Homework 2

You can download a PDF of this assignment here.

Instructions

You may work in groups of up to 4 and submit a single assignment for the group. For computational problems, please show your work; for conceptual questions, please explain your reasoning. Solutions may be neatly hand-written and scanned or typeset. Please submit your solution to Moodle **in PDF format**.

Due: Friday, March 19, 23:59 AoE

Exercises

Exercise 1. Consider the following `Bouncer` object:

```
class Bouncer {
    public static final int DOWN = 0;
    public static final int RIGHT = 1;
    public static final int STOP = 2;
    private boolean goRight = false;
    private int last = -1;
    int visit () {
        int i = ThreadID.get();
        last = i;
        if (goRight)
             return RIGHT;
        goRight = true;
        if (last == i)
            return STOP;
        else
            return DOWN;
    }
}
```

Suppose n threads call the `visit()` method. Argue that the following hold:

1. At most one thread gets the value `STOP`.
2. At most $n-1$ threads get the value `DOWN`.

3. At most $n-1$ threads get the value `RIGHT`.

Exercise 2. So far in this course, we have assumed that all threads have IDs that are reasonably small numbers. In Java, however, thread IDs can be arbitrary `long` values (see `getId()` documentation). In this exercise, we will see how to use `Bouncer` objects as above to create unique IDs that are reasonably small compared to the number of threads.

Consider a 2D triangular array of `Bouncer` objects arranged as follows:

{: width="50%"}

Suppose each thread performs the following procedure: All threads start by calling `visit()` on `Bouncer` 0. Whenever a thread visits a `Bouncer`, if the `Bouncer` returns `STOP`, the thread adopts the number of the `Bouncer` as its ID. If `DOWN` is returned, the thread then visits the `Bouncer` below; if `RIGHT` is returned, the thread visits the `Bouncer` to the right.

1. Show that for a sufficiently large array of `Bouncer`s, every thread will eventually `STOP` at some `Bouncer`, thereby adopting its ID.
2. If the number n of threads is known in advance, how many `Bouncer`s are required to ensure that all threads adopt an ID?

Exercise 3. Consider the following histories of executions of read/write registers (variable), `r` and `s`. Please explain your answers to the following questions.

{: width="100%"}

1. Restricting attention *only* to register `r`, is the execution sequentially consistent? Linearizable?

2. Restricting attention *only* to register `s`, is the execution sequentially consistent? Linearizable?

3. Is the entire execution (including both registers) sequentially consistent? Linearizable?