CHALMERS
EXAMINATION / TENTAMEN

Course code/kurskod	Course name/kursnamn			
TDA384	Princ. of conc. prog.			
Anonymous code Anonym kod		Examination date Tentamensdatum	Number of pages Antal blad	Grade Betyg
TDA384-0110-AXX		2023-03-13	13	5

* I confirm that I've no mobile or other similar electronic equipment available during the examination.
Jag intygar att jag inte har mobiltelefon eller annan liknande elektronisk utrustning tillgänglig under eximinationen.

Solved task Behandlade uppgifter No/nr		Points per task Poäng på uppgiften	Observe: Areas with bold contour are to completed by the teacher. Anmärkning: Rutor inom bred kontur ifylles av lärare.
1	X	17	
2	X	17	
3	X	9	
4	X	15	
5	X	10	
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
Bonus poäng			
Total examination points Summa poäng på tentamen		68	

3

a)

```
boolean[] enter = {false, false};
int yield = 0; // could be 1 instead
```

SHARED

Thread t_0		Thread t_1	
`while(true){`		`while(true){`	
3 `enter[1]=true;`		`enter[0] = true;`	12
4 `yield = 0;`		`yield = 1;`	13
5 `await(!enter[1]`		`await(!enter[0]`	14
`|| yield != 0);`		`|| yield != 1);`	
`// CS`		`// CS`	
8 `enter[1] = false;`		`enter[0] = false;`	17
`}`			

Above you see the modified code.

State := (line T_0, line T_1, enter[0] enter[1], yield)

Ans: In the transition table no entry of (8, 17, –, –, –) exists so there is mutual exclusion.

From the later constructed transition schema it appears that the threads would alternate between having their critical sections.

