COMPSCI 2SD3
Assignment 2
Prakhar Saxena
400451379

1.

```
const N = 10

range T = 0..N

ACCOUNT = ACCOUNT[0],
ACCOUNT[i:T] = (deposit[x:1..N] -> ACCOUNT[i+x]
                     |when (i>=1)
                     withdraw[y:1..i] -> ACCOUNT[i-y]).
```

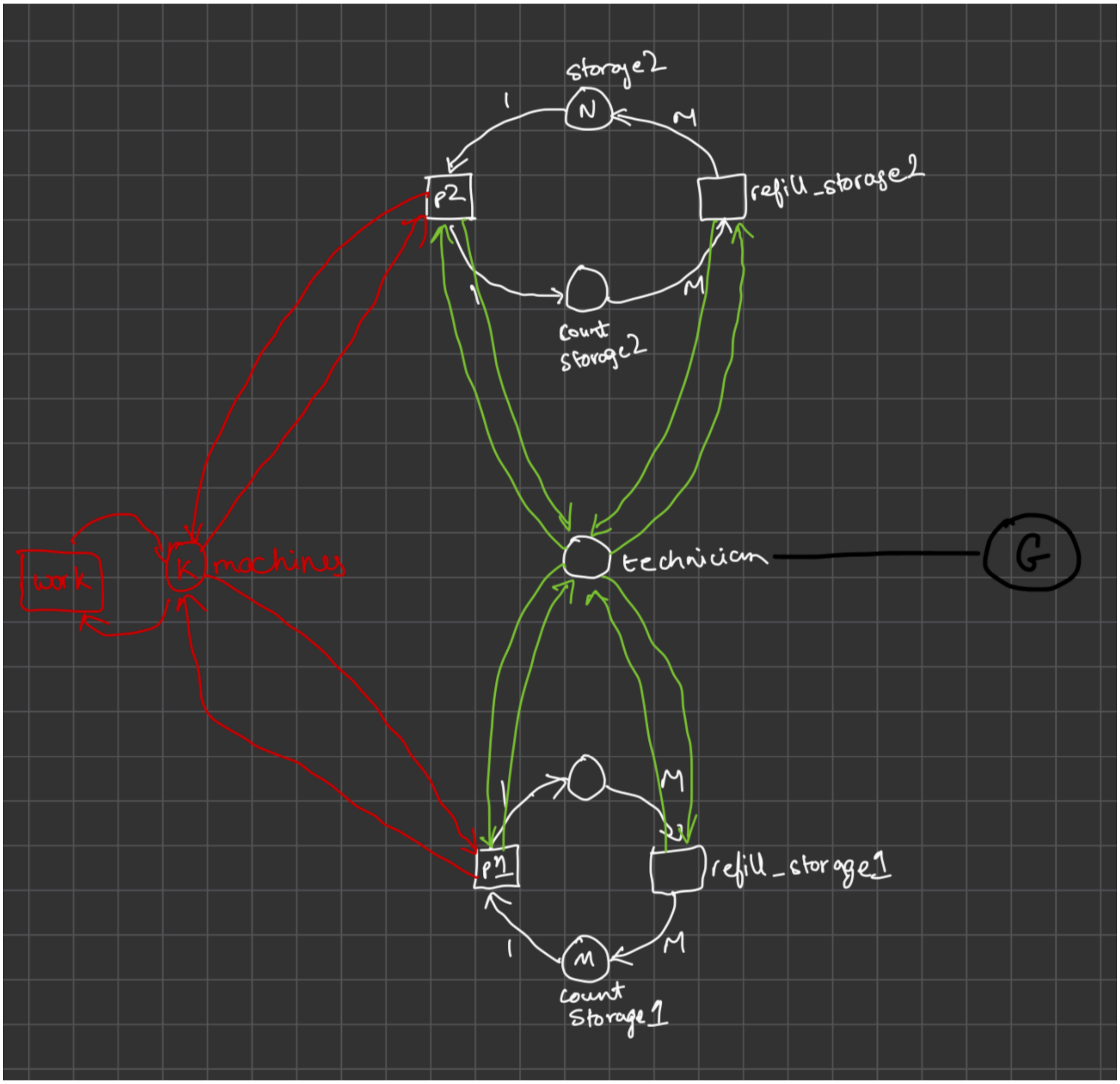
This implementation takes inspiration from The Garden Problem on LN6. I give a range for the total amount of money the bank account can have. For depositing money as long as the value is in the range, we can deposit it. However, for withdrawing money, we can only withdraw when the account balance is more than or equal to 1. Because obviously, we cannot withdraw money from a 0 balance account.

