2SD3 Assignment 2

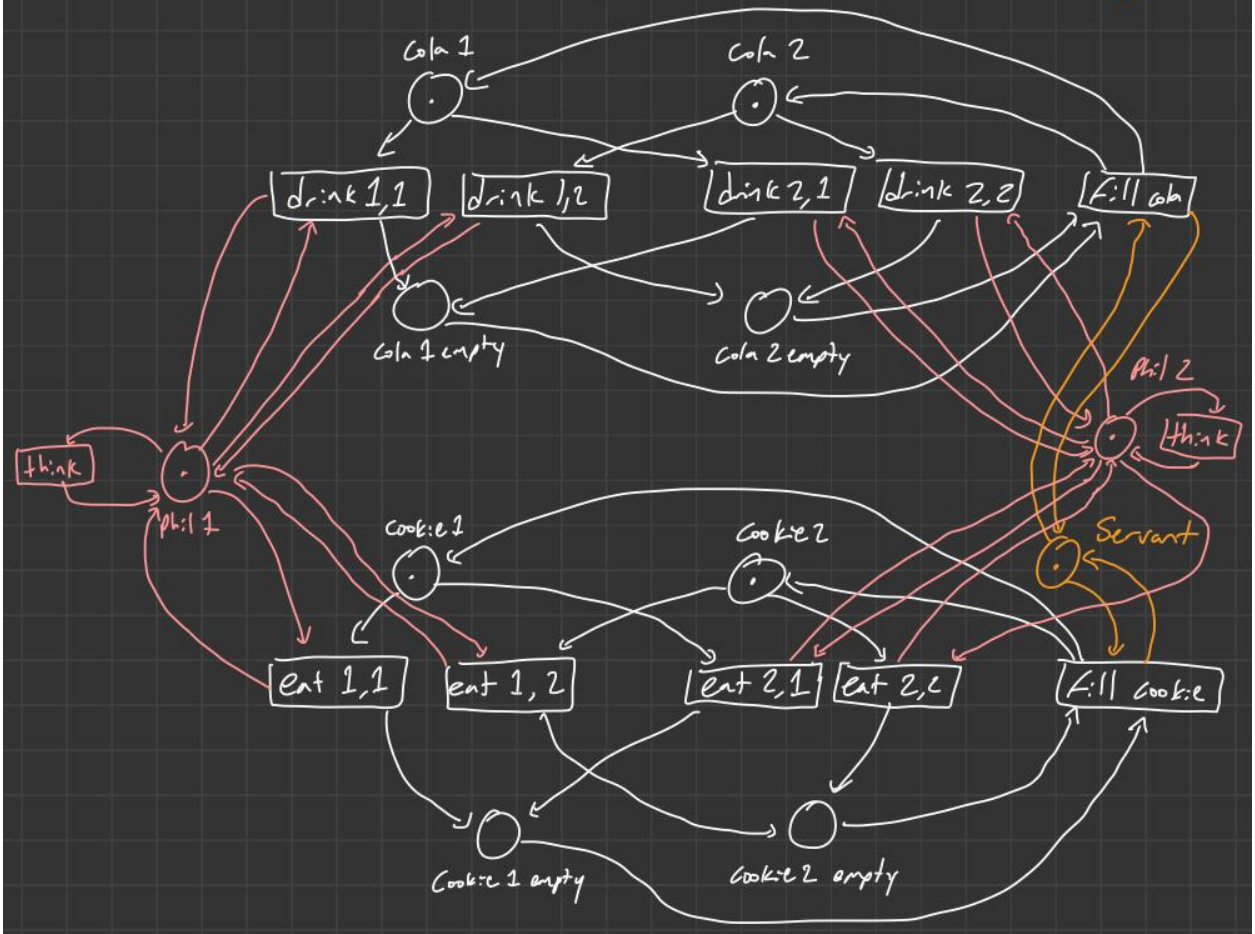
Rochan Muralitharan – muralr3

1. A)

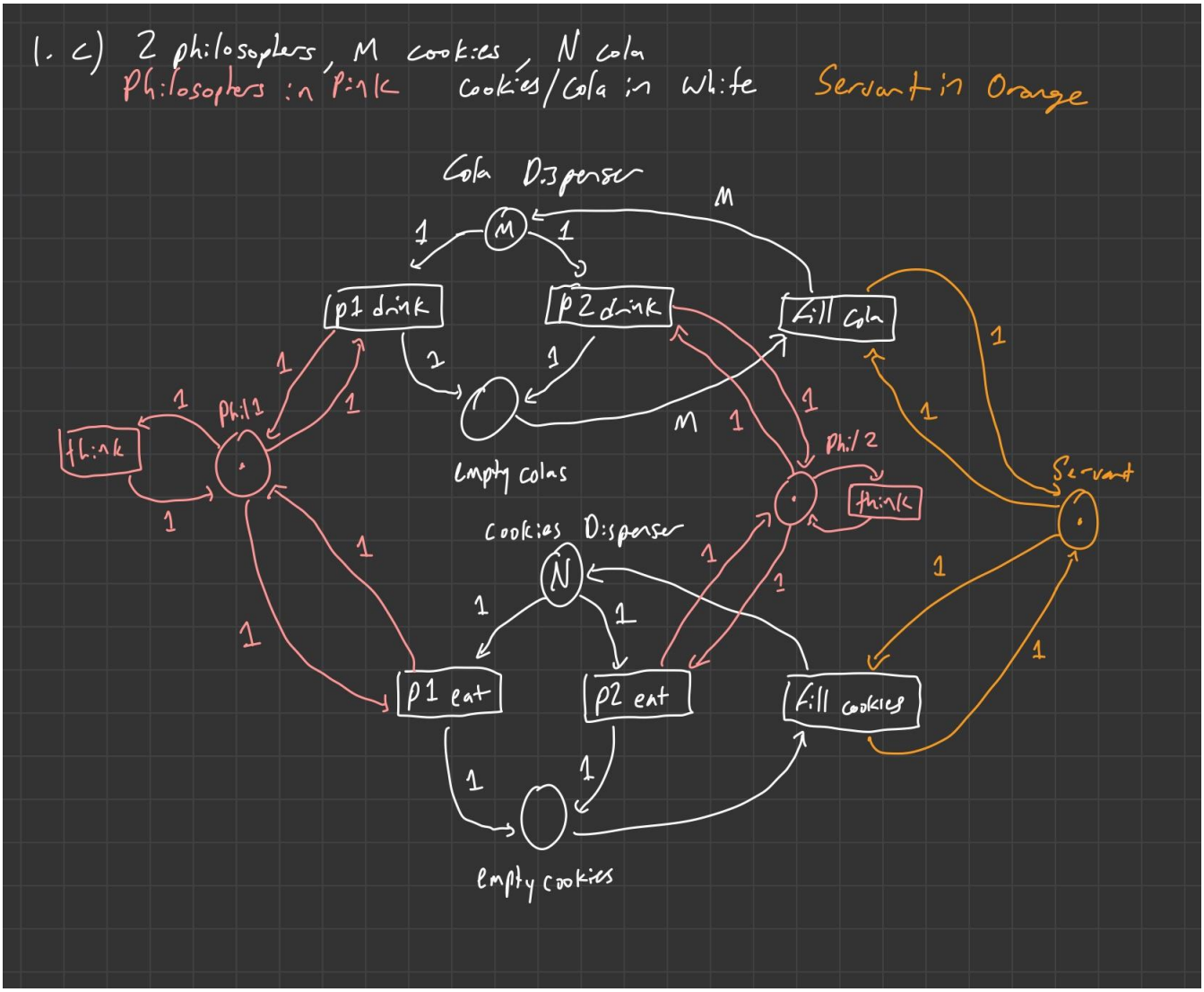
```
const P = 2
PHILOSOPHER = (think -> PHILOSOPHER | get_cookie -> eat_cookie -> PHILOSOPHER
                | get_cola -> drink_cola -> PHILOSOPHER).
||PHILOSOPHERS = (forall[i: 1..P] philosopher[i]:PHILOSOPHER).


SERVANT = (fill_cookies -> SERVANT | fill_cola -> SERVANT).

const M = 2
COOKIES = COOKIES[0],
COOKIES[i:0..M] = (when (i < M) get_cookie -> COOKIES[i+1]
                    | when (i==M) fill_cookies -> COOKIES[0]).

const N = 2
COLA = COLA[0],
COLA[i:0..N] = (when (i < N) get_cola -> COLA[i+1]
                | when (i==N) fill_cola -> COLA[0]).


property PCOOKIES = PCOOKIES[M],
PCOOKIES[i:0..M] = (when (i==M) fill_cookies-> PCOOKIES[M]).

property PCOLA = PCOLA[N],
PCOLA[i:0..N] = (when (i==N) fill_cola-> PCOLA[N]).

||COOKIE_COLA = (PHILOSOPHERS || SERVANT || COOKIES || COLA || PCOOKIES || PCOLA ).
```

