COMS 407: Final Project Presentation

Markov Chainsaw Massacre

Charles Dudley, Ranai Srivastav, Varad Kulkarni

Introduction

Many real-world cyber-physical systems have an aspect of uncertainty (i.e. a probabilistic nature) in the decisions that the agent can make, which traditional finite state machine model checking methods fail to encapsulate. Sometimes, the behavior of a system is influenced by the environment it operates in. For instance the driver in a car has the ability to decide whether to accelerate or brake, however, it may or may not do so depending on road conditions such as ice, gravel, etc. This creates a need for using probabilistic models such as Markov Chains, and Markov Decision Processes (MDP's) to model systems, and verify these models using Probabilistic Model Checking[link].

In this report we present a study of multiple real world traffic scenarios modeled in the form of Markov Decision Processes (MDP's). We use the StormPy and PRISM Model checkers[**?**] to perform Probabilistic Model Checking on these scenarios. With these tools we will measure the probabilistic properties of these systems with respect to safety properties. This allows us to quantitatively examine the best and worst case performance and provide some probabilistic guarantee about the system.

Terminology

1. Agent: The decision-making entity in a scenario. The agent is capable of making non-deterministic decisions.

2. Markov Chain: A finite state machine in which every transition is assigned a probability value between 0 and 1. Markov Chain M_c is mathematically defined as $M_c : (S, P, s_0, \Sigma, L)$ with

- Probability transition function, $P : S \times S \to [0, 1]$
- Initial state, s_0
- Set of atomic propositions, Σ
- Labeling function, L

3. Markov Decision Process: A Markov chain which specifically refers to the agent. Every transition is assigned an action as well as a probability value. Markov Decision Process M_p is mathematically defined as $M_p : (S, s_0, A, \sigma, \Sigma, L)$ where

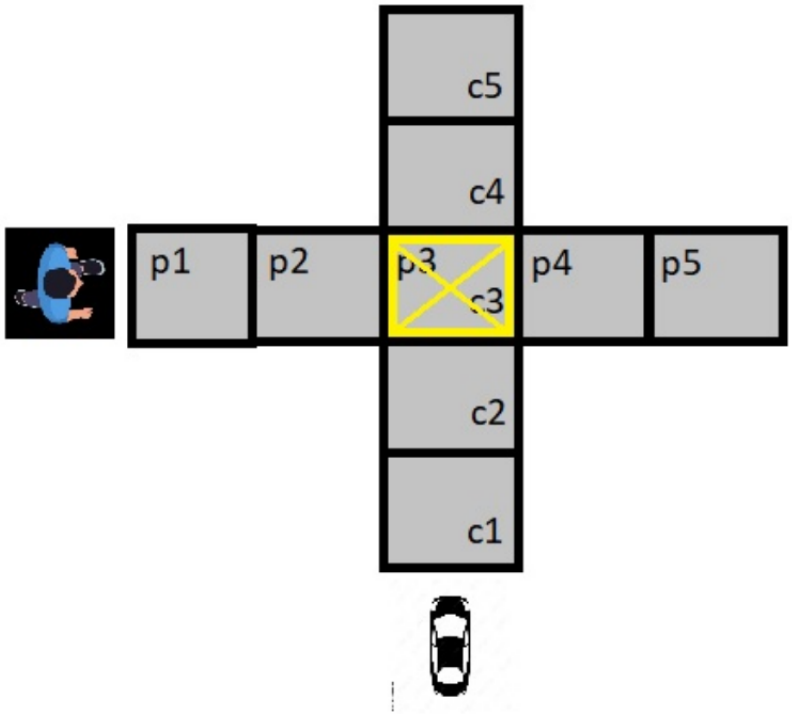
- Set of states, S
- Initial state, $s_0 \in S$
- Finite set of actions, A
- Probability transition function, $\sigma : S \times A \times S \to [0, 1]$:
- Set of atomic propositions, Σ
- Labeling function, L

4. Policy: A rule, which when applied to an MDP, tells the agent to make specific decisions at every state. It hence resolved nondeterminism.

