Parallel Programming Languages and Systems, Assignment 1

s1140740

1 Question 1

If we were to list the instructions that get executed on our machine they would look something like this:

```
# int x = 10, y = 0;
STORE(x=10);
STORE(y=0);
# compare x and y to see if we should continue looping
READ(x);
READ(y);
CMP(x, y);
# x becomes 9
READ(x);
STORE(x=x-1);
# compare x and y to see if we should continue looping
READ(x);
READ(y);
CMP(x, y);
# x becomes 8
READ(x);
STORE(x=x-1);
...
```

`P1` one would execute the loop continuously while `x` is not equal to `y` and `P2` would busy wait until `x` is equal to `y`.

When `x` becomes 0, that is when `x = x - 1` is executed and `x = 0` is stored in memory from `P1`, there are 3 possibilities of which STORE instructions of `P2` get executed before `x` and `y` are read in `P1` to compare `x` and `y` as the condition-checking part of the while loop:

1. **`P2` has executed both `x = 8` and `y = 2`**

   - blah blah

2. **`P2` has executed neither `x = 8` nor `y = 2`**

3. **`P2` has executed `x = 8`**

There are four possibilities:

1. **`P1` reads its own value of `x` and `y`**

   Thus, `x = 0` and `y = 0`. Since `0 == 0`, `P1` will break out of the loop. Depending on when the value of `y` from `P2` is stored in memory this program will terminate with three different values of `y`. Moreover, there is no dependency on `x` in the rest of `P1`, thus its value will be the same in all three cases. After the loop is exited what remains in `P1` is to read the value of `y`, increment it, and store it in memory:

   ```
   READ y
   INCREMENT y
   STORE y
   ```

   Since only the `READ` and the `STORE` are memory operations there are three possible times when value of `y` from `P2` can be stored:

   1. Before the `READ`

      `P1` is going to read value of `y` from `P2`, increment it by one and store it. Thus the program will terminate with `x = 8` and `y = 3`.

   2. Between the `READ` and the `STORE`

      `P1` is going to read its own value of `y`, that is 0, increment it to 1 while `P2` stores 2 into `y`, and then store 1 into `y`. Thus the program will terminate with `x = 8` and `y = 1`.

   3. After the `STORE`

      `P1` is going to read its own value of `y`, that is 0, increment it to 1, and store 1 into `y`. After this, `P2` stores 2 into `y`. Thus the program will terminate with `x = 8` and `y = 2`.

2. **`P1` reads its own value of `x` but value of `y` from `P2`**

   Thus, `x = 0` and `y = 2`. Since all instructions of `P2` have executed, only `P1` remains active. Further, since `0 != 2`, `P1` will decrement `x` until `x == y`. However, since `P2` is done and the only other write to `y` is outside of the loop this condition will never become true. Assuming no overflow, `P1` will decrement `x` all the way to negative infinity and the program will never terminate.

3. **`P1` reads its own value of `y` but value of `x` from `P2`**

   Thus, `x = 8` and `y = 0`. Since `8 != 2`, `P1` will enter the loop and start decrementing `x`. Depending on when the value of `y` from `P2` is stored in memory this program will behave differently:

   1. `P2` stores `y = 2` before `x` reaches 2

      `P1` will decrement `x` until it becomes 2 when it will exit the loop, increment the value of `y` to be 3 and the program will terminate with `x = 3` and `y = 2`.

   2. `P2` stores `y = 2` after `x` reaches 2 but before `P1` breaks out of the while loop

      Hence, `x < y` and thus `P1` will decrement `x` to negative infinity and the program will never terminate.

   3. `P2` stores `y = 2` after `P1` breaks out of the while loop

      This way `P1` is in the same situation as in case 1 (except `x` is 0 now) and depending on when `P2` stores `y` we get three possible outcomes, each with different value of y:

      1. Before the `READ`

         Program terminates with `x = 0`, `y = 3`.

      2. Between the `READ` and the `STORE`

         Program terminates with `x = 0`, `y = 1`.

      3. After the `STORE`

         Program terminates with `x = 0`, `y = 2`.

4. **`P1` reads values of `x` and `y` from `P2`**

   Thus, `x = 8` and `y = 2`. Since all instructions of `P2` have executed, only `P1` remains active. Further, since `8 != 2`, `P1` will decrement `x` until `x == y`, that is until both `x` and `y` are equal to 2 (`P2` is done and there is no one to modify value of `y`). At that point it will exit the loop. Last instruction of `P1` is `y = y + 1` and hence the program will terminate with `x = 2` and `y = 3`.