Stateflow

- ⇒ What is Stateflow
- ⇒ Examples
- ⇒ Extended uses
- ⇒ Semantics and problems

What is Stateflow?

A state machine design tool integrated within Simulink:

- Stateflow produces Simulink blocks, fed with Simulink inputs and producing Simulink outputs,

- a Stateflow block can “execute” Simulink blocks as actions,

provides a seamless integration of state machines into a block diagram formalism.

What is Stateflow made of?

Stateflow integrates three basic components:

1. hierarchical and parallel state machines borrowed from Statecharts,
2. control flow diagrams allowing to design complex transitions between Stateflow states,
3. truth tables allowing to design complex actions.

Stateflow Features

Several features are worth to be noticed:

- Stateflow encompasses both Mealy and Moore machines:
  - actions associated with both states and transitions,
  - entry actions performed when entering a state,
  - during actions, performed when remaining in the state,
  - exit actions, performed when leaving the state,
  - condition actions, performed before leaving the source state,
  - transition actions, performed after leaving the source state but before entering the destination state.

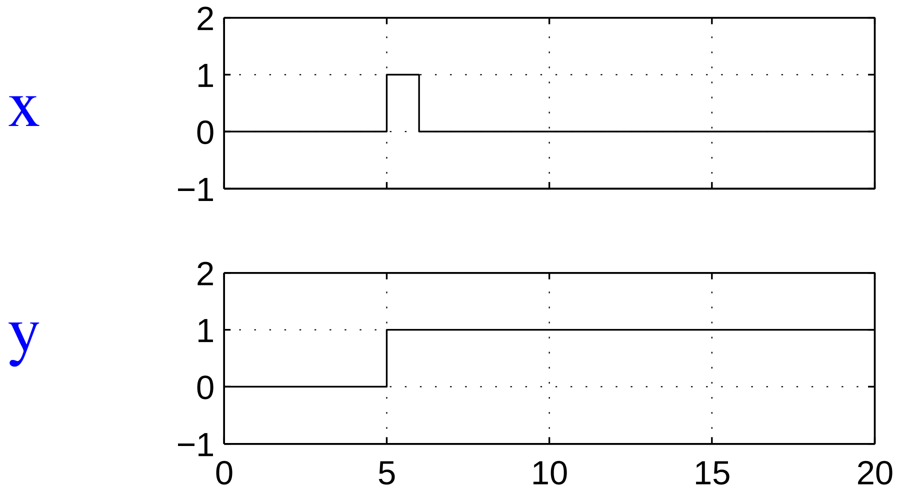
More Stateflow Features

- large variety of means for describing actions:
  - event broadcasting,
  - Stateflow scripts (assignments, etc.),
  - truth tables,
  - Simulink block “execution”,
  - Matlab functions.

Example: The Once Subsystem in Stateflow

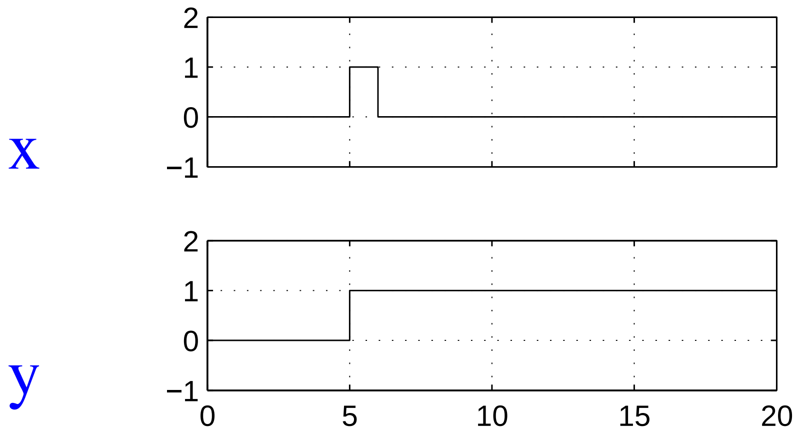
Design an automaton with one boolean input and one boolean output such that the output

- is initially false
- becomes true as soon as the input becomes true
- stays true for ever as soon as it has become true once



