Introduction to Stateflow

Martin Fränzle

STATEFLOW is a tool belonging to the Matlab/Simulink family of products. It extends Simulink's quite rudimentary means for modelling state machines doing discrete-state control and decision making. Building on a variant of Harel's statechart notation [2], a STATEFLOW model uses a syntactically sugared version of transition diagrams to represent event-triggered, i.e. reactive, computational processes. The major extensions to classical transition diagrams are hierarchy (i.e., state nesting), orthogonality (a.k.a. concurrency or parallelism), and symbolic variables as found in conventional programming languages. Closely related statecharts notations are used in the STATEMATE high-level modelling toolbox and in the UML for behavioural modelling [1], albeit with notably different semantics. Furthermore, STATEMATE as well as the UML are primarily targeted towards control-dominated applications, whereas STATEFLOW provides a seamless integration with Simulink such that the control-oriented view of statecharts and the dataflow-oriented view of Simulink blocks can be freely mixed, as perceived appropriate.

Exercise 1: Getting started with STATEFLOW

This tutorial-type exercise shall give you a basic introduction to the use of the STATEFLOW tool and the syntax and semantics of STATEFLOW statecharts. It does so by guiding you through the design of a simple stop-watch.

1.1 Starting STATEFLOW

In order to start STATEFLOW, you first have to start Matlab by entering

```
matlab &
```

in a shell. Then enter

```
stateflow
```

at the Matlab prompt.

In response, two windows pop up. One is the STATEFLOW block library (named "sflib"), which contains an untitled STATEFLOW block icon and a STATEFLOW examples block. The other window (named "untitled" displays an untitled Simulink model containing exactly one untitled STATEFLOW block.

The STATEFLOW block is ready to accommodate a state machine, which can be embedded into the context of a Simulink model by drawing the latter on the surrounding canvas. Should you need more than one state machine in your Simulink model then you can obtain further STATEFLOW blocks by dragging them from the STATEFLOW block library onto

the canvas. Analogously, you could drag Simulink blocks from the Simulink palette onto the canvas in order to embed the state machine into an environment model or to provide glue components between multiple STATEFLOW blocks. For the stop-watch, however, one STATEFLOW block plus a quite trivial environment — we will come to that later — suffices.

For the moment, you complete your work at the Simulink level by naming the untitled STATEFLOW block and the untitled overall model: You label the STATEFLOW block by clicking in the text area underneath the block and replacing the text "Untitled" with a decent name, e.g. "Stop-watch". You name the encompassing Simulink model by choosing "`Save As`" from the File menu of the Simulink model window and entering an appropriate file name. File names for models should end in "`.mdl`"; the prefix of the file name with the extension "`.mdl`" stripped of yields the model name.

1.2 Creating the state chart

Currently, the statechart in the STATEFLOW block of your Simulink model is empty. In order to create a corresponding chart, you have to open the STATEFLOW editor on your STATEFLOW block by double-clicking into that block. A window as in Fig. 1 pops up. The topmost button

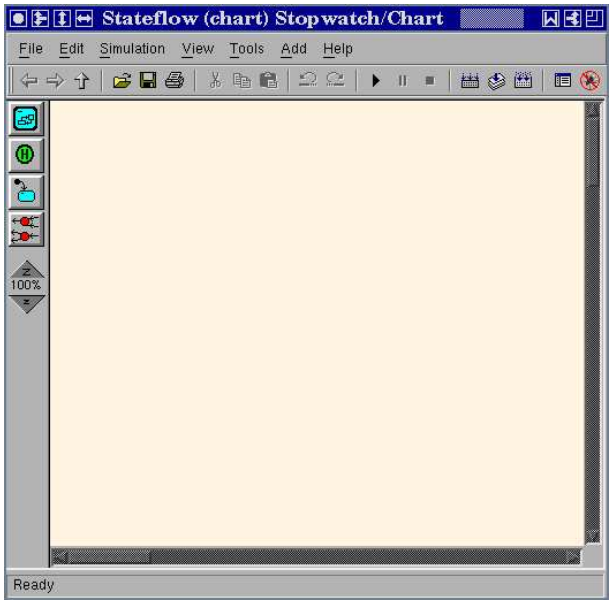

Figure 1: The STATEFLOW editor window

on the left side is the "state tool", which you must select (by left-clicking it once) in order to drag a state onto the canvas. You can either drag it directly by holding the mouse button pressed, or you can select the "state tool" by a single left-click and place the state by moving the cursor into the drawing area and left-click to place the state. Once you dragged a state, you can (and should) name it using the keyboard. You may resize states by dragging at their corners.

