Option Solutions (Chapter 7, numbers 7.6 and 7.7 in book)

6 Based on Dekker/Dijkstra algorithm:

```
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).

NEIGHBOUR1 = (flag1.setTrue -> TEST),
TEST       = (flag2[b:Bool] ->
             if(b) then
               (flag1.setFalse -> NEIGHBOUR1)
             else
               (enter -> exit -> flag1.setFalse -> NEIGHBOUR1)
           )+{{flag1,flag2}.BoolActions}.

NEIGHBOUR2 = (flag2.setTrue -> TEST),
TEST       = (flag1[b:Bool] ->
             if(b) then
               (flag2.setFalse -> NEIGHBOUR2)
             else
               (enter -> exit-> flag2.setFalse -> NEIGHBOUR2)
           )+{{flag1,flag2}.BoolActions}.

property SAFETY = (n1.enter -> n1.exit -> SAFETY | n2.enter -> n2.exit ->
SAFETY).

||FIELD = (n1:NEIGHBOUR1 || n2:NEIGHBOUR2 || {n1,n2}::FLAGS || SAFETY).

progress ENTER1  = {n1.enter}  //NEIGBOUR 1 always gets to enter
progress ENTER2  = {n2.enter}  //NEIGHBOUR 2 always gets to enter

/* greedy neighbours - make setting the flags high priority - eagerness to enter*/

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

/* progress violations show situation where neither neighbour enters
*  each continually retests the lock
*/
```

7. Peterson's Algorithm for two processes (Peterson G.L. 1981):

```
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]
            ).

set   CardActions = {set1,set2,[1],[2]}
range Card =  1..2
CARDVAR     = VAL[1],
VAL[i:Card] = (set1 -> VAL[1]
            |set2 -> VAL[2]
            |[i]  -> VAL[i]
            ).

||VARS = (flag1:BOOLVAR || flag2:BOOLVAR || turn:CARDVAR).

NEIGHBOUR1 = (flag1.setTrue -> turn.set2 -> TEST),
TEST       = (flag2[b:Bool] -> turn[c:Card] ->
              if(b && c==2) then
                TEST
              else
                (enter -> exit -> flag1.setFalse -> NEIGHBOUR1)
            )+{{flag1,flag2}.BoolActions, turn.CardActions}.

NEIGHBOUR2 = (flag2.setTrue -> turn.set1 -> TEST),
TEST       = (flag1[b:Bool] -> turn[c:Card] ->
              if(b && c ==1) then
                 TEST
              else
                (enter -> exit-> flag2.setFalse -> NEIGHBOUR2)
            )+{{flag1,flag2}.BoolActions, turn.CardActions}.

property SAFETY = (n1.enter -> n1.exit -> SAFETY | n2.enter -> n2.exit -> SAFETY).

||FIELD = (n1:NEIGHBOUR1 || n2:NEIGHBOUR2 || {n1,n2}::VARS || SAFETY).

progress ENTER1  = {n1.enter}  //NEIGBOUR 1 always gets to enter
progress ENTER2  = {n2.enter}  //NEIGHBOUR 2 always gets to enter

/* greedy neighbours - make setting the flags high priority - eagerness to enter*/

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

/* progress violation does not now occur due to the turn indicator
*/
```