# SLUGS

## SLUGS: SmalL bUt Complete GROne Synthesizer

- Slugs is a stand-alone reactive synthesis tool for generalized reactivity(1) synthesis.
- Free, open source and available at: https://github.com/VerifiableRobotics/slugs

## Using SLUGS

Step 1: Model the 2-player game in .structuredslugs format

Step 2: Convert the file to .slugsin using: slugs/tools/Struc../compiler.py
$ python slugs/tools/Struc../compiler.py [filename].structuredslugs

Step 3: Run slugs with .slugsin file as input and the option you need
$ slugs [filename.slugsin] --explicitStrategy --jsonOutput > [output_filename]

# The Structured Slugs Language

**Variable Definitions:**

```
[INPUT]
a
b:0...10

[OUTPUT]
c:2...8
d
```

**Safety Formula G (...):**

```
[ENV_TRANS]
a -> (a' <-> ! a)
b' = b + 1

[SYS_TRANS]
d -> (c' = 3)
```

**Initial conditions:**

```
[ENV_INIT]
! a
b = 1

[SYS_INIT]
d
c = 4
```

**Conjunction of Liveness Formulas G F (...):**

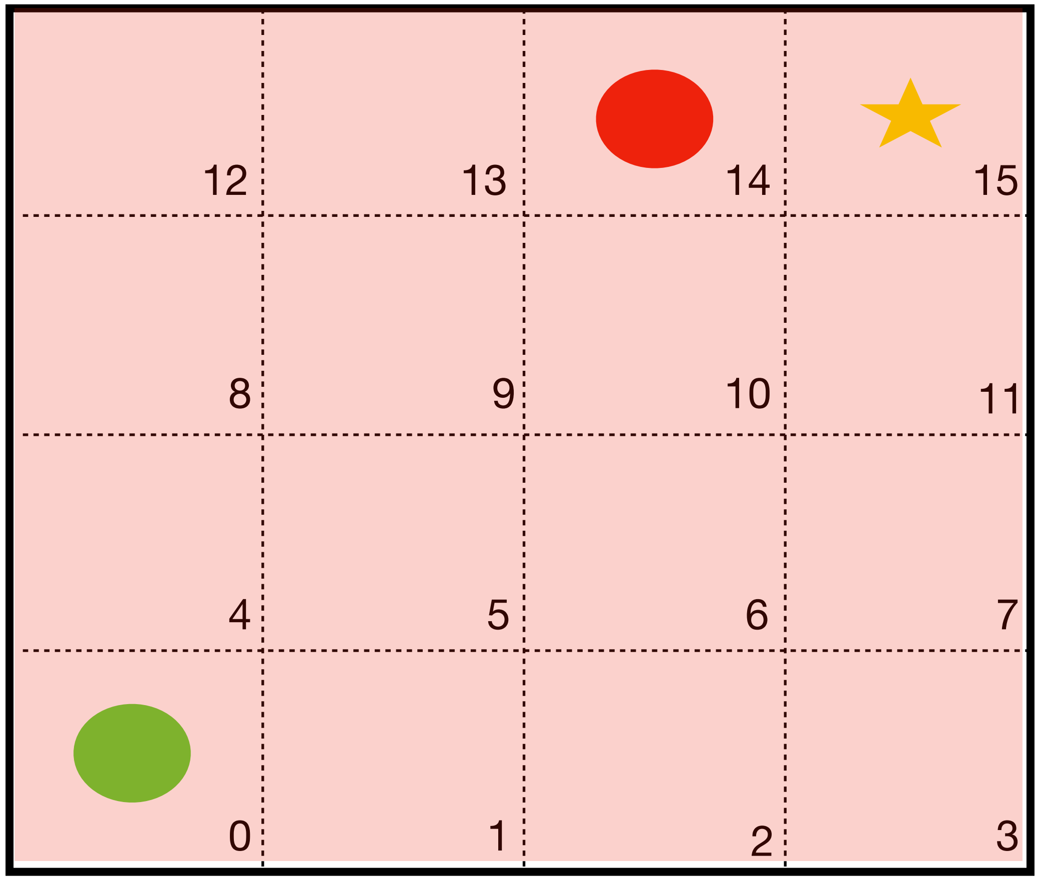
```
[ENV_LIVENESS]
! a | (b = 3)

[SYS_LIVENESS]
d
c = 2
```

