

## Notes for Lecture 15, 3/1/11, CS 6110

**Experiments with an unfair mutex:** In `mutex-unfair.prm` the individual progress claim (expressed by the `never` automaton) fails even under weak fairness. This is because one of the `myproc` processes can get infinitely often disabled, thus requiring nothing of weak fairness.

**Experiments with a fair mutex:** In `mutex-fair.prm` the individual progress claim (expressed by the `never` automaton) succeeds under weak fairness. This is because no process that makes a mutex request gets disabled by the other process.

**Experiments with dl1.prm:** Notice the *really sneaky* initialization of the probable-owner graph! This is hard to understand, relying on Promela's default 0 initialization which makes the PO graph to be sink directed at node 0. Oh well, let's live with that for now.

This protocol does not pass the assertion that infinitely often *some* lock is granted—even under weak fairness. Again the “flickering” mutex variable nullifies whatever weak fairness was going to deliver.

**Experiments with dl2.prm:** To understand sharply what was going on, I separated out

```
q_len_ch[me] ? [count];
```

from

```
atomic {q_len_ch[me] ? [count] && mutex[me]==0 ->
```

counting on the fact that `q_len_ch[me]` is drained only by **this** process. Still no luck – the flickering mutex killeth.

**Experiments with dl3.prm:** OK a mega precondition is slapped:

```
( ([<m1]&&([<m2]&&([<m3] ) -> ( ([<11] && ([<12] && ([<13] )
```

Now things pass. But who knows whether I've restricted the behavioral space of interest too much? Hence, need to probe more!

**Experiments with dl4.prm:** I now embark on preconditioning with something meaningful. It is good to run the precondition by itself and see what happens:

```
[ ](qlengt -> <> pomeisme
```

Obviously this formula can also gloriously fail.

```
#define qlengt (len(q_len_ch[0])>0) #define pomeisme (po[0]==0)
```

**Experiments with dl5.prm:** This is a variant focusing on node 1.

**Experiments with dl6.prm:** Tried

```
[ ](a -> <> b) -> [ ](c -> <> d)
```

where

```
#define a (len(q_len_ch[1])>0)
```

```
#define b (po[1]==1)
```

```
#define c (locked[0]==0)
```

```
#define d (locked[0]==1)
```

Even then, no luck!

**Thing for you to do!:** Realize nature of priority inversion and add priority to `acquire` through the *least bit of change*. Bring your result to next week's class (3/8/11) and discuss your solutions in class!