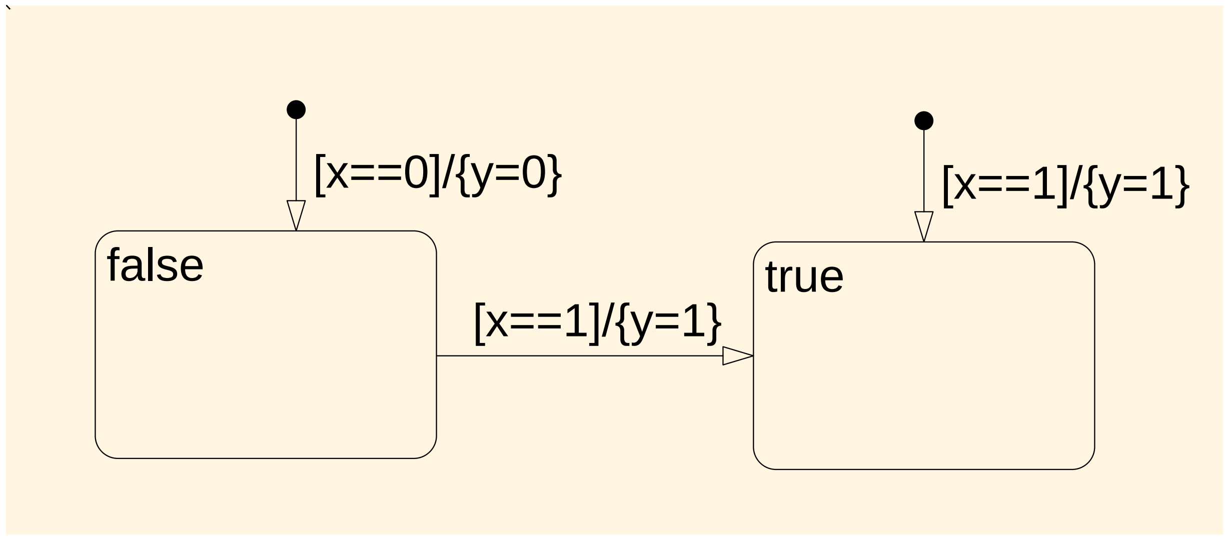
Time offset: 0

Solution



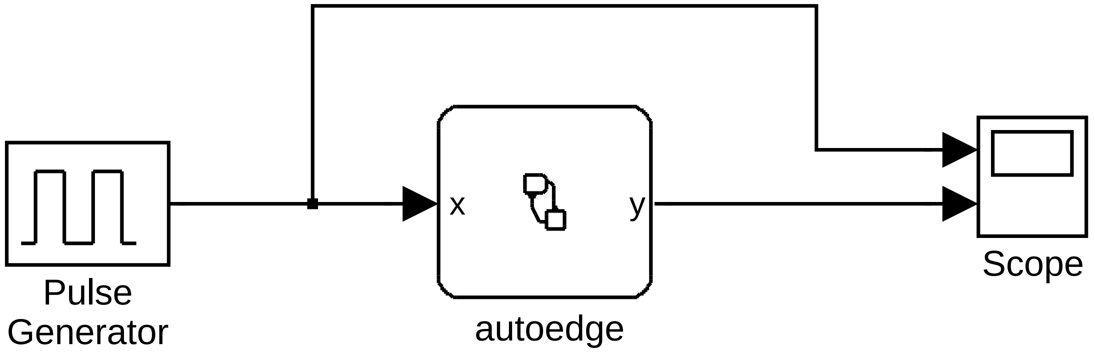
Time offset: 0

edgeauto/autoedge



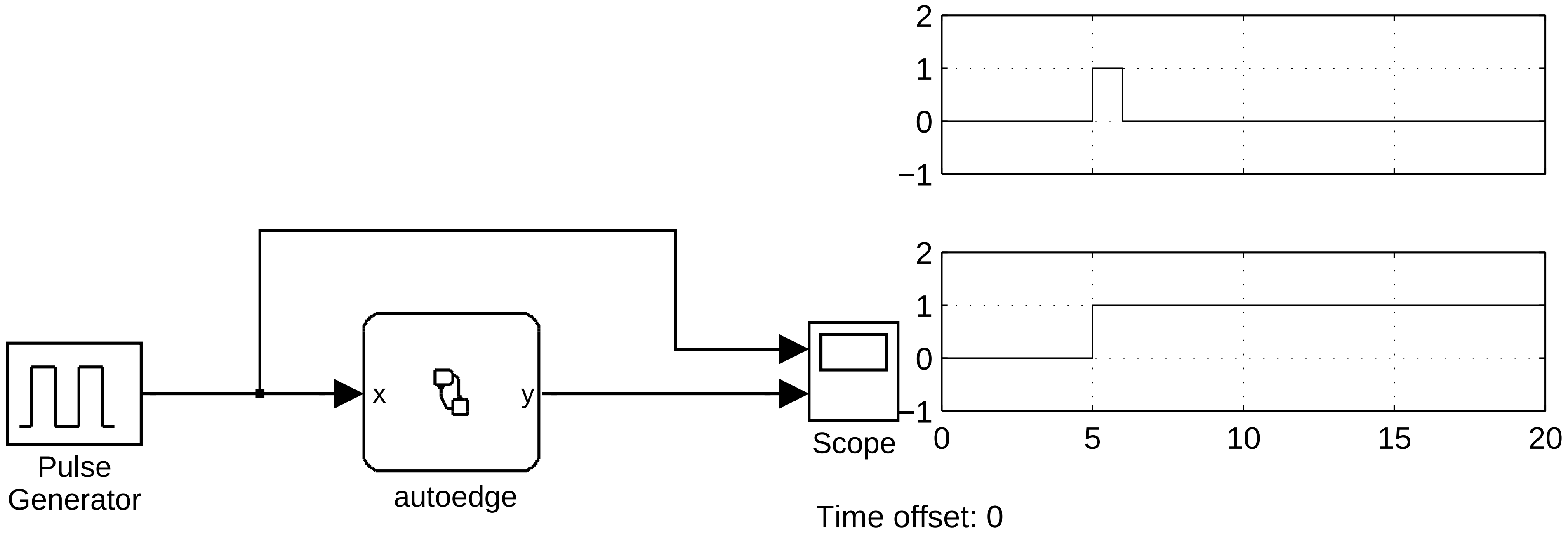
Checking the Solution

Shows how Stateflow integrates with Simulink



Checking the Solution

Shows how Stateflow integrates with Simulink

