Mirza Ovcina
2002721
Cs375

1.1.

<P1=1, P3=1, P5=1> and t1 t3 ready to fire

1.2.

The choice is non-deterministic. Both *t1* and *t3* are enabled and either one on both can fire. Since there is no mechanism enforcing order, choice is random.

1.3.

<P2 = 1, P5 = 1, P4 = 1>

1.4.

We will not be able to fire last *t3* because we would have consumed *p5* on first *t3* transition.

1.5.

t1 and t3
t2 and t4
t2 and t3
t1 and t4.

1.6.

No conflicting transitions. All transitions are executing serially.

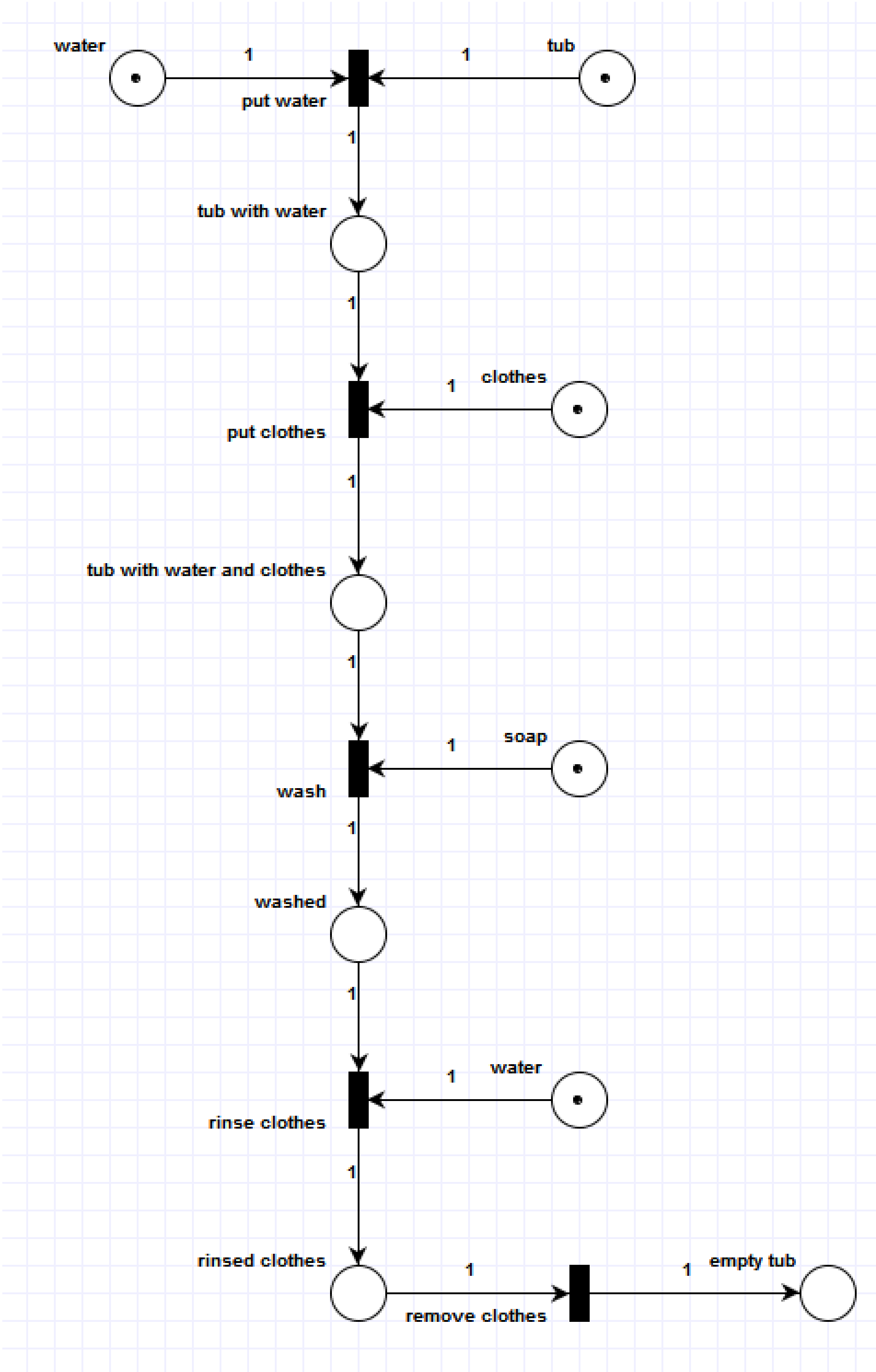
1.7.

