POT ERROR CHECK
Fill.1 -> c1.check -> c2.check -> c1.get -> c2.get

POT MUTEX
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 }.

DINING SAVAGES MODEL
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).

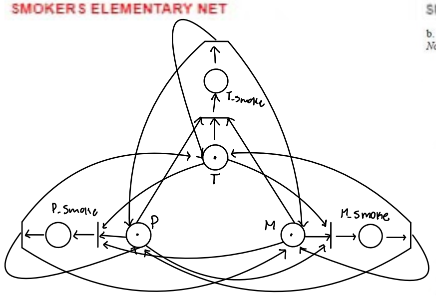
OPERATING SYSTEM BINARY SEMAPHORE
const Max = 1
range Int = 0..Max
SEMAPHORE(N=0) = SEMA[N],
SEMA[v:Int] = (up->SEMA[v+1]
|when(v>0) down->SEMA[v-1]),
SEMA[Max+1] = ERROR.

ACCESS = (down -> control_access -> up -> ACCESS).
||CONTROL = (user:ACCESS || system:ACCESS ||
{user,system}::SEMAPHORE(1)).

DINING PHIL MODEL (SEQUENTIAL PICKING FORK)
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))

SMOKERS ELEMENTARY NET



to see if 2 FSPs are Bisimilar, look to see if each state from both processes are reachable from the initial state with the same trace.

MODEL CHECKING
a i) φ = EG r: We have L(s0) = {r} and L(s2) = {q,r}. Clearly r ∈ s0, s2. So s0 |= φ HOLDS and s2 |= φ HOLDS

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

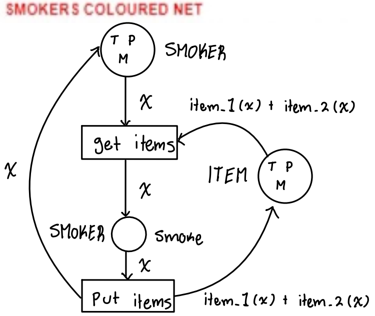
b)
"If the process is enabled infinitely often, then it runs infinitely often."
Let p:: "the process is enabled". q: "the process runs"
LTL: G(Fp ⇒ Fq)

c)
"If the process is enabled infinitely often, then it runs infinitely often."
Let p:: "the process is enabled". q: "the process runs"
CTL: AG(EFp ⇒ 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 travelling elevator."
Atomic Predicates: predicates: floor=2, direction=up, direction=down, ButtonPres2, floor=6, etc.

LTL: G((floor = 5 ∧ ButtonPress = 2 ∧ ((F floor = 6 ∧ F ¬ButtonPress = 6)) ∧ (F floor = 2 ∧ F direction = up))) ∧ (F(floor = 6 ⇒ ButtonPress = 6)))
CTL: AG(floor = 5 ∧ ButtonPress = 2) ∧ AG(EF floor = 6 ∧ EF ¬ButtonPress = 6) ∧ AG(EF floor = 2 ∧ EF direction = up) ∧ AG(EF(floor = 6 ⇒ ButtonPress = 6))

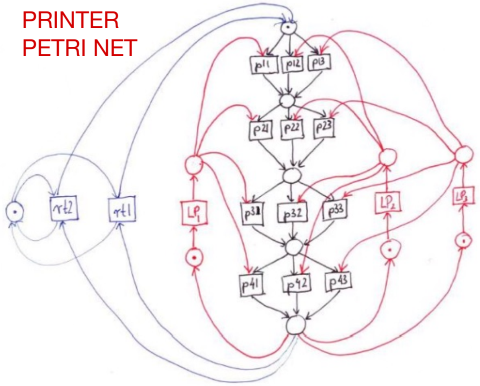
SMOKERS COLOURED NET



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

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

PRINTER
PETRI NET



Dining savage net is similar

NEIGHBOURS & FLAGS
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).

OFFICE PRINTER/TONER FSP
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.

$\Phi ::= \bot \mid \top \mid p \mid (\neg\Phi) \mid (\Phi \wedge \Phi) \mid (\Phi \vee \Phi) \mid (\Phi \Rightarrow \Phi) \mid$
$AX\Phi \mid EX\Phi \mid A[\Phi U \Phi] \mid E[\Phi U \Phi] \mid$
$AG\Phi \mid EG\Phi \mid AF\Phi \mid EF\Phi$ CTL

where p ranges over atomic formulas/descriptions.

- $\bot$ - false, $\top$ - true
- $AX, EX, AG, EG, AU, EU, AF, EF$ are **temporal connections**.
  all pairs, each starts with either A or E
- A means "along All paths" (inevitably)
- E means "along at least (there Exists) one path" (possibly)
- X means "neXt state"
- F means "some Future state"
- G means "all future states (Globally)"
- U means "Until"
- X, F, G, U cannot occur without being preceded by A or E.
- every A or E must have one of X, F, G, U to accompany it.

$\Phi ::= \bot \mid \top \mid p \mid (\neg\Phi) \mid (\Phi \wedge \Phi) \mid (\Phi \vee \Phi) \mid (\Phi \Rightarrow \Phi) \mid$
$(G\Phi) \mid (F\Phi) \mid (X\Phi) \mid (\Phi\ U\ \Phi) \mid (\Phi\ W\ \Phi) \mid (\Phi\ R\ \Phi)$

where p ranges over atomic formulas/descriptions.