1. b) 2 philosophers, 2 cookies, 2 cola

Philosophers in Pink Cookies/Cola in White Servant in Orange

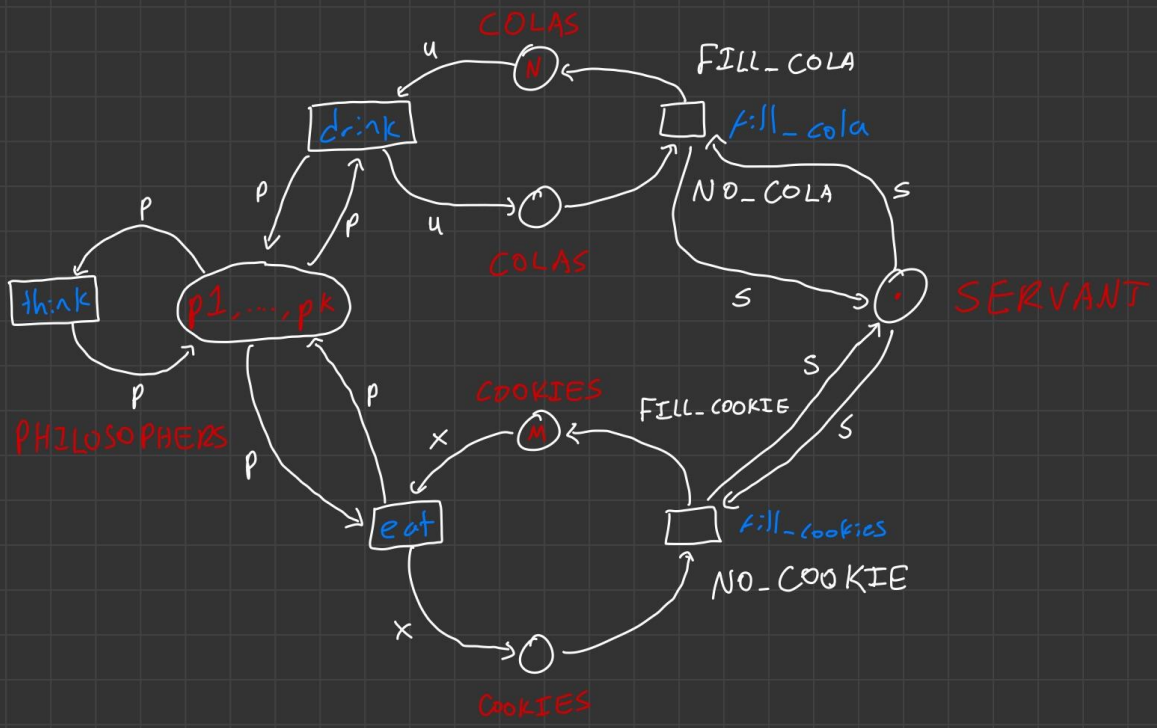


1. c) 2 philosophers, M cookies, N cola
Philosophers in Pink Cookies/Cola in White Servant in Orange



1.d)

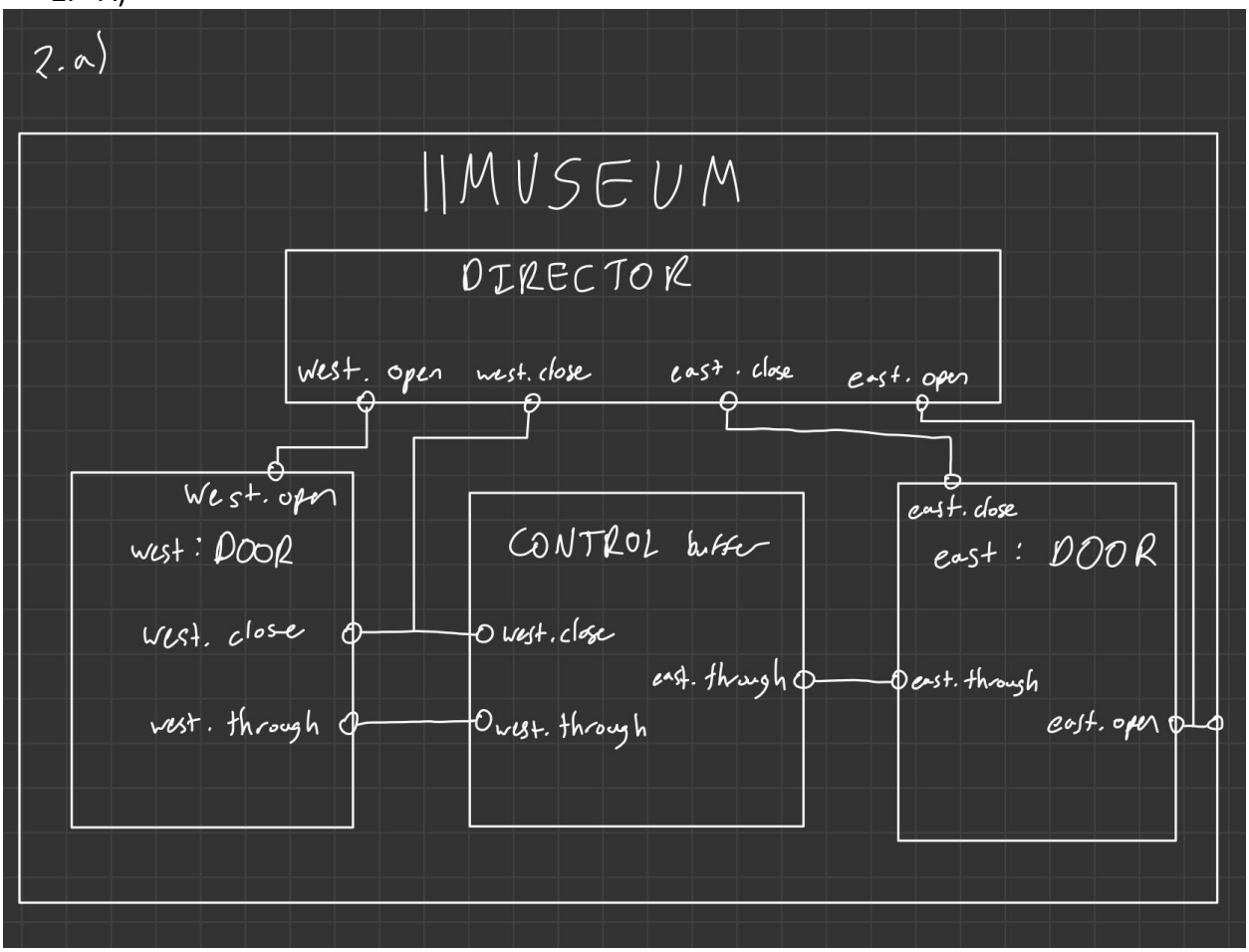
```
colour PHILOSOPHERS = with p1|p2 .... |pk
colour SERVANT = with serv
colour COOKIES = Integers
colour COLAS = Integers
var p : PHILOSOPHERS
var s : SERVANT
var x,y: COOKIES
var u,v: COLAS
fun FILL_COOKIES y = M
fun NO_COOKIES y = 0
fun FILL_COLA v = N
fun NO_COLA v = 0
```



