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/compiler.py
$ python slugs/tools/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 Structruced 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 (...):

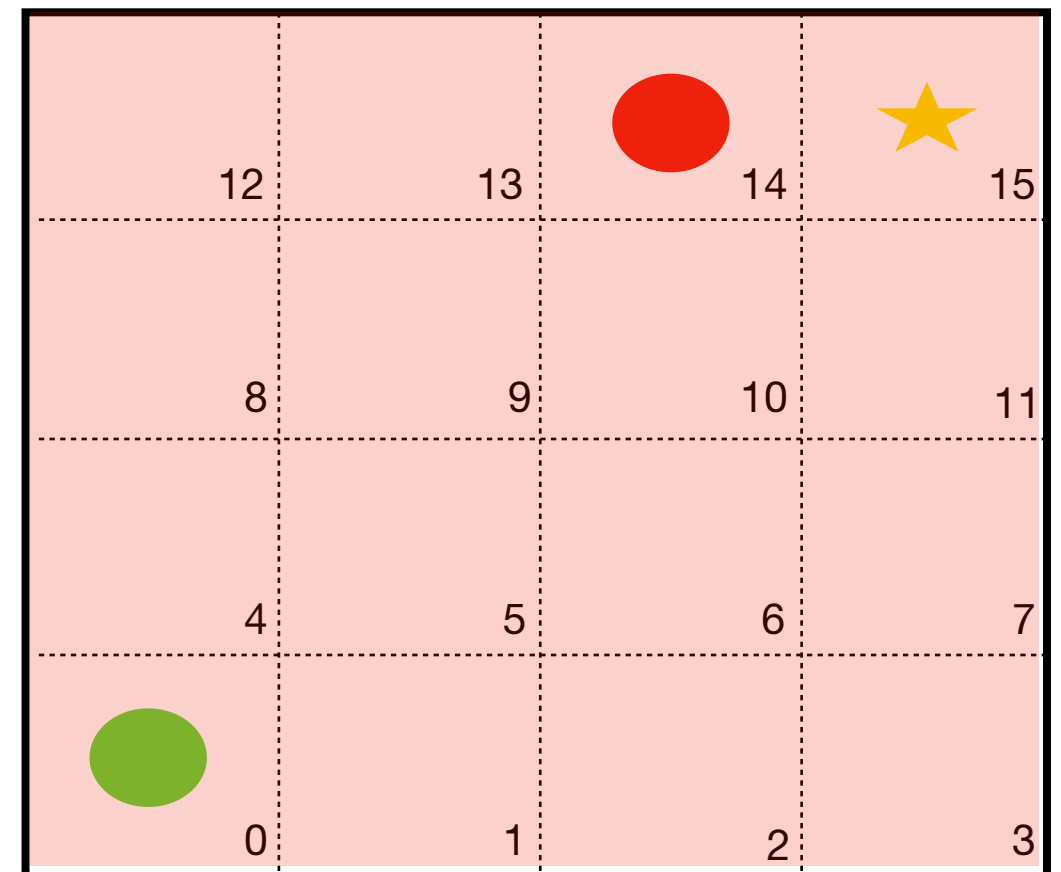[ENV_LIVENESS]
! a | (b = 3)

[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)
```
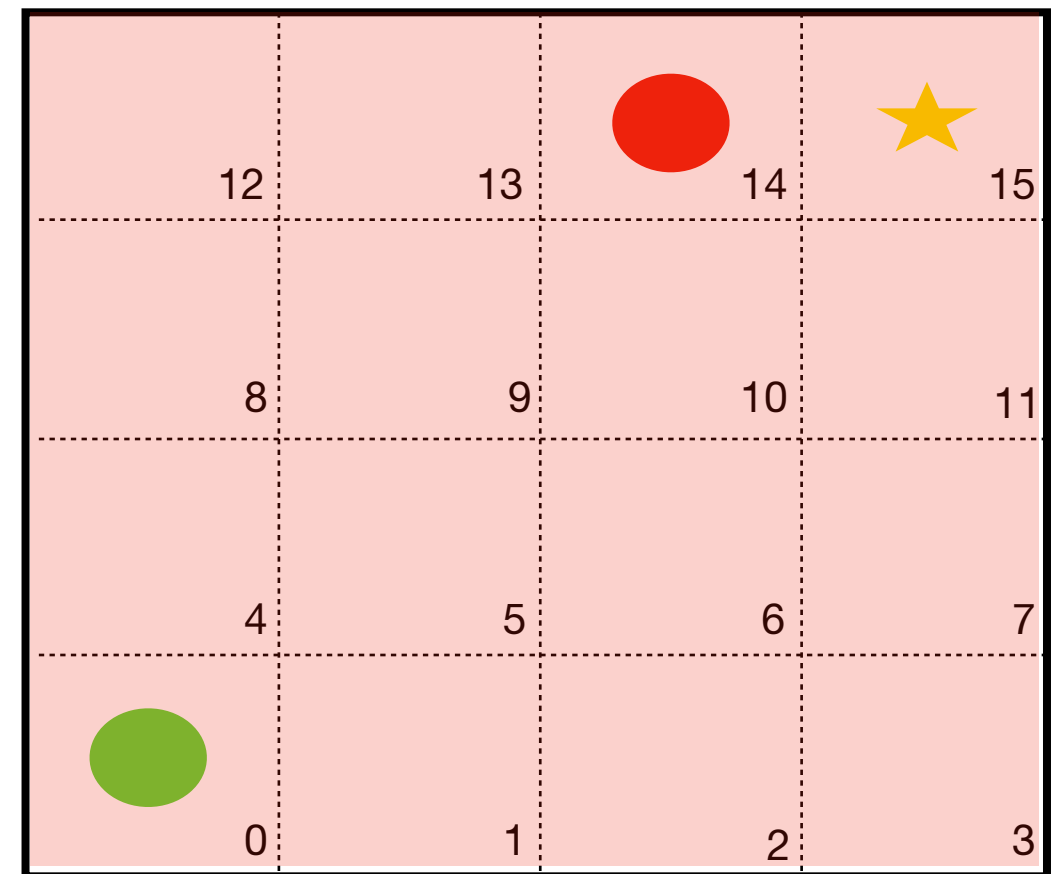
Encode the properties as either safety or liveness LTL specifications