2.a)

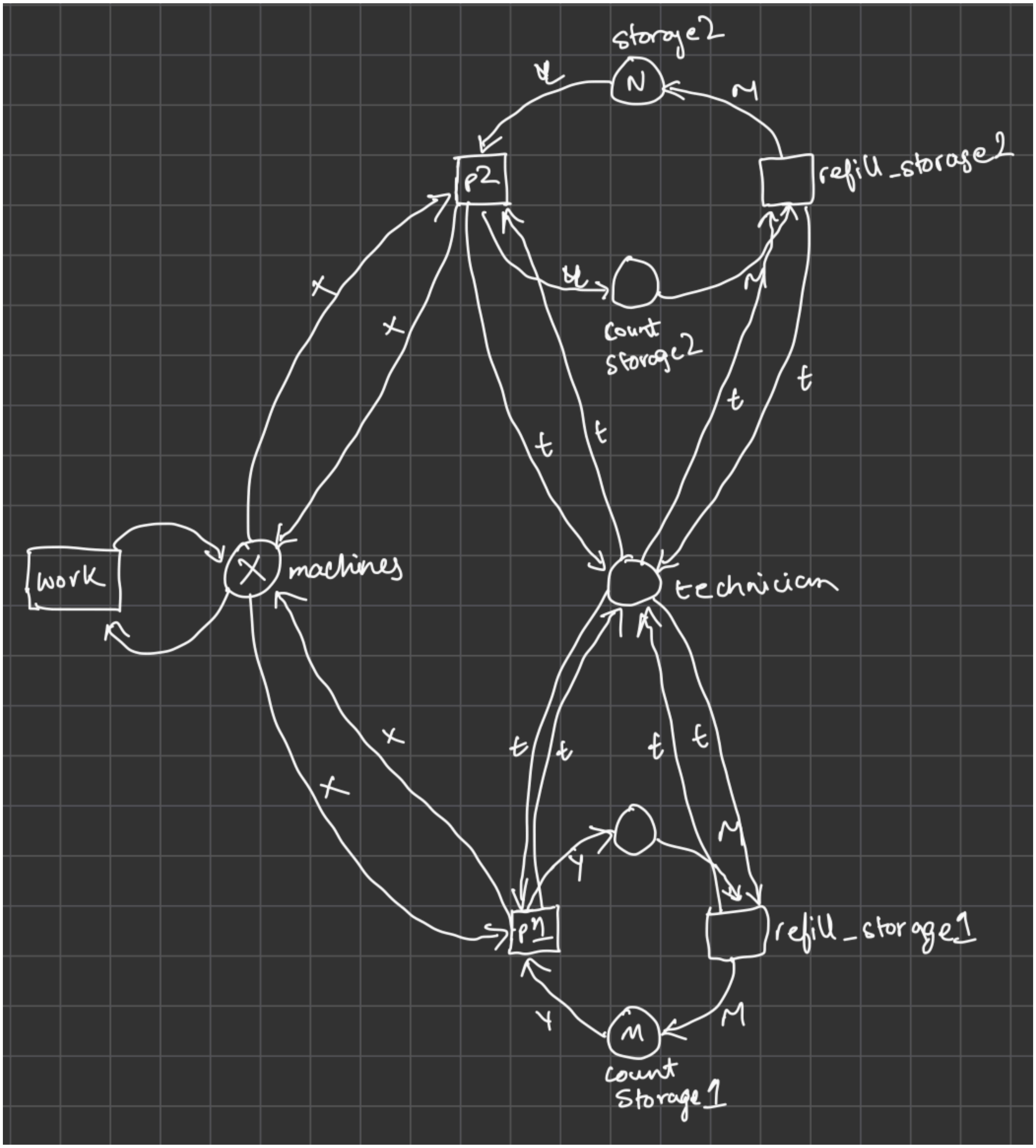
```
const I = 4
range X = 1..I

const J = 3
range p1 = 0..J

const K = 3
range p2 = 0..K

MACHINE = (work -> MACHINE | part1_replace -> MACHINE | part2_replace -> MACHINE)

|| MACHINES = (forall[i:X]machine[i]:MACHINE)

TECHNICIAN = ( part1_replace -> TECHNICIAN | part2_replace -> TECHNICIAN ) |
refill_storage1 -> TECHNICIAN | refill_storage2 -> TECHNICIAN )

STORAGE1 = STORAGE1[J]
STORAGE1[i:p1] = (when (i>0) part1_replace -> STORAGE1[i-1]
| when (i==0) refill_storage1 -> STORAGE1[J] )

STORAGE2 = STORAGE2[K]
STORAGE2[i:p1] = (when (i>0) part2_replace -> STORAGE1[i-1]
| when (i==0) refill_storage2 -> STORAGE1[K]

property P_part1 = P_part1[J]
P_part1[i:p1] = (when (i==0) refill_storage1 -> P_part1[J])

property P_part2 = P_part2[K]
P_part2[i:p2] = (when (i==0) refill_storage2 -> P_part2[K])

||MACHINE_SHOP = ( MACHINES || STORAGE1 || STORAGE2 || TECHNICIAN || P_part1 ||
P_part2 ) << {refill_storage1,refill_storage2}
```