5. Parallel Composition: A combination of an MDP and an MC that encapsulates both of their properties and is defined as follows

Overview of Scenarios

Scenario 1: Car-Pedestrian



Our first scenario involves a car and a pedestrian with intersecting trajectories. The car travels along a vertical trajectory, beginning in state c_1 and ending in state c_5, while the pedestrian travels along a horizontal trajectory, beginning in state p_1 and ending in state p_5. These two trajectories cross in the center, at c_3 and p_3, creating potential for collisions.

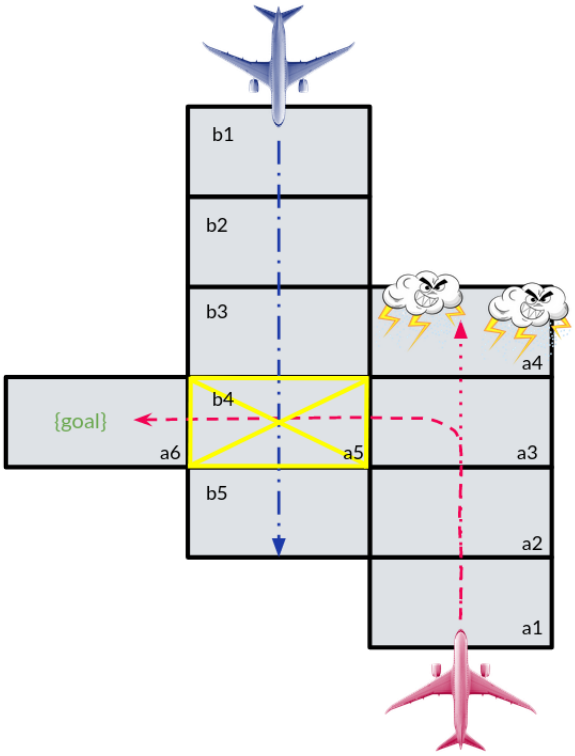
The pedestrian is modeled using a Markov Chain which defines its deterministic behavior. At any time, the pedestrian may either stay in its current state or stumble into the next state.

The car is an agent which is capable of executing two actions: `accelerate` and `brake`. One of these actions must be executed for each transition in this model. Execution of the action `accelerate` results in high probability of moving the car ahead; Execution of the action `brake` results in high probability of keeping the car in place. A policy will be applied to this agent to ensure the desired behavior is executed. The effectiveness of this policy will be examined in later sections.

The English specifications for this system are as follows:

- The car must never collide with a pedestrian.
- The car must reach its destination eventually.

Scenario 2: Aircraft Storm Avoidance



Our second scenario involves two aircraft, also with intersecting trajectories. Aircraft A has planned trajectory of a_1 through a_4. However, this trajectory is blocked by an impassable storm, therefore it must change course to navigate through a_3 and then to a_5 and a_6 to succeed.

Aircraft A is capable of travelling at two different airspeeds, `fast` or `slow`. Additionally, aircraft A is capable of executing four actions: `go`, `accelerate`, `slow`, and `turn`. Execution of the action `go` results in high probability the agent will proceed at its current rate of speed; Execution of the action `accelerate` (only possible while the agent is `slow`) results in high probability the agent will increase its speed to `fast` and move forward; Execution of the action `slow` (only possible while the agent is `fast`) results in high probability the agent will reduce its speed to `slow` and maintain its position; Execution of the action `turn` (only possible while the agent is `slow` and in a_3) results in high probability the agent will move to a_5;

An additional aircraft, aircraft B, has a trajectory which intersects aircraft A's maneuver. Aircraft B's trajectory moves through states b_1 to b_5, intersecting with aircraft A's trajectory at b_4 and a_5. A policy will be applied to aircraft A to ensure desired behavior is executed, and this policy will be examined in later sections.

The English specifications for this system are as follows:

- The agent must never collide with another aircraft.
- The agent must eventually reach its goal state, a_6.

Modeling the Scenarios

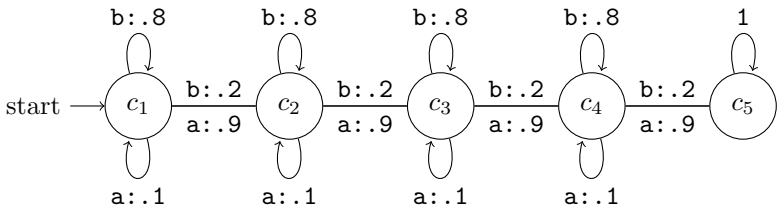
Car-Pedestrian

For the **Car-Pedestrian** scenario we define the following MDP, M_c, and MC, M_p, for the car and pedestrian, respectively.

