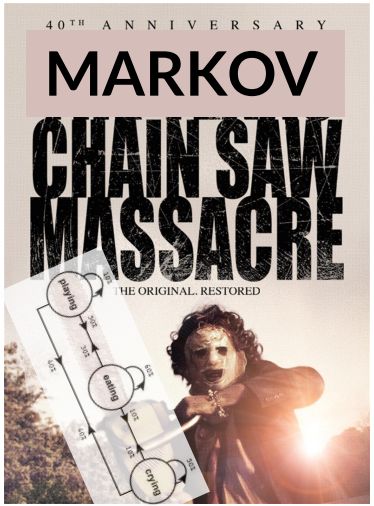


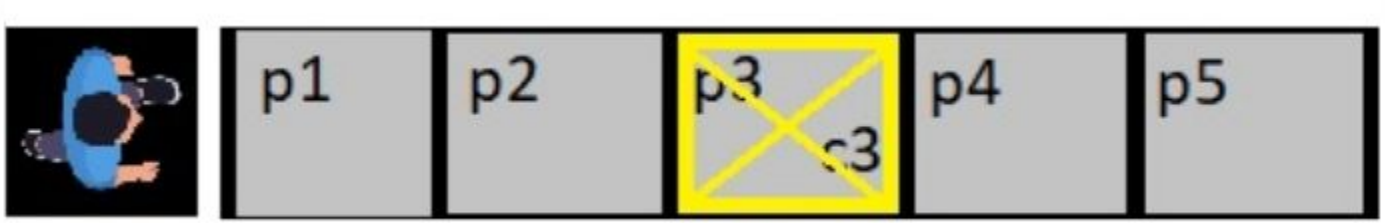
Markov Chainsaw Massacre

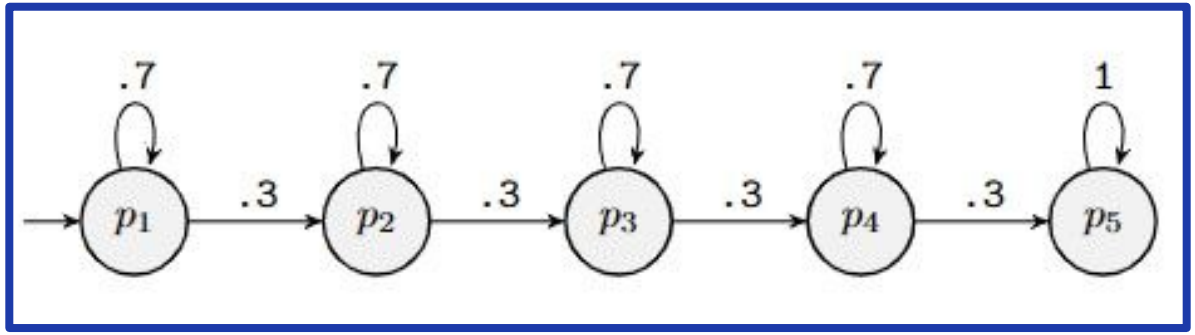
Probabilistic Model Checking

Presented by Charles Dudley, Ranai Srivastav, Varad Kulkarni

Recap: Theory

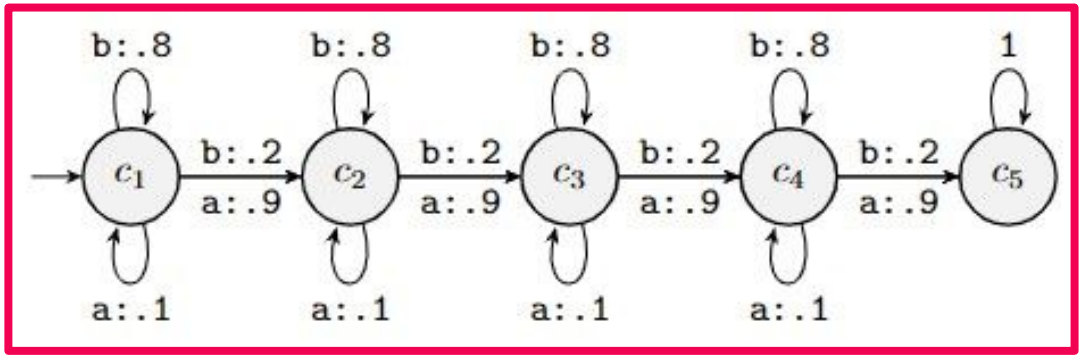
Markov Chains

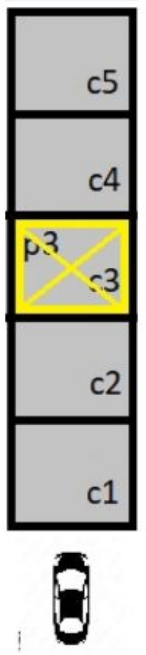




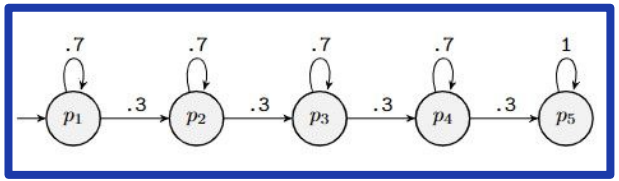
Markov Decision Processes

- **Nondeterministic Agent**
- Selects **action** at each state
- **Policy / Strategy** defines behavior

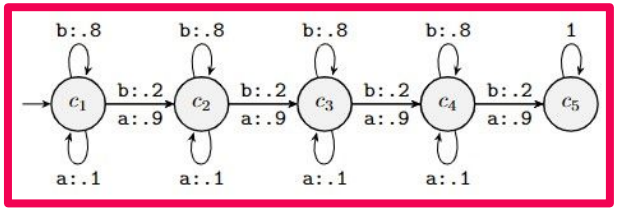




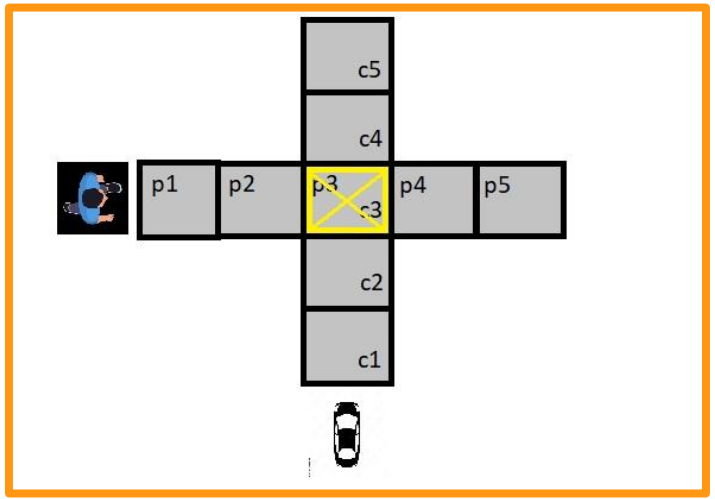
Parallel Composition of MC and MDP



+



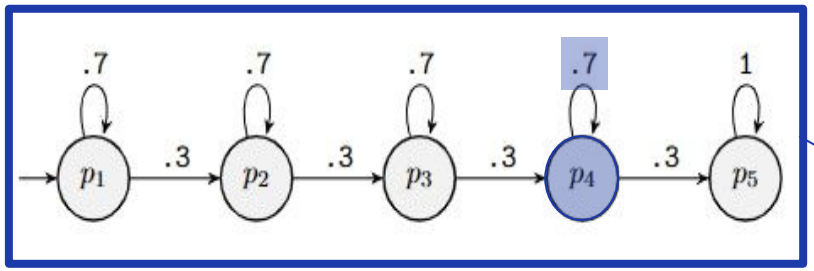
=



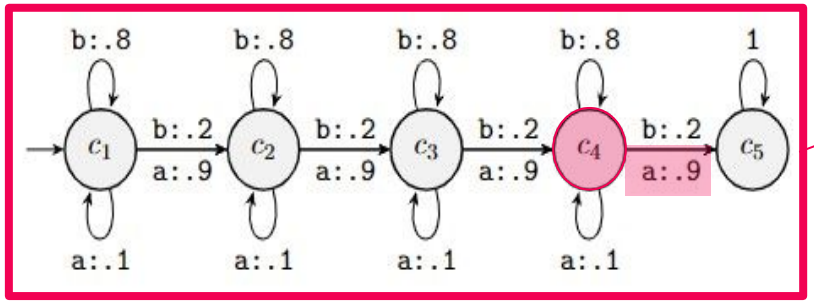
Parallel Composition of MC and MDP

- Capture joint behavior of combined systems
- Assume an MDP M_1 and MC M_2 and a parallel composition $M = M_1 \,||\, M_2$
  - States of M are a cross product of M_1 and M_2
  - The transition probabilities are multiplied
  - The result is another MDP
- Still **non-deterministic**!

