Concurrency Theory, Assignment Lecture 5.5

Krasimir Georgiev

September 24, 2015

The following PGLEc with MSP program `Init` initializes x and y to 0 and creates the semaphores sx and sy.

```
x = 0;
sx = new;
y = 0;
sy = new;
```

The following PGLEc with MSP program `X` performs the assignment $x = 2y+7$ by first locking the semaphores sx and sy in that order, performs the actions and then unlocking the semaphores sy and sx in that order. Note that because the else-branches are jumps to the corresponding lock check, the program first actively waits until sx becomes available, then actively waits until sy becomes available. This ensures that in the inner block the values of x and y will be stable assuming all concurrent agents follow similar guard pattern.

```
L0;
+ sx.+lock {;
    L1;
    + sy.+lock {;
        incr x y;
        incr x y;
        incr x 7;
        sy.-lock;
    }{;
        ##L1;
    };
    sx.-lock;
}{;
    ##L0;
};
```

The following PGLEc with MSP program `Y` performs the assignment $y = 3x + 5$ by first locking the semaphores sx and sy in that order (!), performs the actions and then unlocking the semaphores sy and sx in that order.

```
L0;
+ sx.+lock {;
    L1;
    + sy.+lock {;
        incr y x;
```

```
        incr y x;
        incr y x;
        incr y 5;
        sy.-lock;
    }{;
        ##L1;
    };
    sx.-lock;
}{;
    ##L0;
};
```

Note that to ensure deadlock-free execution, all concurrent programs should lock the semaphores in the same order with respect to some global ordering of the semaphores. For example, if in `Y` the order of locking the semaphores was sy, sx instead, then the following scenario leads to deadlock: `X` locks sx, `Y` locks sy; now `X` actively waits for sy which is locked by `Y` and symmetrically `Y` actively waits for sx which is locked by `X`.

The source code of this assignment can be found on GitHub.