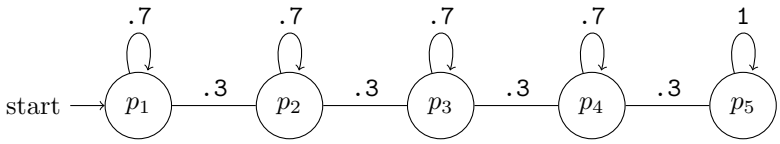
M_c:

Actions:

1. `a`: `accelerate`
2. `b`: `brake`

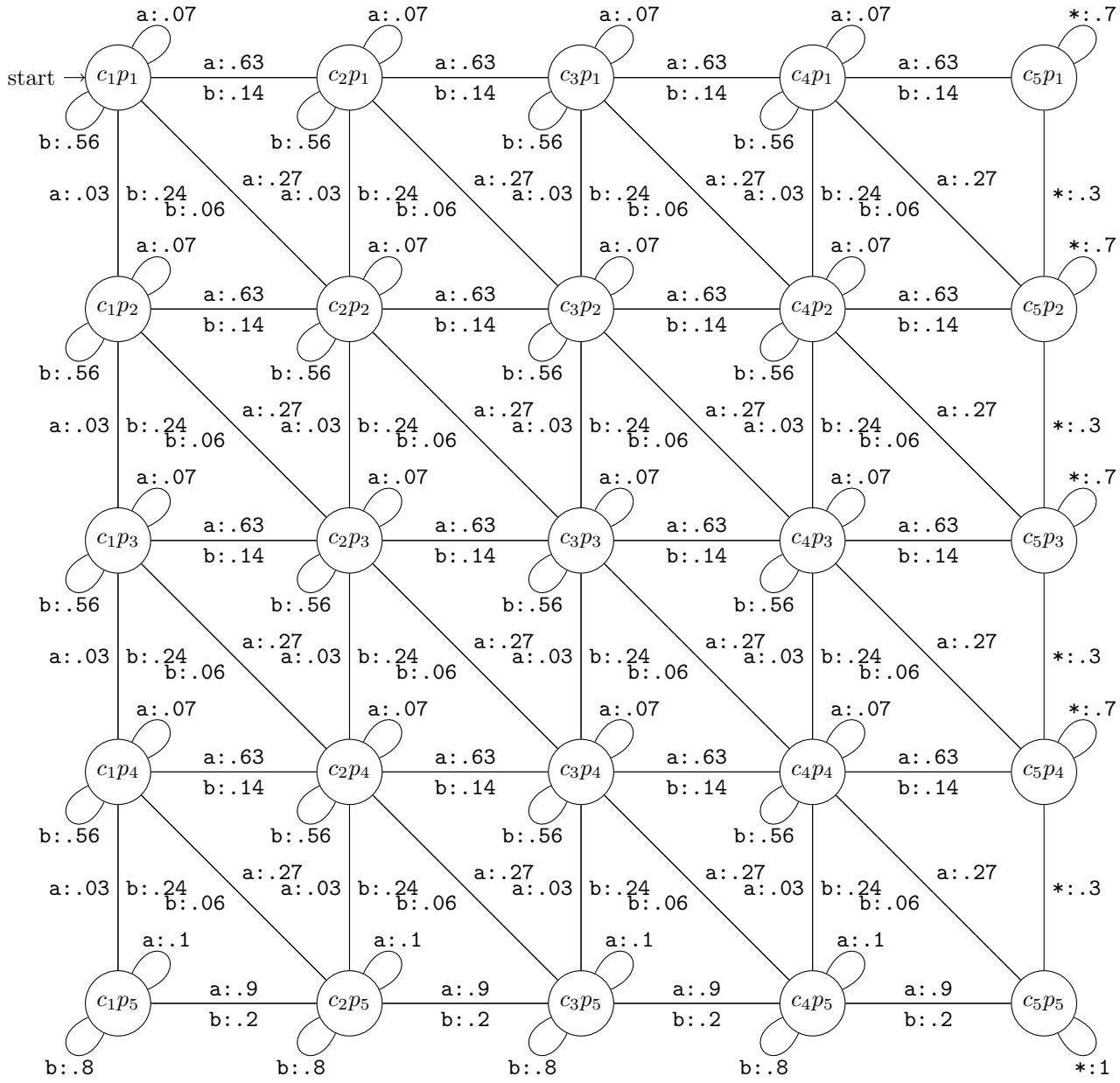


M_p:



We apply parallel composition to these systems to obtain a combined MDP which models their joint behavior. The resultant probabilities for each transition will be a product of the individual transitions.