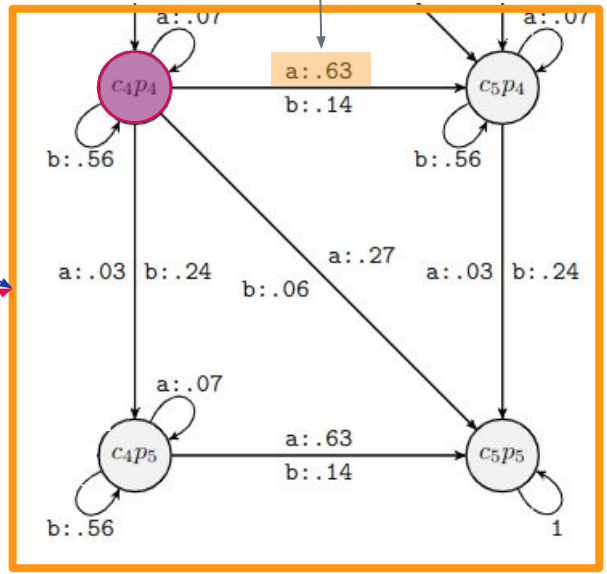
Parallel Composition of MC and MDP



+

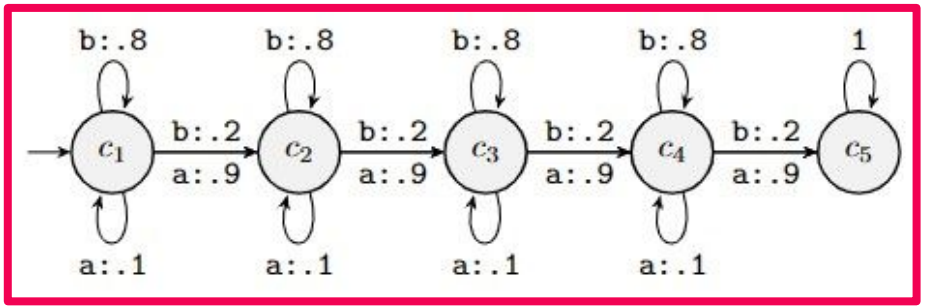


P(car accelerates and goes to next state) * P(ped stays)
0.9 * 0.7 = 0.63



Policy/Strategy

- What action should the agent take in a given state?
  - Should consider environment
- Resolves non-determinism
- Meaningless in isolation
- Powerful in parallel composition



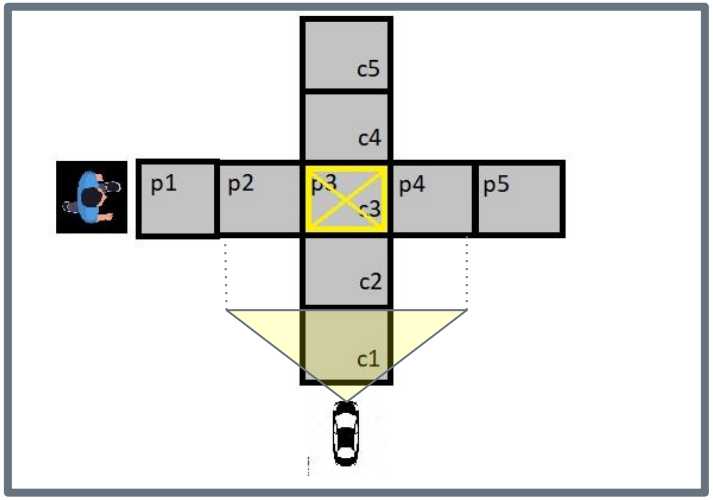
Car Pedestrian Scenario

Applying a Policy

- The car must:
  - Reach the end state, and
  - Not collide with a pedestrian
- Spec:
  - G (F c5 & !(*c3p3*))
- More formally:

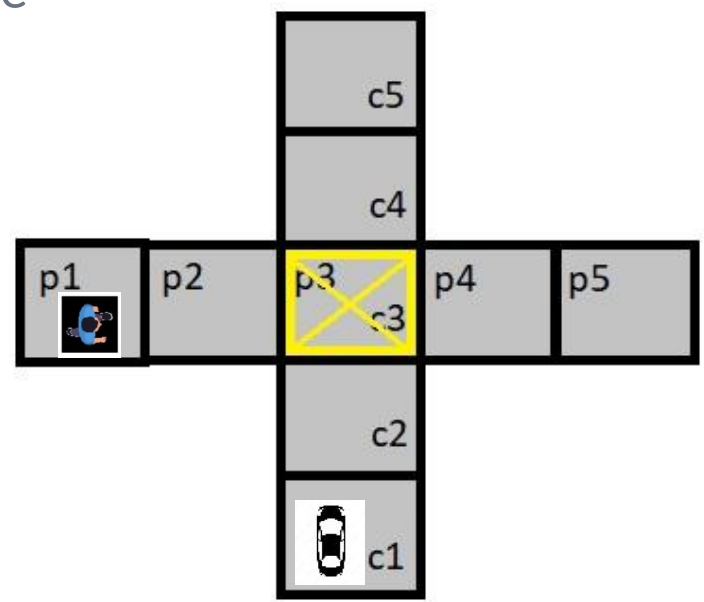
Brake	*if c1p3, c2p3, c2p2*
Accelerate	*otherwise*