Drawing transitions between states is done by first placing the cursor at a straight borderline of the source state such that the cursor changes to crosshairs. Then click-drag the mouse to the border of the destination state. When the transition snaps to the border of the

destination state, release the mouse button. The trigger and action parts are added to the transition by labelling them. Therefore, select the transition by clicking it. Then click the "?" character that appears alongside the transition and enter the label. Finally, press the `Esc` key to deselect the transition label and exit edit mode for this label. The syntax of transition labels is — as far as we will need it —

$$event \quad [condition] \quad / \quad action$$

where $event$ can be dropped if no triggering event is needed, $[condition]$ is replaced by the empty string if no condition is needed, and similarly $/action$ is dropped if no action is to be taken upon that transition.

The initial state is marked using the "default transition tool", which is the third button in the left column of the editor window. GUI-wise, its functionality is akin to that of the "state tool": you select it and drag the default transition until it snaps to the state to be marked initial.

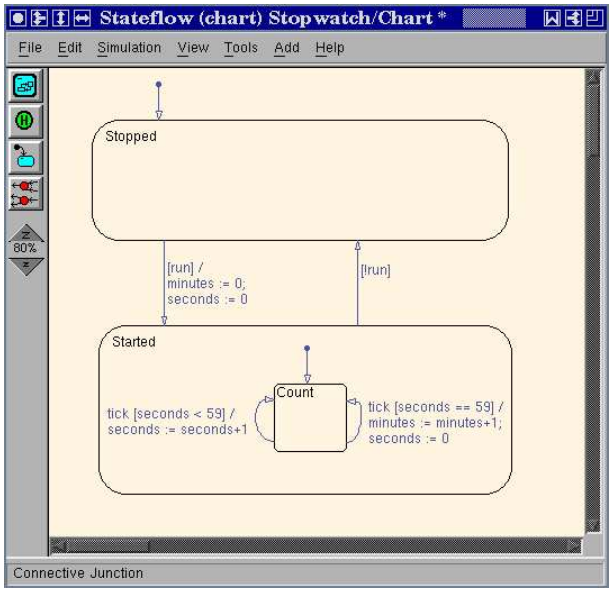

Figure 2: The simple stop-watch

Using these techniques, you should now create a statechart akin to that shown in Fig. 2. Make sure that you understand what this chart is doing.

1.3 Defining input from and output to the Simulink environment

Once we have created the statechart, we have to interface it to its environment, thereby also declaring its globally visible data items.[1] For our model, we have to define an input *event*

[1] A chart may also have local events and local data items that are hidden from the outside and thus inaccessible to the environment. These can be declared using

`Add → Event → Local`

and

`Add → Data → Local`

`tick`, an input *variable* `run`, and two integer variables `minutes` and `seconds` to be output to the environment. To define the input event, select

`Add` → `Event` → `Input from Simulink`

from the STATEFLOW editor's pull-down menu and enter `tick` for the event name. Concerning the `Trigger` type, select `Rising edge` (which is the default) and try the other variants later. For the input variable, select

`Add` → `Data` → `Input from Simulink`

from the STATEFLOW editor's pull-down menu and enter `run` for the name. As a type, you should select `boolean`. For the outputs, select

`Add` → `Data` → `Output to Simulink`

from the STATEFLOW editor's pull-down menu and enter `minutes` (`seconds`, resp.) for the name. As a type, you may select `uint32` for `minutes` and `uint8` for `seconds`. For `seconds`, it makes sense, to additionally limit the range to $0, \ldots, 59$ by adding the according values under "`Limit range`". You can check the success of your operations by selecting

`Tools` → `Explore`

from the STATEFLOW editor's pull-down menu, thus opening a browser for the data dictionary of the STATEFLOW model.

1.4 Adding the Simulink environment

In order to complete our model, we have to embed it into a Simulink environment. Therefore open the Simulink library bowser, e.g. via the Matlab launch pad under

`Simulink` → `Library Browser .`

Select "`Sources`" from the library browser and drag a "`Pulse generator`" to the canvas containing the Simulink model (currently only consisting of the STATEFLOW block). Connect the pulse generator's output to the STATEFLOW block's trigger input (that with the edge symbol on it) by dragging a line. Double click on the pulse generator block in order to change its properties. Reduce the `Period` to 1 second.

Furthermore, drag two constant blocks to the canvas, change their value to 0 and 1 by double clicking into them and editing their properties, and connect them to the input port labelled "`run`" of the STATEFLOW block via a `manual switch` (to be found in the Simulink palette "`Signal routing`", as shown in Fig. 3.

Finally, select "'`Sinks`" from the library browser and drag two "`Display`s" to the Simulink drawing area. Rename the displays to meaningful names, e.g. "`Min`" and "`Sec`", and connect them to the corresponding outputs of the STATEFLOW block by dragging lines. Your model should now resemble Fig. 3.

