2020 Exam

1.

a)

fill.1
c1.check
c2.check
c1.get
c2.get

b)

```
range Burgers = 0..2
CLIENT = ( check -> get -> CLIENT ).

POT = POT[0],
POT[p: Burgers] = ( when p > 0 check -> POT[p]
| get -> POT[p-1]
| fill[n: Burgers] -> POT[n] ).

LOCK = (acquire->check->release->LOCK).
||LOCKPOT = (LOCK || POT).

COOK = ( fill[p: 1..2] -> COOK )+{fill[0]}.
||DS = ( c1: CLIENT || c2: CLIENT || {c1,c2}::LOCKPOT || COOK )
/{ {c1, c2}.check/check, {c1, c2}.get/get }.
```

c)

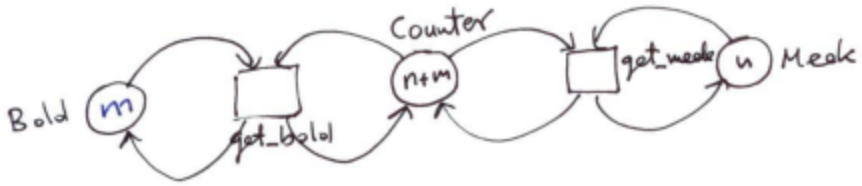
DRAW THE NET

2.

a)

```
set Bold = {bold[1..2]}
set Meek = {meek[1..2]}
set Customers = {Bold,Meek}

CUSTOMER = (getcheese->CUSTOMER).
COUNTER = (getcheese->COUNTER).

 ||CHEESE_COUNTER = (Customers:CUSTOMER || Customers::COUNTER).
```

b)



c)
||CHEESE_COUNTER = (Customers:CUSTOMER ||
Customers::COUNTER)>>{Meek.getcheese}.

If we use progress BOLD = {Bold.getcheese} progress MEEK = {Meek.getcheese} clearly Mees.getcheese gets starved, as Bold.getcheese will always get executed. The same with the second solution. A choice between meek and bold will always be solved in favor of bold.

3.
a)
SAVAGE = (get_serving -> SAVAGE).
COOK = (fill_pot -> COOK).

const K = 3
range Savage = 1..K
||SAVAGES = (forall[i: Savage] savage[i]:SAVAGE).

const M = 3
range Servings = 0..M
POT = POT[0],
POT[s: Servings] = (
	when (s > 0) get_serving -> POT[s-1]
	| when (s == 0) fill_pot -> POT[M]
).

||SYSTEM = (SAVAGES || COOK || POT).

b)
DRAW THE NET

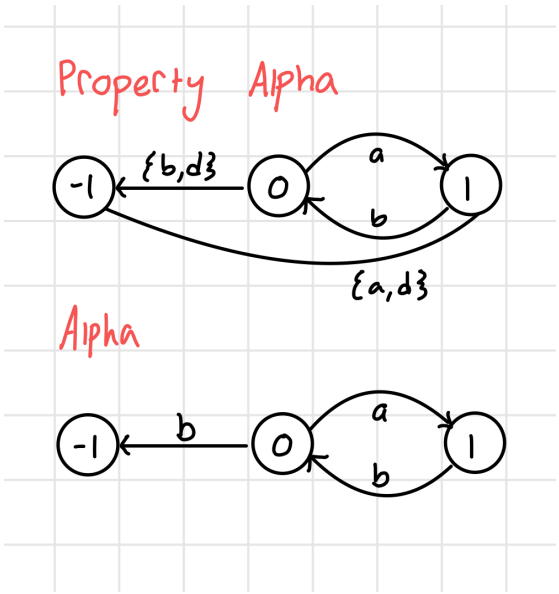
4.
const Max = 1
range Int = 0..Max
SEMAPHORE(N=0) = SEMA[N],
SEMA[v:Int] = (up->SEMA[v+1]
|when(v>0) down->SEMA[v-1]),
SEMA[Max+1] = ERROR.