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)
```
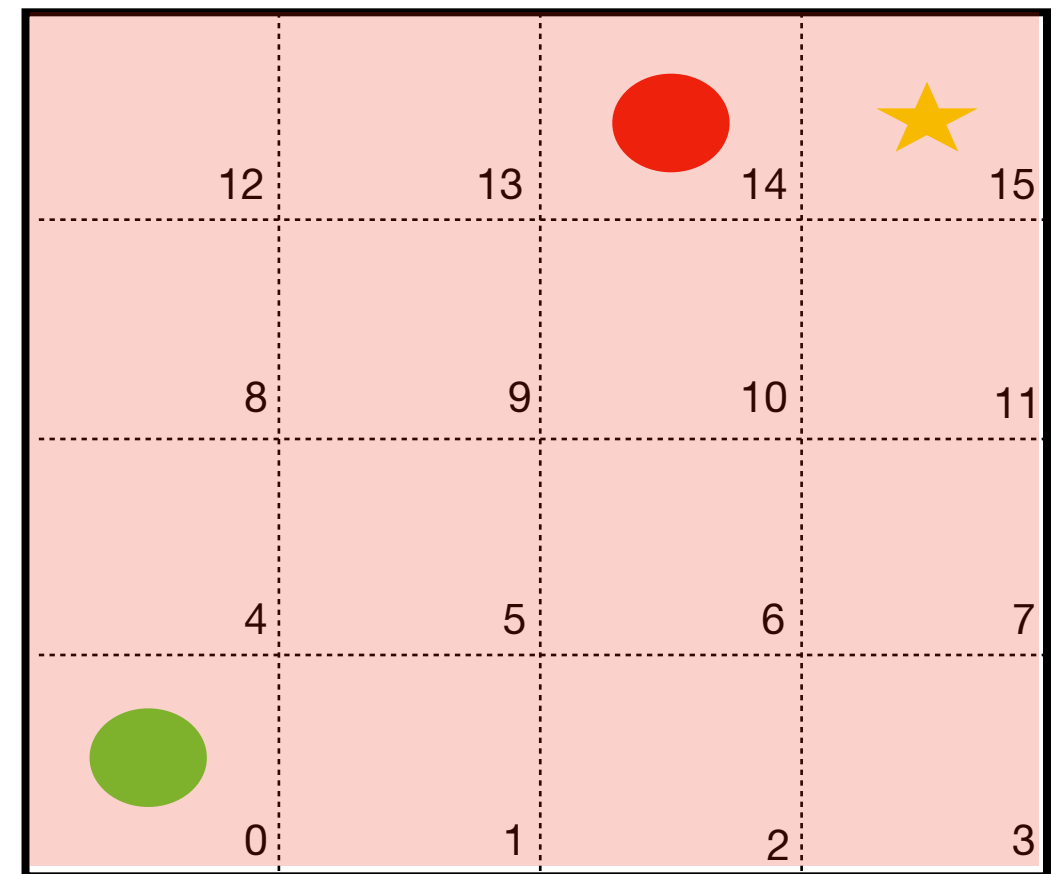
Encode the properties as either safety or liveness LTL specifications

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'
```
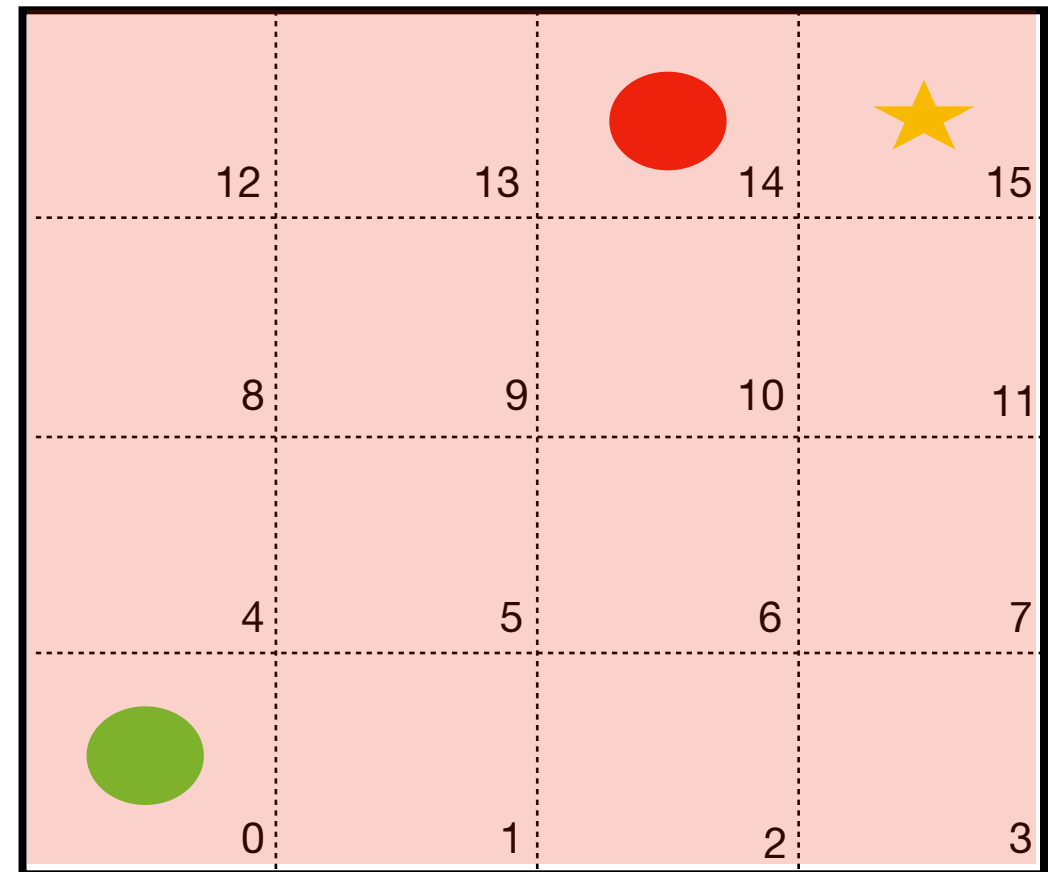
This problem is unrealizable

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
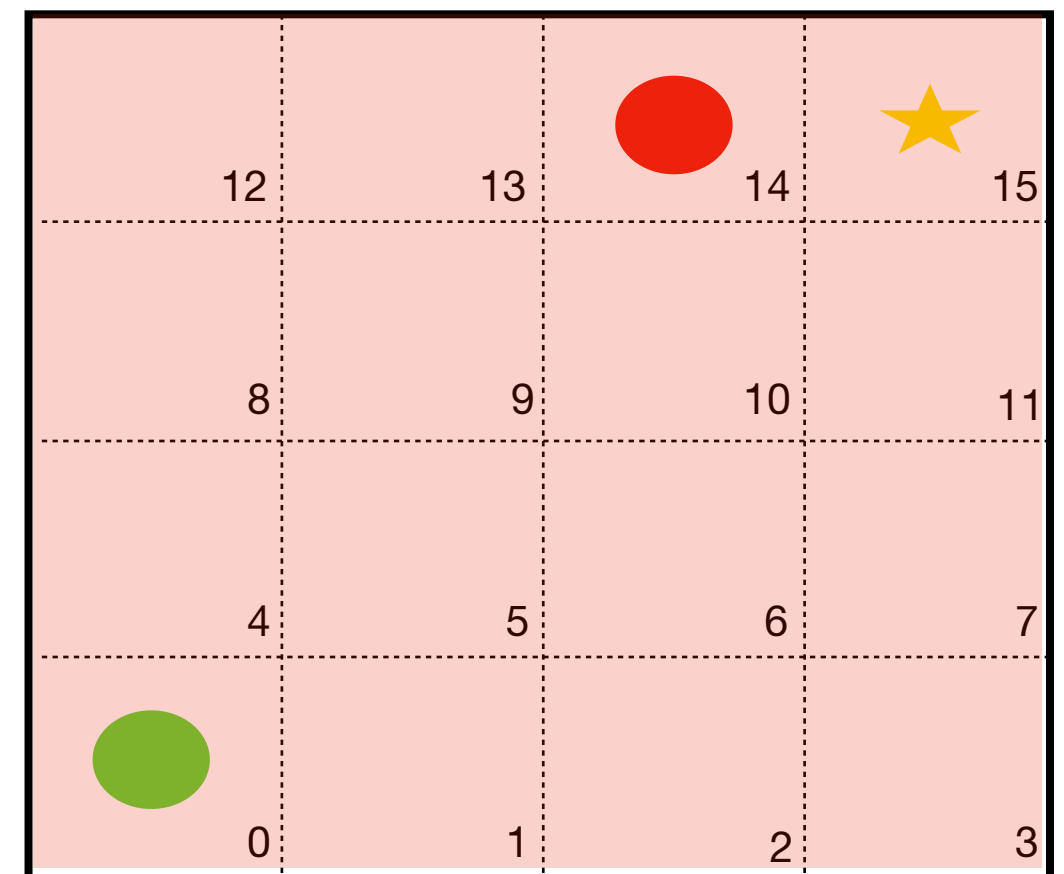
Example:

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

$G (F (o\_state = 15) \wedge F (o\_state = 0))$

=

```
[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

$$[\text{send}]\ (s=2) \rightarrow p_{loss} : (s'=3)\&(lost'=lost+1) + (1-p_{loss}) : (s'=4);$$

| action | guard | probability | update | probability | 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
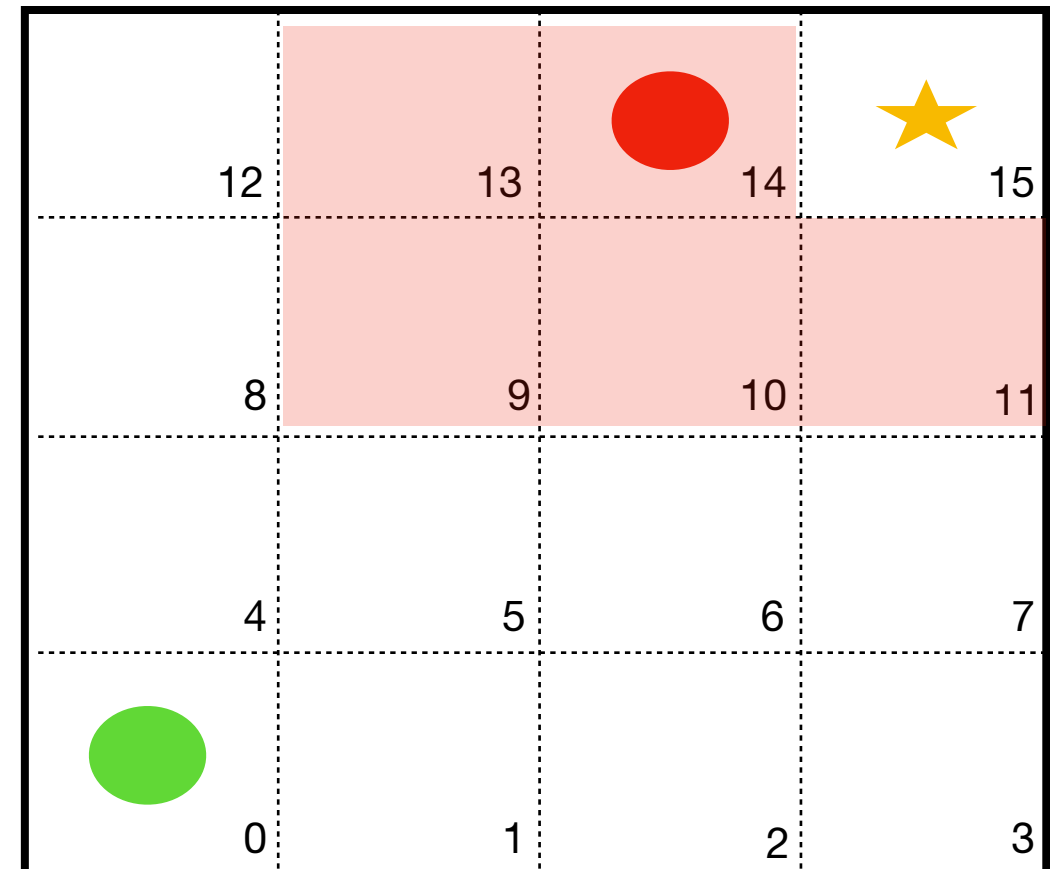
```
 1  dtmc
 2  module obstacle