- This is NOT an optimal policy



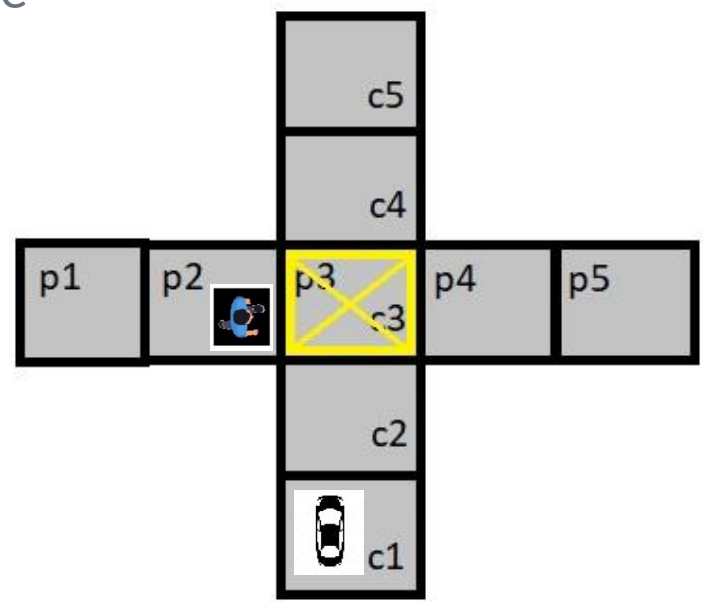
Initial System - Policy

Action: Accelerate



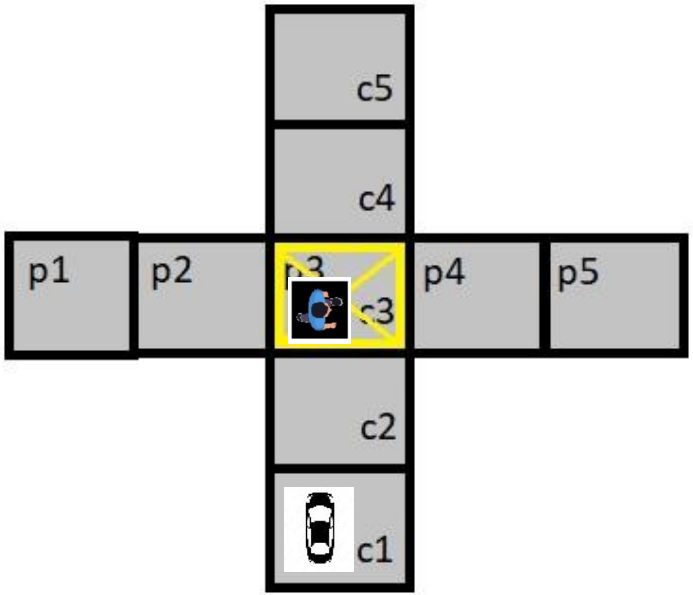
Applying the Policy

Action: Accelerate



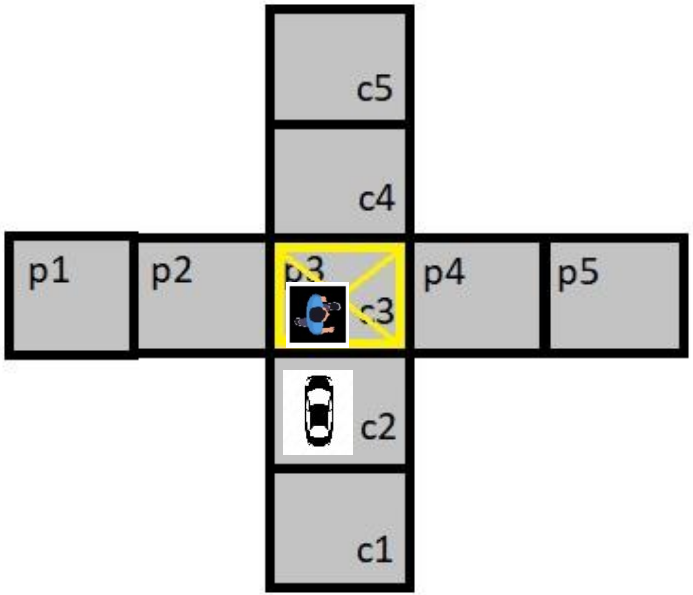
Applying the Policy

Action: Brake



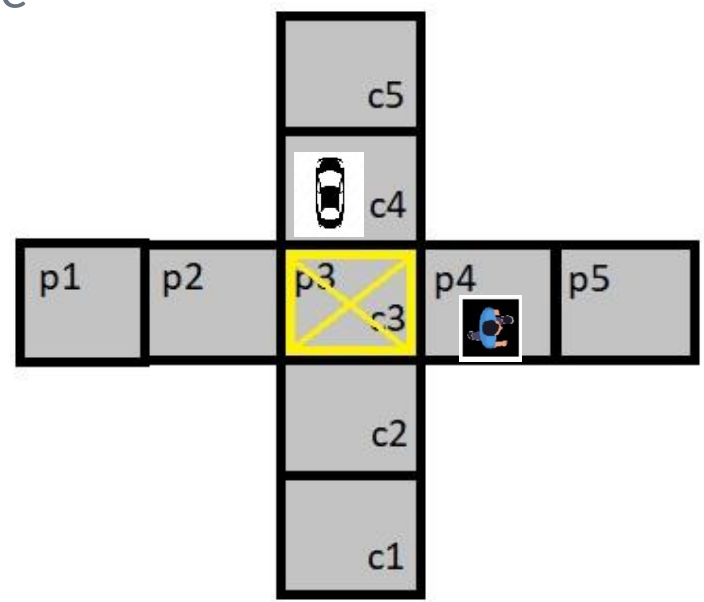
Applying the Policy

Action: Brake



Applying the Policy

Action: Accelerate

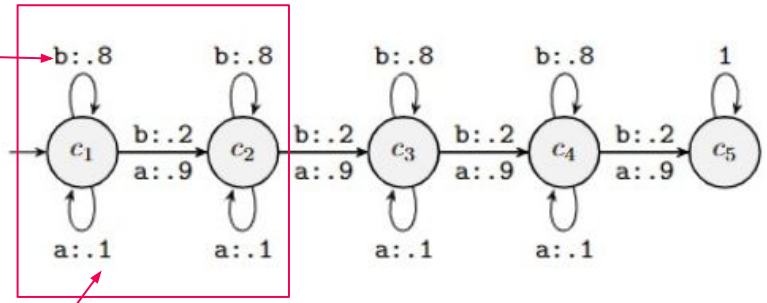


Prism: Modeling Language

- A modeling language for probabilistic model checking
  - Similar syntax to Promela

```
mdp

module M

s:[1..5];

    [brake]      s=1 -> 0.8:(s'=1) + 0.2:(s'=2)
    [accelerate] s=1 -> 0.1:(s'=1) + 0.9:(s'=2)

    [brake]      s=2 -> 0.8:(s'=2) + 0.2:(s'=2)
    [accelerate] s=2 -> 0.1:(s'=2) + 0.9:(s'=2)
```

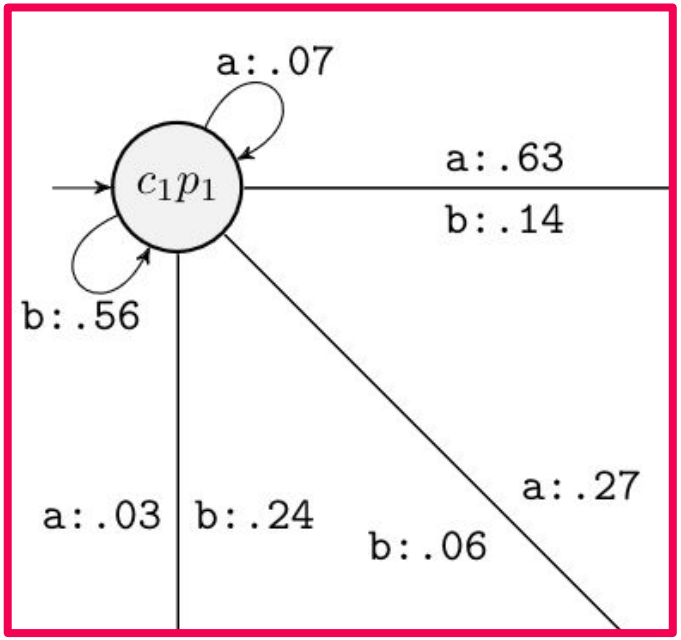
Action label



Transition probabilities and state transitions

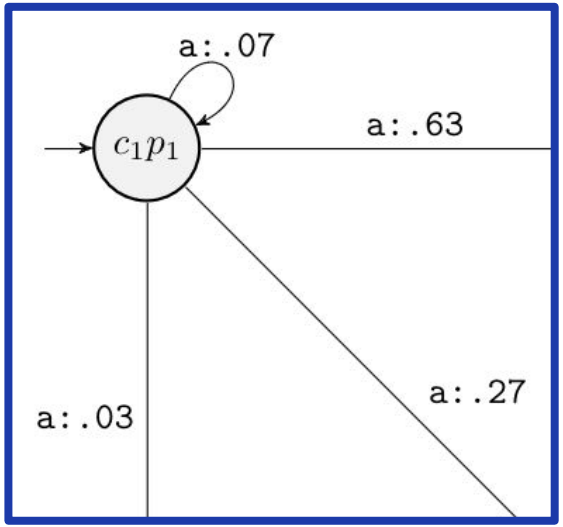
Implementation - Before Policy

```
dtmc

module car5_ped5_parallelcomposition

    c : [1..5] init 1;
    p : [1..5] init 1;
       [accelerate] c=1 & p=1 -> 0.07:(c'=1) & (p'=1) +
                                 0.63:(c'=2) & (p'=1) +
                                 0.03:(c'=1) & (p'=2) +
                                 0.27:(c'=2) & (p'=2);

       [brake]      c=1 & p=1 -> 0.56:(c'=1) & (p'=1) +
                                 0.14:(c'=2) & (p'=1) +
                                 0.24:(c'=1) & (p'=2) +
                                 0.06:(c'=2) & (p'=2);
```