No deadlock. Even though right side of this pitri net can be prevented from executing, left side is free to fire as many times as it wishes, therefore no deadlock.

1.8.

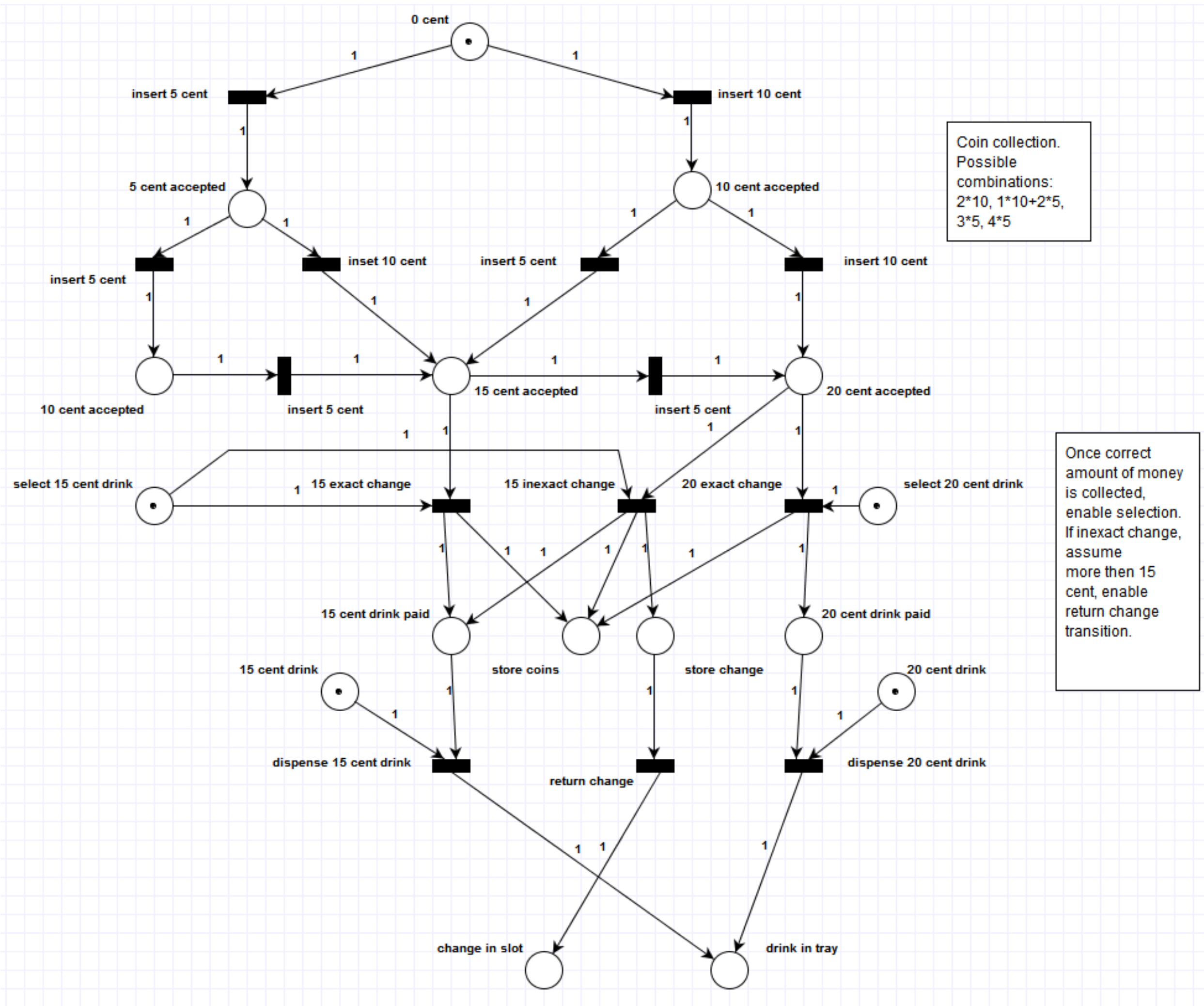
Yes, if *t1* never executes and *p5* has zero resources, *t3* will not be active leading to starvation of right side.

2.



water
1
1
tub
put water
1
tub with water
1
1
clothes
put clothes
1
tub with water and clothes
1
1
soap
wash
1
washed
1
1
water
rinse clothes
1
rinsed clothes
1
1
empty tub
remove clothes

3.



