Statechart Modelling for ARGESIM Benchmark C10 'Dining Philosophers Problem II' using Simulink/Stateflow

Voin Legourski, Yilin Huang, Ondrej Cevan, F. Breitenecker, Vienna Univ. of Technology, Austria

Simulator: Simulink is a MATLAB extension allowing rapid and accurate building of computer models of dynamical systems using block diagram notation. With Simulink one can model complex nonlinear systems using continuous and discrete-time components. Particularly important for our comparison is the state-flow extension to Simulink. Stateflow provides a powerful environment with which one can add finite state machines into the Simulink models. It is build around the state-chart formalism.

Model: The system described by the 'Dining philosophers' Problem" does consist of five philosophers who are sitting at a round table, on the table in front of each of them is located a bowl of food. To eat from the bowl a philosopher needs two chopsticks. The problem lies within the sticks, as there are only five sticks available to the philosophers– one between each bowl, two neighbors cannot eat at the same time. The philosophers can be in 3 resp. 4 states: thinking, eating and waiting (waiting for left and right stick; Figure 1- Stateflow model).

If any of the philosophers wants to grab a chopstick a request needs to be sent to the chopstick subsystem in order to check whether the chopstick is available. A chopstick subsystem controls use of the chopsticks, with two inputs and two outputs. The inputs are function calls that trigger two function-call subsystems: *chopstickR* and *chopstickL* (Figure 2). The outputs signal whether the chopstick is available for one of the two philosophers that compete for the chopstick.

The subsystems (*chopstickR/L*) are modeled by a state-chart and if a philosopher calls the subsystem once, the state-chart is activated and checks whether the subsystem is already using the chopstick. The calling philosopher will receive a "ticket" for the chopstick in case the condition returns true, which means that the chopstick can be used. If the philosopher calls the subsystem the second time (while using the stick) the condition *[havingChopstick]* returns the ticket for the chopstick.

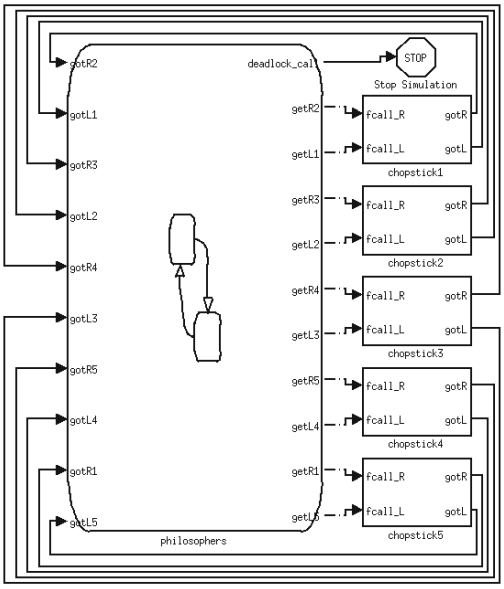

Figure 1. "Dining Philosophers" model, top level

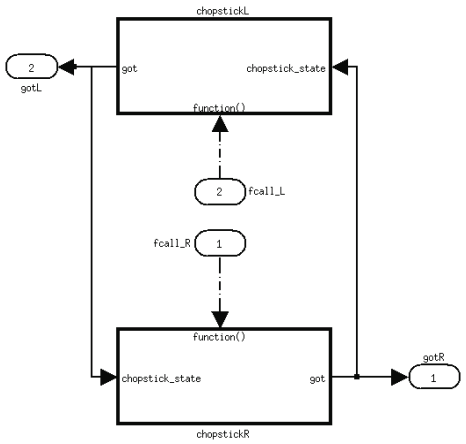

Figure 2. Function call subsystems of a chopstick

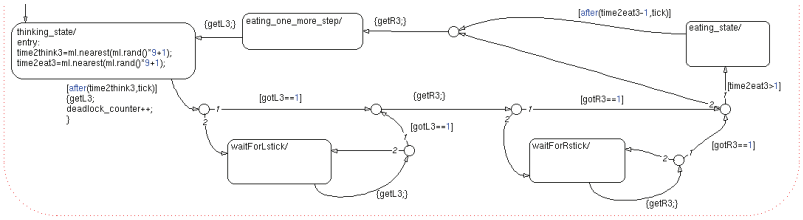

Figure 3. Stateflow diagram of philosopher 3

The philosophers' state-chart consists of one graphical function and one super state called philosophers and which contains six parallel (AND) states (Figure 3). These are independent and can be active at the same time. Five of these states represent philosophers and the sixth one has a deadlock monitoring function.