 3      o_state : [0..15] init 14;
 4
 5      [] o_state = 9 ->  1/2 : (o_state' = 9)
 6                       + 1/2 : (o_state' = 13);
 7      [] o_state = 13 -> 1/3 : (o_state' = 9)
 8                       + 1/3 : (o_state' = 13)
 9                       + 1/3 : (o_state' = 14);
10      [] o_state = 10 -> 1/4 : (o_state' = 9)
11                       + 1/4 : (o_state' = 10)
12                       + 1/4 : (o_state' = 14)
13                       + 1/4 : (o_state' = 11);
14      [] o_state = 14 -> 1/3 : (o_state' = 13)
15                       + 1/3 : (o_state' = 10)
16                       + 1/3 : (o_state' = 14);
17      [] o_state = 11 -> 1/2 : (o_state' = 10)
18                       + 1/2 : (o_state' = 11);
19  endmodule
20
21
22  module agent
23      a_state : [0..15] init 0;
24
25      [] a_state = 0 -> (a_state' = 4);
26      [] a_state = 4 -> (a_state' = 8);
27      [] a_state = 8 -> (a_state' = 12);
28      [] a_state = 12 -> (a_state' = 13);
29      [] a_state = 13 -> (a_state' = 14);
30      [] a_state = 14 -> (a_state' = 15);
31      [] a_state = 15 -> (a_state' = 15);
32  endmodule
33
34  label "crash" = a_state = o_state;
35  label "goal" = a_state = 15;
36
```
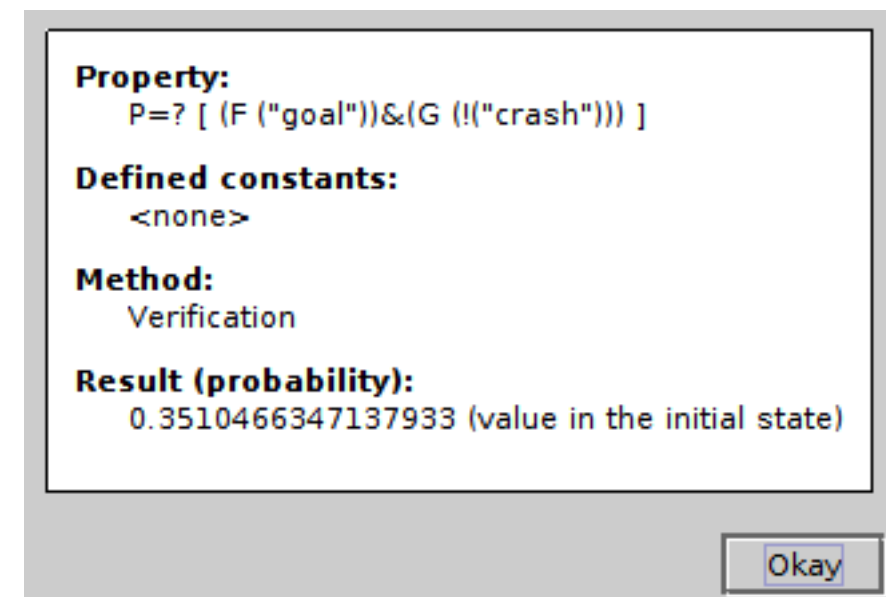

The **agent** must reach the starred location and never crashes into the **moving obstacle**.

