1a

i).

ALARM = ALARM[5],

ALARM[t:0..5] = if (t==0)

then (beep -> STOP)

else (snooze -> ALARM

|tick[t] -> ALARM[t-1]

|off -> STOP).

ii). When minimised:

PROGRAM = (write -> WRITTEN[1]),

WRITTEN[i:1..3]

= (compile -> ok -> run -> STOP

|when (i<3) compile -> error[i] -> fix[i] -> WRITTEN[i+1]

|when (i==3) compile -> ERROR).

iii).

property HAPPY = LIFE,

LIFE = (wakeup -> eat -> REST),

REST = (sleep -> LIFE

|rest -> REST

|read -> READ

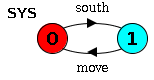
|surf -> SURF),

READ = (read -> READ | eat -> REST),

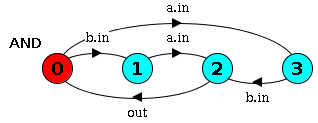
SURF = (surf -> SURF | eat -> REST).

1b

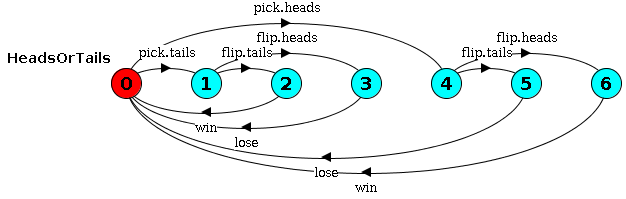
i).



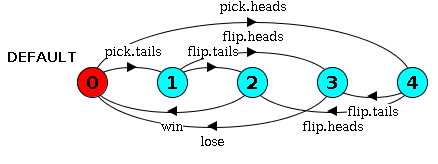
ii).



iii).



or minimised:



1c

i).

ii).

Yes. The first thread acquires the lock on this object when it enters the synchronised method. However, when it calls wait() it on the same object, it releases the lock a places itself in the wait set of that object. When another thread calls foo(), on the entrance to the synchronised method it waits for the lock to be released, then acquires it and prints “Hello World!”.

2a

FLEXDEBT = FD[0][0],

FD[d:0..M][frozen:0..1]

= (when (d>0) deposit -> FD[d-1][frozen]

|when (d<M && !frozen) withdraw -> FD[d+1][frozen]

|when (!frozen) freeze -> FD[d][1]

|when (frozen) unfreeze -> FD[d][0]).

2b

public class FlexDebit {

private final int maxDebt;

private int debt;

private boolean frozen;

public FlexDebit(int maxDebt) {

this.maxDebt = maxDebt;

debt = 0;

frozen = false;

}

public synchronized deposit() {

while (debt == 0) wait();

debt--;

notifyAll();

}

public synchronized withdraw() {

while (debt == maxDebt || frozen) wait();

debt++;

notifyAll();

}

public synchronized freeze() {

while (frozen) wait();

frozen = true;

notifyAll();

}

public synchronized unfreeze() {

while (!frozen) wait();

frozen = false;

notifyAll();

}

}

2c

1)

Assuming both users are connected to the same account

set Users = {lina,will}

||SYSTEM = (Users:USER || Users::FLEXDEBT || ADMIN)

/{{freeze,unfreeze}/Users.{freeze, unfreeze}}.

2)

property NO\_WITHDRAW\_WHEN\_FROZEN = NWWF,

NWWF = (Users.withdraw -> NWWF

|freeze -> unfreeze -> NWWF).

~~property NO\_WITHDRAW\_WHEN\_FROZEN~~

~~= (freeze -> FROZEN~~

~~| {bob, bill}.withdraw -> NO\_WITHDRAW\_WHEN\_FROZEN~~

~~| {bob, bill}.deposit -> NO\_WITHDRAW\_WHEN\_FROZEN~~

~~),~~

~~FROZEN = ({bob, bill}.deposit -> FROZEN~~

~~| unfreeze -> NO\_WITHDRAW\_WHEN\_FROZEN~~

~~)+{bob.withdraw, bill.withdraw}.~~

3)

progress LINA = {lina.withdraw}

progress WILL = {will.withdraw}

progress ALWAYS\_WITHDRAW = ??

(This could also be written as

progress ALWAYS\_WITHDRAW[u:Users] = {u.withdraw})

[Piazza post](https://piazza.com/class/itv4vb3e5wm8i?cid=40)

Yes as it will starve will of withdrawals, every time one user can withdraw, the scheduler will choose lina.

**Another approach** - Not completely sure if this is right.

progress ALWAYS\_WITHDRAW = {will.withdraw , lina.withdraw}

No, the question asks that **either** of the users to be able to withdraw. Although lina would withdraw over will every time, it would still be the case that either of the users is able to withdraw at anytime. It doesn’t have to apply to both Will and Lina, just Lina will do.