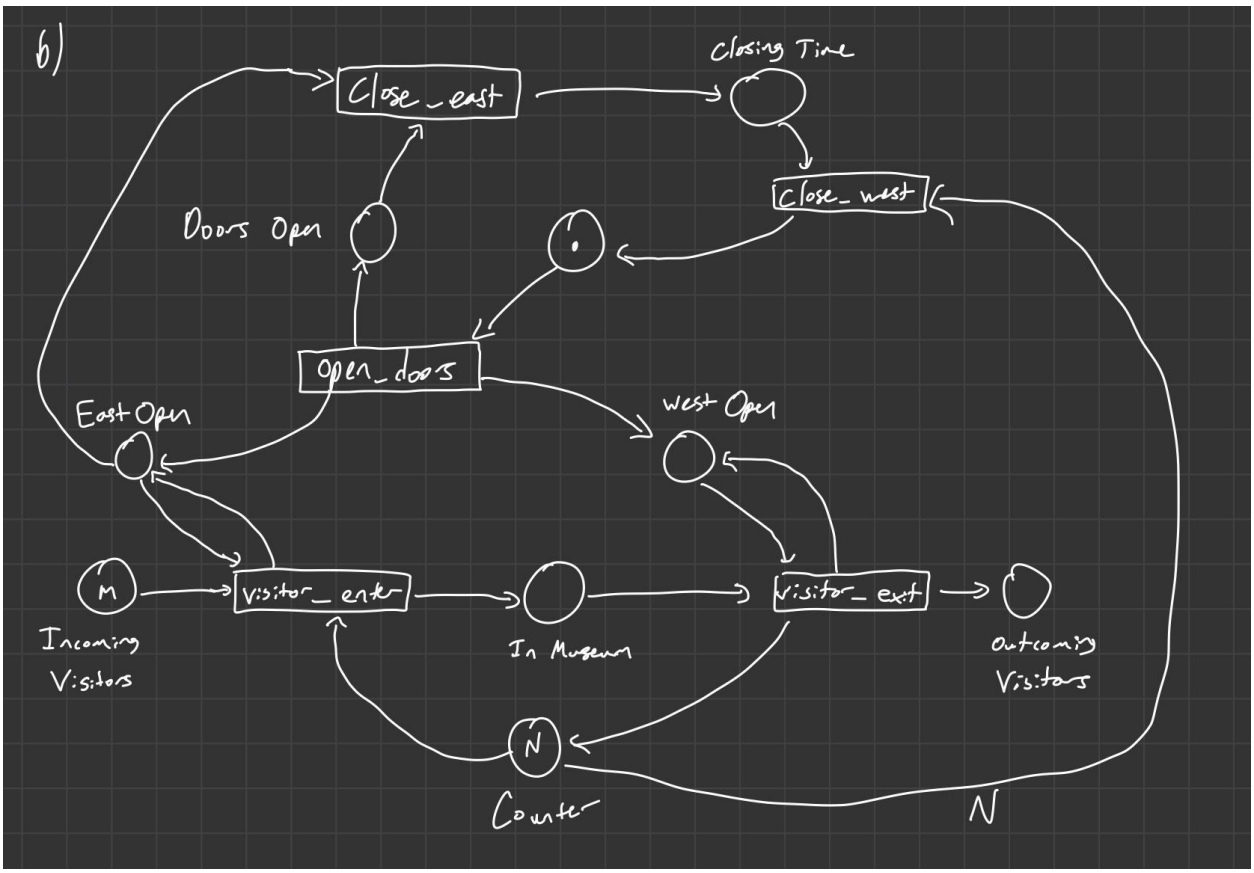
e) The first and most obvious difference between the FSP and the Petri Nets is that the FSP is an algebraic display of the process while the Petri Nets are a visual display of the process. The elementary petri net is a good way to show every single process and state, however it gets confusing when there are a lot of processes/states, so for my elementary petri net I used 2 philosophers and a capacity of 2 for both dispensers. The place/transition net condenses the number of transitions making it less complex than the elementary petri net. This net allows us to show a larger number of processes. For my place transition petri net, I used 2 philosophers. Coloured Petri nets are also less complex than elementary petri nets, but also can use 1 state to represent multiple states, which for in our case would be philosophers. The coloured petri net represent *k* philosophers with one state.

