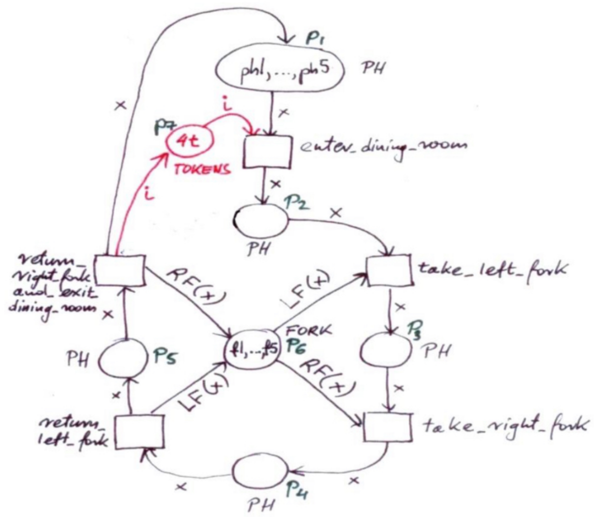
1) Diagram



$M(P_1) + M(P_2) + M(P_3) + M(P_4) + M(P_5) = ph_1 + ph_2 + ph_3 + ph_4 + ph_5$

$LF(M(P_3)) + RF(M(P_4)) + M(P_6) = f1 + f2 + f3 + f4 + f5$

$M(p7) + M(p5) = 4t$

i1) $M(P_1) + M(P_2) + M(P_3) + M(P_4) + M(P_5) = PH$

i2) $LF(M(P_4)) + RF(M(P_5)) + M(P_6) = FORK$

i3) $M(p7) = TOKEN$

Assume M reachable from initial marking

a) $M(p_4) + M(p_5) \neq \emptyset$

Means philosophers are putting down forks, as $M(p_4)$

Exists p_i in p_4 and p_j in p_5, meaning the transitions can be fired.

b) $M(p_4) + M(p_5) = \emptyset$

B/c $M(p_4) + M(p_5) = \emptyset$, using i1 makes it that all the philosophers are not eating as they are in p_1, p_2, p_3. Transition enter room fires b/c token present and there is a philospher in p_1. p_2 fires b/c all the forks are present and thus p_3 can also fire, leading to p_4.

.

2)

a) N customers M pumps

```
const N= 3
const M=2
range C=1..N
range P=1..M
range  A=1..2    //gas amount

CUSTOMER =(prepay[a:A]->  pump  [x:A]->
           if (x=a) then CUSTOMER else ERROR).

||CUSTOMERS_ALL= CUSTOMER[N]: CUSTOMER.
DELIVER = ( CUSTOMER[ i : C].prepay[a: P]->CUSTOMER.deliver [a]-> DELIVER ).

PUMP= ( CUSTOMER[i : C].deliver[p:P] -> CUSTOMER[N] .pump[p]-> PUMP).
```

b) FIFO

Append the following

```
property FIFO= (CUSTOMER[i :C].prepay[G]->FIFO[i])

FIFO [i: C]= ( CUSTOMER[i]. pump[p:P]->FIFO |
               CUSTOMER[o : N]. pump[p:P]  .prepay [G]->
               FIFO[i,o]).

FIFO[i:C][o:C] = (CUSTOMER[i][o].deliver[G]->FIFO[i]).

|| FINAL_2 =(FIFO||FINAL)
```

customers are served in the order they pay so ∴ holds

c) CODE

```
||  STATION   = forall[p:P](pump[p]::PUMP || DELIVER) /{deliver/deliver [P], pump/pump [P], prepay/prepay[P] }.

|| FINAL =(STATION || CUSTOMERS_ALL) \{ CUSTOMER[N].deliver}.
```

3)

a) set Bold = {bold[1..2]}
set Meek = {meek[1..2]}

set Customers = {Bold, Meek}
progress

CUSTOMER = (getcheese → cheese → CUSTOMER).

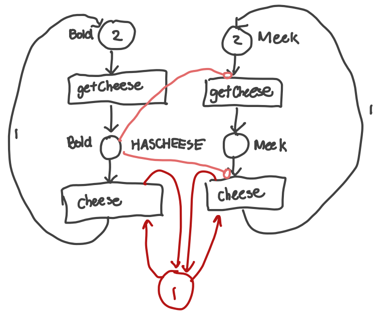
where getcheese is buying cheese and cheese is already obtaining it.