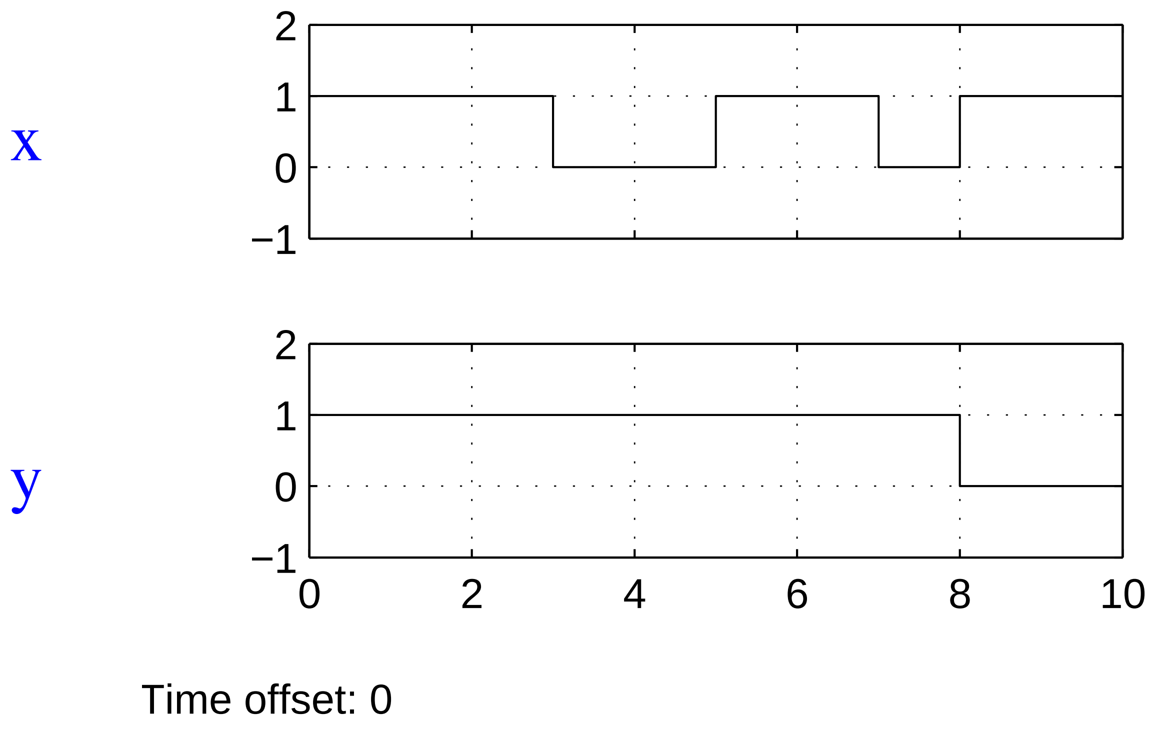


Example:Monitoring Properties

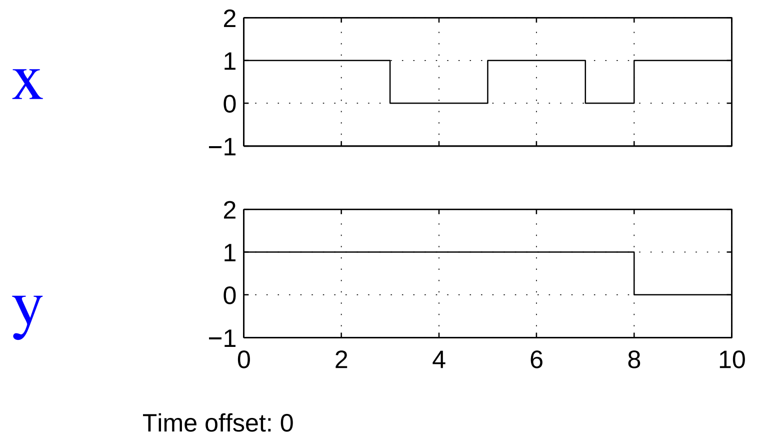
Design a property observer that monitors the property:

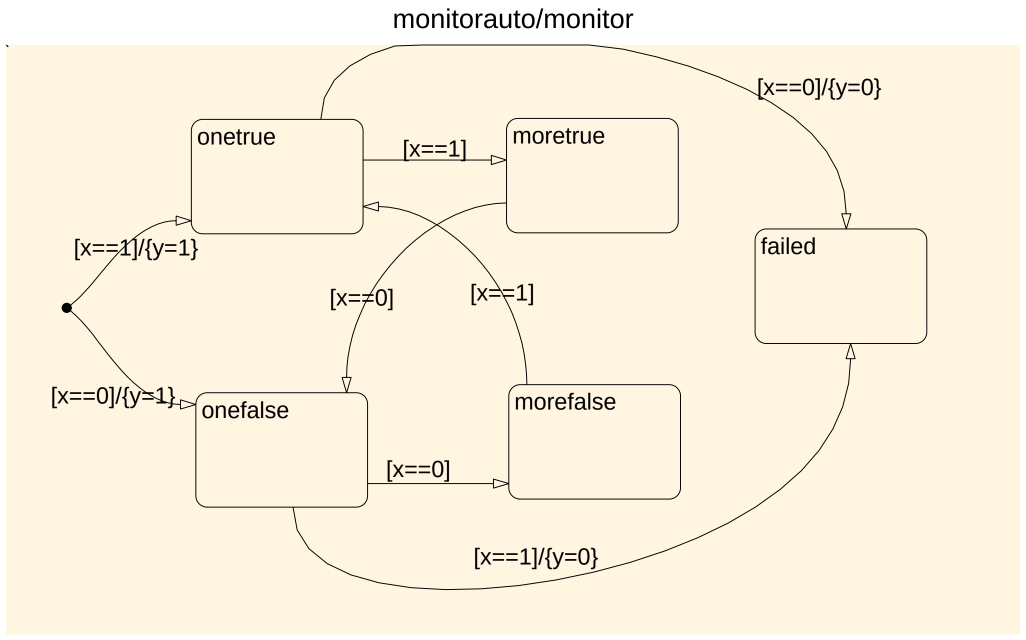
$$\boxed{\text{a signal doesn't change its value in two consecutive samples}}$$

Typical bevaviour:



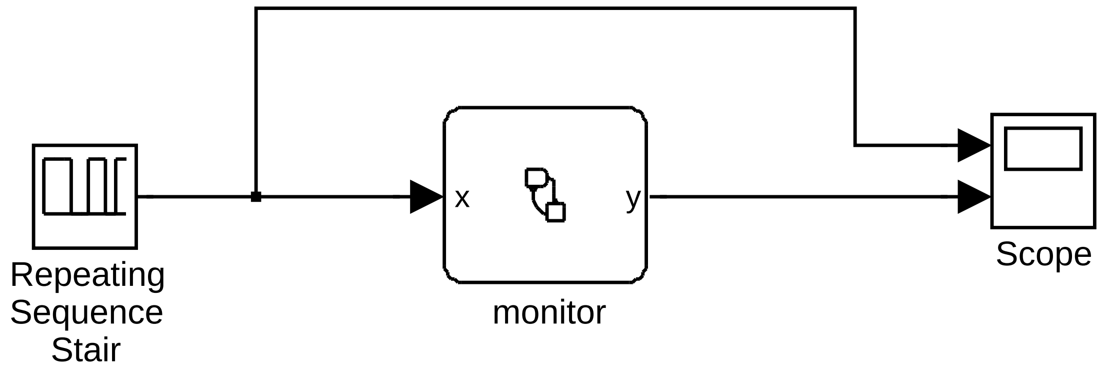
Monitoring Properties: Solution





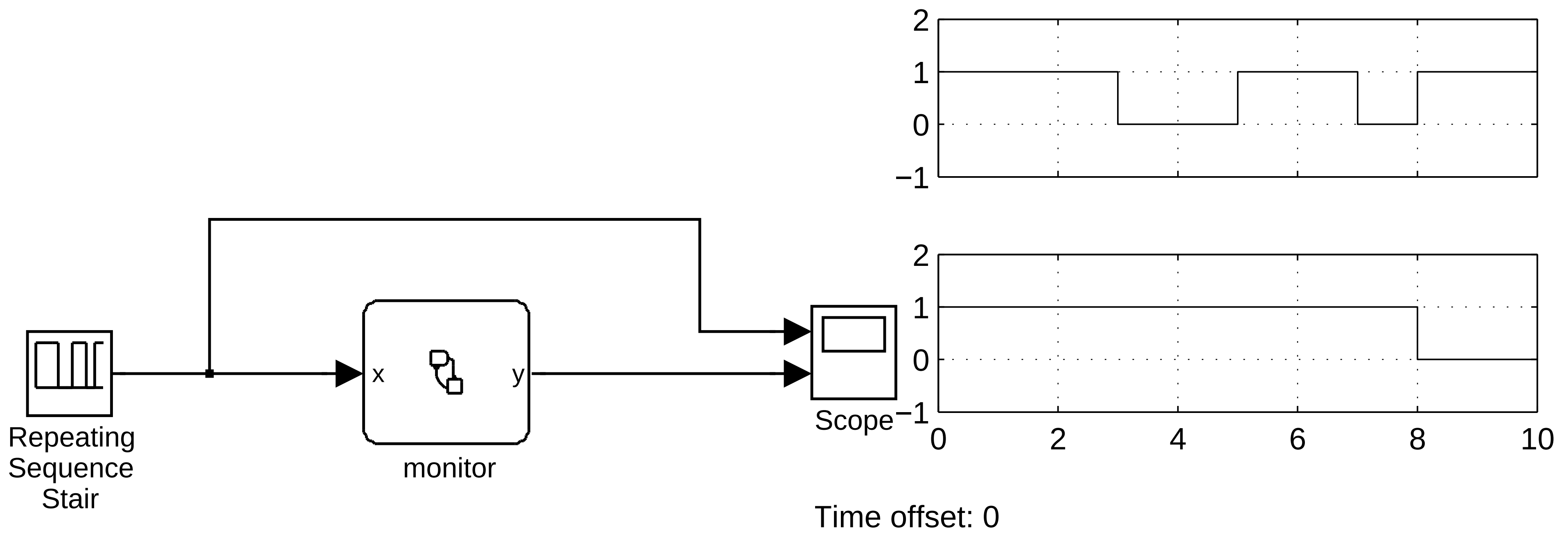
Checking the Solution

Shows how Stateflow integrates with Simulink



Checking the Solution

Shows how Stateflow integrates with Simulink



More Examples

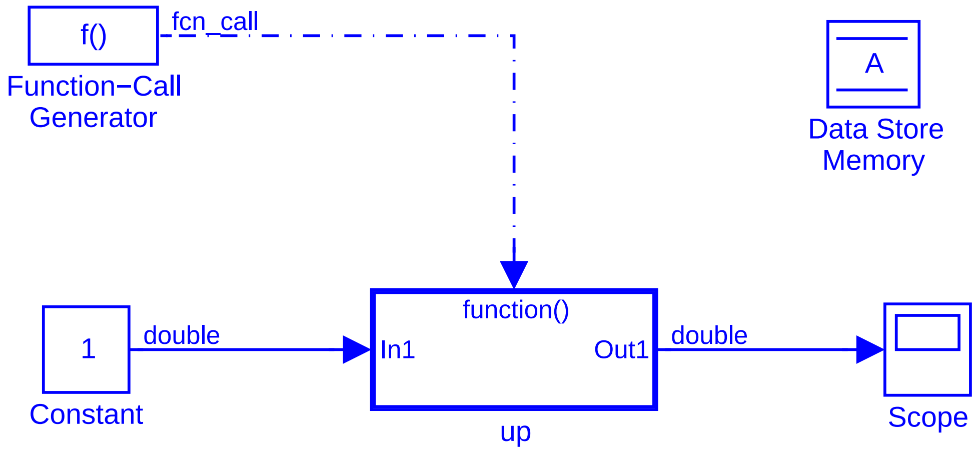
A very popular usage: Mode Automata

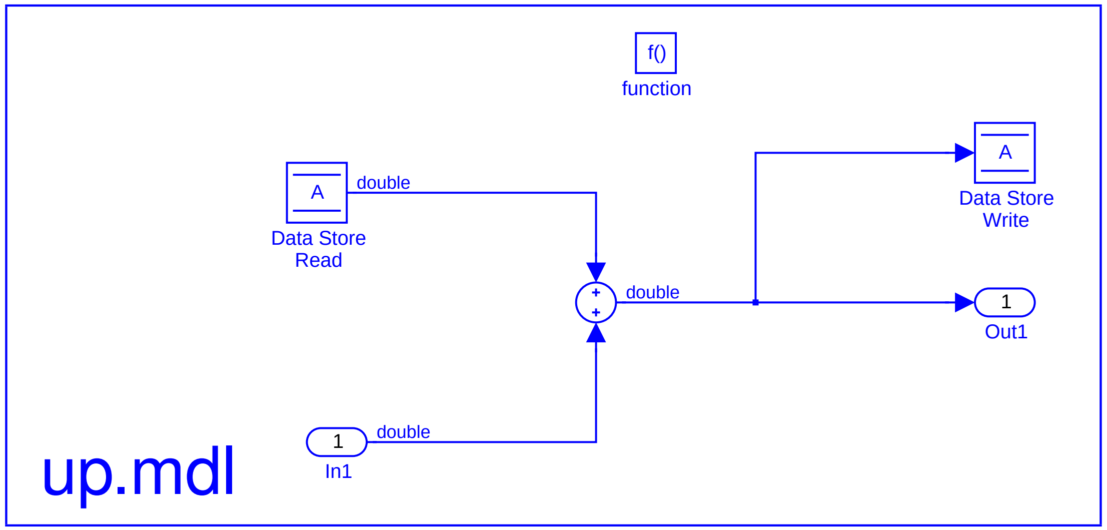
A Stateflow chart is used to trigger several controller modes

A cooperative way of designing complex systems:

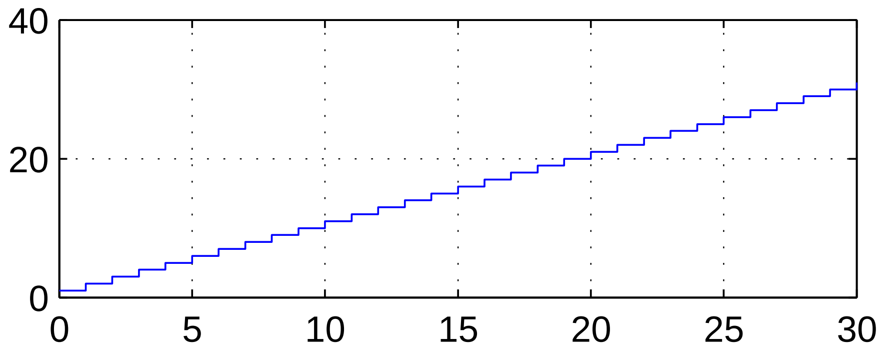
- teams begin to agree on shared variable names,
- then each team can independently design and validate its own mode.

The Up team builds the “up” mode . . .