Implementation - After Policy

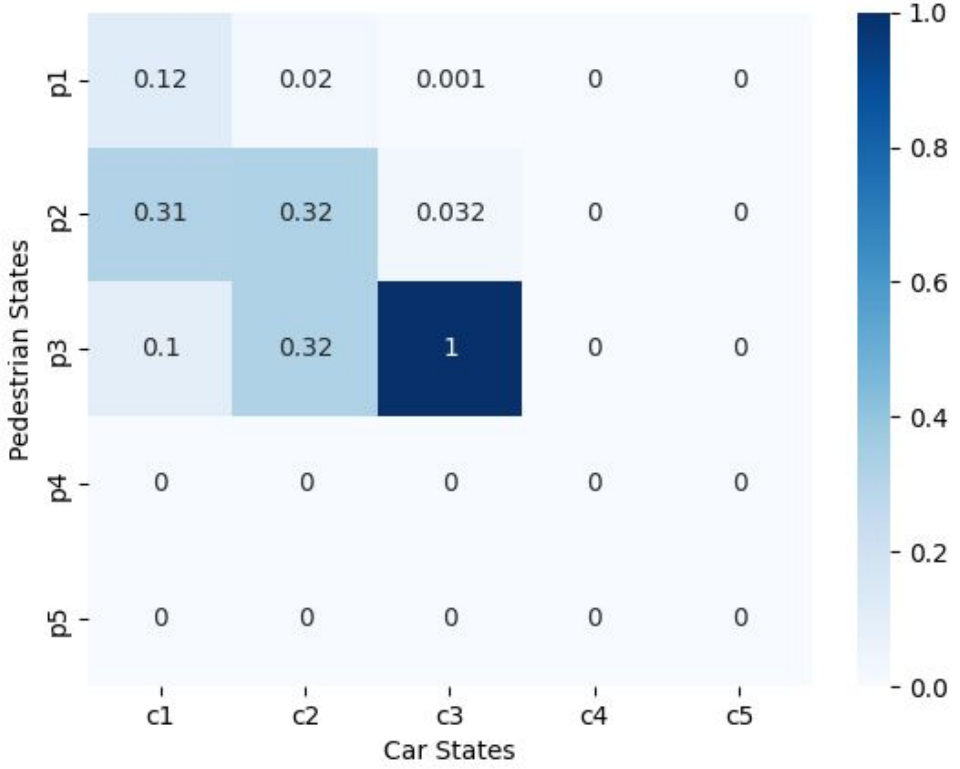
```
dtmc

module car5_ped5_parallelcomposition_policy

    c : [1..5] init 1;
    p : [1..5] init 1;
       [] c=1 & p=1 -> 0.07:(c'=1) & (p'=1) +
                       0.63:(c'=2) & (p'=1) +
                       0.03:(c'=1) & (p'=2) +
                       0.27:(c'=2) & (p'=2);
```



Analysis of Policy - Collisions

PRISM specification:

P=? [F (c=3 & p=3)]



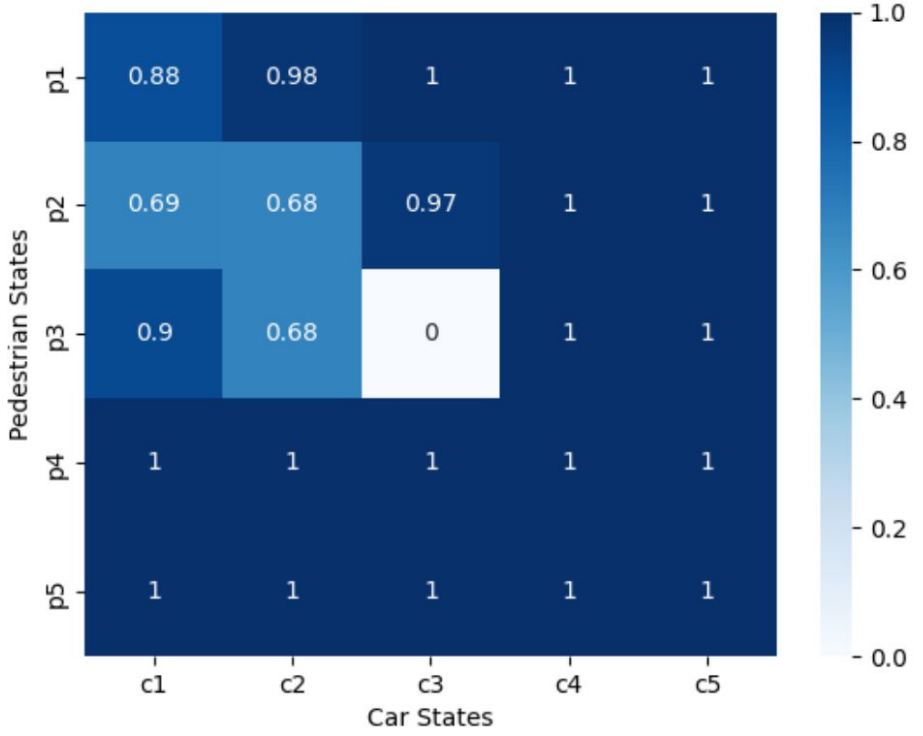
Analysis of Policy - Safety

PRISM specification:

P =?

[(G !(c = 3 & p = 3)) &

(F (c=5 & p = 5))]



Limitations of StormPy

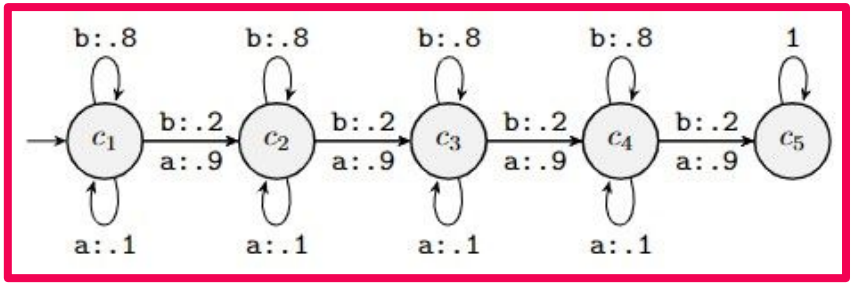
- Little documentation
  - How to automatically perform parallel composition?
    - Using Python script to create explicit definition
  - How to generate optimal policy?
  - How to efficiently apply policy to system?

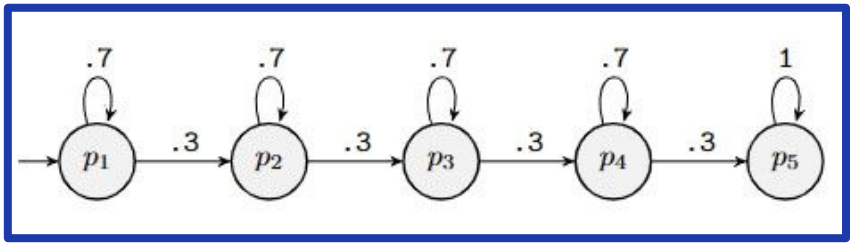
PRISM - The Model Checker

- Extensive documentation
- Painless (and automatic) parallel composition
- Efficient policy generation and analysis
- GUI
  - Helpful for debugging model and specs

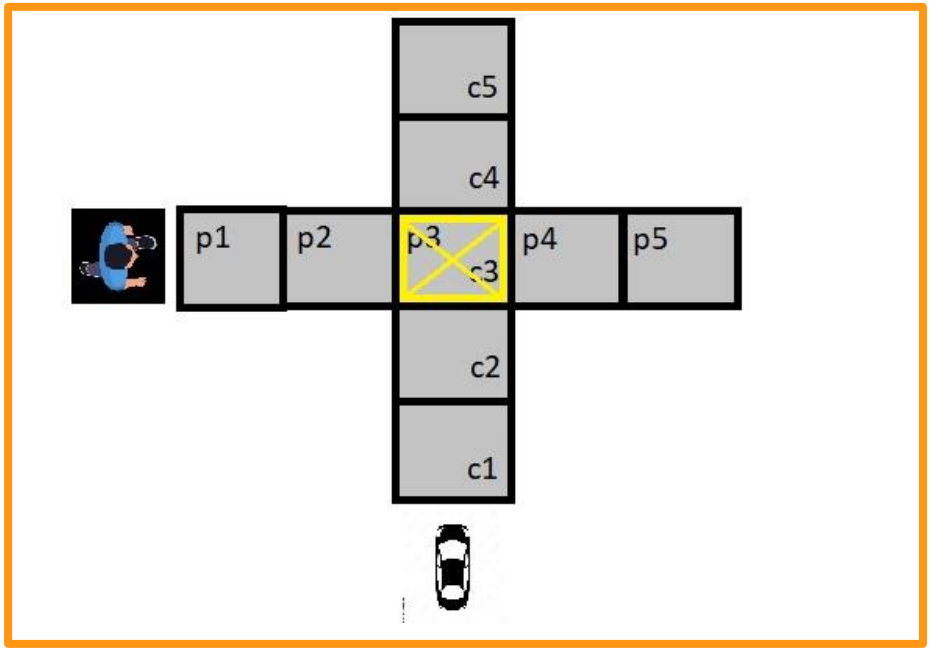
Implementation

```
mdp

module the_mdp

    c : [1..5] init 1;

    [accelerate] c < 5 -> 0.9:(c'=c+1) + 0.1 : (c'=c);
    [brake]      c < 5 -> 0.8:(c'=c) + 0.2 : (c'=c+1);
    [accelerate] c = 5 -> 1:(c'=c);
    [brake]      c = 5 -> 1:(c'=c);

endmodule

module the_mc

    p : [1..5] init 1;

    [accelerate] p < 5 -> 0.7:(p'=p) + 0.3 : (p'=p+1);
    [accelerate] p = 5 -> (p'=p);

    [brake] p < 5 -> 0.7:(p'=p) + 0.3 : (p'=p+1);
    [brake] p = 5 -> (p'=p);

endmodule
```