Although parallel states execute concurrently, they are not activated at the same time. State-flow determines when to activate them during simulation according to priorities. This means, the states are activated in a defined order which can be configured by the user. This feature was used to solve the conflict of simultaneous access. Here the priorities were set in such a way that philosophers to the right side of a stick have higher priority than those to the left of it.

Because the philosophers are sitting in a ring (*philosopher1-philosopher2-philosopher3-philosopher4-philosopher5-philosopher1*), the *philosopher5* has lower priority then *philosopher1* and still needs to win the access to its left chopstick. This was solved by letting *philospher5* pick up its left chopstick one time step before entering "waiting for left stick" state. The same consideration leads us to a very similar solution when our philosophers return their chopsticks.

With this way of assigning and returning chopsticks to/from philosophers we created a model that needs 0 time steps for "housekeeping" in which it is possible to receive both chopsticks during one transition (from "thinking" into the "eating" state) – if both sticks are available. Eating state is divided into two states: *eating_state* and *eating_one_more_step*. Entering and leaving a state takes at least one time step.

A-Task: In order to evaluate the model, we needed to add some more variables into the state-flow charts for measuring time to wait and to export the data to the MATLAB workspace. The chopstick utilization was evaluated with an extra state-flow chart which simply counted the number of time steps a chopstick was used. The simulation was started by MATLAB and stopped if a deadlock was. The simulation ran for 5 356 896 time steps. MATLAB evaluation results in Table 1 and Table 2.

B-Task: In order to perform this task as fast as possible we removed from the model commands needed for the evaluation of time to wait and chopstick utilization. The simulations were executed with a simple FOR loop from the MATLAB environment.

state	p1	p2	p3	p4	p5	all
thinking	5.4935 2.6344	5.5095 2.6300	5.4968 2.6350	5.4961 2.6276	5.5084 2.6306	5.5008 2.6315
waiting	11.7189 7.8350	11.7295 7.8203	11.7178 7.8299	11.7276 7.8398	11.7213 7.8277	11.7230 7.8305
eating	5.5118 2.6265	5.4954 2.6252	5.5045 2.6280	5.4900 2.6303	5.5053 2.6321	5.5014 2.6284

Table 1: Philosophers times in respective states (mean and standard deviation)

c1	c2	c3	c4	c5	all
0.9245	0.9245	0.9241	0.9244	0.9243	0.9244

Table 2: Chopstick utilization (mean)

Before the start, the *time2deadlock* variable was initialized. The resulting times are exported into MATLAB workspace. After performing 50 runs the maximum termination time lay with 21,788,589 time steps, the minimum with 68,994 and the average simulation took 4,666,830 time steps.

A deadlock state can be reached only if all philosophers decide to leave the thinking state at the same time step, and thus grab their left chopsticks concurrently. To detect this, every time a philosopher leaves thinking state the global variable *deadlock_counter* is increased by one. This variable is checked at the end of each time step by the state *deadlock_checking* (the sixth (AND) state in the philosophers super state). If equaling less than five, the value of the variable is reset to 0. In the case all the philosophers increased the variable, *deadlock_counter* equals 5 (equivalent to a deadlock). Thus the simulation is terminated and the current time sent to MATLAB.

Resumé: Stateflow indeed provided a solid base for modeling and performing the tasks of this benchmark. The problems of simultaneous access respectively the correct implementation of accurate access to the chopsticks was achieved by a simple, yet efficient workaround. Simulation results were easily acquired and exported into MATLAB.

F. Breitenecker, Vienna Univ. of Technology, Austria

Received: June 28, 2007
Revised: November 17, 2007; January 18, 2008
Accepted: February 10, 2008