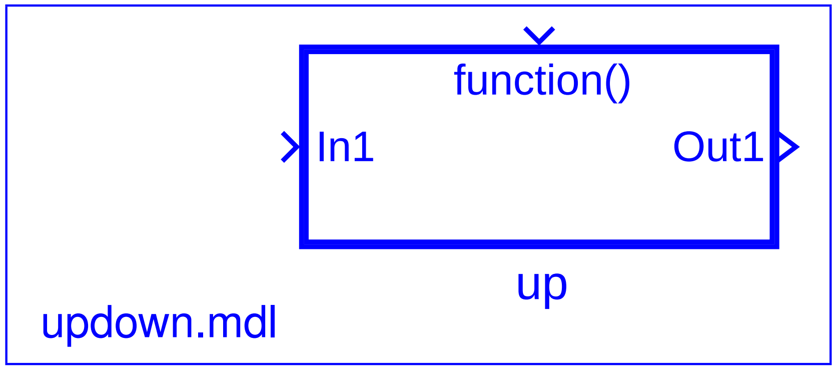




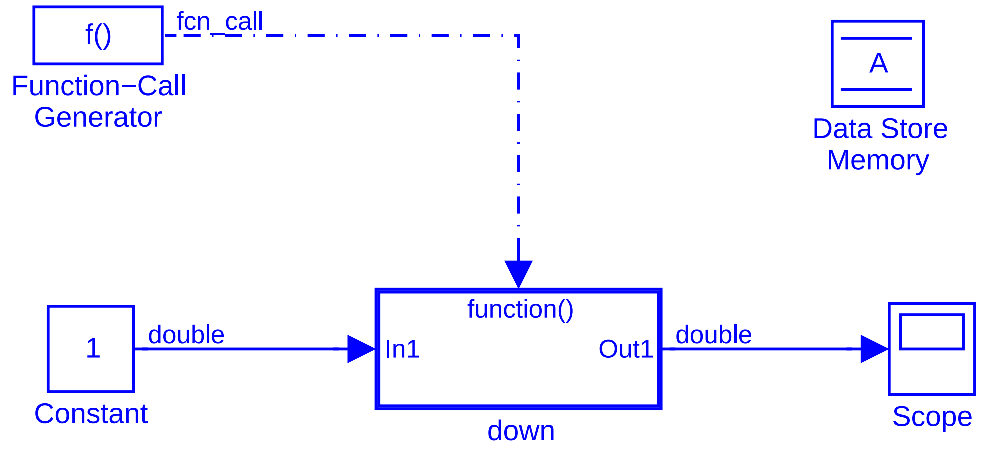
. . . , tries it and saves it

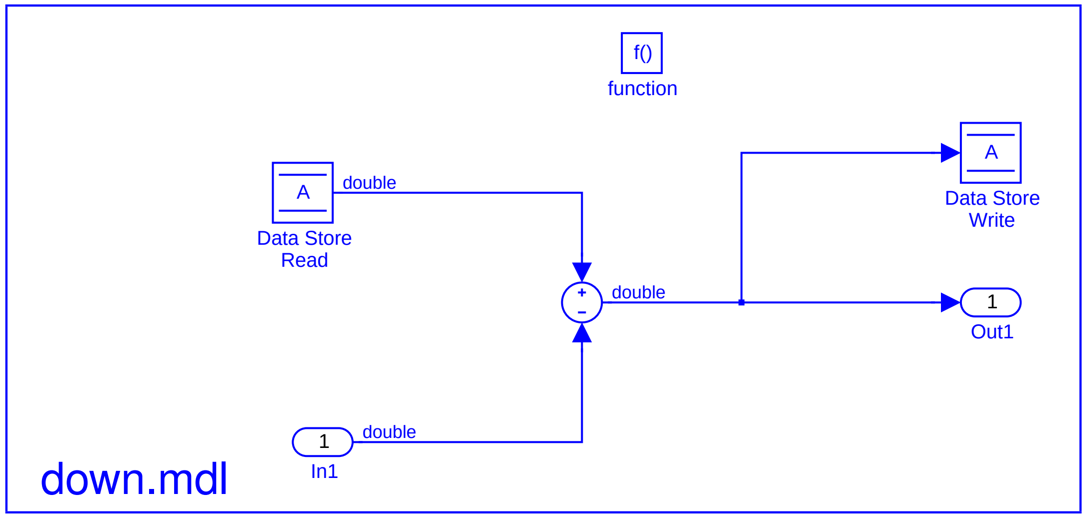


Time offset: 0

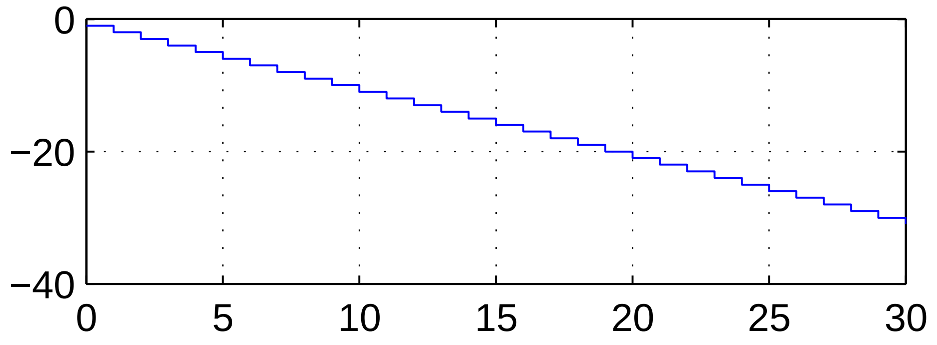


The Down team builds the “down” mode . . .