# SLUGS Example

**Model all the allowed transitions of the agent and the obstacle in LTL**

```
[ENV_TRANS]
o_state == 0 -> (o_state' == 1) || (o_state' == 4) || (o_state' == 0)
o_state == 4 -> (o_state' == 0) || (o_state' == 8) || (o_state' == 4) || (o_state' == 5)
.
.
[SYS_TRANS]
a_state == 0 -> (a_state' == 1) || (a_state' == 4) || (a_state' == 0)
a_state == 4 -> (a_state' == 0) || (a_state' == 8) || (a_state' == 4) || (a_state' == 5)
```




**The agent must reach the starred location and never crashes into the moving obstacle.**

# SLUGS Example

**Model all the allowed transitions of the agent and the obstacle in LTL**

```
[ENV_TRANS]
o_state == 0 -> (o_state' == 1) || (o_state' == 4) || (o_state' == 0)
o_state == 4 -> (o_state' == 0) || (o_state' == 8) || (o_state' == 4) || (o_state' == 5)
.
.
[SYS_TRANS]
a_state == 0 -> (a_state' == 1) || (a_state' == 4) || (a_state' == 0)
a_state == 4 -> (a_state' == 0) || (a_state' == 8) || (a_state' == 4) || (a_state' == 5)
```

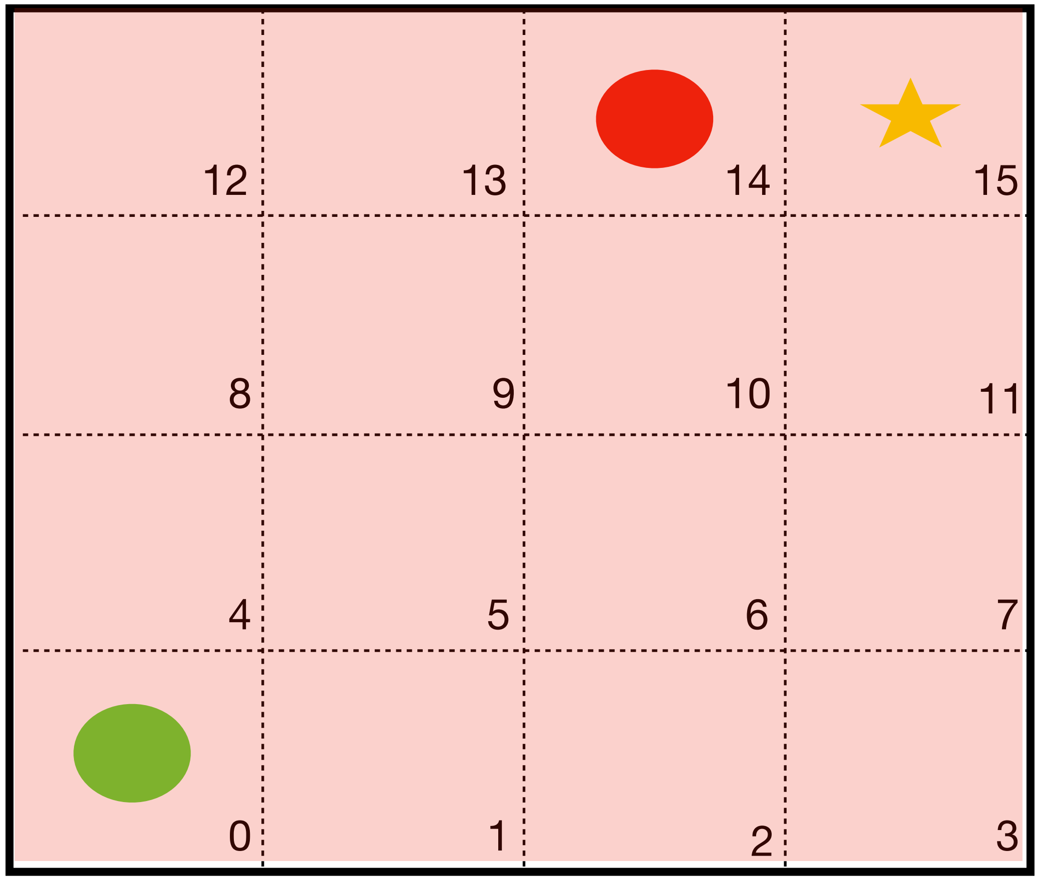
**Encode the properties as either safety or liveness LTL specifications**