b)

c)



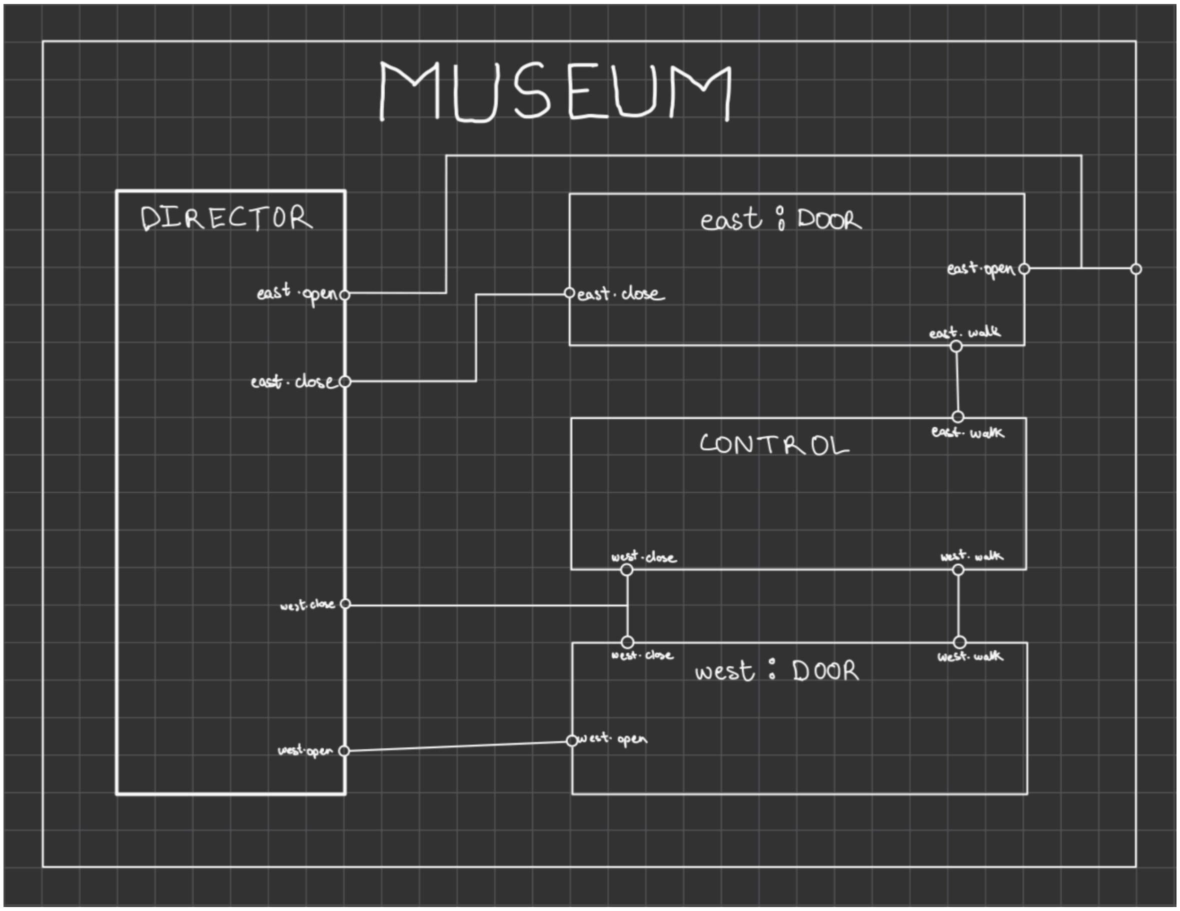
G = { refill_storage1, refill_storage2 } { p1, p2 } d)