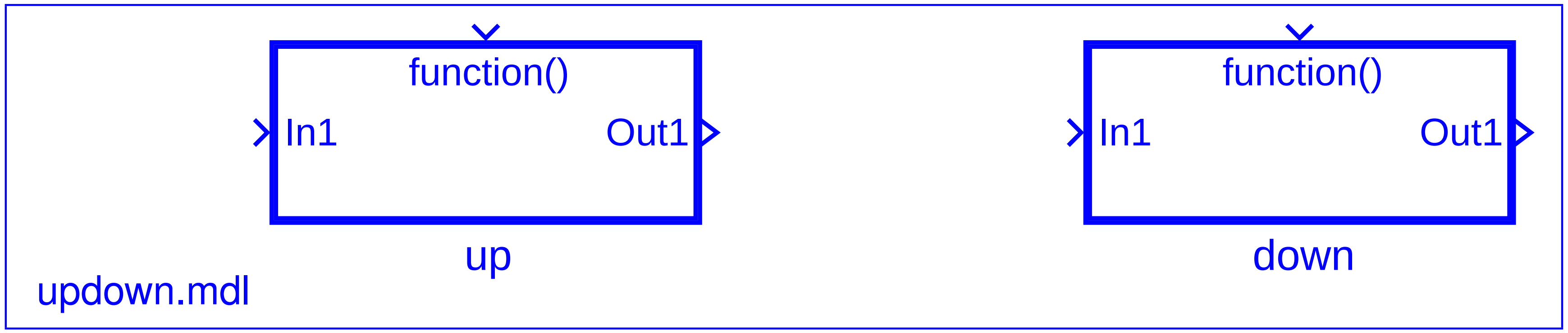




. . . , tries it and saves it

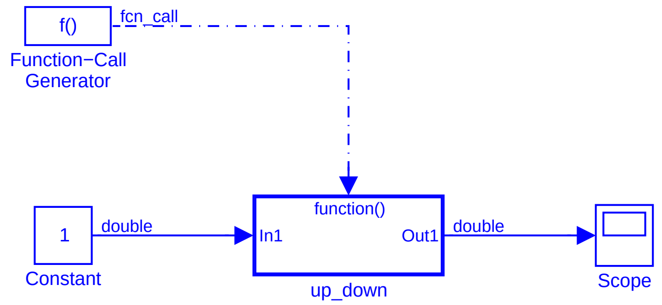


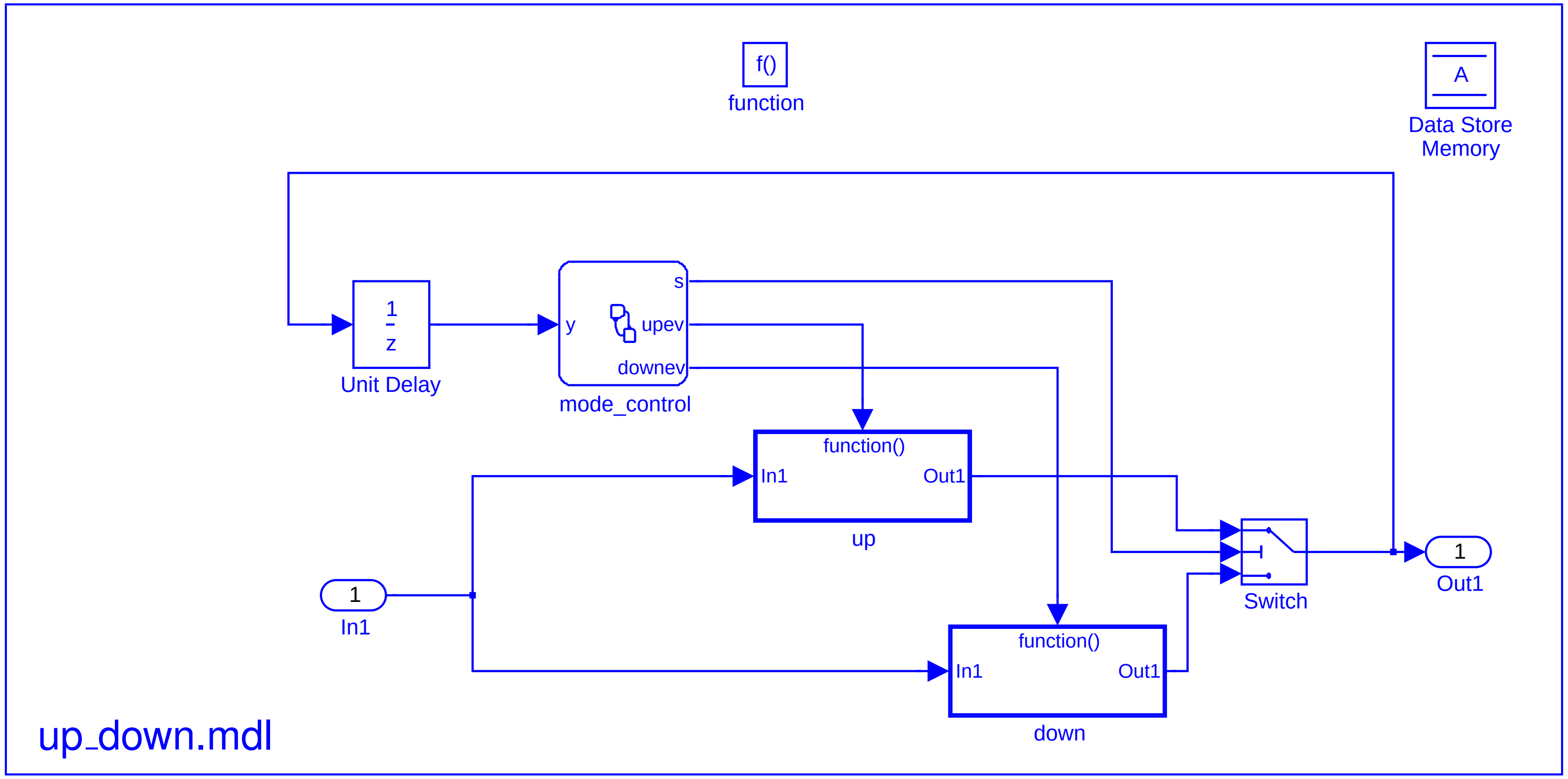
Time offset: 0



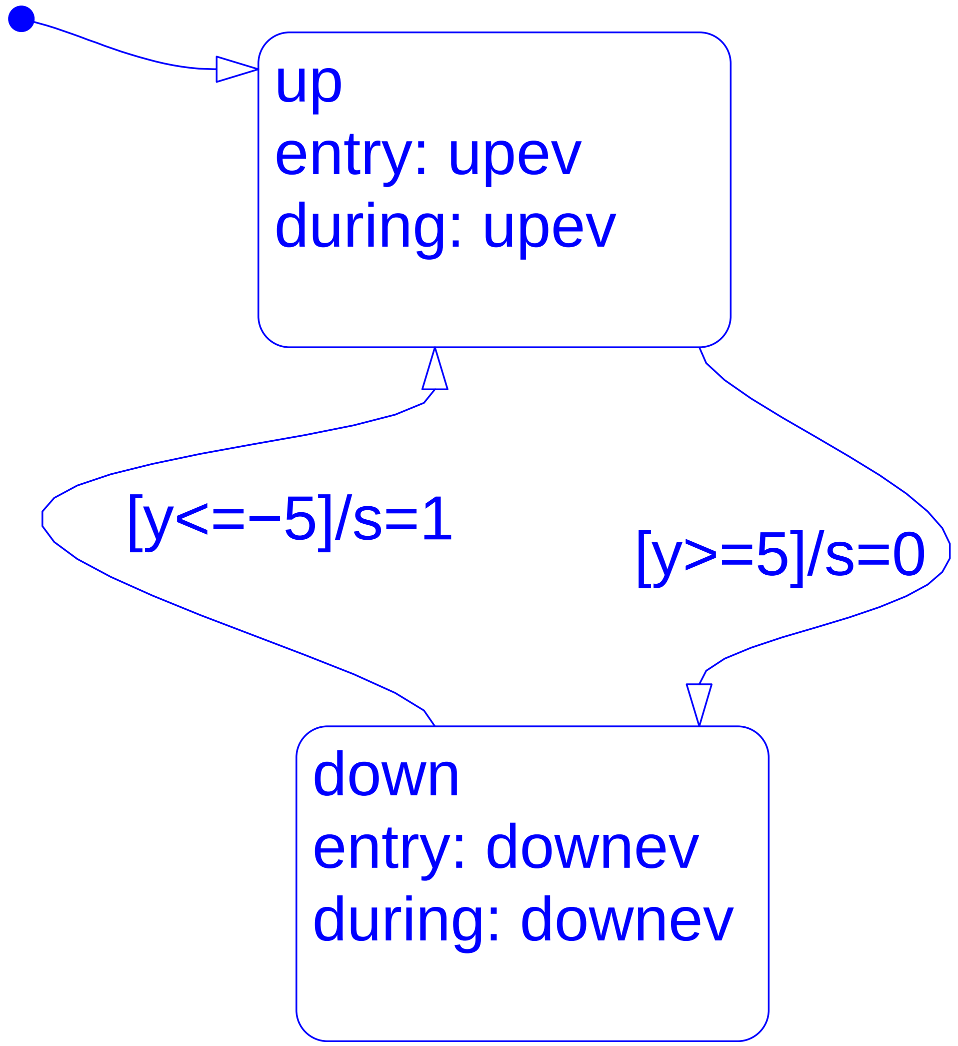
The Simulink library looks like a Java “static class”

