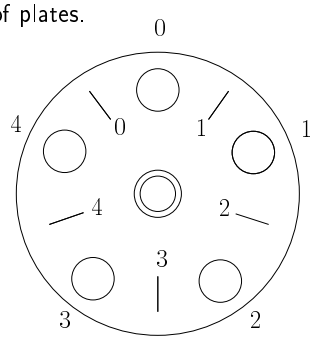
◇ The Dining Philosophers ◇

Five philosophers live in a college; they spend most of their time thinking, but occasionally become hungry. The college has a communal dining room, with a circular table and five chairs. In the middle of the table is a large bowl of rice, and the table is set with five plates. There are also five chopsticks, one between each pair of plates.



When a philosopher is hungry, he enters the dining room, sits down in his chair, picks up the chopsticks on either side of his plate (first the one on the left, then the one on the right), and eats. Two chopsticks are needed to eat rice, so if one of the chopsticks is already in use, he has to wait. When the philosopher has finished eating he puts down the chopsticks, gets up from the chair, and leaves the dining room.

We will model this system in CSP, and analyse its behaviour. The relevant components are the five philosophers, which we will model as processes $PHIL_0 \ldots PHIL_4$, and the five chopsticks, which we will model as processes $CHOP_0 \ldots CHOP_4$.

Using the symbols $\oplus$ and $\ominus$ to denote addition and subtraction modulo 5 (so that $4 \oplus 1 = 0$ and $0 \ominus 1 = 4$), philosopher $PHIL_i$ will sit in seat i and use chopsticks i and $i \oplus 1$.

The alphabet of $PHIL_i$ is

$$\begin{aligned}\alpha(PHIL_i) = \{&sitdown_i, getup_i,\\ &pickup.i.i, pickup.i.(i \oplus 1),\\ &putdown.i.i, putdown.i.(i \oplus 1)\}\end{aligned}$$

In the events $pickup.i.i$ etc. the "." is being used purely as a symbol.

The event $pickup.i.i$ represents $PHIL_i$ picking up chopstick i, and so on.

We will ignore the actions of eating, thinking, and entering and leaving the dining room.

Because the alphabets of the processes $PHIL_i$ are mutually disjoint, there can be no direct interaction between the philosophers. The only way in which they affect each other will be as a consequence of the fact that they are competing for access to the chopsticks.

The relevant events for the chopsticks are the $pickup$ and $putdown$ events. $CHOP_i$ can potentially be picked up or put down by either $PHIL_i$ or $PHIL_{i\ominus 1}$.

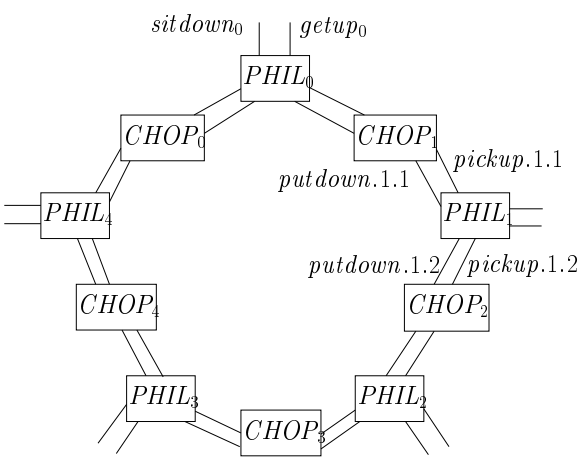
$$\begin{aligned}\alpha(CHOP_i) = \{&pickup.i.i, pickup.(i \ominus 1).i,\\ &putdown.i.i, putdown.(i \ominus 1).i\}\end{aligned}$$

We will define the system as a concurrent combination of the philosophers and the chopsticks.

$$PHILS = \|^{i \in 0..4}\ PHIL_i$$

$$CHOPS = \|^{i \in 0..4}\ CHOP_i$$

$$COLLEGE = PHILS \,\|\, CHOPS$$



Each process $PHIL_i$ simply cycles through a sequence of six events:

$$\begin{aligned}PHIL_i = {}& sitdown_i \rightarrow pickup.i.(i \oplus 1) \rightarrow pickup.i.i \rightarrow\\ & putdown.i.(i \oplus 1) \rightarrow putdown.i.i \rightarrow\\ & getup_i \rightarrow PHIL_i\end{aligned}$$

Each process $CHOP_i$ can be repeatedly picked up and put down, but there is a choice of who picks it up:

$$\begin{aligned}CHOP_i = {}& pickup.i.i \rightarrow putdown.i.i \rightarrow CHOP_i\\ \Box\ & pickup.(i \ominus 1).i \rightarrow putdown.(i \ominus 1).i \rightarrow CHOP_i\end{aligned}$$

Now we can look at the possible behaviour of $COLLEGE$. Suppose all the philosophers sit down in order, and then each one picks up the chopstick to his left.

What can happen next? Each $PHIL_i$ can only do $pickup.i.i$, which requires synchronisation with $CHOP_i$. However, $CHOP_i$ has just done $pickup.(i \ominus 1).i$ and therefore can only do $putdown.(i \ominus 1).i$ next. This means that there is no possible next event for $COLLEGE$. We have a deadlock.

How can we modify $COLLEGE$ to remove the possibility of deadlock? There are a number of obvious but unsatisfactory ideas.

- ◇ Provide two chopsticks for each philosopher. But if the chopsticks represent scarce resources, this may not be feasible.
- ◇ Provide a single extra chopstick, in the middle of the table, which can be used by any of the philosophers. Similarly, this may not be feasible.
- ◇ Modify the definition of just one of the philosophers, so that the chopsticks are picked up in the opposite order. This will work (although it takes some thought to be sure) but it breaks the symmetry of the system.

