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

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

PRISM Example

Modeling the obstacle and agent

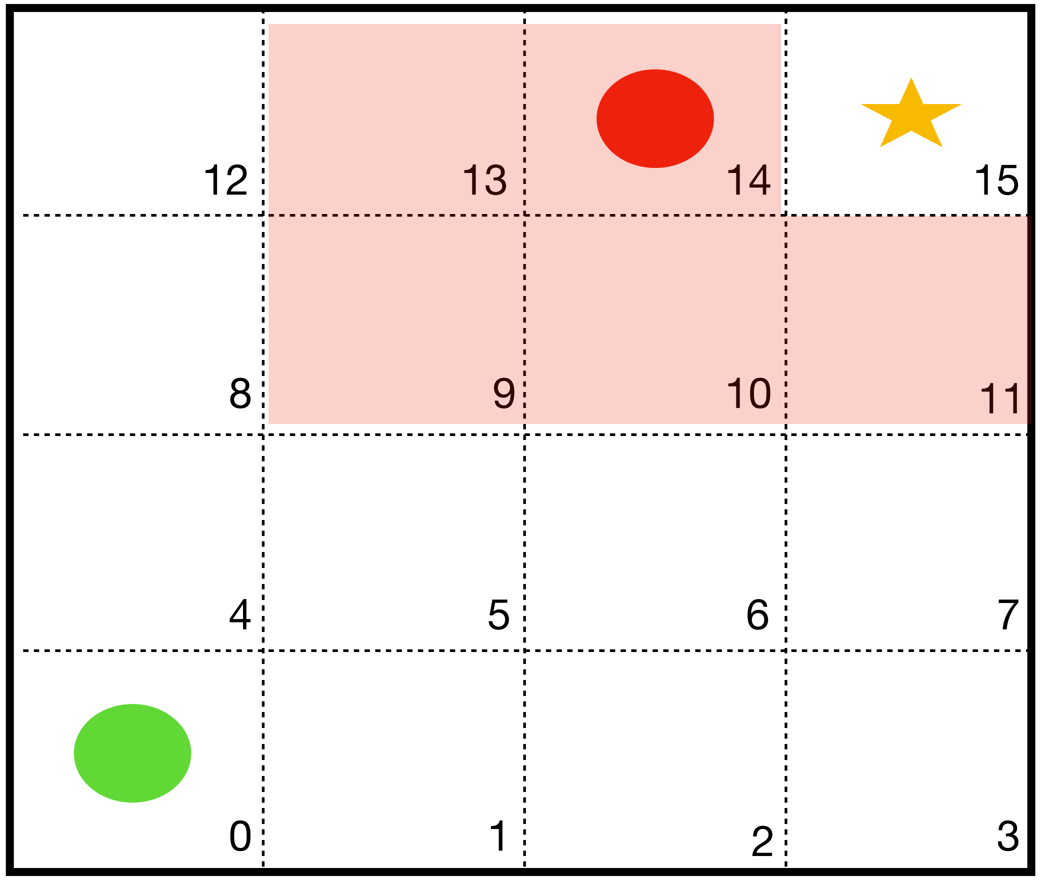
```
dtmc
module obstacle
    o_state : [0..15] init 14;

    [] 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

...or you could synthesize a controller from specifications. That brings us to SLUGS!

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)
```

Encode the properties as either safety or liveness LTL specifications

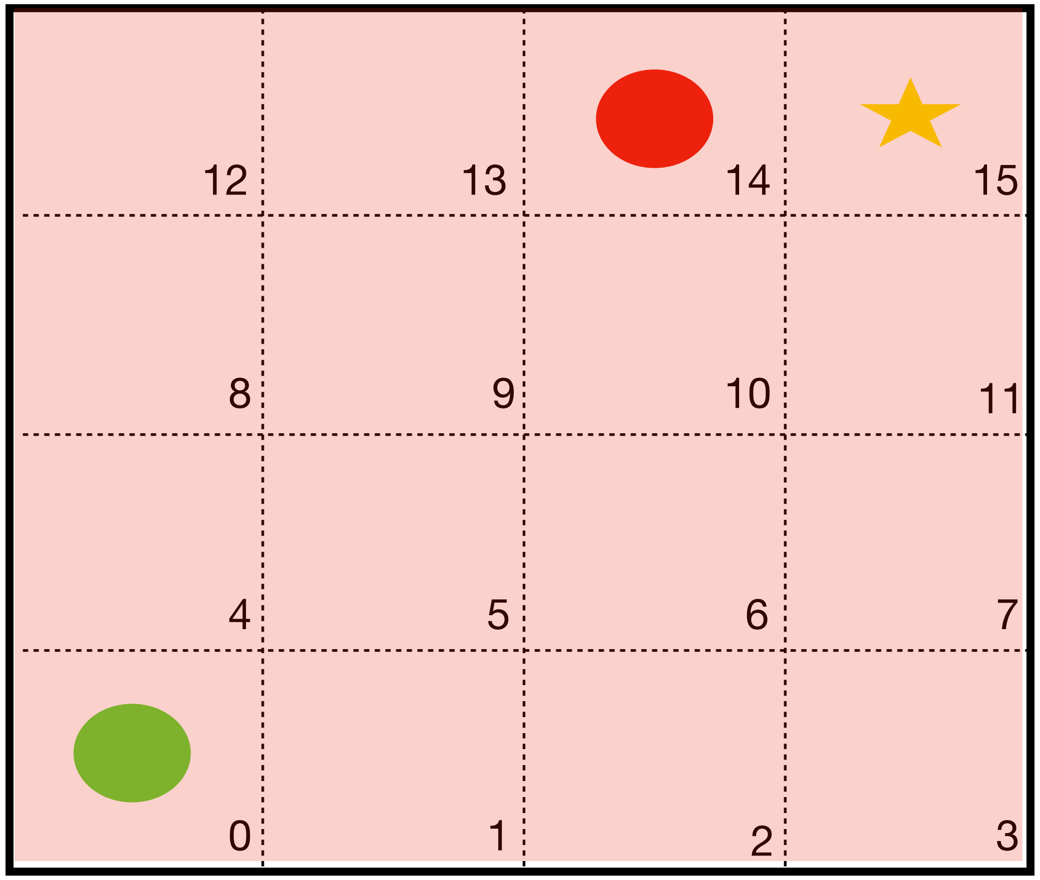
```
sys_liveness
    a_state == 15

sys_liveness
    a_state == 0
```

the agent must visit locations 15 and 0 infinitely often

```
sys_trans
    o_state != a_state
    o_state' != a_state
    o_state != a_state'
```

the agent and obstacle must never crash



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

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