ACCESS = (down -> control_access -> up -> ACCESS).

||CONTROL = (user:ACCESS || system:ACCESS || {user,system}::SEMAPHORE(1)).

5.
Won't be on the exam

6.
a & b)



c)
Alpha1 = (a->({a,d}->Beta | b->Alpha1) | {b,d}->Beta).

WITH THE SAME BETA (Alpha1 instead of Alpha, but not actually necessary to include)

7.
```
FORK = (
        reserve_right -> get_right -> put_right -> FORK
        | reserve_left -> get_left -> put_left -> FORK
).
 PHIL = (think -> reserve_forks -> GET).
GET = ( get_right -> get_left -> eat -> PUT),
PUT = (
        put_left -> put_right -> PHIL
        | put_right -> put_left -> PHIL
).

||DINERS(N=5) = ( forall[i:1..N]( phil[i]:PHIL || {phil[i].right,phil[(i+1)%N].left}::FORK))
```

8.

a i) $\varphi = $ EG r: We have L(s0) = {r} and L(s2) = {q,r}. Clearly r $\in$ s0, s2. So s0 |= φ HOLDS and s2 |= φ HOLDS

a ii) $\varphi = $ G(r $\vee$ q): We have L(s0) = {r}, L(s1) = {p,t,r} L(s2) = {q,r} and L(s2) = {p,q}. Clearly r $\vee$ q $\in$ s0, s2. Now s1 is reachable from s0, and s2 is reachable from s1, putting us in an infinite path where r $\vee$ q $\in$ s1, s2. Furthermore, s3 is reachable from s0 (q $\in$ s3), and s2 is reachable from s2, putting us in the same infinite path where r $\vee$ q $\in$ s1, s2. So So s0 |= φ HOLDS and s2 |= φ HOLDS.

b)

"If the process is **enabled** infinitely often, then it **runs** infinitely often."

Let p be the statement: "the process is enabled".
Let q be the statement: "the process run"

LTL: G(Fp $\Rightarrow$ Fq)

c)

"If the process is **enabled** infinitely often, then it **runs** infinitely often."

Let p be the statement: "the process is enabled".
Let q be the statement: "the process run"

CTL: AG(EFp $\Rightarrow$ EFq)

d)

"A passenger entering the elevator at 5th floor and pushing 2nd floor button, will never reach 6th floor, unless the 6th floor button is already lightened or somebody will push it, no matter if she/he entered an upwards or upward traveling elevator."

Atomic Predicates: predicates: floor=2, direction=up, direction=down, ButtonPres2, floor=6, etc.

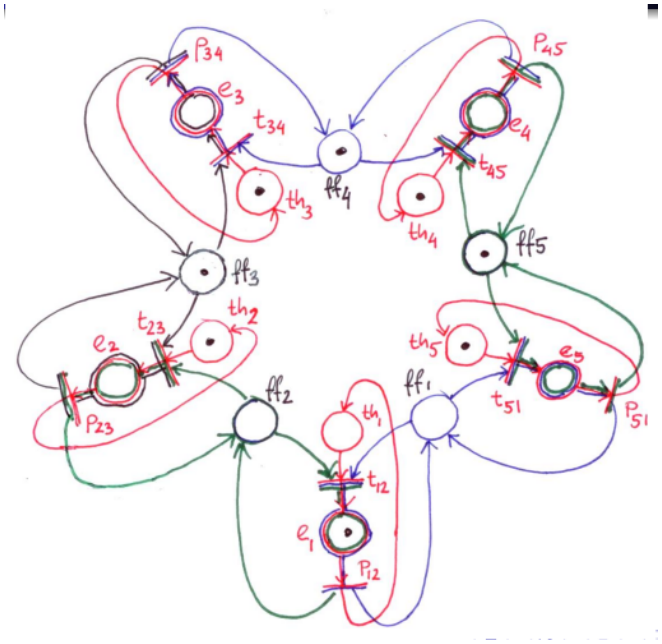
LTL: G((floor = 5 $\wedge$ ButtonPress = 2 $\wedge$ ((F floor = 6 $\wedge$ F $\neg$ButtonPress = 6)) $\wedge$ (F floor = 2 $\wedge$ F direction = up))) $\wedge$ (F(floor = 6 $\Rightarrow$ ButtonPress = 6)))