$M_c||M_p$:

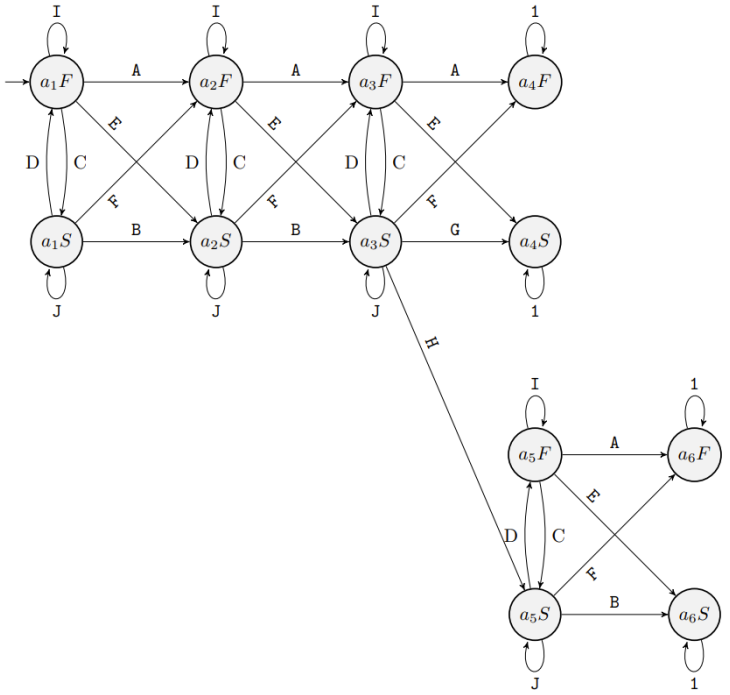


Here c_ip_j refers to the current state of the car and pedestrian.

Aircraft Storm Avoidance

For the **Aircraft Storm Avoidance** scenario we define the following MDP, M_A, and MC, M_B, for aircraft A and B, respectively.

M_A:



Ledger

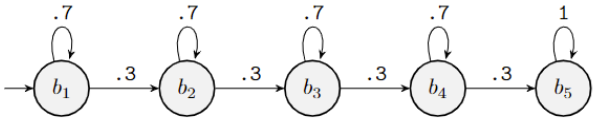
A:	go: .8 slow: .09
B:	go: .5 acc: .02
C:	go: .02 slow: .2
D:	go: .09 acc: .2
E:	go: .08 slow: .7
F:	go: .01 acc: .7
G:	go: .5 acc: .02 turn: .1
H:	turn: .9
I:	go: .1 slow: .01
J:	go: .4 acc: .08

Here a_1F represents the agent being at state 1 at a `fast` airspeed, a_2S represents state 1 at a `slow` airspeed, and so on.

We have assigned probabilities to this model in order to imitate a reasonably realistic system. If the agent is `fast`, it has a higher probability of moving forward than if it is `slow`. If it is `fast` and tries to slow down (executing action `slow`), it is quite likely to still move forward as opposed to staying in the same block. If the aircraft is `slow` and tries to speed up (executing action `accelerate`), it will most likely change speed to `fast` and move ahead. If the arcraft tries to maintain speed (executing action `go`), it may end up accelerating or decelerating due to factors such as air resistance and tailwinds.

The intruder aircraft is modeled as follows. Its transition probabilities are assigned to model an airspeed that is somewhere in between the agent's two speeds, `fast` and `slow`.