the agent must visit locations 15 and 0 infinitely often

```
[SYS_LIVENESS]
a_state == 15
a_state == 0
```

the agent and obstacle must never crash

```
[SYS_TRANS]
o_state != a_state
o_state' != a_state
o_state != a_state'
```




**The agent must reach the starred location and never crashes into the moving obstacle.**

# SLUGS Example

**Model all the allowed transitions of the agent and the obstacle in LTL**

```
[ENV_TRANS]
o_state == 0 -> (o_state' == 1) || (o_state' == 4) || (o_state' == 0)
o_state == 4 -> (o_state' == 0) || (o_state' == 8) || (o_state' == 4) || (o_state' == 5)
.
.
[SYS_TRANS]
a_state == 0 -> (a_state' == 1) || (a_state' == 4) || (a_state' == 0)
a_state == 4 -> (a_state' == 0) || (a_state' == 8) || (a_state' == 4) || (a_state' == 5)
```

**Encode the properties as either safety or liveness LTL specifications**

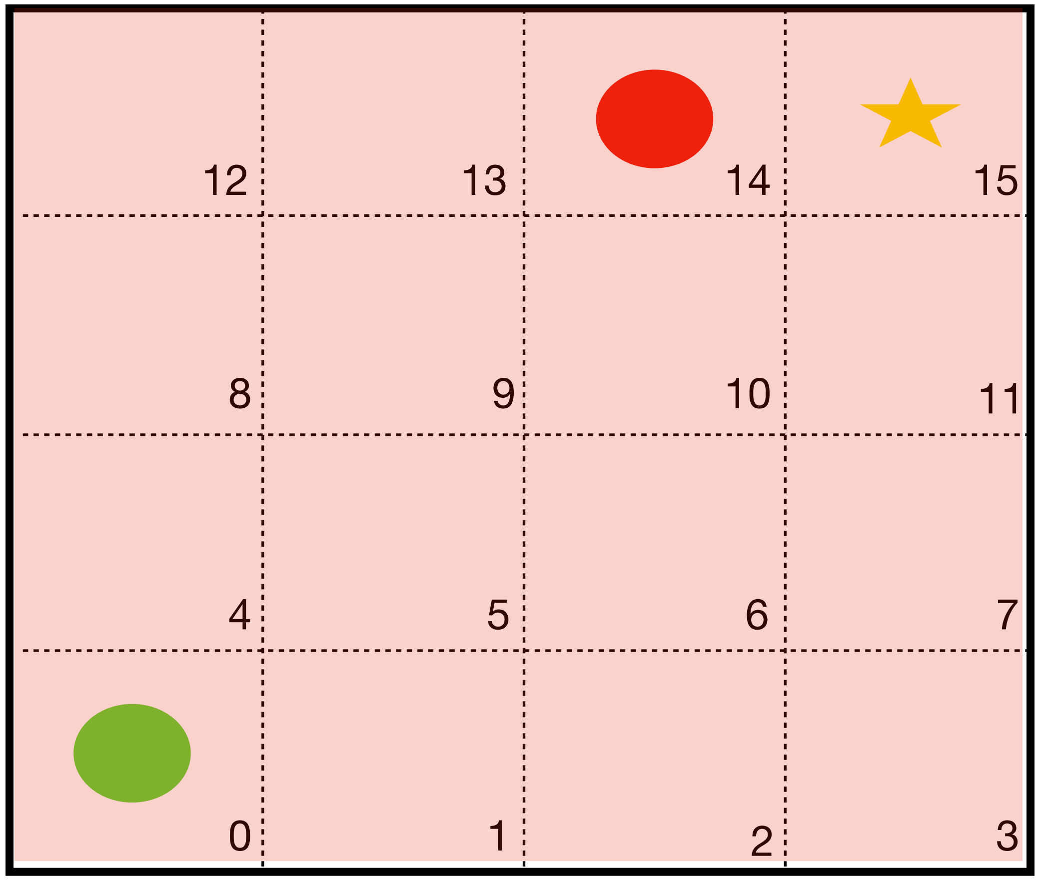
the agent must visit locations 15 and 0 infinitely often

```
[SYS_LIVENESS]
a_state == 15
a_state == 0
```