- $\bot$ - false, $\top$ - true
- $G\Phi, F\Phi, X\Phi, \Phi\ U\ \Phi, \Phi\ W\ \Phi, \Phi\ R\ \Phi$ are **temporal connections**.
- X means "neXt moment in time" LTL
- F means "some Future moments"
- G means "all future moments (Globally)"
- U means "Until"
- W means "Weak-until"
- R means "Release"

Midterm q5

```
const False = 0
const True  = 1
range Bool =  False..True

SEAT = SEAT[False],
SEAT[reserved:Bool]
     = ( reserve     -> SEAT[True]
       | query[reserved] -> SEAT[reserved]
       | when (reserved) reserve -> ERROR   //error of reserved twice
       ).

range Seats = 0..1
||SEATS = (seat[Seats]:SEAT).

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

TERMINAL = (choose[s:Seats] -> acquire
           -> seat[s].query[reserved:Bool]
           -> (when(!reserved)seat[s].reserve -> release-> TERMINAL
              |when(reserved) release -> TERMINAL)
           ).

set Terminals = {a,b}
||CONCERT = (Terminals:TERMINAL || Terminals::SEATS || Terminals::LOCK).
```

SIMPLIFIED MULTIDIM SEMAPHORE
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).

//THIS IS PSEUDOCODE
EDOWN = (if (forall[i:1..N] s[i] > 0)
then (forall[i:1..N] s[i].down) else block).
EUP = (if (forall[i:1..N] execution blocked)
then s[1].up else (forall[i:1..N] s[i].up)).

||COMP = (SEMAS || EDOWN || EUP).

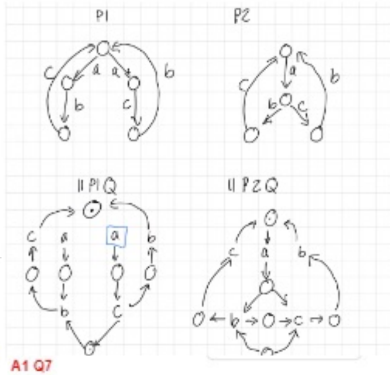
SEMADOMO MULTIDIMENSIONAL SEMAPHORE
const Max = 3
range Int = 0..Max

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

LOOP = (mutex.down -> critical -> mutex.up -> LOOP).

||SEMADEMO = (p[1..3]:LOOP ||
{p[1..3]}::mutex:SEMAPHORE(1)).

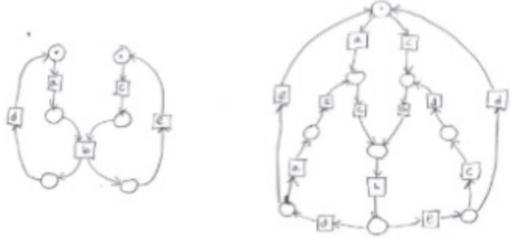
MIDTERM Q7
LTS for P1 and P2 have the same traces, Traces(P1) = Traces(P2) = prefix((a(bc U cb))*). In petri net,there is a deadlock in ||P1Q. So, ||P1Q deadlocks while ||P2Q does not.



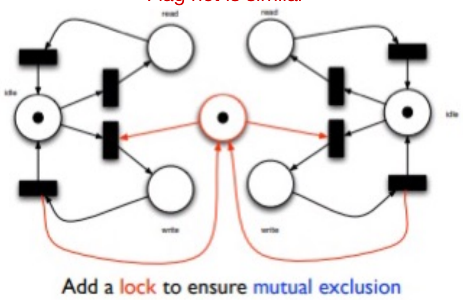
A1 Q7
P = (a -> b -> d -> P).
Q = (c -> b -> e -> Q).
||S1 = (P || Q).

S2 = (a -> S2A | c -> S2B),
S2A = (c -> b -> d -> S2C | c -> b -> e -> S2D),
S2B = (a -> b -> d -> S2C | a -> b -> e -> S2D),
S2C = (e -> S2 | a -> e -> S2A),
S2D = (d -> S2 | c -> d -> S2B).

For the above FSPs, they both share the same LTS diagram, however, since ||S1 has simultaneous actions, its petri net will be show simultaneity whereas S2 will not.

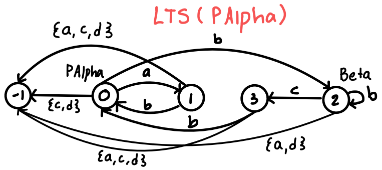


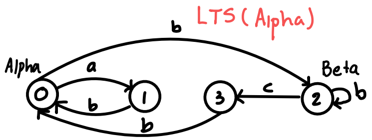
Flag net is similar

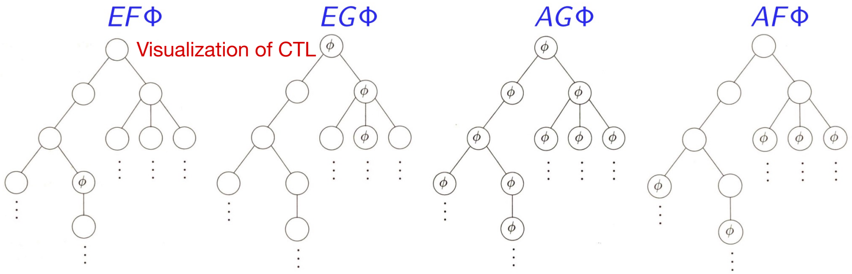


PROPERTY ALPHA AND ALPHA LTS

property Alpha = (a->b->Alpha | b->Beta)+{d}
Beta = (c->b->Alpha | b->Beta) **AND**
Alpha = (a->b->Alpha | b->Beta)+{d}
Beta = (c->b->Alpha | b->Beta)







ALPHA1 = PROPERTY ALPHA FSP

ALPHA1 = (c->ERROR | d->ERROR | a->A1 | b->BETA),
A1 = (b->ALPHA1 | a->ERROR | c->ERROR | d->ERROR),
BETA = (b->BETA | c->B1 | a->ERROR | d->ERROR),
B1 = (b->ALPHA1 | a->ERROR | c->ERROR | d->ERROR).