f) In Java File *Philosophers_A2_Q1.java*
2 Philosophers, capacity for both dispensers is 2

2. A)



```
const N = 5
/*
DIRECTOR models behaviour of museum director
Assumptions:
a) there are 2 doors in museum named east and west respectively
b) director opens east door then west door, then later closes east door then west door
*/
DIRECTOR = ( east.open -> west.open -> east.close -> west.close -> DIRECTOR).

/*
DOOR models behaviour of door
Assumptions:
a) door is initially closed
b) open action models a door opening
c) through action models a person going through the door
d) close action models a door closing
e) Person can only cross the door when the door is open
f) Door can only be closed when it is open
*/
DOOR = (open -> OPEN),
    OPEN = (close -> DOOR | through -> OPEN).

/*
CONTROL models behaviour of control system for museum
Assumptions:
a) control keeps count of people currently in museum
b) control regulates behaviour of doors of museum
c) control has associated state that indicates number of people currently in the museum
d) action east.through increases counter for control
e) action west.through decreases counter for control
f) action west.close determines if museum is empty
g) assumed that museum is intially empty
*/
CONTROL = CONTROL[0],
CONTROL[i:0..N] = (when (i < N) east.through -> CONTROL[i + 1]
                | when (i > 0) west.through -> CONTROL[i - 1]
                | when (i == 0) west.close -> CONTROL[0]).

/*
MUSEUM models behaviour of museum
Assumptions:
a) people enter through east door and exit through west door
*/
||MUSEUM = (east:DOOR || west:DOOR || DIRECTOR || CONTROL).
```

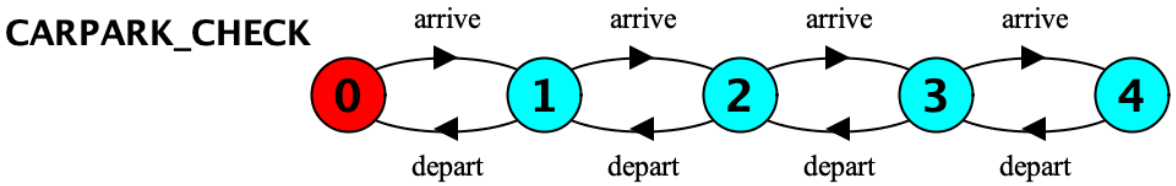


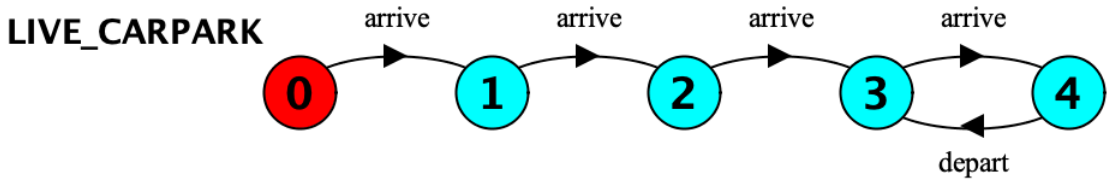
c) In Java File *Museum_A2_Q2.java*

3.

```
CONTROL (N = 5) = SPACES[N],
PARKING_SPACES[i:0..N] = (when(i>0) arrive -> PARKING_SPACES[i-1]
                        | when (i<N) depart -> PARKING_SPACES[i+1]).

ARRIVAL = (arrive -> ARRIVAL).
DEPARTURE = (depart -> DEPARTURE).

||CARPARK = (ARRIVAL || CONTROL(5) || DEPARTURE).

property OVERFLOW (N=5) = OVERFLOW[0],
OVERFLOW[i:0..N] = (arrive -> OVERFLOW[i+1]
                          | depart -> OVERFLOW[i-1]).

||CARPARK_CHECK = (OVERFLOW(5) || CARPARK).

progress ENTER = (arrive)

||LIVE_CARPARK = CARPARK >> (depart).
```

Starvation won't happen when car departure has lower priority than car arrival.