COUNTER = (getCheese → CUSTOMER)

||FINAL = (Customers::CUSTOMER || Customers::COUNTER)

FINAL2 = FINAL << {Bold.getcheese}

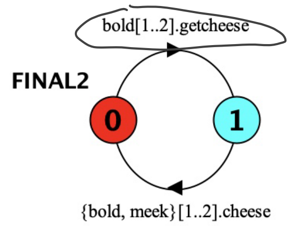
b)



Bold will always execute but meek can be inhibited

ADD THIS:

c) progress BOLD = {Bold.getcheese}
progress MEEK = {Meek.getcheese}



bold can keep getting cheese but meeks can starve

4) const MT = 4
set Bold = {bold[1..2]}
set Meek = {meek[1..2]}
range T = 1..MT
set Customers = {Bold, Meek}

CUSTOMER = (ticket[t:T] → getcheese[t] → cheese → CUSTOMER).
TICKET = TICKET[1],
TICKET[t:T] = (ticket[t] → TICKET[t%MT+1]).

DISPLAY = DISPLAY[1]
DISPLAY[t:T] = (getcheese[t] → DISPLAY[t%MT+1]).

||COUNTER =
(Customers : CUSTOMER || {Customers}:TICKET} ||
{Customers}::COUNTER).

progress BOLD = {Bold.getcheese[T]}
progress MEEK = {Meek.getcheese[T]}
FINAL = COUNTER >> {Meek.getcheese}

5) CODE

6)

```
const N=3
range M=0..N
set P={[M],[M],[M]}

PORT =(send [x:M]->PORT[x]),
PORT[h:M]=(send [x:M]->PORT[x][h]
          |receive[h]->PORT),

PORT [t:S][h:M]
     =(send[x:M]->PORT[x][t][h]
      |receive[h]->PORT[t]).

||APORT=PORT/{send/send[M], receive/receve[M]}.
PRODUCER =PRODUCER[0]
PRODUCER[i:M]  =(send[x:M]->PRODUCER[(i+1)%4]).

RECEIVER=(receive[x:M]-> RECEIVER).
||FINAL=(APORT|| PRODUCER||RECEIVER).
```

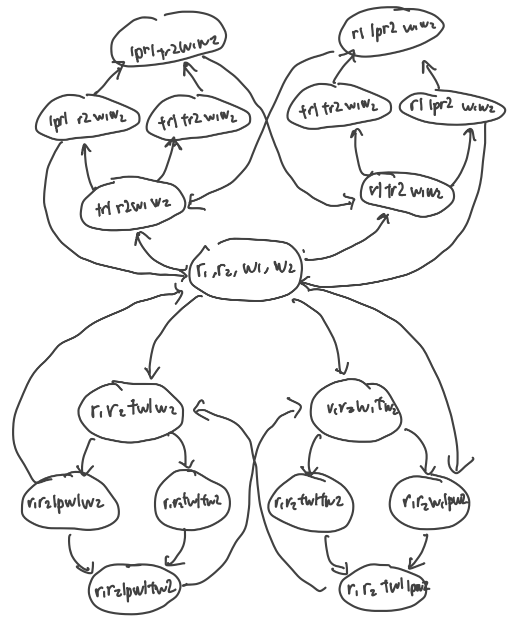
8) 2 Readers and 2 writers

lpri lpr1, lpr2 } actions
lpwi lpw1, lpw2

tri tr1, tr2 } requests
twi tw1, tw2

ri r1, r2 } netral states
wi w1, w2

4 tuple for all actions



Satisfies liveness, safety, non-blocking, and no strict sequencing

7)

a)

i) $\neg p \Rightarrow r$

$s_0 \models \neg p \Rightarrow r$ since $L(s_0)=\{r\}$

$s_2 \models \neg p \Rightarrow r$ since $r \notin L(s_2)$ and $p = L(s_2)$

ii) $\neg EGr$

$s_0 \not\models \neg EGr$ since r can go to s_1 and loops infinitely. $s_0 \to s_1 \to s_1 \ldots$

$s_2 \models \neg EGr$ b/c $r \notin L(s_2)$

iii) $E(tUq)$

$s_0 \not\models E(tUq)$ b/c s_0 has no t so there is no path

$s_2 \not\models E(tUq)$ b/c s_2 has no t so no path

iv) Fq

$s_0 \models Fq$ b/c $s_0 \to s_2$ and s_2 has q

$s_2 \models Fq$ b/c $s_2 \to s_0 \to s_2$ and s_2 has q

b) p precedes s and t on all paths

CTL: $A(pU(s \wedge t))$

LTL: $pU(s \wedge t)$

c) Between $q \wedge r$, p is never true but t is always true

CTL: $AG(q \Rightarrow A[\,(p=\bot \wedge t=\top)Ur])$

LTL: $q \Rightarrow (p=\bot \wedge t=\top)\,U r$

d) Φ is always true infinitely along every path starting at s

CTL: $AG(AF\Phi)$

LTL: $G(F\Phi)$

e) When $p \to q$, no r until t

CTL: $(pEXq) \Rightarrow AX(A(r=\bot\, U\, t))$

LTL: $pXq \Rightarrow r=\bot\, U\, t$

f) Between q and r, p is never true

CTL: $AG(q \wedge r \Rightarrow p=\bot)$

LTL: $G(q \wedge r \Rightarrow p=\bot)$