CTL: AG(floor = 5 $\wedge$ ButtonPress = 2) $\wedge$ AG(EF floor = 6 $\wedge$ EF $\neg$ButtonPress = 6) $\wedge$ AG(EF floor = 2 $\wedge$ EF direction = up) $\wedge$ AG(EF(floor = 6 $\Rightarrow$ ButtonPress = 6))

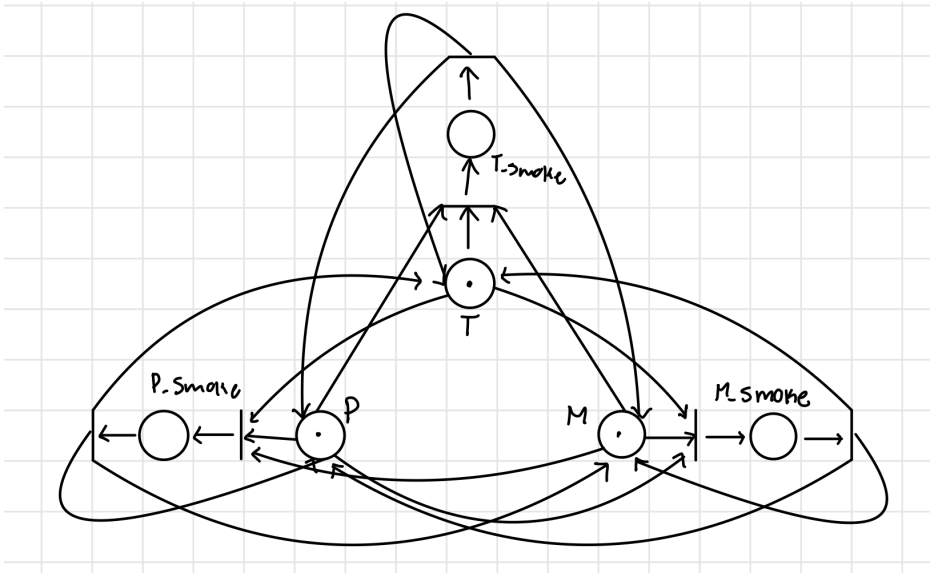
9.

a)

dining phil solution to mimic:

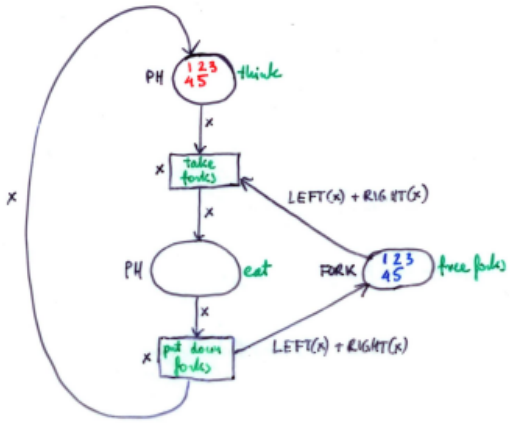


Smokers solution:



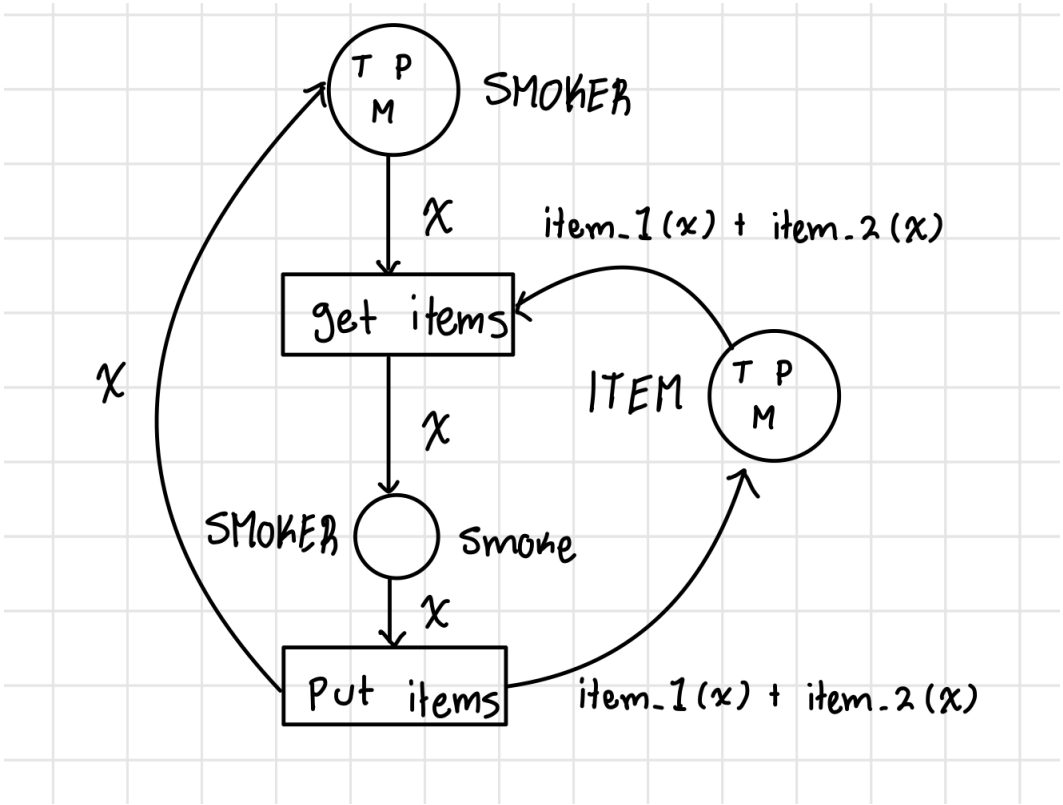
b)

Dining phil solution to mimic:




```
colour PH = with ph1 | ph2 | ph3 | ph4 | ph5
colour Fork = with f1 | f2 | f3 | f4 | f5
LEFT : PH → FORK,    RIGHT : PH → FORK
var x :  PH
fun LEFT x = case of ph1 ⇒ f2 | ph2 ⇒ f3 | ph3 ⇒ f4 |
                    ph4 ⇒ f5 | ph5 ⇒ f1
fun RIGHT x = case of ph1 ⇒ f1 | ph2 ⇒ f2 | ph3 ⇒ f3 |
                     ph4 ⇒ f4 | ph5 ⇒ f5
```

Smokers solution:



Colour SMOKER = with SMOKER_T | SMOKER_P | SMOKER_M
Colour ITEM = with TOBACCO | PAPER | MATCH

SHOULD WRITE FULL ITEM NAME IN NET
Var x: SMOKER
Fun item_1 x = case of SMOKER_T -> PAPER | SMOKER_P -> TOBACCO | SMOKER_M -> TOBACCO
Fun item_2 x = case of SMOKER_T -> MATCH | SMOKER_P -> MATCH | SMOKER_M -> PAPER

2021 Exam

1.
a)