Policy Generation and Analysis - Optimal



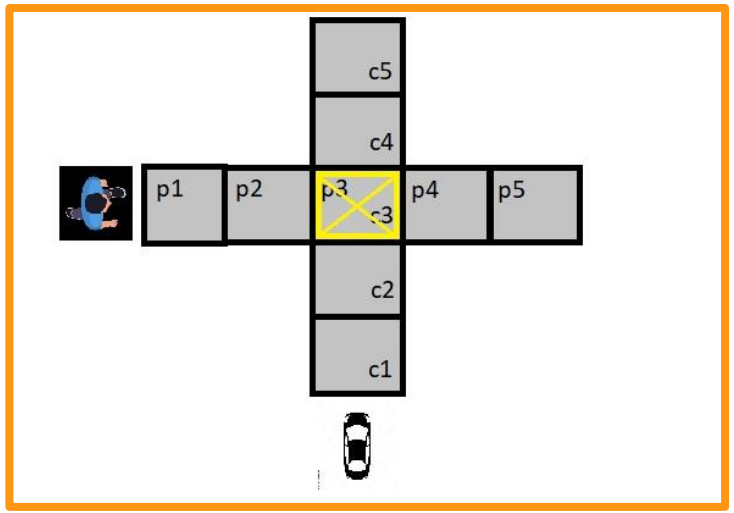
What is the maximum probability of avoiding collision?

Pmax=? [G !(c=3&p=3)]

Policy Generation and Analysis - Optimal

Pmax=? [G !(c=3&p=3)]

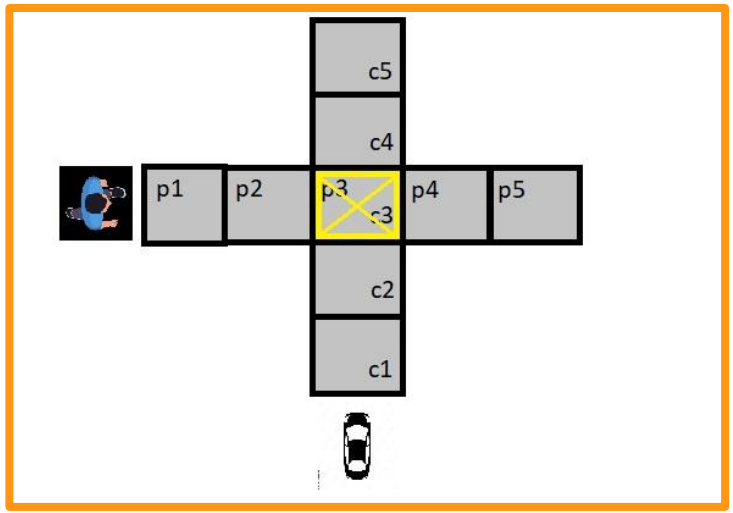
Pmax= 0.886758485



Policy Generation and Analysis - Optimal

Pmax=? [G !(c=3&p=3)]

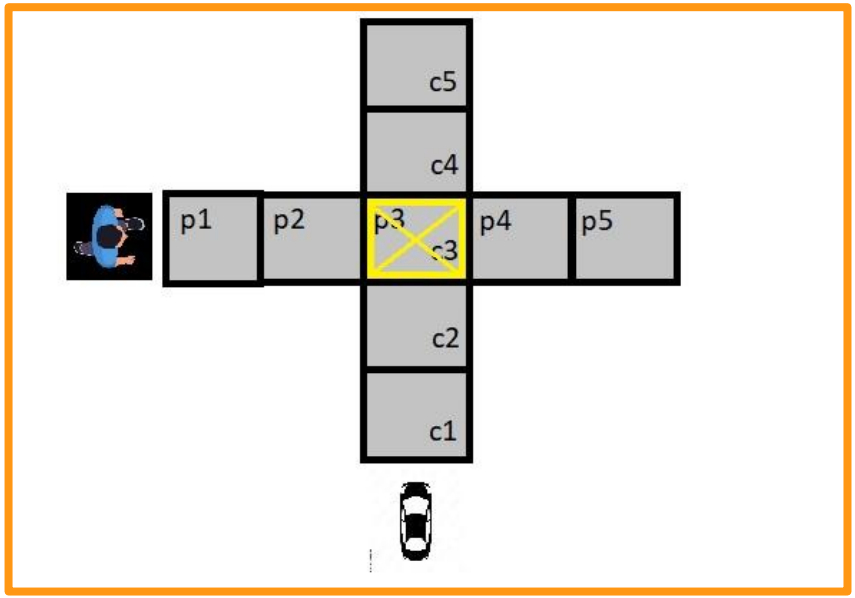
Pmax= 0.886758485



Policy: (a,b):action

(1,1):accelerate
(1,2):brake
(1,3):brake
(1,4):brake
(1,5):brake
(2,1):accelerate
(2,2):brake
(2,3):brake
(2,4):brake
(2,5):brake
(3,1):accelerate
(3,2):accelerate
(3,4):brake
(3,5):brake
(4,1):brake
(4,2):brake
(4,3):brake
(4,4):brake
(4,5):brake
(5,1):brake
(5,2):brake
(5,3):brake
(5,4):brake
(5,5):brake

Policy Generation and Analysis - Optimal



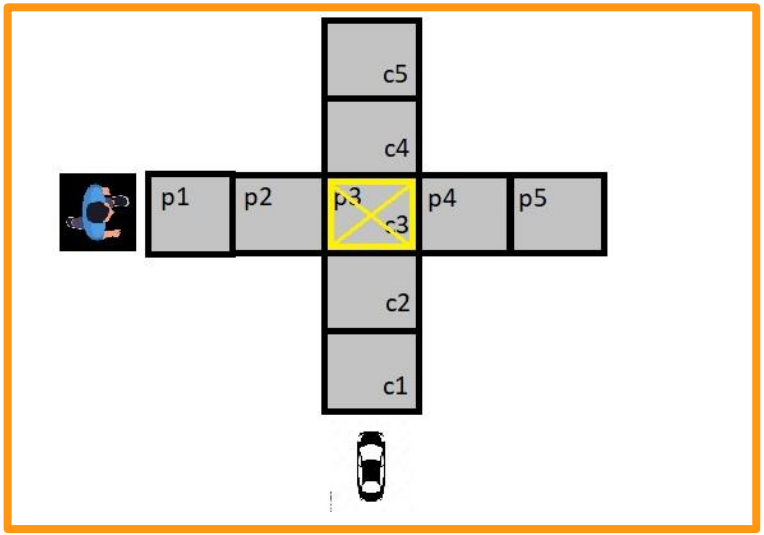
What is the maximum probability of collision AND agents reach their goal?