storage2
N
k
M
refill_storage2
p2
k
M
count storage2
x
x
t
t
t
t
work
X
machines
technician
x
x
t
t
t
t
M
Y
p1
refill_storage1
Y
M
M
count Storage1

$X = m_1, .... m_k$

e)
Elementary nets and FSP models have significant common features, especially in their ability to model concurrent systems. However, while Petri nets allow for formal proofs of system properties, FSP models are more geared towards verification rather than proof, making formal analysis somewhat more difficult. Petri nets offer a structured approach to demonstrating system properties mathematically, whereas FSP emphasizes the practical verification of behaviors and interactions within a system.

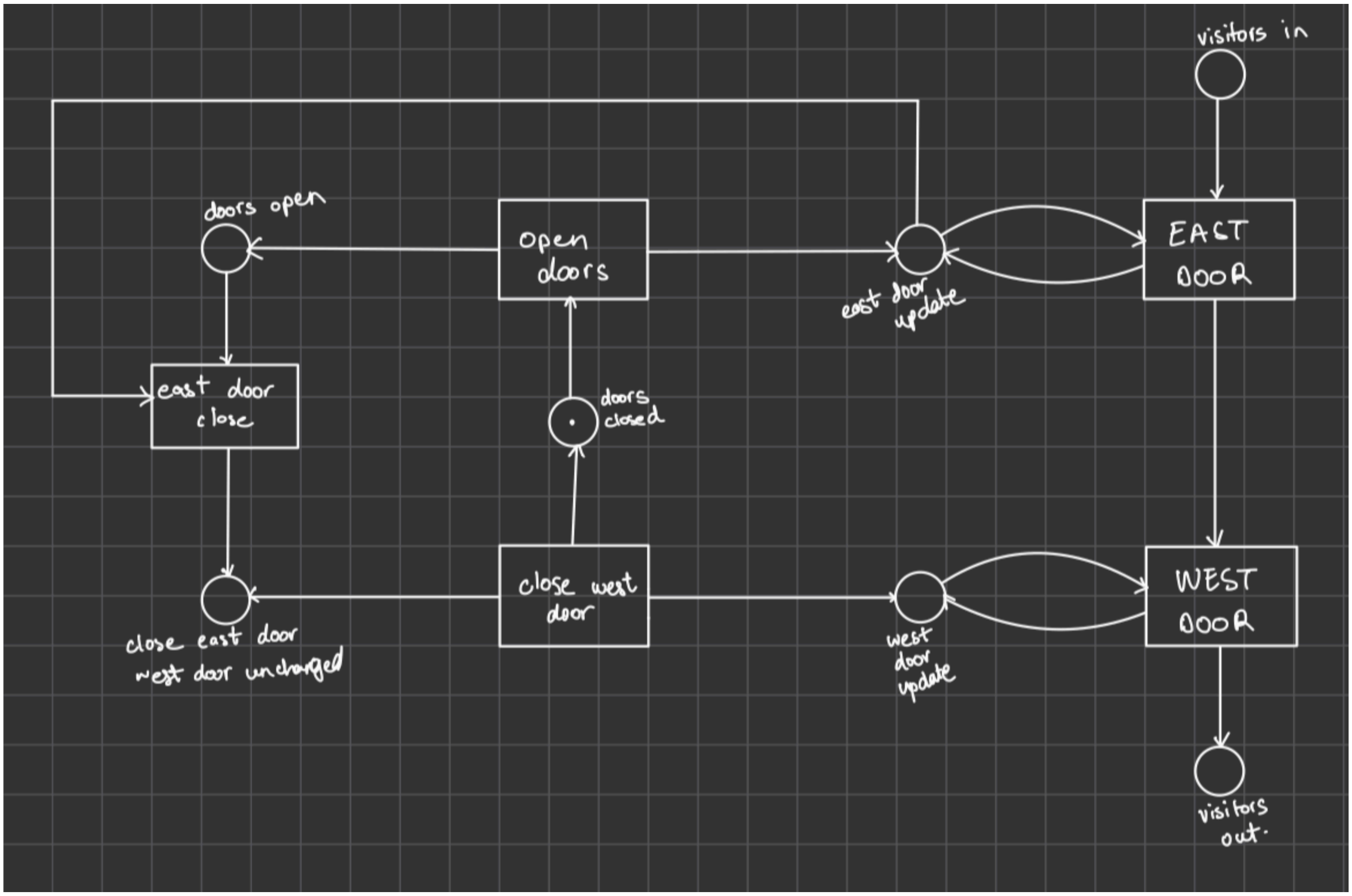
3.
a)