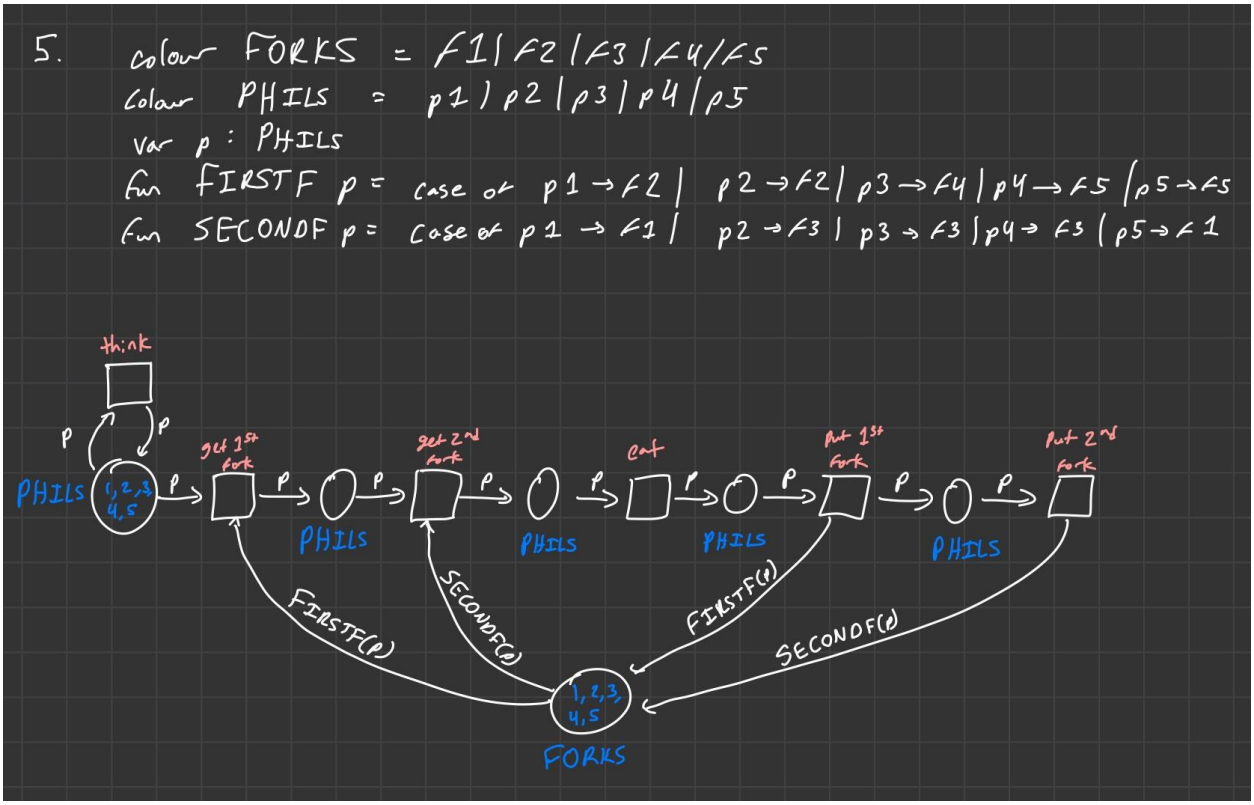




4.

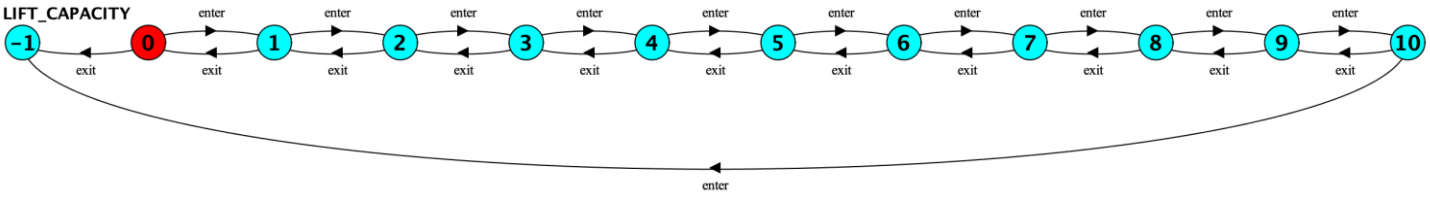
```
FORK = (reserve.left -> take.left -> put.left -> FORK
        | reserve.right -> take.right -> put.right -> FORK).

PHILOSOPHER = (think -> (reserve.left -> reserve.right -> USE_FORKS || reserve.right -> reserve.left -> USE_FORKS)).

USE_FORKS = (take.left -> take.right -> eat -> PUT_FORKS
             | take.right -> take.left -> eat -> PUT_FORKS).

PUT_FORKS = (put.left -> put.right -> PHILOSOPHER
             | put.right -> put.left -> PHILOSOPHER).

||DINING_PHILOSOPHERS (N=5) = (forall[i:1..N]
 (philosopher[i]:PHILOSOPHER || {philosopher[i].right, philosopher[(i+1)%N].left}::FORK)).
```

5.