Pmax=? [G (!(c=3&p=3)&
(F (c=5&p=5)))]

Policy Generation and Analysis - Optimal

Pmax=? [G (!(c=3&p=3)& (F (c=5&p=5)))]

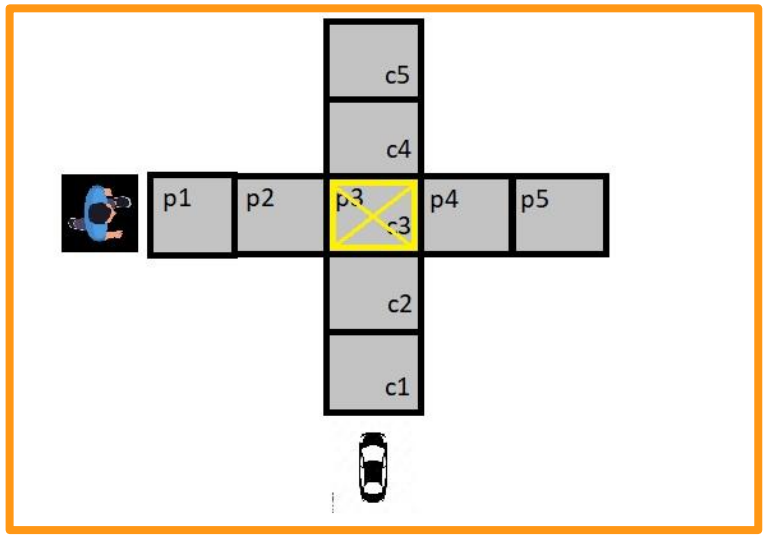
Pmax= 0.8867583356



Policy Generation and Analysis - Optimal

Pmax=? [G (!(c=3&p=3)& (F (c=5&p=5)))]

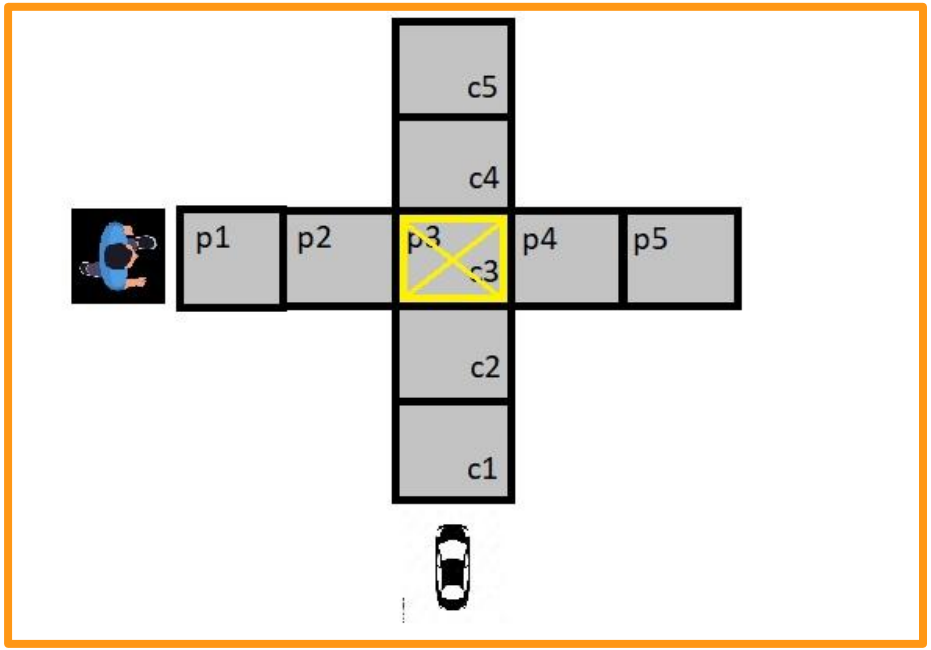
Pmax= 0.8867583356



Policy: (a,b):action

(1,1):accelerate	(4,2):accelerate
(1,2):brake	(4,3):accelerate
(2,1):accelerate	(1,5):accelerate
(2,2):brake	(2,5):accelerate
(1,3):brake	(3,5):accelerate
(2,3):brake	(4,4):accelerate
(3,1):accelerate	(4,5):accelerate
(3,2):accelerate	(5,1):accelerate
(1,4):accelerate	(5,2):accelerate
(2,4):accelerate	(5,3):accelerate
(3,4):accelerate	(5,4):accelerate
(4,1):accelerate	(5,5):accelerate

Policy Generation and Analysis - Attacker

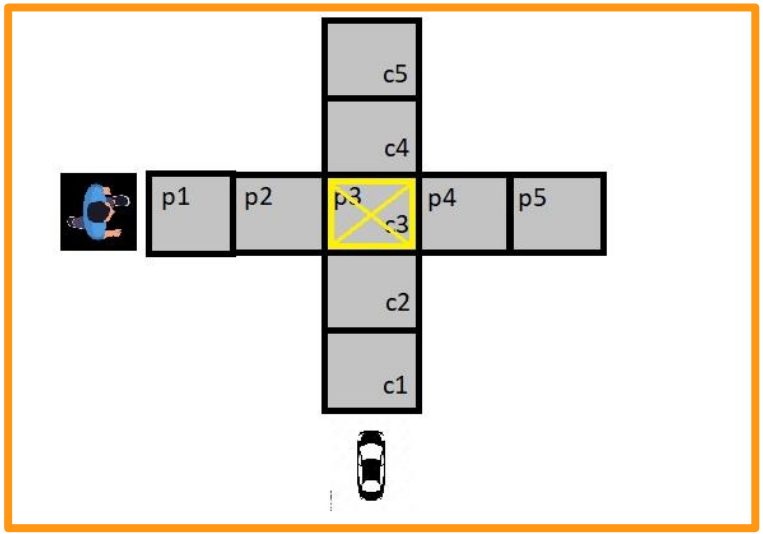


What is the maximum probability of collision?

Pmax=? [F (c=3&p=3)]

Policy Generation and Analysis - Attacker

Pmax=? [F (c=3&p=3)]



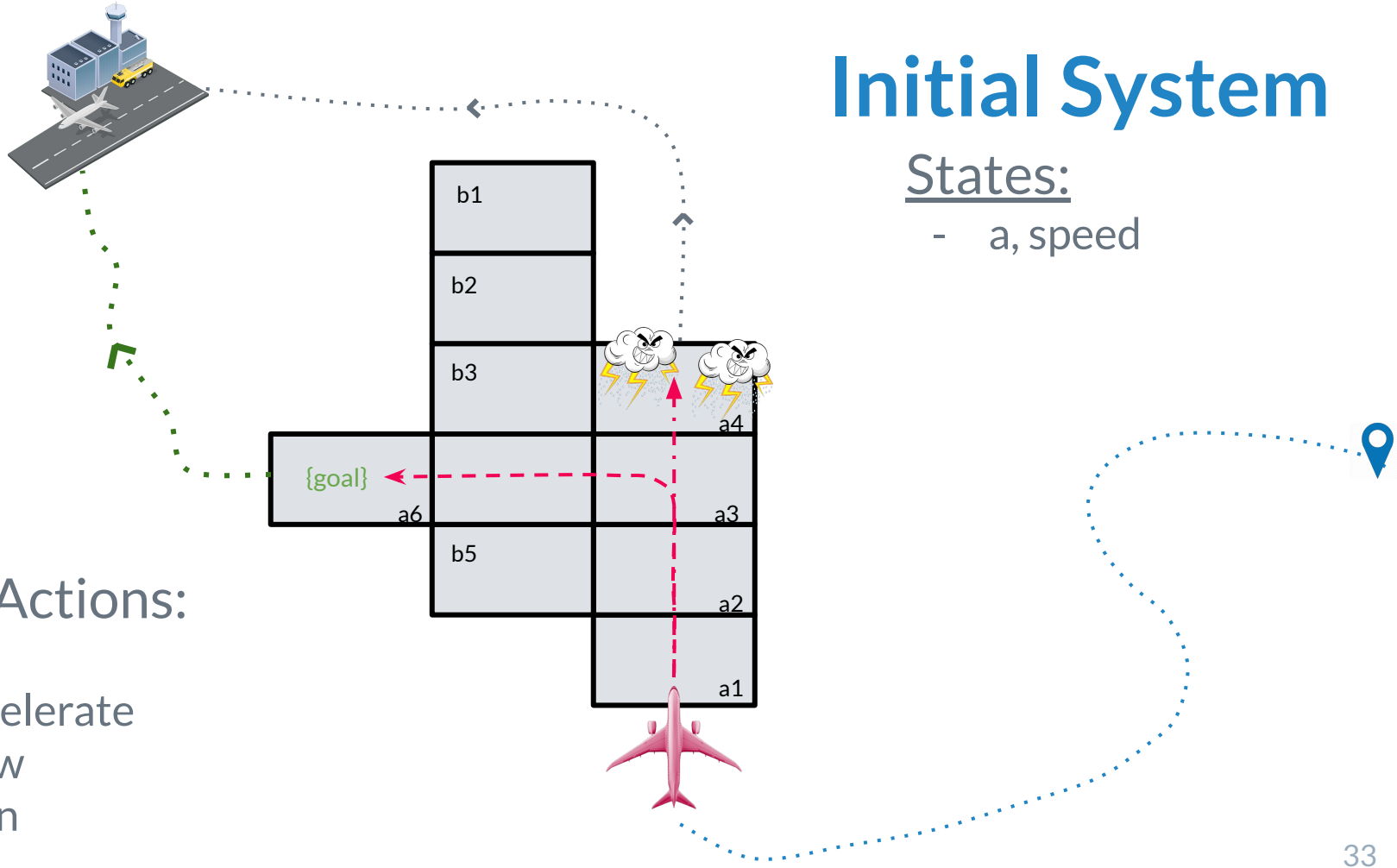
Pmax= 0.632279

Policy: (a,b):action

(1,1):brake
(1,2):accelerate
(1,3):accelerate
(2,1):brake
(2,2):accelerate
(2,3):accelerate
(3,1):brake
(3,2):brake