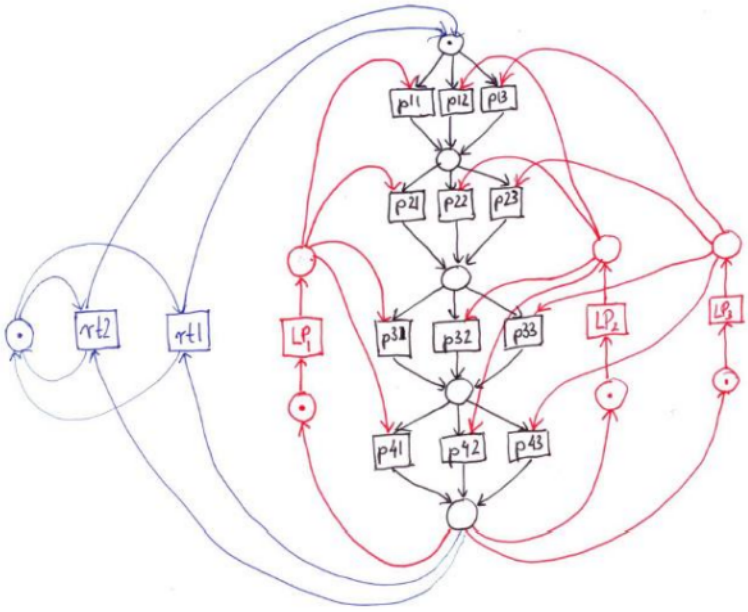
```
const J=3
range Jobs = 0..J

PRINTER = PRINTER [3],
PRINTER[j: Jobs] = (
        when j==0 replace_toner->PRINTER[J]
        |when j>0 print_job -> PRINTER[j-1]
).

USER = (print_job->USER).

const M = 2
range Users = 0..M

||USERS = (forall[i:Users] user [i]:USER).
TECHNICIAN=(replace_toner->TECHNICIAN).

||OFFICE=(USERS||PRINTER||TECHNICIAN) /{user[Users].print_job/print_job}.
```

Description of intended behavior: any USER can print a job if the PRINTER has enough toner, if the printer is empty, then the TECHNICIAN comes to replace the toner.

b)

b. (1) Elementary Petri Nets with 3 users, 2 technicians and 4 print jobs before a new toner is needed.

Notation:
pij – user j does print i
LP_i – user i does local processing where printing is not involved
rti – toner is replaced by technician i



2.

```
const True = 1
const False = 0
range Bool = False..True
set BoolActions={setTrue,setFalse,[False],[True]}

BOOLVAR = VAL[False],
VAL[v:Bool] = ( setTrue -> VAL[True]
| setFalse -> VAL[False]
| [v] -> VAL[v] ).

range Neighs = 0..1
||NEIGHBOURS = (n[Neighs]:BOOLVAR).

LOCK = (acquire -> release -> LOCK).

FIELD = (flag[n:Neighs] -> acquire
        -> (when(False)n[s].setTrue -> release -> FIELD
                | when(True) release -> FIELD)
        ).

set Neighbours = {n1,n2}
||COMP = (Neighbours:FIELD || Neighbours::NEIGHBOURS || Neighbours::LOCK).
```

3.
Same as 2020 Q1

4.
Same as 2020 Q3

5.
Same as 2020 Q4

6.
Same as 2020 Q6

7.
```
const N = 2
const Max = 3
range Int = 0..Max

SEMAPHORE(N=0) = SEMA[N],
SEMA[v:Int] = (up->SEMA[v+1]
|when(v>0) down->SEMA[v-1]),
SEMA[Max+1] = ERROR.

||SEMAS = ( forall[i: 1..N] s[i]:SEMAPHORE).
```

incomplete

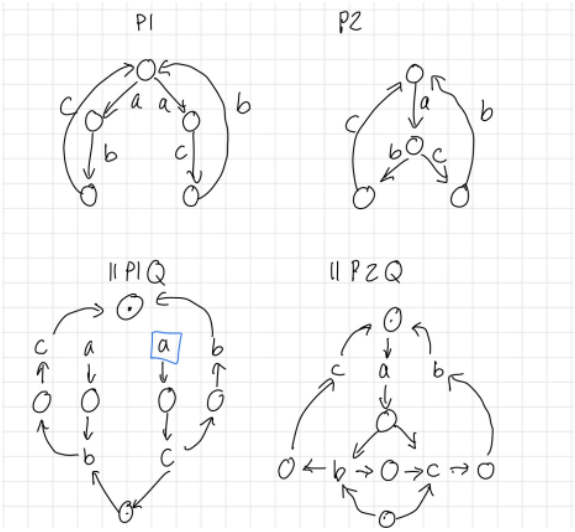
8.
Same as 2020 Q8

9.
Same as 2020 q9

<u>More based on hint doc</u>

Midterm Q7:

If we look at the LTS for P1 and P2 (below), they both have the same traces, Traces(P1) = Traces(P2) = prefix((a(bc U cb))*). But if we look at the petri net, we can see a deadlock (squared in blue) in ||P1Q. So, ||P1Q deadlocks while ||P2Q does not. So they cannot be considered equivalent.