M_B**:**



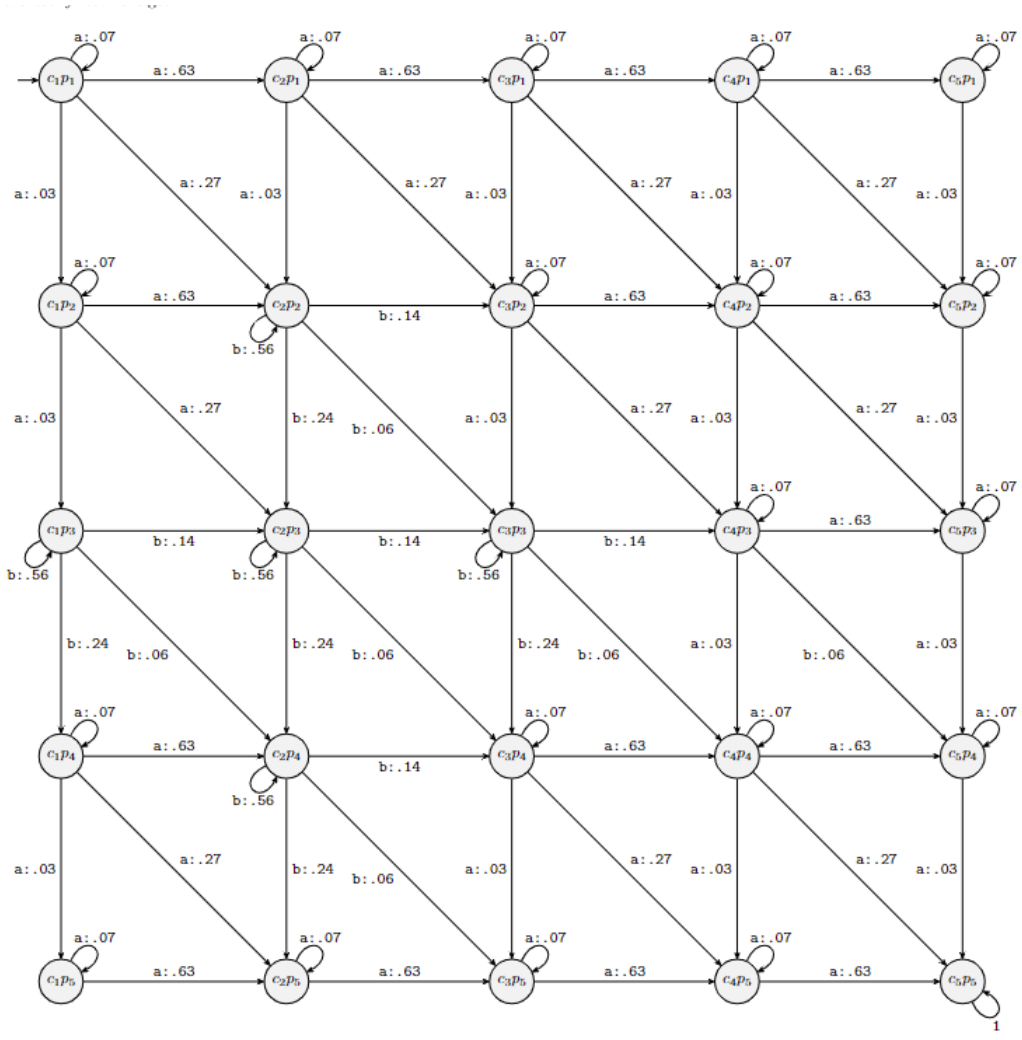
Similarly to the first scenario, we apply parallel composition to these systems to obtain a combined MDP which models their joint behavior. The parallel composition for this combined system will be quite large (60 states). We chose not to elaborate for the sake of brevity.

Car-Pedestrian - Applying the Policy

After closely inspecting our parallely composed MDP for **Car-Pedestrian**, we translated our English specifications into a PCTL specification as `G (F c5 & !(c3p3))`. This specification was applied to the model in the form of the following (sub-optimal) policy:

Brake; *If* c_2p_2, c_2p_3, c_1p_3
Accelerate; *Otherwise*

This policy directs the car to reach its destination as soon as possible while avoiding a collision. After applying this policy, the MDP of the parallelly composed system is as follows:



On closer inspection we can see that the action `accelerate` is not defined for the states specified in our policy.

Due to the relatively simple nature of the car-pedestrian scenario, we were able to directly modify the parallel composition to adhere to the policy. This new MDP has been shown above. However, we knew that this would be highly difficult for our second air traffic scenario which had a much larger state space. After doing some research, we found out that the PRISM model checker is capable of generating an optimal policy while simultaneously performing verification. We will discuss this in more detail in the next section.