The UpDown team combines the two models. . .

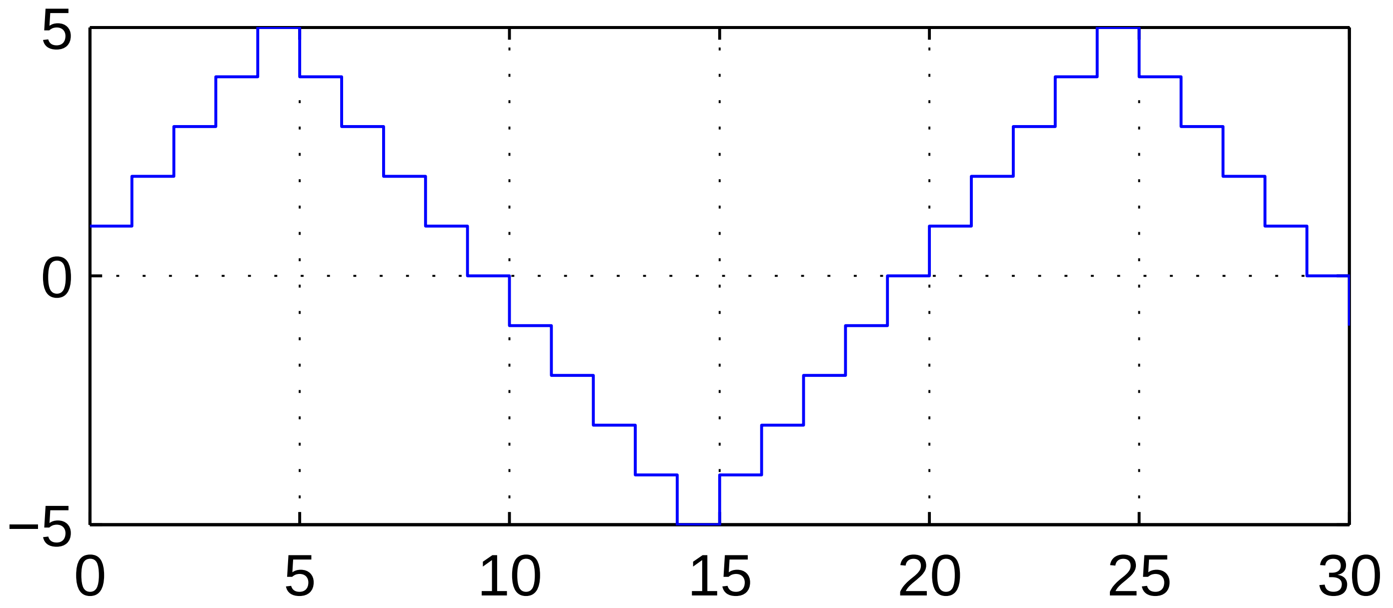




. . . through the activation automaton. . .



…and tries it



Time offset: 0

It works…

Interest and Drawbacks

- Interest :
  - Modular approach

    No redesign

    No complex wiring

- Drawbacks
  - Unsafe features

    when activations are not exclusive

Semantics

Simple Stateflow has a clear semantics. Semantic problems appear with parallel states : parallel simultaneous activities which interact and communicate.

Several solutions:

- non determinism: seldom used because determinism is appealing,
- unique logical solution: (for instance Signal),
- unique constructive solution: (Esterel, SyncCharts),
- restriction to one-way interaction: interactions statically ordered ensuring uniqueness (Lustre/Scade, Simulink?),
- micro-step semantics: (StateCharts, Stateflow)

Stateflow Solution

Micro-step semantics:

an interpretation algorithm based on several orderings (priorities) describes uniquely in which order actions take place.

raises problems of cyclic behaviours which don't terminate. *(What about Statecharts?)*

The Interpretation Algorithm

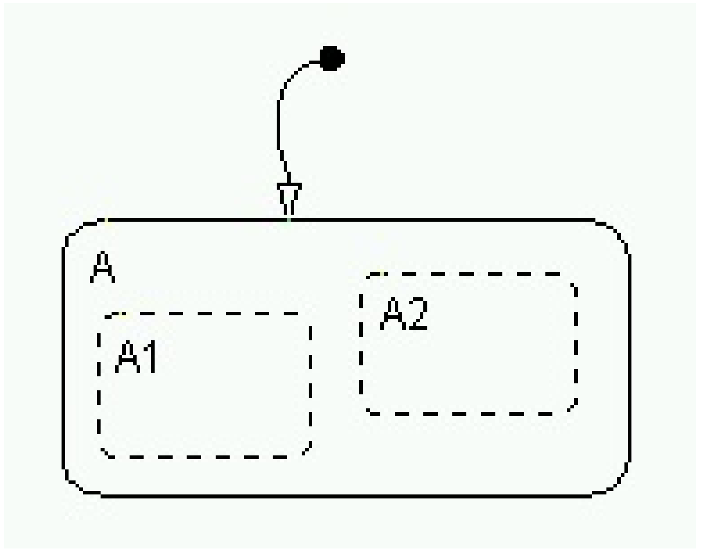
Starts each time an input event (coming from the Simulink world or from the same Stateflow: recursive behaviour) arrives.

