing transitions is permitted.
- The transition can only hold a trigger expression.
- The “immediate” flag can be enabled on demand.
- When the transition is activated, then the target state is (strongly) suspended.

Connectors

This (artificial) SyncChart demonstrates all four connector states:



H* History connector:

- This connector is directly attached to macrostates
- Only incoming transitions can connect.
- The previous state of the macrostate is restored when it is entered through a history connector.

Esterel Studio

`http://www.esterel-technologies.com/`

Esterel Studio features:

- Graphical editor for Statechart dialect “Safe State Machines”, a. k. a. SyncCharts
- Code generator for textual Esterel
- Esterel compiler for C code production
- Interface to backend in host language (C)
- Graphical simulation Validation/Testing environment

To Go Further

- David Harel, Statecharts: A Visual Formulation for Complex Systems, *Science of Computer Programming*, 8(3), June 1987, pp. 231–274
- D. Harel, M, Politi, *Modeling Reactive Systems with Statecharts: The STATEMATE Approach*, McGraw-Hill, ISBN 0-07-026205-5, 1998.
- Charles Andr, Semantics of S.S.M (Safe State Machine), Esterel Technologies Technical Report, April, 2003, `http://www.esterel-technologies.com/v3/?id=50399&dwnID=48`
- Home page of Esterel Technologies: `http://www.esterel-technologies.com/`
- Local information on Esterel-Studio and where to find further documentation: `http://www.informatik.uni-kiel.de/~esterel/`