Aircraft Storm Avoidance

Initial System

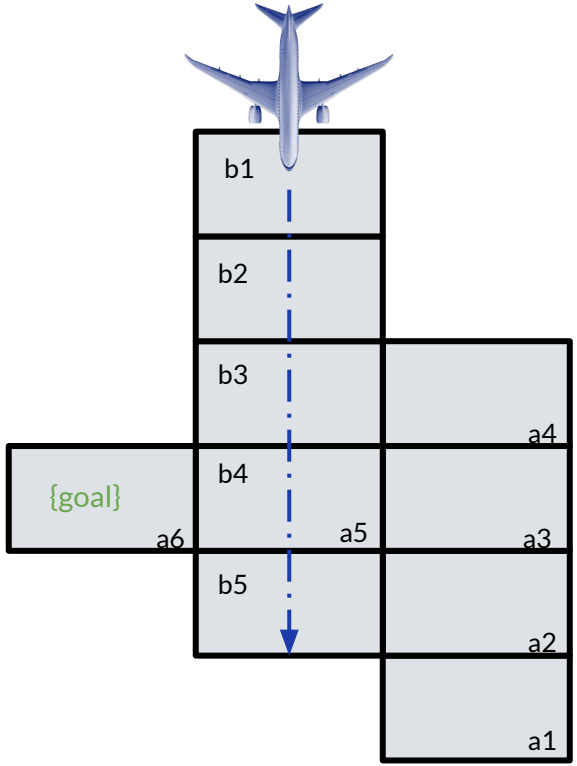


States:

- a, speed

MDP Actions:

- go
- accelerate
- slow
- turn

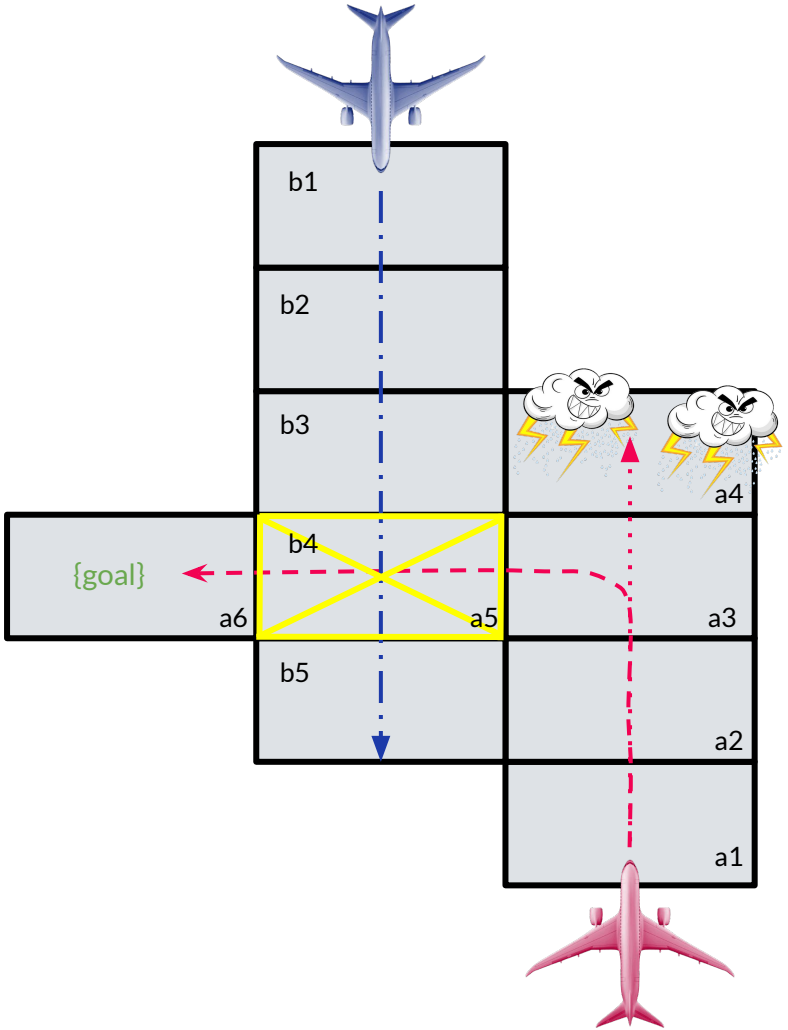


Initial System

Markov Chain

States:

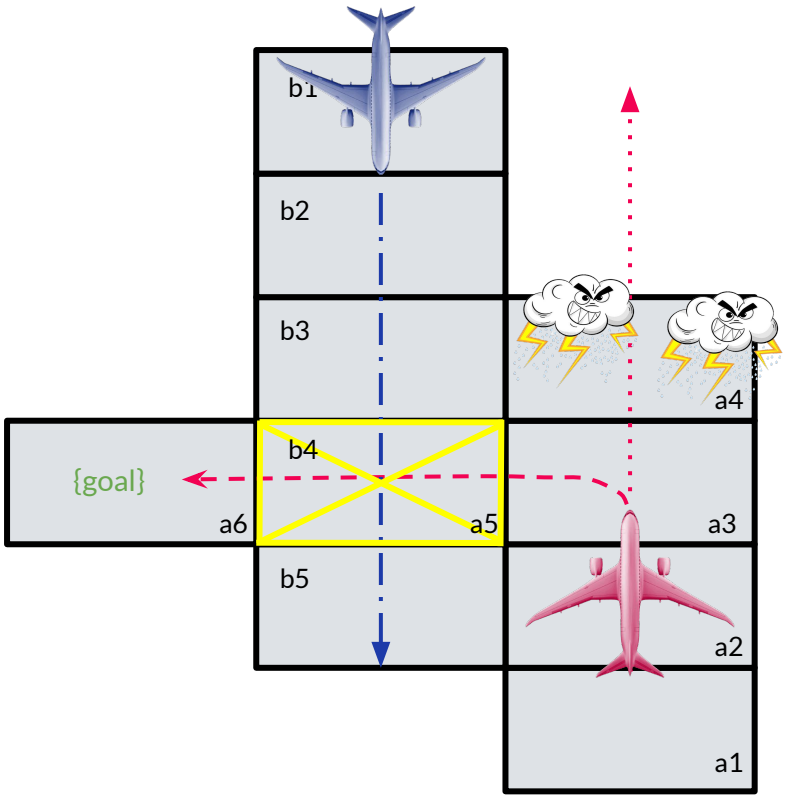
- b



Problem Formulation

- States - (a, speed, b)
- Goals
  - Never collide
  - Reach A6

Applying the Policy



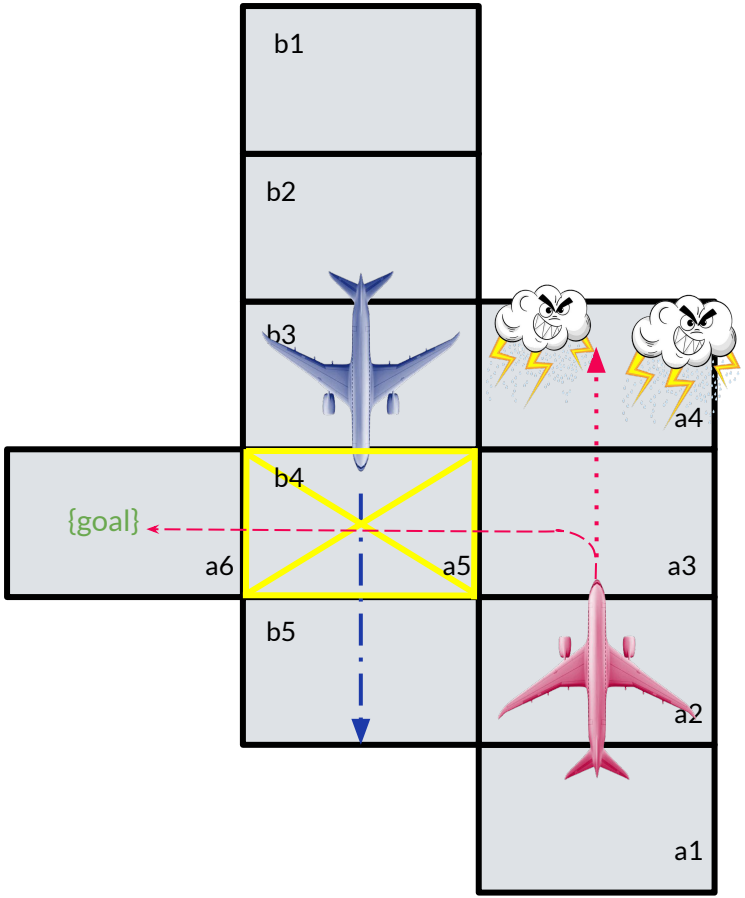
State: (2, slow, 1)