1. search for active states,
2. search for valid transitions,
3. valid transition execution,
4. during action execution.

Search for Active States

performed:

- hierarchically, from top to bottom;
- sequentially for parallel states: graphical two dimension priority: states are searched from top to bottom and from left to right!



Search for Active States

back

performed:

- hierarchically, from top to bottom;
- sequentially for parallel states: graphical two dimension priority: states are searched from top to bottom and from left to right!

This is an unsafe feature:

- first, small variations in the drawing can result in different behaviours,
- parallelism is misleading (actually sequentiality).

verify that the Stateflow behaviour is independent of the ordering of parallel states.

Search for Valid Transitions

transitions are searched based on several criteria,

this is considered harmful even in the Stateflow documentation:

“ Do not design your Stateflow diagram based on the expected execution order of transitions. ”

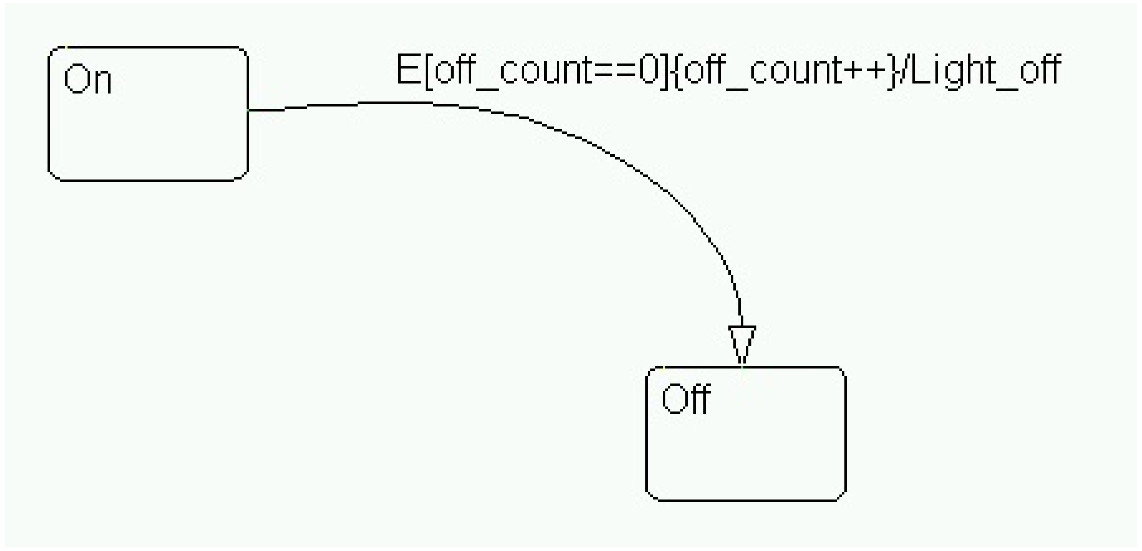
verify that there is at most one valid initial segment at a time outgoing an active state.

Transitions

A transition can be a complex flow graph made of segments joining connective junctions. Each segment can bear a complex label made of:

```
event [condition]{condition_action}/transition_action
```

in which each field is optional.

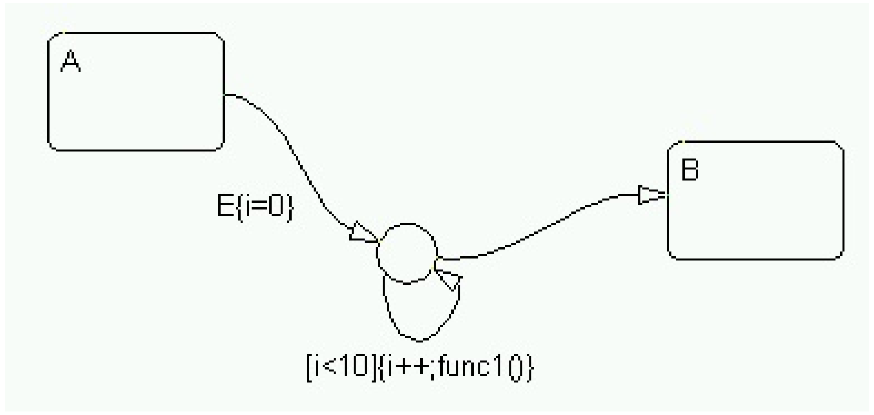


Valid Transitions

Segments outgoing from a connective junctions are searched for validity according to several criteria, among which

"clock-wise, starting from a twelve'o clock position".

Cyclic flow diagrams are allowed leading to "for" and "while" diagrams *(not available in Scade).*



Valid Transitions

Segments outgoing from a connective junctions are searched for validity according to several criteria, among which

“clock-wise, starting from a twelve’o clock position”.

Cyclic flow diagrams are allowed leading to “for” and “while” diagrams *(not available in Scade)*.

Cyclic diagrams can raise non-termination.

Testing for validity of a transition implies executing condition actions found along the transition path (complex side effects).

Labelled Default Transitions

Default transitions (transitions that are executed when entering a super-state) raise a specific problem, indicated in the following warning quoted from Stateflow documentation:

“When labelling default transitions, take care to ensure that there is always at least one valid default transition. Otherwise, a Stateflow chart can transition into an inconsistent state.”

verify that there is always a valid default transition

Executing a valid transition

Once a valid transition has been found, the algorithm is quite simple:

1. execute the exit action of the source state,
2. set the source state to inactive,
3. execute the transition actions of the transition path,
4. set the destination state to active,
5. execute the entry action of the destination state.

Executing a valid transition

Yet, the behaviour can be very complex and lead to non termination because, when an event is broadcast, the interpretation algorithm stops and is stacked and a new intepretation algorithm starts dealing with this event and runs up to completion.

“Broadcasting an event in the action language is most useful as a means of synchronisation among AND (parallel) states. Recursive event broadcasts can lead to definition of cyclic behaviour. Cyclic behaviour can be detected only during simulation.”

Thus possible cyclic behaviours should be detected at translation time.

When there is not Valid Transition

When an active state has no valid output transition, its “during” action is performed and the state remains active.

Safety Checking

The majors points to check for safety:

1. check that the order into which parallel states are considered is irrelevant,
2. check that outgoing transitions from a state are exclusive,
3. check that when entering a super-state, there is always a valid default transition,
4. check the "twelve o'clock rule",
5. check the cyclic behaviour of connective junctions,
6. check the recursive behaviour due to event broadcasting.

In all these aspects, guidelines are needed