the agent and obstacle must never crash

```
[SYS_TRANS]
o_state != a_state
o_state' != a_state
o_state != a_state'
```

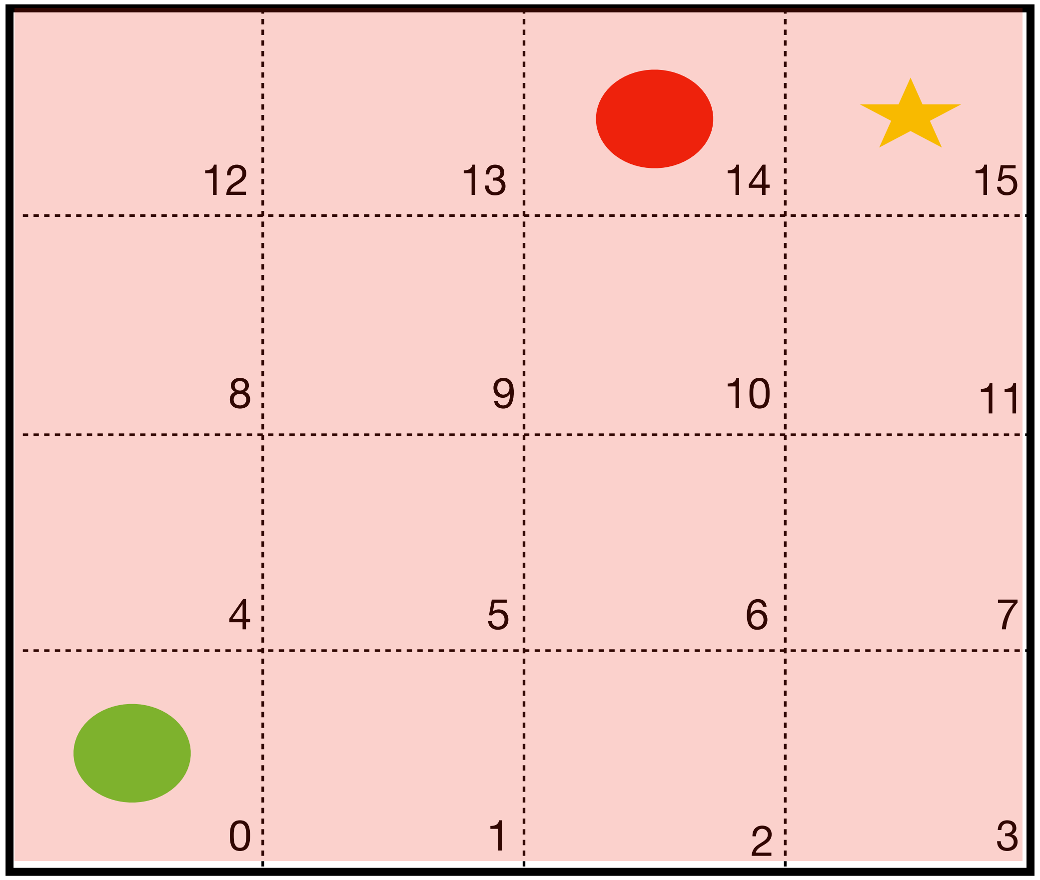
This problem is unrealizable




**The agent must reach the starred location and never crashes into the moving obstacle.**

# SLUGS Example

**Incrementally add reasonable assumptions on the environment until it's realizable**




**The agent must reach the starred location and never crashes into the moving obstacle.**

# SLUGS Example

**Incrementally add reasonable assumptions on the environment until it's realizable**

Example:

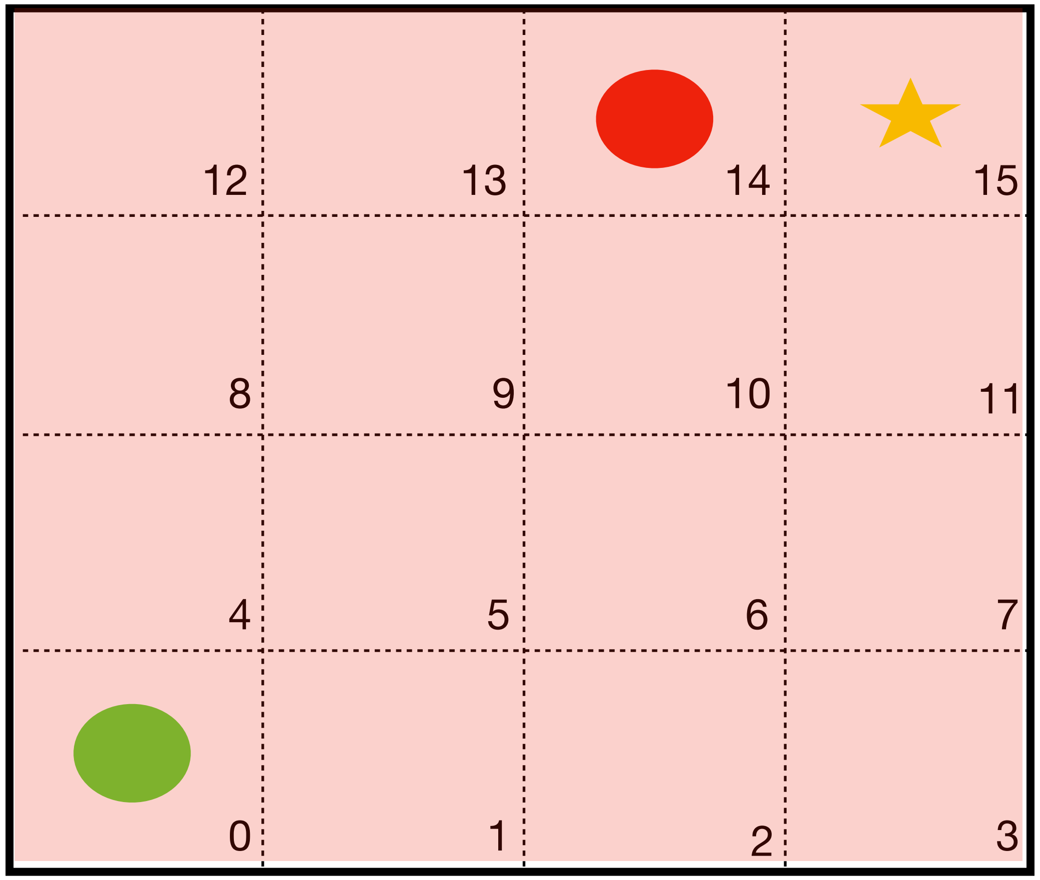
assume that the moving obstacles visits locations 0 and 15 infinitely often

G (F (o_state = 15) ⋀ F (o_state = 0))

=

```
[ENV_LIVENESS]
o_state == 15
o_state == 0
```

Now we can synthesize a reactive controller for this problem




**The agent must reach the starred location and never crashes into the moving obstacle.**

# The PRISM tool

- PRISM: Probabilistic symbolic model checker
  - developed at Birmingham/Oxford University, since 1999
  - free, open source (GPL)
  - versions for Linux, Unix, Mac OS X, Windows, 64-bit OSs
- Modelling of:
  - DTMCs, CTMCs, MDPs + costs/rewards
  - probabilistic timed automata (PTAs) (not covered here)
- Model checking of:
  - PCTL, CSL, LTL, PCTL* + extensions + costs/rewards