Car-Pedestrian - Generating Policies

After the tedious process by which we analyzed our initial policy over the **Car-Pedestrian** system, we looked for other methods to make this process more efficient. We hoped to automate the process of parallel composition and to generate optimal policies with respect to our specifications. PRISM model checker is where we were able to realize this functionality. Details on implementation and results are discussed in coming sections.

Aircraft Storm Avoidance - Generating the Policy

With the larger size of this model and the efficiency we found with PRISM model checker, we did not attempt to manually create a policy or manually implement a parallel composition for this scenario. Details on implementation and results are discussed in coming sections.

Implementation

We provide a Docker container to make it easy to test the code. This docker container can be accessed in two ways. More information can be found in the README in the repo:

1. Open the repo in VSCode. VScode will prompt you to open it as a devcontainer. Click on it to allow it to set up the environment to allow easy code editing.
2. Navigate to the repo. Run the command `docker build .` to build an image. Run the image using the command `docker run -it <C ID>` where `<C ID>` can be found using the command `docker image list`. More information can be found in the README file.

Car-Pedestrian System

To perform probabilistic model checking on the **Car-Pedestrian** system defined above, we used Prism to define the model and StormPy model checker. We created a Python script to generate PRISM code of the scenario MDP with and without the policy applied. We then passed the generated model to a StormPy file containing the system specifications. Upon execution, we obtained a probability value associated with maximum success.

The PRISM file titled `car_ped_parallelcomposition_policy.prism` was used to to create the parallel composition of M_c and M_p and then apply the aforementioned policy. A snippet of the code is provided below:

```
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);
  [] c=1 & p=2 -> 0.07:(c'=1) & (p'=2) + 0.63:(c'=2) & (p'=2) + 0.03:(c'=1) & (p'=3) + 0.27:(c'=2) & (p'=3);
  [] c=1 & p=3 -> 0.56:(c'=1) & (p'=3) + 0.14:(c'=2) & (p'=3) + 0.24:(c'=1) & (p'=4) + 0.06:(c'=2) & (p'=4);
  [] c=1 & p=4 -> 0.07:(c'=1) & (p'=4) + 0.63:(c'=2) & (p'=4) + 0.03:(c'=1) & (p'=5) + 0.27:(c'=2) & (p'=5);
  [] c=1 & p=5 -> 0.07:(c'=1) & (p'=5) + 0.63: (c'=2) & (p'=5);
```

We generate the necessary PRISM files using the generator scripts. These can be run using the following command
`python3 <NAME OF FILE>`, where the files that can be run are

1. `car_ped_parallelcomposition_generator.py`
2. `car_ped_parallelcomposition_auto_generator.py`

These files enumerate through all the possible states (the cross product of the states in both modules) and calculate the probabilities for the resulting transitions by multiplying the individual transitions as explained earlier. This is an inefficient way of doing this since if we need to change the policy, a new script needs to be written. These generated PRISM files were mainly run with StromPy.

After learning more about Prism's parallel composition and policy-generation capabilites, we changed our model-checking process. We created the MDP and MC for the individual entities as separate modules in Prism, which is capable of automatically creating a parallel composition of all listed modules. This new PRISM model can be found in file `car_ped_parallelcomposition_auto.prism`.

To perform model checking, we run a terminal command that calls the Prism file and passes the desired specification. An example of which is as follows:
`prism car_ped_parallelcomposition.prism -pf P =?  [F (c = 3 & p = 3)]' -exportstrat stdout.`

Aircraft Storm Avoidance System

To model the second scenario, we use the parallel composition capability found in PRISM. We implement two modules, both as MDPs. The Markov Chain is also implemented as a MDP but with probabilities for each action being the same. Thus, for each action taken by the MDP, the same action is taken in the MC. This is done to ensure that PRISM is able to synchronize actions across time and force both systems to take transitions simultaneously. PRISM defines synchronizing as choosing the same action for multiple modules at the same timestep given that they have actions with the same name. This is especially useful if for a certain action of one module, you want the other module to do something consequently. In our case, this was helpful in ensuring that each timestep, the pedestrian and the car both had to choose an action. This was done according to the code in the file `car_ped_parallelcomposition_auto.prism` as follows:

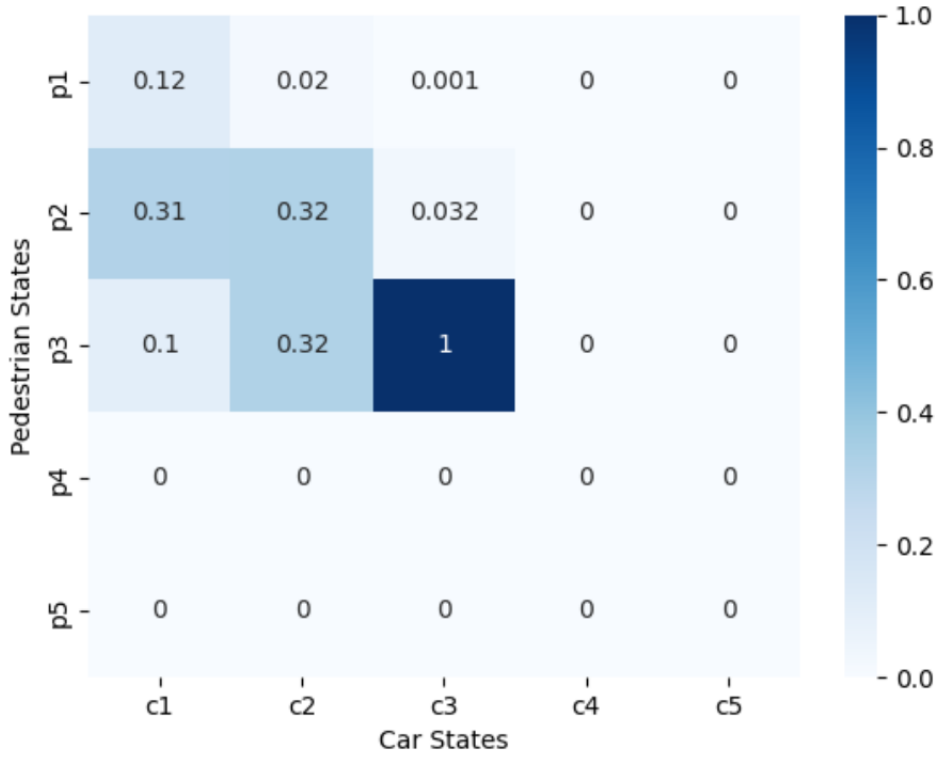
```
mdp

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
```

Probabilistic model checking was conducted over this model using the PRISM model checker. This model checker was used to produce policies for given specifications. An example of a command used to generate and analyze a policy for this model and specification in file `turn_auto.specs` is as follows: `prism turn_auto.prism turn_auto.specs -exportstrat stdout`.