Action: Go

State: (2, fast, 1)

Action: Slow

Applying the Policy



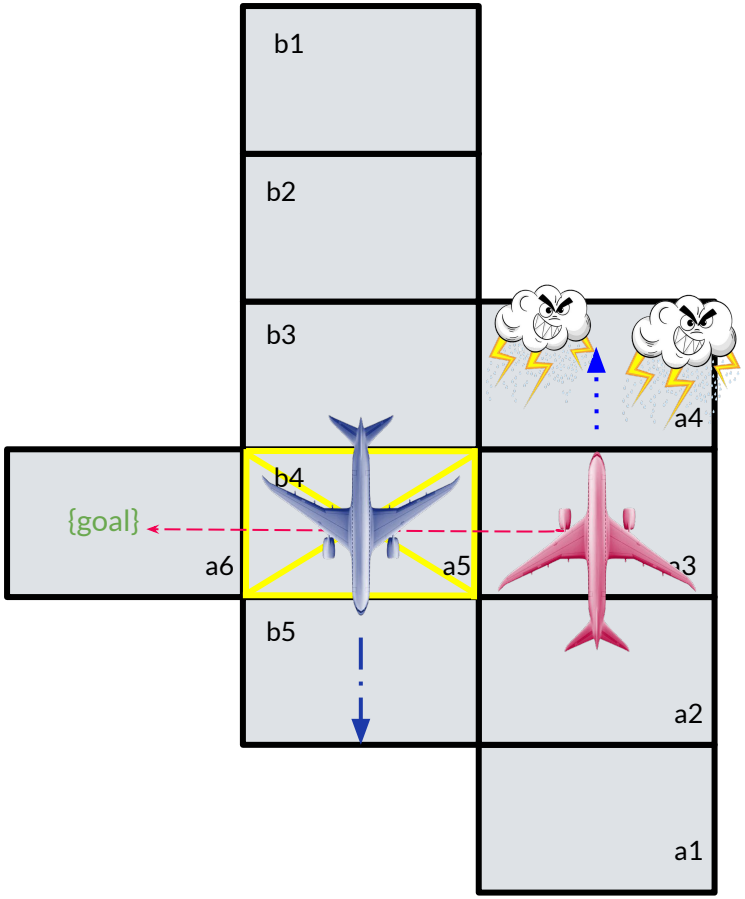
State: (2, slow, 3)

Action: Go

State: (2, fast, 3)

Action: Slow

Applying the Policy



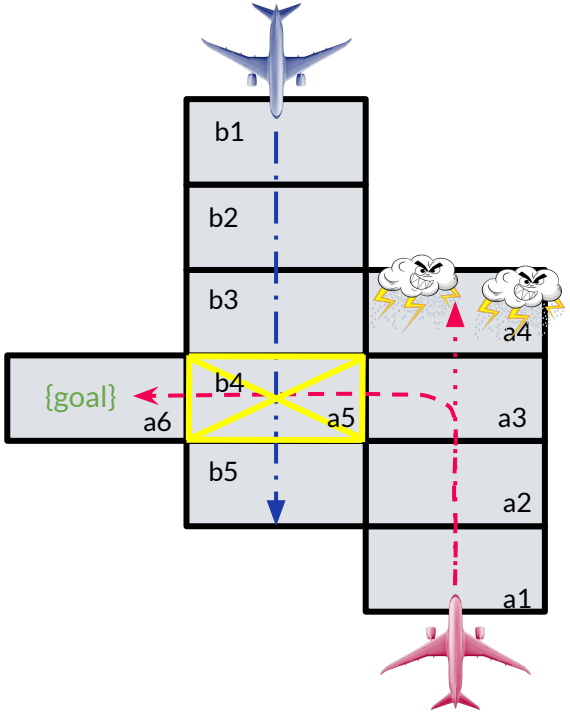
State: (3, slow, 4)

Action: Turn

State: (3, fast, 4)

Action: Slow

Policy Generation and Analysis

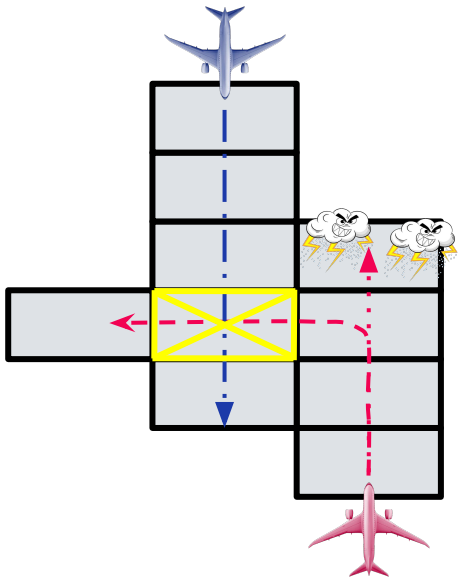


What is the maximum probability of avoiding collision AND reaching goal state?

Pmax=? [G !(a=5&b=4)&
(F (a=6))]

Policy Generation and Analysis

Pmax=? [G !(a=5&b=4)& (F (a=6))]

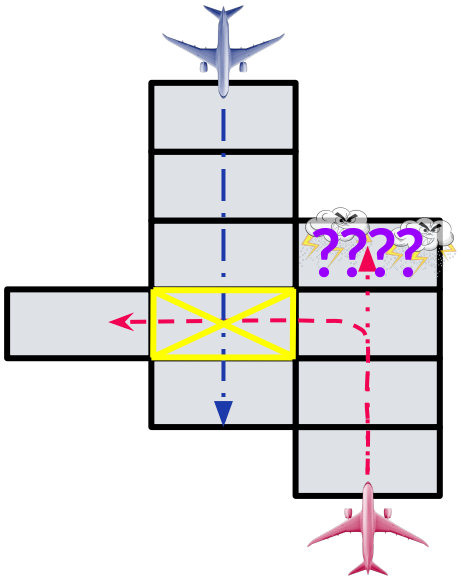


Pmax= 0.7758568

Policy: (a,speed,b):action

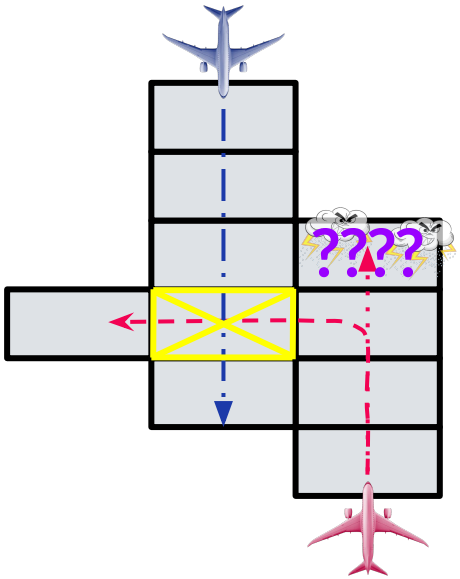
(2,1,1):slow
(3,0,4):turn
(5,0,1):acc
(5,1,1):go
…

Future Work



- What if the storm's presence is an unknown?
- Dynamic goal state
  - Reward function?
- HARD to write specs :(

Future Work



Specs:

- Aircrafts **a** and **b** never collide
- Aircraft **a** does not enter the storm **unless** a collision is eminent

Accomplishments

- Understanding MDPs, MCs, probabilistic model checking
- Using StormPy and PRISM, PCTL
- Easy to use environment setup through Docker containers
- Coming up with scenarios
  - Model checking car-pedestrian scenario
  - Model checking aircraft avoiding storm scenario
  - Dynamically changing goals (Future Work)
  - 4 way intersection model (Future Work)
- Ate Jethro's Wings