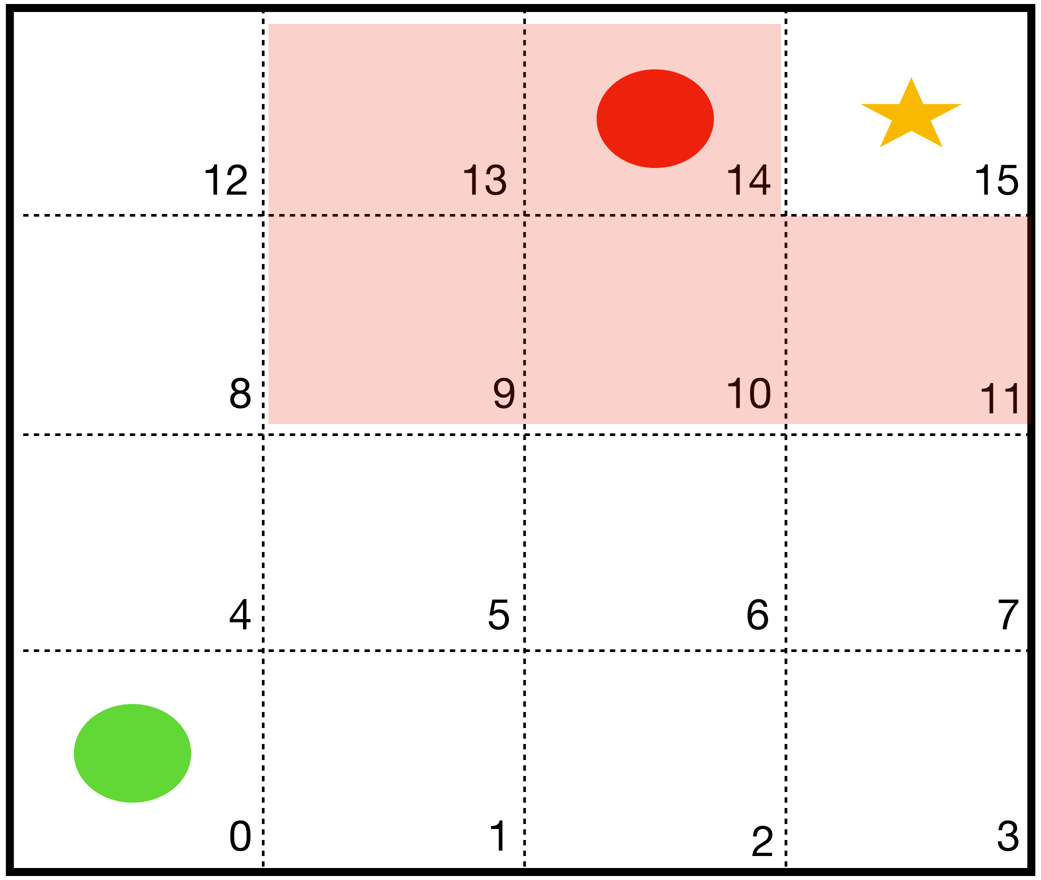
# PRISM modeling language

- Guarded commands
  - describe behaviour of each module
  - i.e. the changes in state that can occur
  - labelled with probabilities (or, for CTMCs, rates)
  - (optional) action labels

$$\underbrace{[\text{send}]}_{\text{action}}\ \underbrace{(s=2)}_{\text{guard}}\ \text{->}\ \underbrace{p_{loss}}_{\text{probability}} : \underbrace{(s'=3)\&(lost'=lost+1)}_{\text{update}} + \underbrace{(1-p_{loss})}_{\text{probability}} : \underbrace{(s'=4)}_{\text{update}};$$