PRISM Example

Modeling the obstacle and agent

```
 1  dtmc
 2  module obstacle
 3      o_state : [0..15] init 14;
 4
 5      [] o_state = 9 ->  1/2 : (o_state' = 9)
 6                       + 1/2 : (o_state' = 13);
 7      [] o_state = 13 -> 1/3 : (o_state' = 9)
 8                       + 1/3 : (o_state' = 13)
 9                       + 1/3 : (o_state' = 14);
10      [] o_state = 10 -> 1/4 : (o_state' = 9)
11                       + 1/4 : (o_state' = 10)
12                       + 1/4 : (o_state' = 14)
13                       + 1/4 : (o_state' = 11);
14      [] o_state = 14 -> 1/3 : (o_state' = 13)
15                       + 1/3 : (o_state' = 10)
16                       + 1/3 : (o_state' = 14);
17      [] o_state = 11 -> 1/2 : (o_state' = 10)
18                       + 1/2 : (o_state' = 11);
19  endmodule
20
21
22  module agent
23      a_state : [0..15] init 0;
24
25      [] a_state = 0 -> (a_state' = 4);
26      [] a_state = 4 -> (a_state' = 8);
27      [] a_state = 8 -> (a_state' = 12);
28      [] a_state = 12 -> (a_state' = 13);
29      [] a_state = 13 -> (a_state' = 14);
30      [] a_state = 14 -> (a_state' = 15);
31      [] a_state = 15 -> (a_state' = 15);
32  endmodule
33
34  label "crash" = a_state = o_state;
35  label "goal" = a_state = 15;
36
```

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
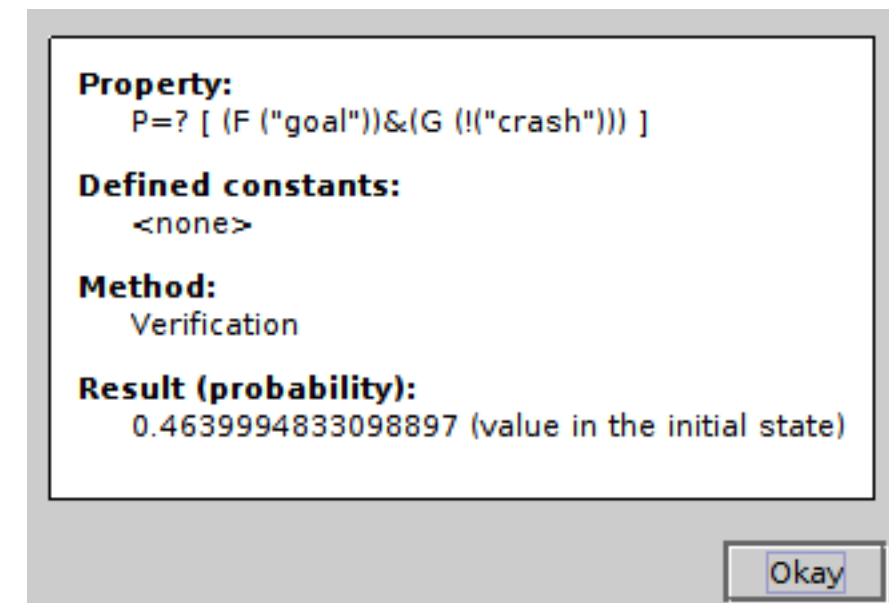
How can we improve the probability?

PRISM Example

Modeling the obstacle and agent

```
1  dtmc
2  module obstacle
3      o_state : [0..15] init 14;
4
5      [] o_state = 9 ->  1/2 : (o_state' = 9)
6                       + 1/2 : (o_state' = 13);
7      [] o_state = 13 -> 1/3 : (o_state' = 9)
8                       + 1/3 : (o_state' = 13)
9                       + 1/3 : (o_state' = 14);
10     [] o_state = 10 -> 1/4 : (o_state' = 9)
11                      + 1/4 : (o_state' = 10)
12                      + 1/4 : (o_state' = 14)
13                      + 1/4 : (o_state' = 11);
14     [] o_state = 14 -> 1/3 : (o_state' = 13)
15                      + 1/3 : (o_state' = 10)
16                      + 1/3 : (o_state' = 14);
17     [] o_state = 11 -> 1/2 : (o_state' = 10)
18                      + 1/2 : (o_state' = 11);
19 endmodule
20
21
22 module agent
23     a_state : [0..15] init 0;
24
25     [] a_state = 0 -> (a_state' = 4);
26     [] a_state = 4 -> (a_state' = 8);
27     [] a_state = 8 -> (a_state' = 12);
28     [] a_state = 12 & o_state = 13 -> (a_state' = 12);
29     [] a_state = 12 & o_state != 13 -> (a_state' = 13);
30     [] a_state = 13 -> (a_state' = 14);
31     [] a_state = 14 -> (a_state' = 15);
32     [] a_state = 15 -> (a_state' = 15);
33 endmodule
34
35 label "crash" = a_state = o_state;
36 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