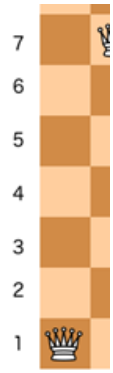
Week 4

Author: Florin Deleanu

Assignment 1- 5 Rooks

wk 4 assignments (III)



- see https://en.wikipedia.org/wiki/Eight_queens_puzzle
- simplifications:
  - 5x5
  - rooks: not in the same row, nor same column
  - and: no direct neighbors

Because of the way I approached this assignment the board size does not matter, we would only have to change the range(5) to range(X),R[i]<X+1 and the solver would solve for a board of size X by X with X number of rooks.

```
# 5x5 rooks problem
#R_1 is the rook on the first column, R_3 rook on third column etc
# R_1=2 means the rook on first column is on the second row etc
R=[Int('R_%i' % (i+1)) for i in range(5)]
print(R)
print("")

#rooks between 1 and 5 in value
valConstraint=[And(R[i]>0,R[i]<6) for i in range(5)]
print(valConstraint)
print("")

#rooks can't be on the same row
rowConstraint=[Distinct(R)]
print(rowConstraint)
print("")

s=Solver()
s.add(valConstraint)
s.add(rowConstraint)

print(s.check())
print(s.model())
```

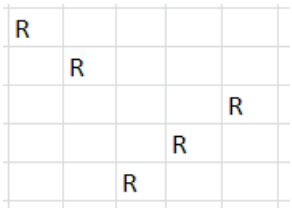
```
[R_1, R_2, R_3, R_4, R_5]

[And(R_1 > 0, R_1 < 6), And(R_2 > 0, R_2 < 6), And(R_3 > 0, R_3 < 6), And(R_4 > 0, R_4 < 6), And(R_5 > 0, R_5 < 6)]

[Distinct(R_1, R_2, R_3, R_4, R_5)]

sat
[R_1 = 1, R_2 = 2, R_3 = 5, R_4 = 4, R_5 = 3]
```

Solution from the above result can be visualized below



Assignment 2- a,b values

Consider the following program:

```
for i := 1 to 10 do
    if a > b then
        b := 2b; a := a - 3
    else
        a := 2a; b := b - 5
    fi
od
```

Find initial values for a and b such that afterwards a = 1000 and b = 999.

Here I reused python list to automate the solving instead of manually typing the values

```
#a,b values assignment

#a[0]=a1
a=[Int('a%i' % (i)) for i in range(1,11)]
print(a)
print("")
b=[Int('b%i' % (i)) for i in range(1,11)]
for i, v in enumerate(b):
     print(i, v)
print("")

#TEST visualize in python
# for i in range(1,11):
#     if a>b:
#         b=2*b
#         a=a-3
#     else:
#         a=2*a
#         b=b-5
s=Solver()

#TEST first is the initial value, then the subsequent values depend on the former ones
# s.add( If(a[0]>b[0], And(b[1]==2*b[0],a[1]==a[0]-3), And( a[1]==2*a[0], b[1]==b[0]-5 ) ) )

#looping the condition
for i in range(1,10):
    s.add( If(a[i-1]>b[i-1], And(b[i]==2*b[i-1],a[i]==a[i-1]-3), And( a[i]==2*a[i-1], b[i]==b[i-1]-5 ) ) )
#last values
s.add(a[9]==1000,b[9]==999)

print(s.check())
print(s.model())
```

```
sat
[a4 = 128,
 a2 = 32,
 b6 = 256,
 b5 = 128,
 a3 = 64,
 b2 = 143,
 a6 = 253,
 b1 = 148,
 a5 = 256,
 a7 = 506,
 a1 = 16,
 b3 = 138,
 b7 = 251,
 a8 = 503,
 b8 = 502,
 b4 = 133,
 a9 = 500,
 b9 = 1004,
 b10 = 999,
 a10 = 1000]
```

These results can be visualized in the spreadsheet below and also checked by the excel with a formula introduced manually from the assignment to prove they are right

H2 f_x =IF(B2>F2,B2-3,2*B2)

A	B	C	D	E	F	G	H	I
							next a	next b
a1	16			b1	148		32	143
a2	32			b2	143		64	138
a3	64			b3	138		128	133
a4	128			b4	133		256	128
a5	256			b5	128		253	256
a6	253			b6	256		506	251
a7	506			b7	251		503	502
a8	503			b8	502		500	1004
a9	500			b9	1004		1000	999
a10	1000			b10	999			

Assignment 3 – Missionaries Cannibals

- 3 wolfs + 3 rabbits (aka. 3 cannibals + 3 missionaries):
  - they must all pass the river to the other shore
  - at *any* shore at *any* time: there should not be more wolfs than rabbits (because the rabbits will be eaten)
  - one boat; for each crossing: at least 1 animal, and max 2 animals
  - (hint: try with e.g. buttons + candy's; there are 11 crossings needed)
  - parse the z3-output and display how the river crossings are done for all time stamps (keep it simple)

