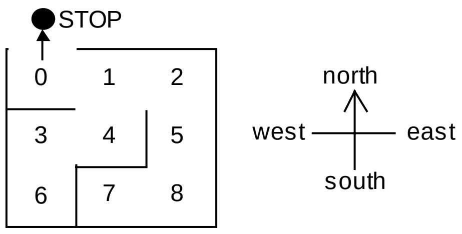
Chapter 6 - Solutions to selected exercises

Exercises

6.1 The figure below depicts a maze. Write a description of the maze in FSP which, using deadlock analysis, finds the shortest path out of the maze starting at any square.



for solution see slides

6.2 One solution to the Dining Philosophers problem permits only four philosophers to sit down at the table at the same time. Specify a `BUTLER` process which, by counting, when composed with the model of section 6.2 permits a maximum of four philosopher to engage in the `sitdown` action before an `arise` action occurs. Show that this system is deadlock-free.

Solution:

```
BUTLER(N=5)    = BUTLER[0],
BUTLER[i:0..N-1] =
   (when (i<N-1) phil[0..N-1].sitdown -> BUTLER[i+1]
   |when (i>0)   phil[0..N-1].arise   -> BUTLER[i-1]
   ).

||BUTLER_DINERS = (BUTLER||DINERS).
```

Chapter 6 - Solutions to selected exercises

6.3 Using the Java timed wait primitive:

```
public final void wait(long timeout)
                     throws InterruptedException
```

modify the `Fork` monitor such that after a wait of 1 second, the call to `get` times out and returns the result false. The `Philosopher` should release the other fork if it holds it and try again if the timeout occurs. Observe the behavior of the resulting system.

```
class Fork {

    private boolean taken=false;
    private PhilCanvas display;
    private int identity;

    Fork(PhilCanvas display, int identity) {
        this.display = display;
        this.identity = identity;
    }

    synchronized void put() {
        taken=false;
        notify();
    }

    synchronized boolean get() throws
            java.lang.InterruptedException {
        if (taken) wait(1000);
        if (taken) //still held after wait of 1 second
             return false;
        else {
             taken=true;
             return true;
        }
   }
}
```