Exercise 12

Exercise 1

```
1  // compute input[0]+...+input[9];
2  int sum(int *input, bool may_overwrite){
3    // allocation site of input: A
4    int n=10;
5
6    // prepare output
7    int *output;
8    if (may_overwrite){
9      output=input;
10   }else{
11     output=new int[n]; // allocation site: B
12   }
13
14   // fill up output
15   output[0]=input[0];
16   int half=n/2;
17   fork{
18     output[0]=input[0];
19     for (int i=1;i<half;i++){
20       int val=output[i-1]+input[i];
21       output[i]=val;
22     }
23   }
24   fork{
25     output[half]=input[half];
26     for (int j=half+1;j<n;j++){
27       int val=output[j-1]+input[j];
28       output[j]=val;
29     }
30   }
31   join;
32
33   return output[half-1]+output[n-1];
34 }
```

Listing 1: Parallel code summing up entries of an array

Consider the function `sum` in Listing 1. It sums up the first `n` elements of `input`, where `n=10`. In this task, we want to verify that `sum` does not induce non-determinism due to the use of multiple threads.

1. Run the flow-insensitive pointer-analysis (see previous lectures) to determine the content of the pointers `input` and `output`. Assume that `input` was allocated at allocation site A, and that line 11 corresponds to allocation site B.

2. Determine the possible values of `n`, `half`, `i` and `j` using abstract interpretation (use the interval domain).

You may summarize lines that contain the same abstract state.

3. Check that the multiple threads do not induce non-determinism using the abstract conflict-free checker from the lecture. It suffices to only consider the abstract states on the lines `18`, `20`, `21`, `25`, `27` and `28`, as they are the only lines that interact with variables shared across threads and that may occur in parallel.

Exercise 2

```
1  void voting(){
2    // BLOCK: A
3    int yes=0;
4    int no=0;
5
6    int n_people=2;
7
8    for (int i=0;i<n_people;i++){
9      fork{
10       // BLOCK: B-i-init
11       int decision = rand() % 2; // decision is 0 or 1
12       if (decision==0){
13         int tmp=no+1; // BLOCK: B-i-readno
14         no=tmp; // BLOCK: B-i-writeno
15       }else{
16         int tmp=yes+1; // BLOCK: B-i-readyes
17         yes=tmp; // BLOCK: B-i-writeyes
18       }
19     }
20   }
21   join;
22   // BLOCK C
23   // print results
24   cout << "Result: " << (float) yes/(no+yes) <<
25           "% voted yes..." << endl;
26 }
```

Listing 2: Parallel voting program

Consider the function `voting` in Listing 2. It encodes a parallel voting system where 2 people decide how to vote and then update the `yes` or `no` counts accordingly. In this task, we want to detect that `voting` behaves non-deterministically.

1. What memory locations (i.e. variables) are the target of multiple write operations? Only these may exhibit a race. In the following, we will call these variables $V_{critical}$.

2. Assume the program generates a trace where both people disagree, i.e., person 0 votes "yes" while person 1 votes "no". Define the Happens-Before (HB) Model for this case. Treat lines `2-6` as a single block, lines `10-12` as 2 blocks (one per thread) and lines `13,14,15,16` as individual blocks. Finally, treat lines `22-25` as a single block. Do not list any other blocks.

3. Extend your HB Model by vector clocks, as discussed in the lecture.

4. For each node in you HB Model, add the set of variables from $V_{critical}$ that this node accesses, and mention if the access is a read or a write.

5. Does the HB Model allow for a race, according to the dynamic race detection algorithm from the lecture?

6. Repeat the steps 2-5, assuming that both people agree, i.e., both vote "yes".

7. Demonstrate how reordering the nodes in the HB Model of step 6 can lead to a different result. Concretely, provide two orderings (that are valid according to the HB) for which the line `24` prints a different value.

8. Now assume that the threads are executed atomically (i.e., once a tread starts, it cannot be interrupted by another thread). You can achieve this by considering only a single block `B-0` for person 0 and a single block `B-1` for person 1. May the dynamic race detection algorithm still report a conflict?