Homework 1 - Theory

Patrick Collins

January 21, 2014

2. Identify safety and liveness properties.

   (a) Liveness. The "good thing" is being served.

   (b) Liveness. The "good thing" is receiving the ball that has beeen thrown.

   (c) Safety. The "bad thing" is waiting for longer than it takes for the first thread to exit its critical section.

   (d) Safety. The "bad thing" is waiting more than one second to print a message.

   (e) Liveness. The "good thing" is priting a message.

   (f) Safety. The "bad thing" is the cost of living decreasing.

   (g) Liveness. The "good thing" is taxes being paid and resources being freed by death.

   (h) Safety. The "bad thing" is to be identified as something other than a Harvard man.

3. Design a producer-consumer protocol with interrupts.
   Alice and Bob each have a can on their windowsill. Alice does the following:

   (a) She waits until her can is down.

   (b) She releases the pets.

   (c) When the pets return, Alice checks whether they finished the food. If they did, she resets her can and knocks over Bob's.

   Bob does the following:

   (a) He waits until his can is down.

   (b) He puts food in the yard.

   (c) He resets his can.

   (d) He pulls the string and knocks Alice's can down.

4. Design a winning strategy, when...

   - You know the initial state of the switch.
     Designate one prisoner as a counter, who thinks of a number $x = 0$. Whenever a prisoner except for the counter enters the cell for his second time, he turns the light on. If a prisoner other than the counter enters the cell and the light is on, he leaves it on, and if he was supposed to turn it on that time, he remembers to turn it on the next time. When the counter enters the cell and the light is on, he turns it off and increments the number x that he is thinking of. When the counter is thinking of P, he announces that every prisoner has visited the room at least once.
   - You do not know the initial state of the switch.
     The same as in the original case, except the counter waits until $x = P + 1$.

5. Design a strategy that allows $P - 1$ of P prisoners to be freed.
   The first prisoner determines the parity of blue hats in front of him. If it is even, he says "Red", otherwise he says "Blue." Every other prisoner then determines the parity of blue hats in front of him. If the parity is even, he starts to think of the same color that the first prisoner said, if it is odd, he thinks of the opposite color. Whenever a prisoner hears an earlier prisoner say "Blue," he switches the color that he is thinking of to the opposite color. Whenever it is a prisoner's turn to answer, he says the color that he is thinking of. Then, every prisoner except for the first one will always correctly guess the color of his hat.

7. Solve for S_n.

$$
\begin{aligned}
S_2 &= \frac{1}{1 - p + \frac{p}{2}} \\
p &= 2 - \frac{2}{S_2} \\
S_n &= \frac{1}{1 - (2 - \frac{2}{S_2}) + \frac{2 - \frac{2}{S_2}}{n}} = \frac{1}{\frac{2}{S_2} + \frac{2 - \frac{2}{S_2}}{n} - 1}
\end{aligned}
$$

8. When is the multiprocessor superior?
   Plugging in the number of processors to Amdahl's law, the multiprocessor is superior when:

$$\frac{1}{(1 - p) + \frac{p}{10}} = \frac{1}{10 - 9p} > 5$$

   so the multiprocessor solution is superior for applications that are at least $88.\overline{8}\%$ parallel.

11. Does/Is the `Flaky` lock...

(a) satisfy mutual exclusion?
Yes. For any two threads A and B, note that:

$$W_A(turn = A) \rightarrow R_A(busy == false) \rightarrow W_A(busy = true) \rightarrow R_A(turn == A) \rightarrow CS_A$$
$$W_B(turn = B) \rightarrow R_B(busy == false) \rightarrow W_B(busy = true) \rightarrow R_B(turn == B) \rightarrow CS_B$$

Suppose, for contradiction, that there exist some executions i, j such that $CS_A^i \not\rightarrow CS_B^j$ and $CS_B^i \not\rightarrow CS_A^j$. Suppose, without loss of generality, that A is about to enter its critical section. Then it must be the case that $R_A(turn == A)$ has just occurred. If any other thread B is about to enter its critical section, then it must be the case that:

$$W_A(turn = A) \rightarrow R_A(turn == A) \rightarrow R_B(turn == B)$$

※

(b) deadlock free?
No. With any number of threads, the following sequence of events can occur:

- The lock is initialized, and `busy = false`.
- Each thread executes line 5-9, i.e. each thread leaves the inner loop before `busy = true`.
- Each thread except for the thread whose ID is currently stored in `turn` returns to the top of the loop and blocks at line 9.
- The remaining thread returns to the top of the loop and blocks at line 9.

The algorithm will now no longer advance.

(c) starvation free?
No, because deadlock implies starvation.

14. Modify `Filter` to be an ℓ-exclusion algorithm.

```
class Filter implements Lock {
  int[] level;
  int[] victim;
  public Filter(int n, int L) {
    level = new int[n];
    victim = new int[n - L];
    for (int i = 0; i < n; i++) {
      level[i] = 0;
    }
  }
  public void lock() {
    int me = ThreadID.get();
    for (int i = 1; i < n - L; i++) {
```

```
        int num_conflicts;
        level[me] = i;
        victim[i] = me;

        while (victim[i] == me) {
           num_conflicts = 0;
           for (int k = 0; k < n; k++){
              if((k != me) && (level[k] >= i)){
                 if(++num_conflicts >= L) return;
              }
           }
        }
     }
  }
  public void unlock() {
     int me = ThreadID.get();
     level[me] = 0;
  }
}
```

15. `FastPath` neither provides mutual exclusion nor is starvation free.

    - `FastPath` does not provide mutual exclusion because if two threads read line 8 at the same time, both will enter their critical sections: one by going through `lock.lock` and the other by exiting `FastPath`'s `lock` method.
    - `FastPath` is not starvation free, because it could be the case that every time one thread gets a chance to execute an instruction, it is stuck on line 8, and other threads have set `y` to a value other than -1.

16. Prove that...

    - At most one thread gets the value `STOP`.
      Suppose some thread A just executed line 14 and it is about to return `STOP`. Then it must be the case that `last == i` and `goRight == true`. All threads which have not yet executed line 11 will return `RIGHT`. All other threads which have executed line 11 return `DOWN`, since `last == i` and no thread can change the value of `i` if it has already executed line 11.
    - At most $n-1$ threads return `DOWN`.
      If some thread has already returned `STOP`, then no more than $n-1$ threads will return `DOWN`, since each thread returns only once. If no thread has yet returned `STOP`, then every thread must have already executed lines 11-13. One of these threads will be the thread such that `last == i`, and this thread will return `STOP`. Therefore no more than $n-1$ threads will return `DOWN`.

- At most $n-1$ threads return `RIGHT`.
  If any thread returns `RIGHT`, then it must be that `goRight == true`, and so at least one thread must have executed line 13, and at least one thread cannot return `RIGHT`. Therefore at most $n-1$ threads return `RIGHT`.