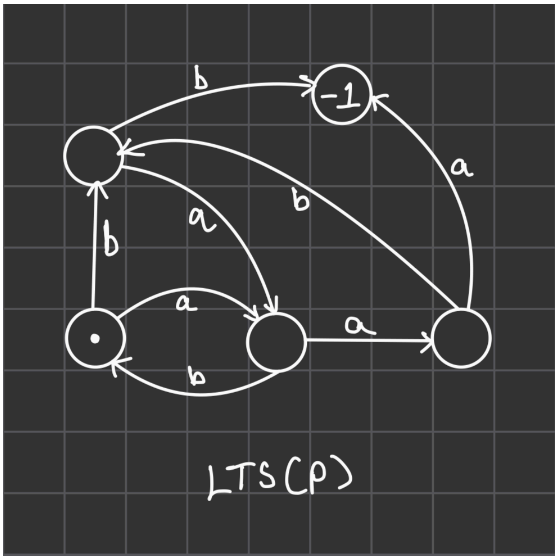
b)

```
DOOR = (open -> OPEN),
OPEN = (cross -> OPEN
                  | close -> DOOR).

DIRECTOR = (east.open -> west.open -> east.close -> west.close -> DIRECTOR).

const T = 4
CONTROL = CONTROL[0],
CONTROL [x:0..T] = (when x < T east.walk -> CONTROL[x+1]
                          | when x > 0 west.walk -> CONTROL[x-1]
                          | when x == 0 west.close -> CONTROL[x]
                          ).

||MUSEUM = (DIRECTOR || east:DOOR || CONTROL || west:DOOR).
```

c)

