Carl Johnson
Homework 1: Answers
CSc 422
Patrick Homer
31 January 2018

Homework 1

1. a) In order for the program to terminate, the while condition in the first arm (`x != y`) has to fail once and the await condition in the second arm (`x == y`) has to pass once. To the best of my knowledge, there are four scenarios by which this can happen, labeled scenarios A, B, C and D below.

Termination Scenario A:

Step	Arm	Instruction	x	y
1	1	`x != y` ≡ T	10	0
2	1	`x = x - 1`	9	|
3	1	`y = y + 1`	|	1
4	1	`x != y` ≡ T	9	1
5	1	`x = x - 1`	8	|
6	1	`y = y + 1`	|	2
7	1	`x != y` ≡ T	8	2
8	1	`x = x - 1`	7	|
9	1	`y = y + 1`	|	3
10	1	`x != y` ≡ T	7	3
11	1	`x = x - 1`	6	|
12	1	`y = y + 1`	|	4
13	1	`x != y` ≡ T	6	4
14	1	`x = x - 1`	5	|
15	1	`y = y + 1`	|	5
16	1	`x != y` ≡ F	5	5
17	2	`x == y` ≡ T	5	5
18	2	`x = 8`	8	|
19	2	`y = 2`	8	2

Following the 5th iteration of the while loop in the first arm (steps 1 - 15), x = 5 and y = 5. If the while condition in the first arm (`x != y`) is checked either immediately before or immediately after (steps 16 and 17 can be swapped with no change in outcome) the await condition in the second arm (`x == y`) is checked, then the while loop in the first arm will terminate, allowing the code in the body of the second arm to execute, leaving us with
x = 8 and y = 2

Termination Scenario B:
(NOTE: Steps 1 - 15 are identical to the tables in scenario A, so they were left out.)

Step	Arm	Instruction	x	y
-	-	...		
16	2	`x == y` ≡ T	5	5
17	2	`x = 8`	8	|
18	2	`y = 2`	|	2
19	1	`x != y` ≡ T	8	2
20	1	`x = x - 1`	7	|
21	1	`y = y + 1`	|	3
22	1	`x != y` ≡ T	7	3
23	1	`x = x - 1`	6	|
24	1	`y = y + 1`	|	4
25	1	`x != y` ≡ T	6	4
26	1	`x = x - 1`	5	|
27	1	`y = y + 1`	|	5
28	1	`x != y` ≡ F	5	5

Following the 5th iteration of the while loop in the first arm (steps 1 - 15), `x` = 5 and `y` = 5. If the await condition in the second arm (`x == y`) is checked and all the code in the body runs (steps 16 - 18) before the while condition in the first arm (`x != y`) is checked again (step 19), then we will have `x` = 8 and `y` = 2. The while condition in the first arm (`x != y`) will still be true, and the while loop will run for three more iterations before
x = 5 and y = 5,
causing the while loop to terminate.

Termination Scenario C:
(NOTE: Steps 1 - 15 are identical to the tables in scenario A, so they were left out.)

Step	Arm	Instruction	x	y
-	-	...		
16	2	`x == y` ≡ T	5	5
17	2	`x = 8`	8	|
18	1	`x != y` ≡ T	8	5
19	1	`x = x - 1`	7	|
20	2	`y = 2`	|	2
21	1	`y = y + 1`	|	3
22	1	`x != y` ≡ T	7	3
23	1	`x = x - 1`	6	|
24	1	`y = y + 1`	|	4
25	1	`x != y` ≡ T	6	4
26	1	`x = x - 1`	5	|
27	1	`y = y + 1`	|	5
28	1	`x != y` ≡ F	5	5

Following the 5th iteration of the while loop in the first arm (steps 1 - 15), `x` = 5 and `y` = 5. if the await condition in the second arm (`x == y`) is checked and only the first instruction in its body (`x = 8`) runs before while condition in the first arm (`x != y`) is checked, then we will have `x` = 8 and `y` = 5. The while condition in the first arm (`x != y`) will still be true. If the second instruction in arm 2 runs immediately after the `x = x + 1` instruction on the next iteration of the loop in arm 1, then we are left with `x` = 7 and `y` = 3 going into the next iteration. Following the next two iterations of the while loop in arm 1, we now have
x = 5 and y = 5,
causing the while loop to terminate.

Termination Scenario D:
(NOTE: Steps 1 - 15 are identical to the tables in scenario A, so they were left out.)