3

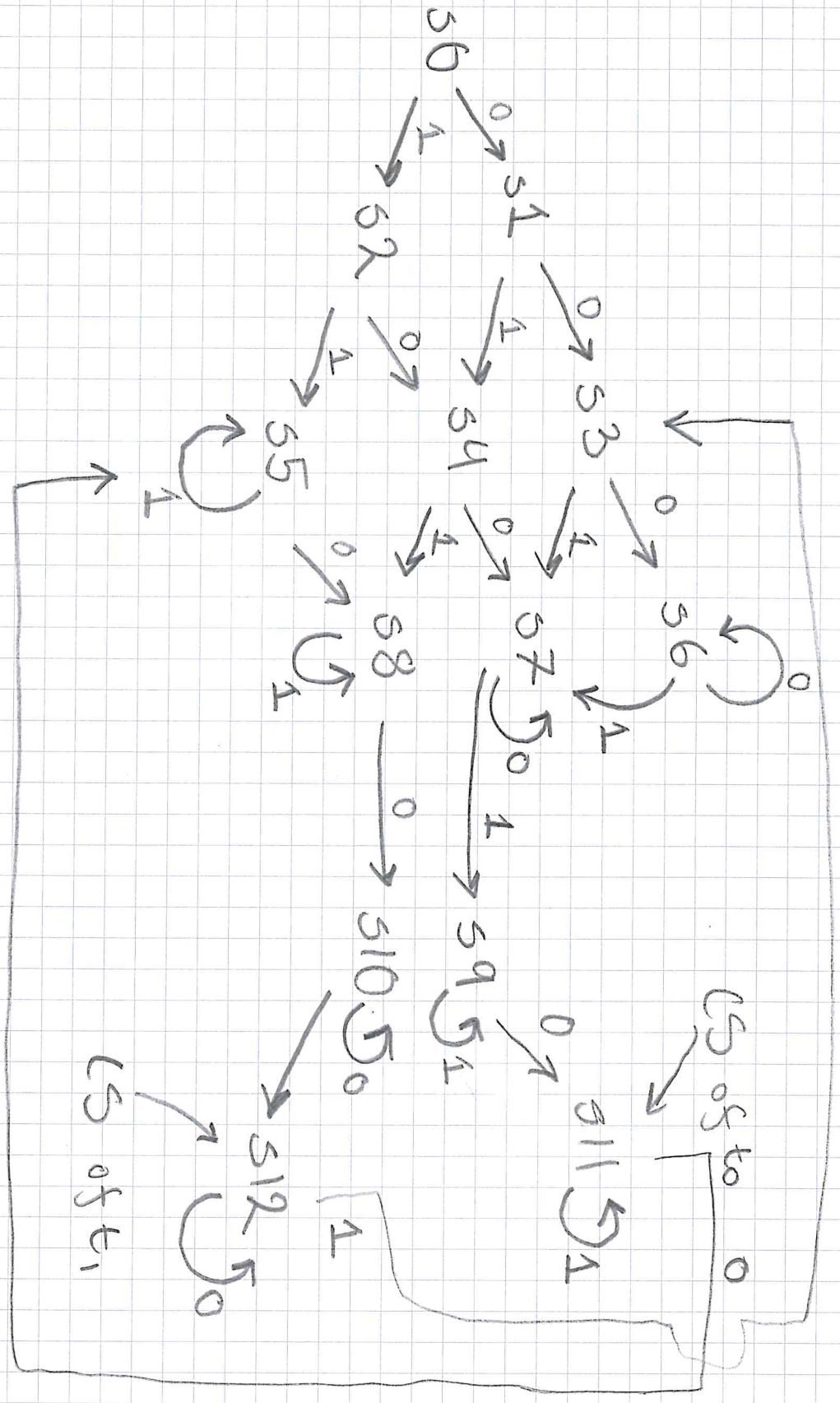
a

State	t_0 moves	t_1 moves
s0 (3,12,f,f,X)	(4,12,f,t,X) ✓	(3,13,t,f,X) ✓
s1 (4,12,f,t,X)	(5,12,f,t,0) ✓	(4,13,t,t,X) ✓
s2 (3,13,t,f,X)	(4,13,t,t,X) ✓	(3,14,t,f,1) ✓
s3 (5,12,f,t,0)	(5,12,f,t,0) ✓	(5,13,t,t,0) ✓
s4 (4,13,t,t,X)	(5,13,t,t,0) ✓	(4,14,t,t,1) ✓
s5 (3,14,t,f,1)	(4,14,t,t,1) ✓	(3,14,t,f,1) ✓
s6 (5,12,f,t,0)	(5,12,f,t,0) ✓	(5,13,t,t,0) ✓
s7 (5,13,t,t,0)	(5,13,t,t,0) ✓	(5,14,t,t,1) ✓
s8 (4,14,t,t,1)	(5,14,t,t,0) ✓	(4,14,t,t,1) ✓
s9 (5,14,t,t,1)	(8,14,t,t,1) ✓	(5,14,t,t,1) ✓
s10 (5,14,t,t,0)	(5,14,t,t,0) ✓	(5,17,t,t,0) ✓
s11 (8,14,t,t,1)	(3,14,t,f,1) ✓	(8,14,t,t,1) ✓
s12 (5,17,t,t,0)	(5,17,t,t,0) ✓	(5,12,f,t,0) ✓

My initial thought was that if 3, 8, 12, 17 was flipped so should the enter index of line 5 and 14 too.

CHALMERS | Anonymous code: TDA384-0110-AXX | Points for question (to be filled in by teacher) | Consecutive page no. Löpande sid nr 3 | Question no. Uppgift nr 1

a)



It appears to work fine. The purpose of enter however is unclear. As await only responds to yield?

Korrektion ?

3

b)

```
int[] enter = {0,0};
int[] yield = {0,0};

t_x
while(true){
  for(int i=1; i<2; i++){          // only one iteration
    enter[x] = i;                  // always == 1
    yield[i] = x;
    await (∀t != x: enter[t] < i   // returns 1 to int, 0 to int
      || yield[i] != x);
    // CS
    enter[x] = 0
  }
```

So for t_0 this means (i.e. !enter[1])

```
enter[0] = 1;
yield[1] = 0;                      // since enter only can be 0 or 1 this is the same as enter[1]==0
await (enter[1] < 1 || yield[1] != 0);
// CS
enter[x] = 0
```

CHALMERS | Anonymous code: TDA384-0110-AXX | Points for question (to be filled in by teacher) | Consecutive page no. Löpande sid nr 5 | Question no. Uppgift nr 1

Conclusion b)

Yes it would behave the same as the (unmodified) figure 2 version the only difference is that we would use zeroes and ones as booleans a bit more like in C.

3

1 False the program has deadlocks see (2) below 2

2. No it is not dead-lock free

calling take will acquire the lock and get stuck on nPedals.down() with the lock acquired by this user no maker will be able to reach nPedals.up() as put will get stuck on lock.down() before accessing the collection. and later increasing nPedals

Swap lines 9 and 10. 5

3 False the order does not matter. Calling nPedals.up() is unproblematic because the pedal has been added to the collection. Neither

<lock>.unlock() or <Sem...>.up() is a blocking operation. 3

4. True Pretty much the same as (3) unlock() and up() are not blocking so the only problem that could arrise would be if nFree.up() would be called prior to store.remove() but this is not the case 3

5. Unsure
There is no check whether the store is empty and returns null as a pedal whilst nFree is incremented none the less. however nPedals should take care of this? ← Indeed 2

6. False Only one lock is needed
But threads acting as users need to be blocked on a differen condition (that is `Belt.isEmpty()`) than makers (instead waiting on Belt.isFull()) for this two Semaphores are needed. 2