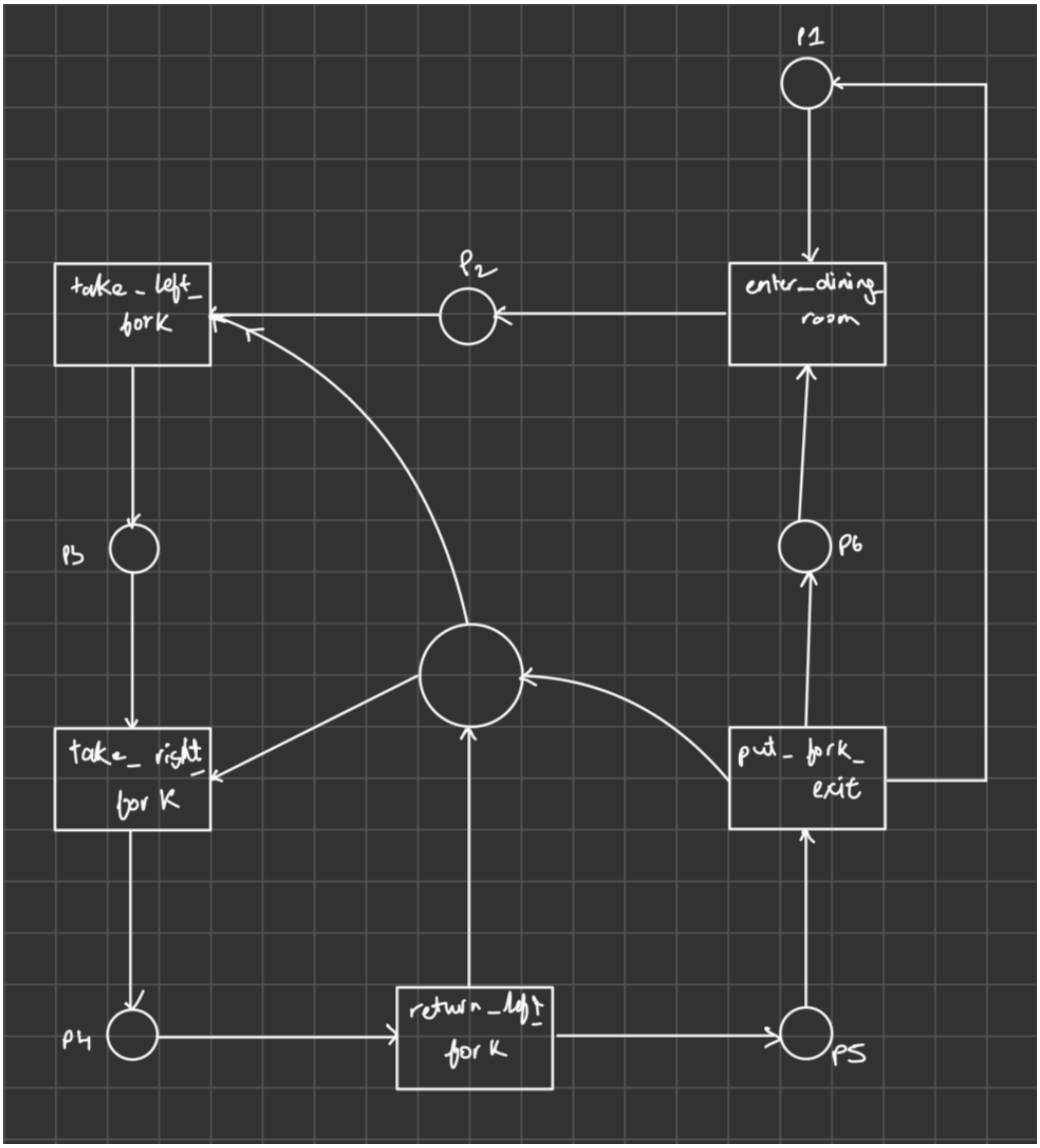


4.



```
P = ( a -> P1 | b -> ( a -> P1 | b -> ERROR)),
P1 = ( a -> ( b -> (a -> P1| b -> ERROR) | a -> ERROR) | b -> P).
```

5.


P1
take_left_fork
P2
enter_dining_room
P3
P6
take_right_fork
put_fork_exit
P4
return_left_fork
P5

P1 = ph 1,2,3,4,5

6.

```
CARPARKCONTROL(N=4) = SPACES[N],
SPACES[i:0..N] = |when(i<N) depart -> SPACES[i+1]
                          (when(i>0) arrive -> SPACES[i-1]
          ).

ARRIVALS   = (arrive -> ARRIVALS).
DEPARTURES = (depart -> DEPARTURES).
||CARPARK = (ARRIVALS||CARPARKCONTROL(4)||DEPARTURES).

||ACTIVE_CARPARK = (OVERFLOW(4) || CARPARK).

progress ENTER = {arrive}

||UPDATE_CARPARK = CARPARK >> {depart}.
```

7.

a)

```
SMOKER_T=( get_resources -> roll_cigarrette -> inhale -> SMOKER_T)
SMOKER_P=( get_resources -> roll_cigarrette -> inhale -> SMOKER_P)
SMOKER_M=( get_resources -> roll_cigarrette -> inhale -> SMOKER_M)

AGENT = ( provide_resources -> initiate_smoke -> AGENT )

GATHER_RESOURCES = ( get_tobacco_paper_match -> distribute_resources ->
GATHER_RESOURCES )

PROCESS = ( start_process -> finsih_process -> PROCESS )

SMOKER_SYSTEM = s_t_p_m:SMOKER_T_P_M || resources:GATHER_RESOURCES ||
agent:AGENT || rules:RPOCESS
```

b)

```
property PICKUP = (s_t.get_paper -> s_t.get_match -> PICKUP
                          | s_p.get_tobacco -> s_p.get_match -> PICKUP
                          | s_m.get_tobacco -> s_m.get_paper -> PICKUP).

PROPERTY = (a1 -> a2 -> PROPERTY
                   | a3 -> a4 -> PROPERTY
                   | a5 -> a6 -> PROPERTY).

SYSTEM = ( elements || resources || rules || PROPERTY ) /
{ a1/a2, a3/a4, a5/a6 }
```