# PRISM Example

## Modeling the obstacle and agent




**The agent must reach the starred location and never crashes into the moving obstacle.**

# PRISM Example

## Modeling the obstacle and agent

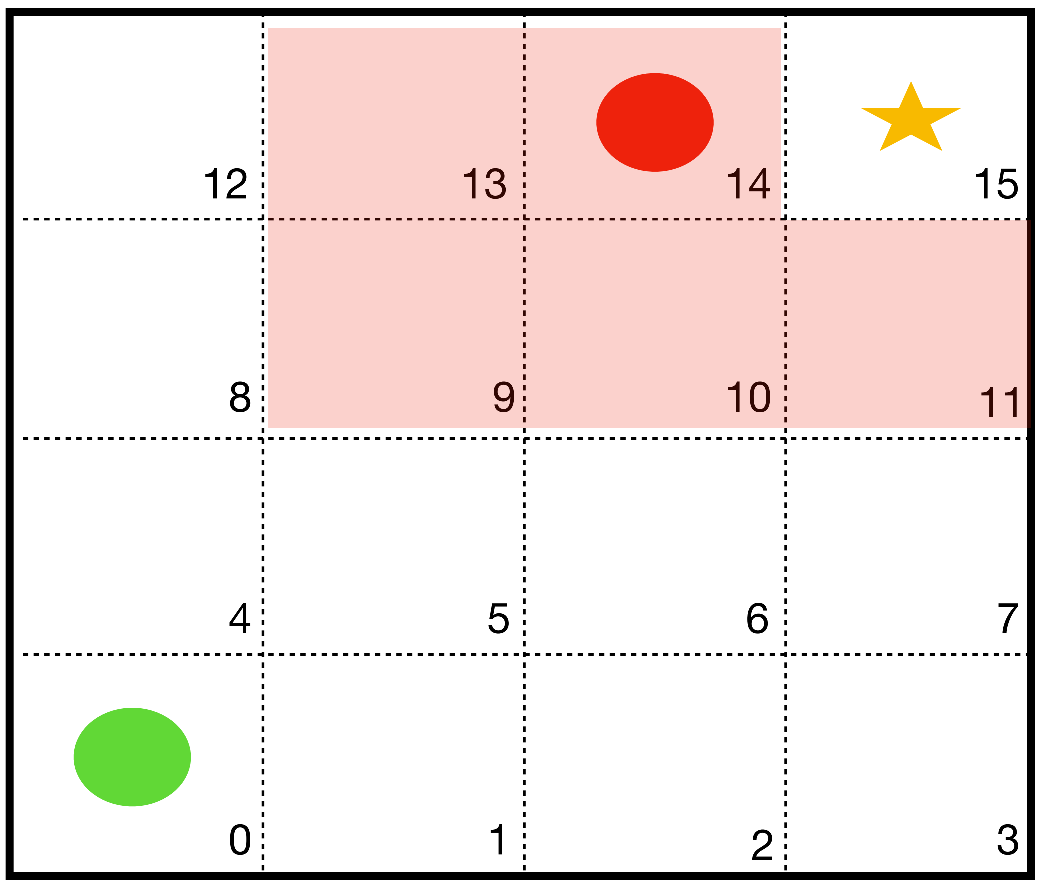
```
dtmc
module obstacle
    o_state : [0..15] init 14;

    [] o_state = 9 ->  1/2 : (o_state' = 9)
                     + 1/2 : (o_state' = 13);
    [] o_state = 13 -> 1/3 : (o_state' = 9)
                     + 1/3 : (o_state' = 13)
                     + 1/3 : (o_state' = 14);
    [] o_state = 10 -> 1/4 : (o_state' = 9)
                     + 1/4 : (o_state' = 10)
                     + 1/4 : (o_state' = 14)
                     + 1/4 : (o_state' = 11);
    [] o_state = 14 -> 1/3 : (o_state' = 13)
                     + 1/3 : (o_state' = 10)
                     + 1/3 : (o_state' = 14);
    [] o_state = 11 -> 1/2 : (o_state' = 10)
                     + 1/2 : (o_state' = 11);
endmodule


module agent
    a_state : [0..15] init 0;

    [] a_state = 0 -> (a_state' = 4);
    [] a_state = 4 -> (a_state' = 8);
    [] a_state = 8 -> (a_state' = 12);
    [] a_state = 12 -> (a_state' = 13);
    [] a_state = 13 -> (a_state' = 14);
    [] a_state = 14 -> (a_state' = 15);
    [] a_state = 15 -> (a_state' = 15);
endmodule

label "crash" = a_state = o_state;
label "goal" = a_state = 15;
```