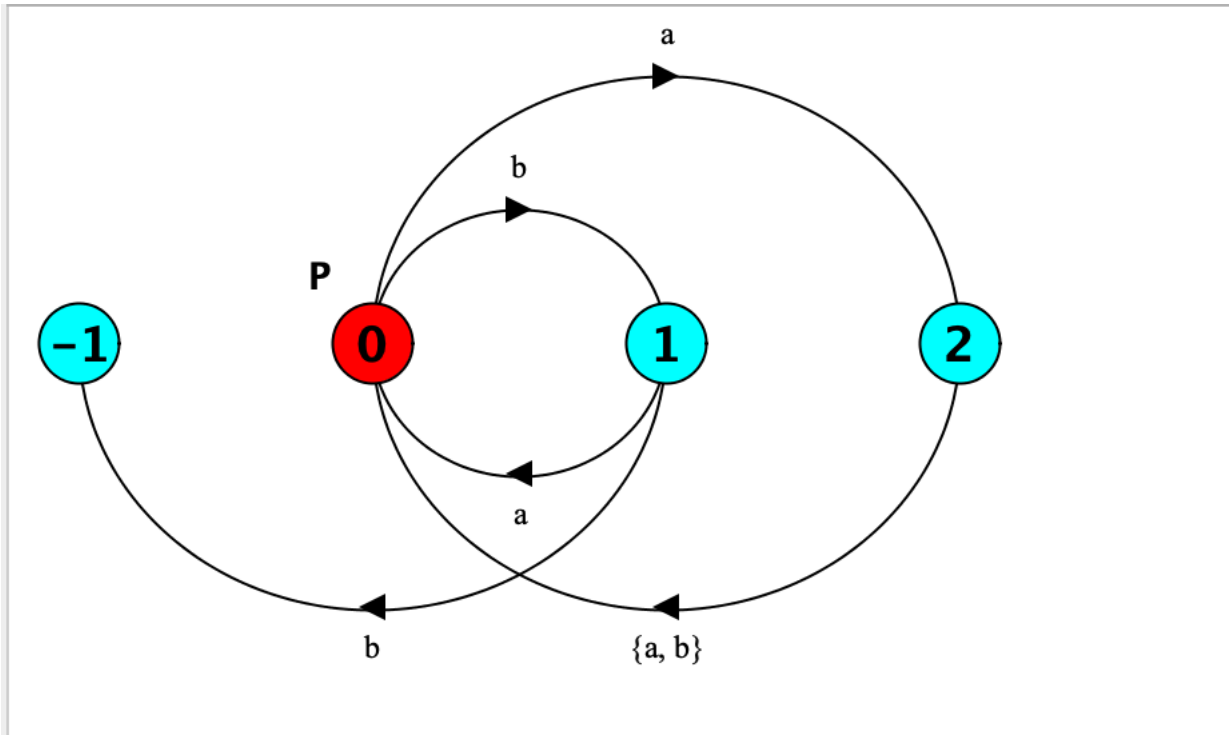


6.

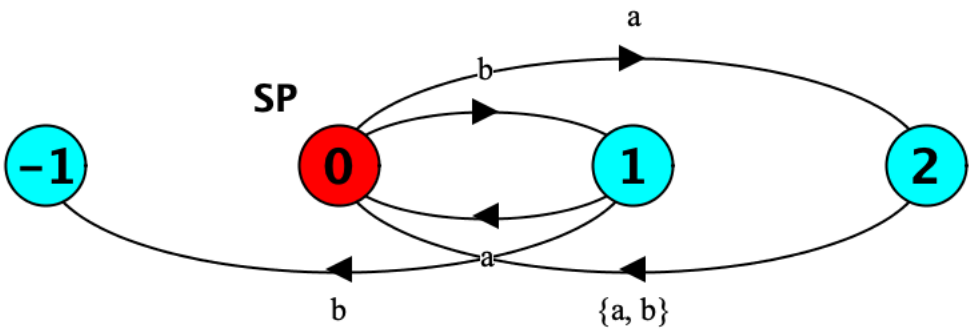
```
const N = 10

property LIFT_CAPACITY = LIFT[0],
LIFT[i:0..N] = (when (i<10) enter -> LIFT[i+1]
                | when(i>0) exit -> LIFT[i-1]).
```