Assignment 3 Q7 (basically Q8 from 2020 and 2021):

(a)

(i) $\neg p \Rightarrow r \equiv \neg(\neg p) \vee r \equiv p \vee r$. We have $L(s_0) = \{r\}$ so $M, s_0 \models \varphi$ holds. We have $L(s_2) = \{p, q\}$ so $M, s_2 \models \varphi$.

(ii) $\neg EG\ r$. This can be translated to "There does not exist at least one path from all future/global states leading to r". We have $L(s_0) = \{r\}$ and $L(s_1) = \{p, t, r\}$, so $r \in s_0, s_1$. Now s_1 is reachable from s_0, and we can follow this path infinitely to stay in s_1. Since we established that $r \in s_0, s_1, M, s_0 \models \varphi$ does NOT hold.

We have $L(s_2) = \{p, q\}$. Since we have that $r \notin s_2$, we can immediately assert that $M, s_2 \models \varphi$ definitely holds as s_2 would have to be included as a possible path.

(iii) $E(t\ U\ q)$. This can be translated to "There exists at least one path in which t can occur until q". We have $L(s_0) = \{r\}$ and $L(s_2) = \{p, q\}$. Although $q \in s_2, t \notin s_0$ and $t \notin s_2$. So $M, s_0 \models \varphi$ does NOT hold and $M, s_0 \models \varphi$ does NOT hold.

(iv) $F\ q$. This can be translated to "Some future state leads to q". We have $L(s_2) = \{p, q\}$, so $q \in s_2$. If we take a path from s_0, then there is an infinite path to s_2, which can be modeled by the trace $s_0 \rightarrow s_2 \rightarrow s_0 \rightarrow s_2 \rightarrow ...$, so $s_0 \models \varphi$ holds. Clearly $s_2 \models \varphi$ holds since $q \in s_2$.

NOTE: For the following parts of this question, they are some assumptions I am making to ensure the models are valid. I will assume the word "between" in a case such as "event q is between event p and event r" means that q is an event that does not take place in at the same time as p or r. I will also assume the word "precedes" in a case such as "event p precedes event q" means that p must happen before q, and they cannot happen at the same time. The opposite can be assumed for "followed" in a case such as "event q is by followed by event p".

(b)

"Event p precedes s and t on all computational paths."
LTL: $G(F\ p \wedge (p \Rightarrow F\ s) \wedge (P \Rightarrow F\ t))$
CTL: $AG(AF\ p \wedge AG(p \Rightarrow F\ s)) \wedge AG(P \Rightarrow F\ t))$

(c)

"Between the events q and r, p is never true but t is always true."
LTL: $G(F\ q \wedge F\ r \wedge (q \Rightarrow (\neg p\ U\ r) \wedge q \Rightarrow (F\ t\ U\ r)))$
CTL: $AG(AF\ q \wedge AF\ r) \wedge AG(q \Rightarrow A(\neg p\ U\ r) \wedge q \Rightarrow A(AF\ t\ U\ r)))$

(d)

"Φ is true infinitely often along every paths starting at s."
LTL: $G(F\Phi)$
CTL: $AG(AF\Phi)$

(e)

"Whenever p is followed by q (after some finite amount of steps), then the system enters an 'interval' in which no r occurs until t."
LTL: $G(P \Rightarrow XG(q \Rightarrow (\neg r\ U\ t)))$
CTL: $AG(P \Rightarrow AXAG(q \Rightarrow A(\neg r\ U\ t)))$

(f)

"Between the events q and r, p is never true."
LTL: $G(F\ q \wedge F\ r \wedge (q \Rightarrow (\neg p\ U\ r)))$
CTL: $AG(AF\ q \wedge AF\ r) \wedge AG(q \Rightarrow (\neg p\ U\ r))$