C10 Dining Philosophers II – AnyLogic

Simulation / Process Flow

Simulator: AnyLogic (www.xjtek.com) is a general-purpose simulator for simulation of both discrete and continuous systems. AnyLogic is fully based on the programming language Java and with knowledge of Java one can write own code and extend the features of AnyLogic.

Model: Five philosophers are sitting around a table. They are all going through the same cycles, starting out with a *thinking-phase*, followed by a *hungry state* and then *eating-phase*. The problem is that every philosopher needs two chopsticks to eat, but between the philosophers it is only one available: each philosopher must share chopsticks with his neighbours, leading to simultaneous access to the same chopstick and occurrence of deadlock.

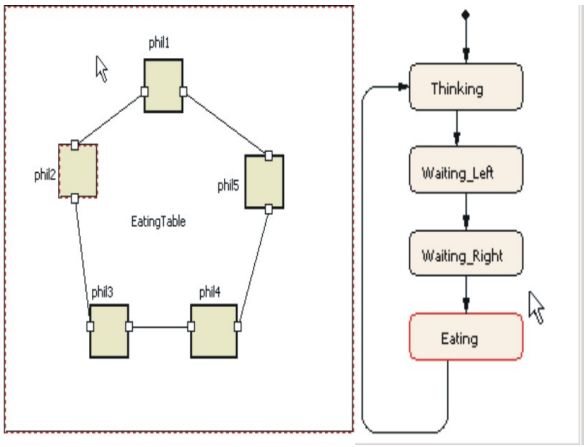

Fig.1: Philosopher objects, communicating with messages to the neighbour (left), state chart for each philosopher (right)

Implementation. We chose to implement our model in an object-oriented manner. First we defined the philosopher object containing a statechart with four different states being: *Thinking*, *Waiting left*, *Waiting right*, *Eating*. Then we defined eating table as the root object with five philosophers as encapsulated objects (Figure 1).

The philosophers are communicating with their neighbours through public ports through which they are sending and receiving message. We defined three message classes: *Left free*, *Right free*, *Block*

The philosophers start with a thinking period, which follows a discrete uniform distribution in the interval [1,10]. When one finishes his thinking period he continues into the waiting left state.

In entering this state he sends a message to his neighbours saying that the chopsticks are free. Him self waits for the same message from his left side. After receiving this message he goes on the waiting right state signalling his right that the chopstick is still free and blocking the left. When he gets the right free signal from his right he starts eating blocking his neighbours. The eating period follows the same discrete distribution as the thinking period after which he returns to the thinking phase.

Task a. Simulation until Deadlock with Utilisation Statistics. The first simulation run, after compiling into Java, stopped with a deadlock at t = 19180; results are shown in the following tables.

	Thinking	Waiting	Eating
Phil1	5.04±2.58	13.98±6.20	5.37±2.54
Phil2	5.07±2.57	13.81±6.12	5.39±2.53
Phil3	5.19±2.51	13.71±6.13	5.25±2.56
Phil4	5.11±2.51	13.71±6.20	5.21±2.58
Phil5	5.19±2.58	13.75±6.16	5.23±2.55
Average	**5.02±2.57**	**13.76±6.25**	**5.29±2.34**

Table 1: Average Times for thinking, waiting and eating for each philosopher and for all together

Ch1	Ch2	Ch3	Ch4	Ch5	ChAll
0.218	0.219	0.217	0.217	0.219	0.2186

Table 2: Average Usage rate of each chopstick and of all chopsticks together

Task b. Handling Simultaneous Access to Chopsticks. AnyLogic handles simultaneous events on a random basis. This means that when two philosophers try to access the same chopstick at the same time, AnyLogic chooses randomly between the two. If the event granted access schedules another event, this event is placed behind the event, which was fighting for access to the chopstick.

Task c. Batch of Simulation Runs with Deadlock Statistics. AnyLogic detects a deadlock when no new events are scheduled and the event list becomes empty. This causes the simulation to stop, in our case when all the philosophers are waiting for the chopstick to the right of them, already having grabbed the left one (results given in Table 3.

Number of runs	50
Maximum time of termination	49465
Minimum time of termination	1417

Table 3: Maximum and minimum deadlock time, batch of 50 simulation runs

Franz Holzer, TU Vienna
fholzer@fsmat.htu.tuwien.ac.at
Henning Nilsen, Univ. Trondheim
nilsennorge@hotmail.com