**The agent must reach the starred location and never crashes into the moving obstacle.**

# PRISM Example

## Modeling the obstacle and agent

```
dtmc
module obstacle
    o_state : [0..15] init 14;

    [] o_state = 9 ->  1/2 : (o_state' = 9)
                     + 1/2 : (o_state' = 13);
    [] o_state = 13 -> 1/3 : (o_state' = 9)
                     + 1/3 : (o_state' = 13)
                     + 1/3 : (o_state' = 14);
    [] o_state = 10 -> 1/4 : (o_state' = 9)
                     + 1/4 : (o_state' = 10)
                     + 1/4 : (o_state' = 14)
                     + 1/4 : (o_state' = 11);
    [] o_state = 14 -> 1/3 : (o_state' = 13)
                     + 1/3 : (o_state' = 10)
                     + 1/3 : (o_state' = 14);
    [] o_state = 11 -> 1/2 : (o_state' = 10)
                     + 1/2 : (o_state' = 11);
endmodule


module agent
    a_state : [0..15] init 0;

    [] a_state = 0 -> (a_state' = 4);
    [] a_state = 4 -> (a_state' = 8);
    [] a_state = 8 -> (a_state' = 12);
    [] a_state = 12 -> (a_state' = 13);
    [] a_state = 13 -> (a_state' = 14);
    [] a_state = 14 -> (a_state' = 15);
    [] a_state = 15 -> (a_state' = 15);
endmodule

label "crash" = a_state = o_state;
label "goal" = a_state = 15;
```

## Model checking

**Property:**
P=? [ (F ("goal"))&(G (!("crash"))) ]

**Defined constants:**
<none>

**Method:**
Verification

**Result (probability):**
0.3510466347137933 (value in the initial state)

Okay

**The probability to eventually reach the goal and never crash is 0.35**

**How can we improve the probability?**

# PRISM Example

## Modeling the obstacle and agent

```
dtmc
module obstacle
    o_state : [0..15] init 14;

    [] o_state = 9 ->  1/2 : (o_state' = 9)
                     + 1/2 : (o_state' = 13);
    [] o_state = 13 -> 1/3 : (o_state' = 9)
                     + 1/3 : (o_state' = 13)
                     + 1/3 : (o_state' = 14);
    [] o_state = 10 -> 1/4 : (o_state' = 9)
                     + 1/4 : (o_state' = 10)
                     + 1/4 : (o_state' = 14)
                     + 1/4 : (o_state' = 11);
    [] o_state = 14 -> 1/3 : (o_state' = 13)
                     + 1/3 : (o_state' = 10)
                     + 1/3 : (o_state' = 14);
    [] o_state = 11 -> 1/2 : (o_state' = 10)
                     + 1/2 : (o_state' = 11);
endmodule


module agent
    a_state : [0..15] init 0;

    [] a_state = 0 -> (a_state' = 4);
    [] a_state = 4 -> (a_state' = 8);
    [] a_state = 8 -> (a_state' = 12);
    [] a_state = 12 & o_state = 13 -> (a_state' = 12);
    [] a_state = 12 & o_state != 13 -> (a_state' = 13);
    [] a_state = 13 -> (a_state' = 14);
    [] a_state = 14 -> (a_state' = 15);
    [] a_state = 15 -> (a_state' = 15);
endmodule

label "crash" = a_state = o_state;
label "goal" = a_state = 15;
```

## Model checking

**Property:**
P=? [ (F ("goal"))&(G (!("crash"))) ]

**Defined constants:**
<none>

**Method:**
Verification

**Result (probability):**
0.4639994833098897 (value in the initial state)

Okay

**The probability to eventually reach the goal and never crash is now 0.46**

**You can keep on improving your controller to get better results**