3

a)

State	p mov	q mov
s1 (2,2,f,f,f)	s3	s2
s2 (2,3,f,t,f)	s4	s5
s3 (3,2,t,f,f)	s6	s4
s4 (3,3,t,t,f)	s7	s8
s5 (2,5,f,t,t)	s8	s1
s6 (5,2,t,f,t)	s1	s7
s7 (5,3,t,t,t)	s2	s7
s8 (3,5,t,t,t)	s8	s3

4

State := (p_i, q_j, flaga, flagb, turn)

p_5 = turn == false || flagb == false

q_5 = turn == false || flaga == false

b) Yes it does since no state is reachable where p_5 and q_5 are true (that is are in there critical sections) at the same time.

2

c) No it does not since turn being false is sufficient for proceeding and no state exist where flaga is false and turn is true.

3

3

4

a)

```
reduce(plus, 0, [2,4])   % initial call
```

returns 6

```
plus(2, reduce(plus, 0, [4]))
```

returns 4

```
plus(4, reduce(plus, 0, [1]))
```

matches basecase

returns 0

reduce(plus, 0, [2,4]) gives 6 as the result.

```
preduce(plus, 0, [2,4])   % initial call
    % H = [2]  R = [4]     third match
preduce(plus, 0, [2])     % second match
```

is received as Hr which evaluates to plus(0,2) → 2

similarly preduce(plus, 0, [4]) gives plus(0,4) → 4

is received as Rr

and finally plus(2,4) is computed by the `Me` process which returns the value 6.

3

5

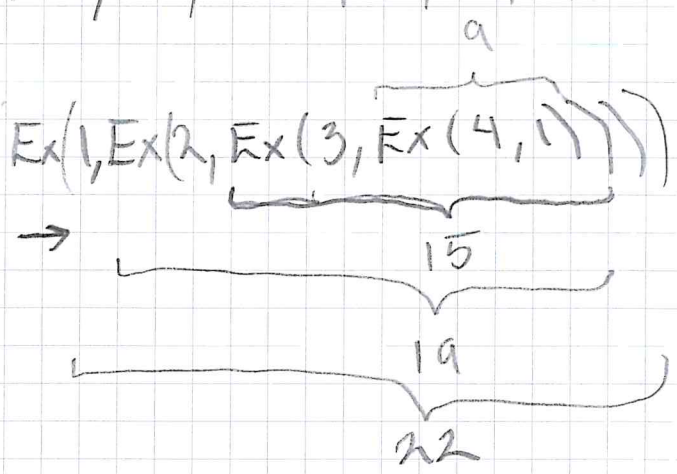
b) No it does not always return the same result. Consider a function where these two conditions does not hold:

* F(A, F(B, C)) == F(F(A, B), C) (Aka associativity)
* F(Element, Start) == Element (Identity element; Identity Function →)

One simple example is

Ex = fun (A, B) -> 2A + B end,

reduce(Ex, 1, [1, 2, 3, 4])



whereas reduce(Ex, 1, [1, 2, 3, 4]) gives Ex(Ex(Ex(1,1), Ex(1,2)), Ex(Ex(1,3), Ex(1,4)))

which obviosly is not the same

you do the math lol

3

6

c) This is not a problem in the example as A (=0) is the identity element of the operation (addition) 2

It is assumed in a parallell folder (i.e. reduce) that the A is the identity element an example of this being no problem other than 2

preduce(plus, 0, list) is preduce(mul, 1, list)

if however the function F is not associative or 2

A is not the identity element

preduce will rarely return the same result as reduce.

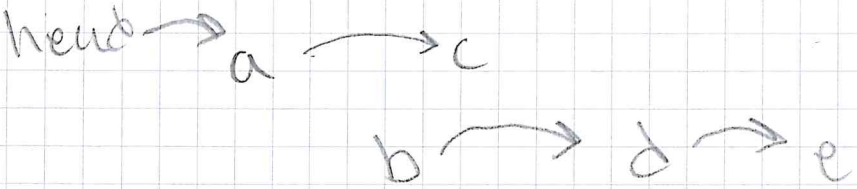
3

a)

1 False

'rm b' 'rm c' could be executed simultaneously if curr is not locked giving a faulty LinkedList

2



2. False

the locks solves this problem, no validation needed.

2

3

b)

```
// line 1,2,3
pred.lock();
curr.lock();
while(curr.key < key){
        pred.unlock();
        pred = curr;
        curr = curr.next();
        curr.lock();
}
// Line 12,13
```

6

The above code is a correct solution. The problem with the code is of course that the programmer calls unlock on line 10 (instead of lock) and that he locks curr inside the while meaning that if curr.key initially is key he will return an unlocked node as curr which as discussed in a) 1 will cause inconsistencies.