However, we do not need local variables for this simple model.

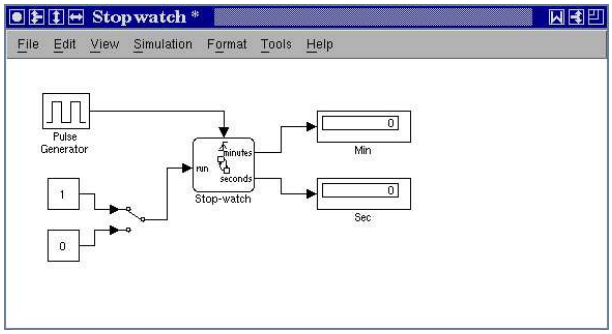

Figure 3: The environment

1.5 Simulating the system

As the simulation algorithm detects stable situation and skips over them, it will terminate immediately whenever the manual switch is set to the 0 ($= \neg$`run`) position. To circumvent this optimisation, select

`Simulation` → `Simulation parameters`

from the pull-down menu and switch "`Solver options, Type`" to `Fixed-step`. Furthermore, change "`Solver options, Fixed step size`" from `auto` to some constant between `1e-4` and `1e-3`, depending on the speed of your computer, in order to slow down the simulation enough to make individual transitions visible.[2] If you furthermore change the "`Stop time`" to `1000` then you have decent time for experimenting with your model during simulation. Now start simulation by selecting

`Simulation` → `Start simulation`

and switch forth and back the run/stop switch of your model by double-clicking on the manual switch symbol while simulation runs. You can stop the simulation by selecting

`Simulation` → `Stop simulation .`

Try out what happens if you over-constrain the range of the data item `minutes` by confining its value range to $0, \ldots, 58$. You can do so by exploring the data dictionary via the browser (activate with `Tools` → `Explore`, or simply `Ctrl-R`, in the STATEFLOW editor window) and editing the appropriate entry.

Exercise 2: Extending the stop-watch

In order to get used to the STATEFLOW tool and to the statechart notation, you shall incrementally extend the stop-watch with additional functionality:

1. add a `lap` switch, allowing you to take lap times by freezing the display while the stop-watch continues to run,[3]

[2] Even though you selected the period of the pulse generator to be 1 second, the simulation does not run in real-time. It runs with a variable speed-up/slow-down, and "1 second" just refers to the virtual time axis used for synchronising different Simulink blocks and for producing x-t diagrams from the simulation.

[3] While this functionality could be implemented by simply disconneting the displays at the Simulink level, you should implement it via statechart mechanisms.

2. add a `store/recall` switch, which stores the current value of the stop-watch to auxiliary memory when the switch is engaged, and which resets the stop-watch to the stored value when it is disengaged. I.e.

   - it invisibly stores the current value x of the stop-watch to some internal memory when the switch is engaged, independently of whether the stop-watch is currently running (in which case it stores the momentary value) or stopped (in which case it stores the stable clock reading),
   - it sets the stop-watch to x when it is disengaged, independently of whether the stop-watch is currently running (in which case it continues to run from x) or stopped (in which case it changes to value x, yet remains stopped).

Make sure that switching above switches does not cause missed updates of the internal counters of the stop-watch. (*Hint:* Use parallel sub-systems. To obtain a parallel composition right-click on the canvas and select `Decomposition` → `Parallel` (`AND`).)

3. remove the reset-to-0 of counter values that is currently coupled to starting the stop-watch. Instead, implement the following reset procedure: the counters are reset to 0 iff a full store-and-recall cycle[4] is performed while the stop-watch is stopped. If the stop-watch is being stopped while the `store/recall` switch is engaged then disengaging the `store/recall` switch should just recall the stored value. It then needs a full further store-and-recall cycle while holding the stop-watch stopped for resetting it to 0.

4. Up to now, your stop-watch reads the switches and updates its state just once every second, giving a notable delay on some operations (e.g., starting the stop-watch — count the number of transitions that happen until the first increment of the internal counters happens). Change you model such that it internally runs with $10\,\mathrm{Hz}$ clock frequency. (*Hint:* Don't forget to enlarge the time step of the simulator; otherwise, you will become very sleepy...)

References

[1] Bruce Powell Douglass and Grady Booch. *Doing Hard Time: Developing Real-Time Systems with UML, Objects Frameworks and Patterns.* Addison-Wesley Publishing Company Inc., 1999.

[2] David Harel and Michal Politi. *Modeling Reactive Systems with Statecharts.* McGraw-Hill Inc., 1998.

[4] by engaging and disengaging the `store/recall` switch