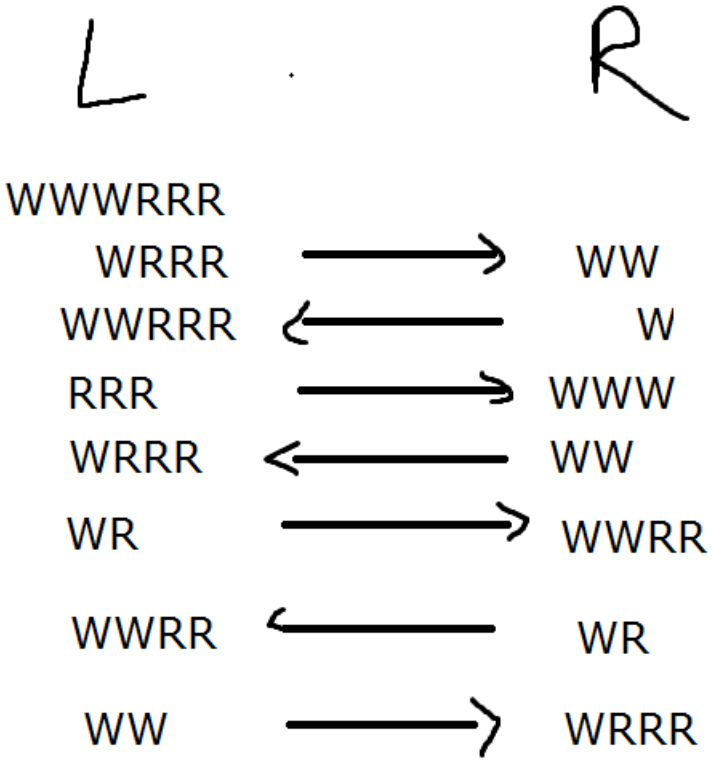
The input code for this can be found in the jupyter notebook file. The function names and definitions should mostly be self explanatory, otherwise I added some comments for better clarity

To analyze the output:

```
sat
[rabbits = [(0, left) -> 3,
            (1, left) -> 3,
            (2, left) -> 3,
            (3, left) -> 3,
            (4, left) -> 3,
            (5, left) -> 1,
            (5, right) -> 2,
            (6, left) -> 2,
            (6, right) -> 1,
            (7, right) -> 3,
            (8, right) -> 3,
            (9, right) -> 3,
            (10, right) -> 3,
            (11, right) -> 3,
            else -> 0],

 wolves = [(0, left) -> 3,
           (0, right) -> 0,
           (1, left) -> 1,
           (2, right) -> 1,
           (3, left) -> 0,
           (3, right) -> 3,
           (4, left) -> 1,
           (5, left) -> 1,
           (6, right) -> 1,
           (7, right) -> 1,
           (8, left) -> 3,
           (8, right) -> 0,
           (9, left) -> 1,
           (10, right) -> 1,
           (11, left) -> 0,
           (11, right) -> 3,
           else -> 2]]
```

This output picture is equivalent to the following movements drawn for better visualisation



In words:

1. WW cross
2. W comes back
3. WW cross
4. W comes back
5. RR cross
6. WR come back
7. RR cross
8. All rabbits are on right side, move the wolves until finished

Assignment 4- Bridge crossing

Albert, Bob, Charlie and Fred want to cross a bridge.

- The bridge can carry at most two people at the same time.
- In order to cross the bridge safely, a torch is needed, and they have only one torch.
- The four men walk at different speeds. It takes Albert 10 minutes to cross the bridge, Bob 5 minutes, Charlie 2 minutes and Fred 1 minute.

What is the fastest way for all four men to cross the bridge?

The input code along with informative comments for this can be found in the jupyter notebook file. The approach for this is similar to the above assignments in the way that I used time to describe the position of each person and crossings are alternative.

To analyze the output:

```
sat
[alb = [3 -> right, 4 -> right, 5 -> right, else -> left],
 fre = [0 -> left, 4 -> left, else -> right],
 bob = [3 -> right, 4 -> right, 5 -> right, else -> left],
 lapsed = [1 -> 2,
           2 -> 4,
           3 -> 14,
           4 -> 15,
           5 -> 17,
           else -> 0],
 cha = [1 -> right, 5 -> right, else -> left]]
```

Better visualized:

	LEFT		RIGHT	Lapsed
t0	abcf	-->		
t1	ab	<--	fc	2
t2	abc	-->	f	4
t3	c	<--	abf	14
t4	cf	-->	ab	15
t5			abcf	17

In words:

1. F,C cross
2. C comes back
3. A,B cross
4. F comes back
5. FC cross