/\*Lab 3 & 4

\*

\* Name: Nejada

\* Surname: Karriqi

\*

\* Name: Jonnatan

\* Surname: Mendoza Escobar

\*

\*/

LAB 3: To extend the alphabets we use the actions in card.CardActions.

WRITER = (write[1]->write[3]->WRITER) + {write[0..3]}.

property MUTEX = ....

||FIELD = (n1:NEIGHBOR1 ||n2: ... || {n1,n2}:: ... || MUTEX).

The below implementation includes the safety property MUTEX and the FIELD process:

const False = 0

const True = 1

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]

).

||FLAGS = (flag1:BOOLVAR || flag2:BOOLVAR).

NEIGHBOR1 = (flag1.setTrue -> TEST), //he is able to enter and get berries, exit and lower the flag when the process has finished.

TEST = (flag2[raised:Bool] -> if (raised) then(flag1.setFalse -> NEIGHBOR1)

else (enter -> pick\_berries -> exit -> flag1.setFalse -> NEIGHBOR1)) + {{flag1, flag2}.BoolActions}.

// Same for Neighbor2

NEIGHBOR2 = (flag2.setTrue -> TEST),

TEST = (flag1[raised:Bool] -> if (raised) then (flag2.setFalse -> NEIGHBOR2)

else (enter -> pick\_berries -> exit -> flag2.setFalse -> NEIGHBOR2)) + {{flag1,flag2}.BoolActions}.

// the issue here is to check if the neighbors exit before the other one enters.

property MUTEX = (n1.enter -> n1.exit -> MUTEX

|n2.enter -> n2.exit -> MUTEX

).

||FIELD = (n1:NEIGHBOR1 || n2:NEIGHBOR2 || {n1,n2}::FLAGS || MUTEX).

The progress properties are used to check that both neighbors have the opportunity to enter:

progress ENTER1 = {n1.enter}

progress ENTER2 = {n2.enter}

There exist options that would prevent always the neighbors progress. Imagine a situation where one of them raises the flag directly after exiting. The other neighbor waiting wouldn’t have any chance to access the field. If both of them would do this, then none of them would really go inside to get berries. So, all they could do is repeat the process without making any progress.

The greedy neighbor gives high priority to the raising flag. The way it does this is by giving priority to each neighbor’s flag setTrue action by leading again to the above situation where no progress is done.

||GREEDY = FIELD << {{n1, n2}. {flag1,flag2}.setTrue}.

If we check the progress properties when we use greedy neighbors, we find progress violations in both.

Progress violation: ENTER2 ENTER1

Trace to terminal set of states:

n2.flag2.setTrue

Cycle in terminal set:

n1.flag1.setTrue

n1.flag2.1

n1.flag1.setFalse

Actions in terminal set:

{n1.{flag1.{setFalse, setTrue}, flag2[1]}, n2.{flag1[1], flag2.{setFalse, setTrue}}}

Progress Check in: 5ms

Warring Neighbors: Peterson's Exclusion Algorithm

Peterson's Exclusion Algorithm below includes the greedy version of the Field process

and the safety and progress properties to check the proper behavior of the process:

const False = 0

const True = 1

range Bool = False..True

range Card = 1..2

set BoolActions = {setTrue, setFalse, [False], [True]}

set CardActions = {set1, set2, [1], [2]} // New set of actions for the turn system

BOOLVAR = VAL[False],

VAL[v:Bool] = (setTrue -> VAL[True]

|setFalse -> VAL[False]

|[v] -> VAL[v] // change the state

).

||FLAGS = (flag1:BOOLVAR || flag2:BOOLVAR).

// Turn indicator

CARDVAR = VAL[1],

VAL[i:Card] = (set1 -> VAL[1]

|set2 -> VAL[2]

|[i] -> VAL[i]

).

||TURNS = (FLAGS || card:CARDVAR).

When a neighbor wants to access the field, he raises his flag and sets the turn indicator to the other neighbor. He has to wait till one of the other neighbor’s conditions (flag raised, turn= 1) change, to have again the chance to enter. Both neighbors will wait their turn to enter, even in the cases when they are greedy.

As in the previous exercise, we need to extend the alphabets with the actions in card.CardActions.

NEIGHBOR1 = (flag1.setTrue -> card.set2 -> TEST),

TEST = (flag2[raised:Bool] -> card[turn:Card] -> if (raised && turn == 2) then (active\_wait -> TEST)

else (enter -> pickberries -> exit -> flag1.setFalse -> NEIGHBOR1)) + {{flag1,flag2}.BoolActions, card.CardActions}.

// Neighbors are symmetric

NEIGHBOR2 = (flag2.setTrue -> card.set1 -> TEST),

TEST = (flag1[raised:Bool] -> card[turn:Card] -> if (raised && turn == 1) then (active\_wait -> TEST)

else (enter -> pickberries -> exit -> flag2.setFalse -> NEIGHBOR2)) + {{flag1,flag2}.BoolActions, card.CardActions}.

// check the process' safety

property MUTEX = (n1.enter -> n1.exit -> MUTEX

|n2.enter -> n2.exit -> MUTEX

).

||FIELD = (n1:NEIGHBOR1 || n2:NEIGHBOR2 || {n1,n2}::TURNS || MUTEX).

Progress properties are used to check if both neighbors are able to pick berries.

progress BERRIES1 = {n1.pickberries}

progress BERRIES2 = {n2.pickberries}

We can see that by using the Peterson's Exclusion Algorithm the progress properties are satisfied even when the neighbors are greedy.

||GREEDY\_FIELD = FIELD << {{n1, n2}.{flag1, flag2}.setTrue}.