0 cent
insert 5 cent
insert 10 cent
5 cent accepted
10 cent accepted
insert 5 cent
inset 10 cent
insert 5 cent
insert 10 cent
10 cent accepted
insert 5 cent
15 cent accepted
insert 5 cent
20 cent accepted
select 15 cent drink
15 exact change
15 inexact change
20 exact change
select 20 cent drink
15 cent drink paid
store coins
store change
20 cent drink paid
15 cent drink
20 cent drink
dispense 15 cent drink
return change
dispense 20 cent drink
change in slot
drink in tray
Coin collection. Possible combinations: 2*10, 1*10+2*5, 3*5, 4*5
Once correct amount of money is collected, enable selection. If inexact change, assume more then 15 cent, enable return change transition.

4.1.

takeLeftFork takes one token from left fork place if it exists.

takeRightFork takes one token from right fork place if it exists.

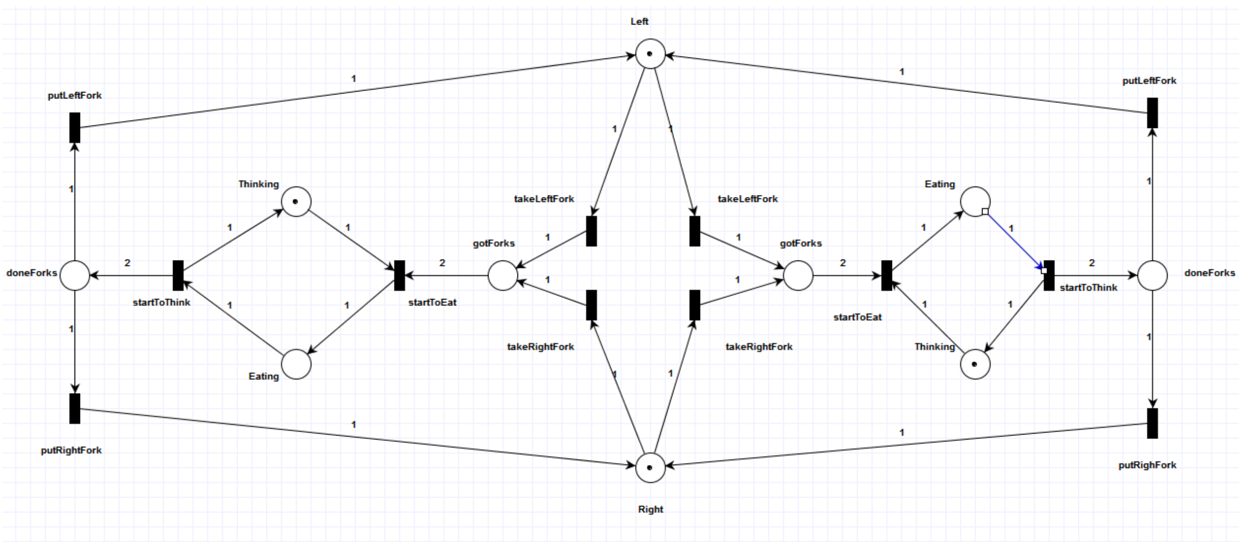
putLeftFork: inserts one token into left fork place

putRightFork: inserts one token into right fork place

startToEat:
requires one token from left and right forks (gotForks) as well as one token from thinking. Once all required tokens are acquired, action fires and state changes to eating.

startToThink:
Once this action is fired, token is removed from eating and startToThink returns two tokens for putLeftFork and putRightFork to consume and also puts one token into thinking place.

4.2



4.3

*Note: if processes are not mentioned, assume they have zero tokens
Philosopher #1

1. <Left=0, Right=1, gotForks=1, takeLeftFork>
2. <Left=0, Right=0, gotForks=2, takeRightFork>
3. <gotForks=0, Eating=1, Thinking=0, startToEat>
4. <Eating=0, Thinking=1, doneForks=2, startToThink>
5. <doneForks=1, Left=1,Right=0, putLeftFork>
6. <doneForks=0, Left=1,Right=1, putRightFork>

Philosopher #2

7. <Left=1, Right=0, gotForks=1, takeRightFork>
8. <Left=0, Right=0, gotForks=2, takeLeftFork>
9. <gotForks=0, Thinking=0, Eating=1, startToEat>
10. <Thinking=1, Eating=0, doneForks=2, startToThink>
11. <doneForks=1, Left=1, Right=0, putLeftFork>
12. <doneForks=0, Left=1, Right=1, putRightFork>

4.4.

Yes, if philosopher P1 takes left or right fork and philosopher P2 takes the remaining fork, neither will be able to eat as two forks are required in order to fire that action resulting in deadlock.

4.5.

Yes, either of the philosophers can stay in the eating process indefinitely starving the other. There is also no mechanism preventing either of the philosophers from looping though the sequence of thinking, picking up forks, eating, thinking picking up forks and so on.