7.




```
SP = (a -> (b -> SP | a -> SP) | b -> (a -> SP | b -> ERROR)).
```



8. A)

```
SMOKER_T = (get_paper -> get_match -> roll_cigarrette -> smoke_cigarrette -> SMOKER_T).
SMOKER_P = (get_tobacco -> get_match -> roll_cigarrette -> smoke_cigarrette -> SMOKER_P).
SMOKER_M = (get_tobacco -> get_paper -> roll_cigarrette -> smoke_cigarrette -> SMOKER_M).

TOBACCO = (delivered -> picked -> TOBACCO).
PAPER = (delivered -> picked -> PAPER).
MATCH = (delivered -> picked -> MATCH).

AGENT_T = (can_deliver -> deliver_paper -> deliver_match -> AGENT_T).
AGENT_P = (can_deliver -> deliver_match -> deliver_tobacco -> AGENT_P).
AGENT_M = (can_deliver -> deliver_tobacco -> deliver_paper -> AGENT_M).

RULE = (can_deliver -> smoking_completed -> RULE).

||SMOKERS = (s_t:SMOKER_T || s_p:SMOKER_P || s_m:SMOKER_M).

||RESOURCES = ({s_m,s_p}::TOBACCO|| {s_t,s_m}::PAPER || {s_t,s_p}::MATCH).

||AGENT_RULE = ({s_m,s_p,s_t}::RULE || {s_m,s_p}::AGENT_T || {s_m,s_t}::AGENT_P || {s_t,s_p}::AGENT_M).

||CIGARETTE_SMOKERS = (SMOKERS || RESOURCES || AGENT_RULE)/{
    s_t.get_paper/s_t.picked,s_m.get_paper/s_m.picked,
    s_p.get_paper/s_p.picked,s_t.deliver_paper/s_t.delivered,
    s_m.deliver_paper/s_m.delivered, s_p.deliver_paper/s_p.delivered,
    s_t.smoking_completed/s_t.smoke_cigarrette,
    s_m.smoking_completed/s_m.smoke_cigarrette,
    s_p.smoking_completed/s_p.smoke_cigarrette
    }.
```

b)

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

||CIGARETTE_SMOKERS_SAFETY = (SMOKERS || RESOURCES || AGENT_RULE || CORRECT_PICKUP)/{
    s_t.get_paper/s_t.picked,s_m.get_paper/s_m.picked,
    s_p.get_paper/s_p.picked,s_t.deliver_paper/s_t.delivered,
    s_m.deliver_paper/s_m.delivered, s_p.deliver_paper/s_p.delivered,
    s_t.smoking_completed/s_t.smoke_cigarrette,
    s_m.smoking_completed/s_m.smoke_cigarrette,
    s_p.smoking_completed/s_p.smoke_cigarrette
    }.
```

C)

```
SMOKER_T =(need_tobacco -> get_paper -> get_match-> roll_cigarrette -> smoke_cigarrette -> SMOKER_T).
SMOKER_P =(need_paper -> get_tobacco -> get_match-> roll_cigarrette -> smoke_cigarrette -> SMOKER_P).
SMOKER_M =(need_match -> get_tobacco -> get_paper-> roll_cigarrette -> smoke_cigarrette -> SMOKER_M).

TOBACCO = (delivered -> picked -> TOBACCO).
PAPER = (delivered -> picked -> PAPER).
MATCH = (delivered -> picked -> MATCH).

AGENT_T = (can_deliver -> need_tobacco -> deliver_paper -> deliver_match -> AGENT_T).
AGENT_P = (can_deliver -> need_paper -> deliver_match -> deliver_tobacco -> AGENT_P).
AGENT_M = (can_deliver -> need_match -> deliver_tobacco -> deliver_paper -> AGENT_M).

RULE = (can_deliver -> smoking_completed -> RULE).

||SMOKERS = (s_t:SMOKER_T || s_p:SMOKER_P || s_m:SMOKER_M).

||RESOURCES = ({s_m,s_p}::TOBACCO|| {s_t,s_m}::PAPER || {s_t,s_p}::MATCH).

||AGENT_RULE = ({s_m,s_p,s_t}::RULE || {s_m,s_p}::AGENT_T || {s_m,s_t}::AGENT_P || {s_t,s_p}::AGENT_M).

||CIGARETTE_SMOKERS = (SMOKERS || RESOURCES || AGENT_RULE)/{
    s_t.get_paper/s_t.picked,s_m.get_paper/s_m.picked,
    s_p.get_paper/s_p.picked,s_t.deliver_paper/s_t.delivered,
    s_m.deliver_paper/s_m.delivered, s_p.deliver_paper/s_p.delivered,
    s_t.smoking_completed/s_t.smoke_cigarrette,
    s_m.smoking_completed/s_m.smoke_cigarrette,
    s_p.smoking_completed/s_p.smoke_cigarrette
    }.
```

D)

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

||CIGAGRETTE_SMOKERS_SAFETY = (SMOKERS || RESOURCES || AGENT_RULE || CORRECT_PICKUP)/{
    s_t.get_paper/s_t.picked,s_m.get_paper/s_m.picked,
    s_p.get_paper/s_p.picked,s_t.deliver_paper/s_t.delivered,
    s_m.deliver_paper/s_m.delivered, s_p.deliver_paper/s_p.delivered,
    s_t.smoking_completed/s_t.smoke_cigarrette,
    s_m.smoking_completed/s_m.smoke_cigarrette,
    s_p.smoking_completed/s_p.smoke_cigarrette
    }.
```