Step	Arm	Instruction	x	y
-	-	...		
16	2	`x == y` ≡ T	5	5
17	2	`x = 8`	8	|
18	1	`x != y` ≡ T	8	5
19	1	`x = x - 1`	7	|
20	1	`y = y + 1`	|	6
21	1	`x != y` ≡ T	7	6
22	1	`x = x - 1`	6	|
23	1	`y = y + 1`	|	5
24	2	`y = 2`	|	2
25	1	`x != y` ≡ T	6	2
26	1	`x = x - 1`	5	|
27	1	`y = y + 1`	|	3
28	1	`x != y` ≡ T	5	3
29	1	`x = x - 1`	4	|
30	1	`y = y + 1`	|	4
31	1	`x != y` ≡ F	4	4

Following the 5th iteration of the while loop in the first arm (steps 1 - 15),

x = 5 and y = 5. if the await condition in the second arm (`x == y`) is checked and only the first instruction in its body (`x = 8`) runs before while condition in the first arm (`x != y`) is checked, then we will have x = 8 and y = 5. The while condition in the first arm (`x != y`) will still be true. If the second instruction in arm 2 runs immediately after two more full iterations of the while loop in arm 1, then we are left with x = 6 and y = 2 going into the next iteration. Following the next two iterations of the while loop in arm 1, we now have
x = 4 and y = 4,
causing the while loop to terminate.

b) Yes, there are circumstances where the program does not terminate. If it becomes the case that x > y before the while condition in the first arm (`x != y`) is ever evaluated as false, then the while loop will run infinitely.

As x$\to -\infty$ and y$\to \infty$, the ranges of x and y values from that point forward are entirely disjoint from one another.
Here is an example:

Infinite loop scencario:
(NOTE: Steps 1 - 15 are identical to the tables in part (a) scenario A, so they were left out.)

Step	Arm	Instruction	x	y
-	-	...		
16	2	`x == y` ≡ T	5	5
17	2	`x = 8`	8	|
18	1	`x != y` ≡ T	8	5
19	1	`x = x - 1`	7	|
20	1	`y = y + 1`	|	6
21	2	`y = 2`	|	2
22	1	`x != y` ≡ T	7	2
23	1	`x = x - 1`	6	|
24	1	`y = y + 1`	|	3
25	1	`x != y` ≡ T	6	3
26	1	`x = x - 1`	5	|
27	1	`y = y + 1`	|	4
28	1	`x != y` ≡ T	5	4
29	1	`x = x - 1`	4	|
30	1	`y = y + 1`	|	5
31	1	`x != y` ≡ T	4	5
-	-	...		

Following the 5th iteration of the while loop in the first arm (steps 1 - 15),
x = 5 and y = 5. if the await condition in the second arm (`x == y`) is checked and only the first instruction in its body (`x = 8`) runs before while condition in the first arm (`x != y`) is checked, then we will have x = 8 and y = 5. The while condition in the first arm (`x != y`) will still be true. If the second instruction in arm 2 runs immediately after the next full iteration of the while loop in arm 1, then we are left with x = 7 and y = 2 going into the next iteration. Following the next three iterations of the while loop in arm 1, we now have
x = 4 and y = 5
Since arm 2 has completed its instructions, x only decreases and y only increases from this point forward. Therefore, the condition `x != y` will always be true, causing an infinite loop.

2. a) $(\forall i : 1 < i < m\ (\forall j : 1 < j < n : a[i] < b[j]))$

   c) $\neg((\exists i : 1 < i < m : a[i] = 0) \wedge (\exists j : 1 < j < n : b[j] = 0))$

   e) $(\forall i : 1 < i < m\ (\exists j : 1 < j < n : a[i] = b[j]))$

3. a) In weakly fair scheduling, the program may or may not terminate. If the await condition (`x == 0`) isn't checked immediately after `x` is decremented to zero by the atomic code in the second arm (`x = x - 1`) and immediately before it is decremented again, then `x` will continue to decrement past zero. Even if `x` overflows and reaches zero more than once, the await condition (`x == 0`) may still not be checked at the right time, since it does not remain true when it becomes true.

   b) In strongly fair scheduling, the program will terminate if `x` overflows and reaches zero multiple times. In this case, the the await condition (`x == 0`) is infinitely often true, and strongly fair scheduling must at some point check the condition while it is true. If overflow is not taken into account, than the program may or may not terminate, as strongly fair scheduling requires that the condition be infinitely often true.

   c) In weakly fair scheduling, the program may or may not terminate. Just like (a) above, the await condition (`x == 0`) does not remain true when it becomes true, so the await condition (`x == 0`) may still not be checked at the right time, causing the program to never terminate.

   d) In strongly fair scheduling, the program will terminate. In this scenario the await condition (`x == 0`) is infinitely often true, and strongly fair scheduling must at some point check the condition while it is true, causing the program to terminate.