Prof. Bernd Finkbeiner, Ph.D.
Dipl.-Inf. Rüdiger Ehlers
Dipl.-Inf. Hans-Jörg Peter
Michael Gerke, B.Sc.

Reactive Systems Group
Saarland University

Embedded Systems 08/09 – Second STATEFLOW Tutorial

The aim of this tutorial is to introduce slightly more elaborate and concise STATEFLOW constructs. Furthermore, it will provide some insight into sensing and measuring environment entities with polling real-time systems.

Exercise 1: State actions

As self loops — i.e. transitions with coinciding source and target state — are extremely frequent in everyday STATEFLOW models, there is special syntax for avoiding[1] these. States can be equipped with so-called *state actions*. Such actions are, depending on their specific type, executed upon entering the state, upon exiting the state, or when staying in a state because no outgoing transition fires. The latter form of state action constitutes a replacement for self loops that is both more concise and more efficient upon simulation than a self loop.

There are two forms of such state actions that are executed when staying in a state, i.e. when the state chart is triggered, yet no outgoing transition fires:

1. A *during* action, which is executed upon *every activation* (i.e. triggering) of the state chart unless some transition originating in the state fires, has the syntax

   `during:` *action*

   where the syntax of actions is as for the action part of a transition. Such an action is (almost[1]) equivalent to a self-loop transition labelled with empty guard (i.e., guard `[true]`) and action part *action*.

2. An *event-triggered action* executed upon *every occurrence of the respective event* unless some transition originating in the state fires:

   `on` *event*`:` *action*

   Such an action is (almost[1]) equivalent to a self-loop transition labelled with guard *event* and action part *action*.

[1] In fact, you may consider the new notation introduced here as merely being an abbreviation. Yet, there are notable semantic differences: First, the `during` and `on` state actions introduced herein do always have lower priority than all transitions originating from the respective state. Second, state actions are — in contrast to outgoing transitions, including self-loops — not considered to actually leave the state, which is providing a visible difference in the interaction with "temporal logic events", cf. exercise 2.

For associating such actions to a state, you have to extend the state name with a slash ("/") and to add additional lines with `during` or `on` actions to the state name: e.g. extended state name

```
Adams_adventurous_state/
on God_speaking: utter_wish_for_Eve
on Eve_whispering: eaten_apples++
```

defines the state name to be "`Adams_adventurous_state`" and associates two state actions to that state. These state actions are "`utter_wish_for_Eve`", to be issued upon event "`God_speaking`", and increment of variable `eaten_apples` whenever Eve is whispering (and no outgoing transition saves Adam...).

Exercise 2: "Temporal logic events"

A further frequently encountered operation is counting event occurrences in order to perform a certain action only if a certain number of event occurrences has happened, e.g. in dividing frequencies. STATEFLOW offers so-called *temporal logic events* for this purpose (thereby actually abusing the term "temporal logic"). You may use

`after(`*number*`,`*event*`)`

as an event in the trigger condition of a transition, where *number* is a natural number (or an expression evaluating to such) and *event* is an event name. The semantics is that the corresponding transition will be enabled iff at least *number* instances of *event* have occurred since entering the transition source state and the remaining part of the trigger condition is true. E.g. a transition with label

```
after(4,Calls_from_God) / march
```

will be enabled after seeing four `Calls_from_God`, yet only if the chart was *stably* in the source state of the transition during those four calls.[2]

[2] Note that it is here where `during` or `on` actions behave notably different from self-loops: self-loops leave a state (thus resetting the "count" underlying the `after` event) while `during` or `on` actions don't leave the state.