Instead we will try to control the way in which the philosophers sit down, the idea being that if only 4 philosophers are seated at any one time, then even if everyone picks up the left chopstick, one philospher will be sitting on the left of an empty place, and can pick up the chopstick to his right.

As we have seen, the behaviour of a system can be controlled by adding another process in parallel, and taking advantage of the fact that certain events require synchronisation.

We can define a process $BUTLER$ with alphabet

$$\alpha(BUTLER) = D \cup U,$$

where

$$\begin{aligned} D &= \{sitdown_0, \ldots, sitdown_4\} \\ U &= \{getup_0, \ldots, getup_4\}, \end{aligned}$$

to control the sitting down and getting up of the philosophers. $BUTLER$ is defined in terms of auxiliary processes $BUTLER_0, \ldots, BUTLER_4$, all with alphabet $\alpha(BUTLER)$.

$$\begin{aligned} BUTLER_0 &= x : D \to BUTLER_1 \\ BUTLER_i &= x : D \to BUTLER_{i+1} \\ &\quad \Box\ y : U \to BUTLER_{i-1}\ \ 1 \leqslant i \leqslant 3 \\ BUTLER_4 &= y : U \to BUTLER_3 \\ BUTLER &= BUTLER_0 \end{aligned}$$

The notation in the second line is shorthand for

$$BUTLER_i = z : (D \cup U) \to P(z)$$

where

$$\begin{aligned} P(z) &= BUTLER_{i+1} \text{ if } z \in D \\ &= BUTLER_{i-1} \text{ if } z \in U. \end{aligned}$$

Now we can define

$$NEWCOLLEGE = COLLEGE \parallel BUTLER$$

△ Convince yourself that $NEWCOLLEGE$ does not deadlock. How formal can you be?

We could consider checking the entire state space of the system, to discover whether or not it can deadlock. Since each philosopher has 6 states and each chopstick has 3 states, the total number of possible states of $COLLEGE$ is $6^5 \times 3^5$, or about 1.8 million, though not all of these will be reachable (since the states of the chopsticks must be consistent with the states of the philosophers). Since the effect of $BUTLER$ is to restrict the number of states, this is also a limit on the number of states of $NEWCOLLEGE$. Systems of this complexity are within the scope of current software tools such as FDR.

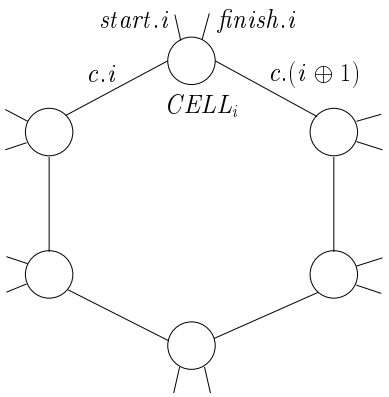
◇ The Cyclic Scheduler ◇

Suppose there are a number of processes which we need to control. Each process can be started by a $start$ event and uses a $finish$ event to indicate that it has finished. Suppose also that we want to start the processes in order, returning to the first when the last has been started; when a process has finished it can be started again, but only when its turn comes round.

We will define a *scheduler*, which uses $start$ and *finish* events to control the processes. The scheduler will be implemented as a collection of cells (processes), each of which communicates with one of the processes being controlled, and also with other cells.

The next slide has a diagram of the case where there are 6 processes to control. $\oplus$ denotes addition modulo the number of processes.

The idea is that each cell waits for a signal on the c channel to its left, which means that the process to its left has been started. Then the cell starts its own process, and sends a signal on the c channel to its right, to tell the next cell that it can start its process. Also, each cell has to wait for its process to do $finish$ before starting it again.



In order to start everything off, one cell must begin by starting its process instead of waiting.

$$\begin{aligned} STARTCELL_i \;=\; & start.i \rightarrow c.(i \oplus 1) \rightarrow \\ & (\;\; finish.i \rightarrow c.i \rightarrow STARTCELL_i \\ & \Box\;\; c.i \rightarrow finish.i \rightarrow STARTCELL_i) \end{aligned}$$

The other processes wait for $c.i$

$$WAITCELL_i = c.i \rightarrow STARTCELL_i$$

It is convenient to define

$$\begin{aligned} CELL_0 &= STARTCELL_0 \\ CELL_i &= WAITCELL_i \quad (i > 0) \end{aligned}$$

and then

$$\begin{aligned} SCHED \;=\; & (\|^{0 \leqslant i < n}_{\{start.i, finish.i, c.i, c.(i \oplus 1)\}} CELL_i) \setminus \\ & \{c.i \mid 0 \leqslant i < n\} \end{aligned}$$

There are three properties which we would like to verify for the scheduler. The first is that for each i, the events $start.i$ and $finish.i$ happen alternately, beginning with $start.i$. The second is that the events $start.0, \ldots, start.(n-1)$ happen in the correct cyclic order. The third is deadlock-freedom.

For the first property, we can define a process specifying alternation of $start$ and $finish$ for each cell:

$$ALT_i = start.i \rightarrow finish.i \rightarrow ALT_i$$

and combine them in parallel to produce a specification for the scheduler as a whole.

$$ALTSPEC = \|^{0 \leqslant i < n}_{\{start.i, finish.i\}} ALT_i$$

In this parallel combination the alphabets are all disjoint, and no synchronisation is required. It is simply an independent parallel combination of the ALT processes.

The specification

$$ALTSPEC \sqsubseteq_T SCHED$$

can be checked with FDR.

For the second property, define

$$CYCLE_i \;=\; start.i \rightarrow CYCLE_{i \oplus 1} \;\; (0 \leqslant i < n)$$

and specify

$$CYCLE_0 \sqsubseteq SCHED \setminus \{finish.i \mid 0 \leqslant i < n\}.$$