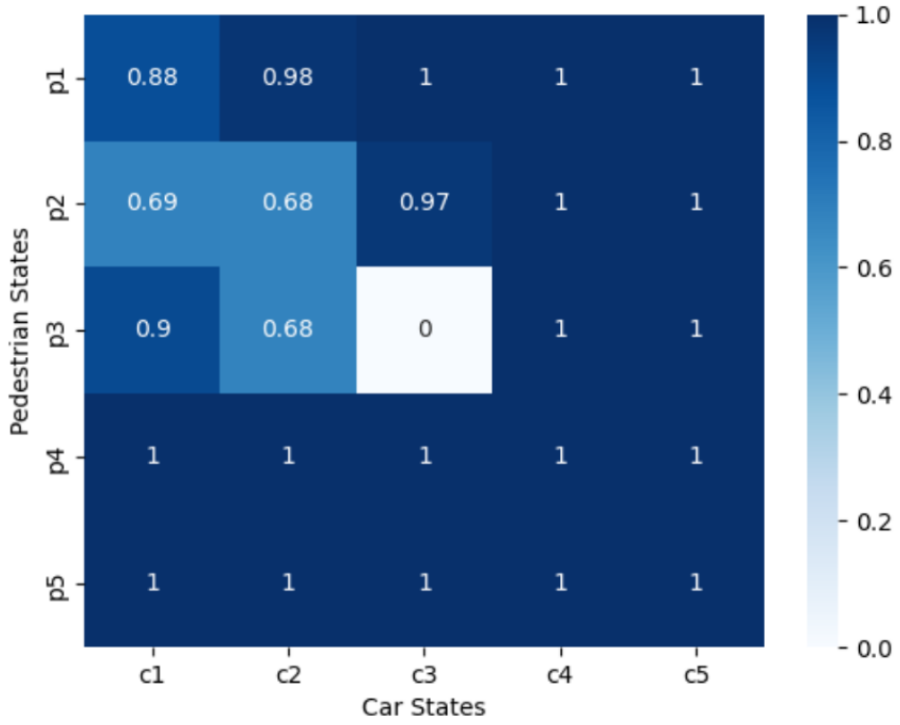
Results

The bulk of our analysis was conducted in the form of running PRISM commands with specifications including `Pmax =?` and `Pmin =?`. These commands reason over non-determinsitic systems (MDPs) to measure the maximum or minimum probability of satisfaction of the given spec for any policy. As mentioned in the implementation, we also generated policies using PRISM.

Car-Pedestrian - Manual Policy:



The above heatmap colorcodes the probability of satisfaction of $\Diamond(c = 3 \wedge p = 3)$ given our manually implemented policy for the **Car-Pedestrian** system. Since we define $c = 3$ and $p = 3$ as a collision, the probability of collision in this state is 100%. The probability of collision at $c = 1$ and $p = 1$ can also be thought of as the probability of satisfaction of the specification over the entire system.



The above heatmap colorcodes the probability of satisfaction of $\square\neg(c = 3 \wedge p = 3) \wedge \Diamond(c = 5 \wedge p = 5)$ (avoid collision and reach goal) given our manually implemented policy for the **Car-Pedestrian** system. Here we can observe that the probability of satisfaction of the specification is 100% once the pedestrian or the car passes the intersection, and 88% in the initial state.

Car-Pedestrian - Worst-Case Policy:

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

The above policy was generated using PRISM on the automatic parallel composition of the **Car-Pedestrian** scenario. Here `(x,y):z` represents the state $a_x b_y$ selecting action `z`. This was generated using the specification `Pmax=? [ F(c=3 & p=3) ]` as a parameter to model checking `car_ped_parallel_composition_auto.prism`. The resulting policy attempts to maximize the probability of collisions, which is achieved with probability 0.632. A log containing the full output for this experiment can be found in `car_ped_output_max_collide.txt`.

Car-Pedestrian - Optimal Policy Attempt 1:

```
Exporting strategy as actions below:
```

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

(5,5):brake
```

The above policy was also generated using PRISM on the automatic parallel composition of the **Car-Pedestrian** scenario. This was generated using the specification `Pmax=? [ G (!(c=3&p=3)) ]` as a parameter to model checking `car_ped_parallel_composition_auto.prism`. The resulting policy attempts to maximize the probability of avoiding collisions, which is achieved with probability 0.887. A log containing the full output for this experiment can be found in `car_ped_output_max_noCollide.txt`.

Car-Pedestrian - Optimal Policy Attempt 2:

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

The above policy was also generated using PRISM on the automatic parallel composition of the **Car-Pedestrian** scenario. This was generated using the specification `Pmax=? [ G ( !(c=3 & p=3) & (F (c=5 & p = 5))) ]` as a parameter to model checking `car_ped_parallel_composition_auto.prism`. The resulting policy attempts to maximize the probability of avoiding collisions and reaching the end states, which is achieved with probability 0.887. A log containing the full output for this experiment can be found in `car_ped_output_max_noCollide_reach.txt`.

Aircraft Storm Avoidance - Optimal Policy:

A policy was also generated using PRISM on the automatic parallel composition of the **Aircraft Storm Avoidance** scenario. This was generated using the specification `Pmax=? [ G !(a=5&b=4)&(F (a=6)) ]` as a parameter to model checking `turn_auto.prism`. The resulting policy attempts to maximize the probability of avoiding collisions and Aircraft A reaching the end state (avoiding the storm). This specification

can be satisfied with probability 0.776. A log containing the full output and policy for this experiment can be found in `turn_auto.out`. The format of the policy is similar to the previous examples, with `(x,Z,y):w` denoting state $a_x b_y Z$ ($Z \in \{\texttt{fast}, \texttt{slow}\}$) being assigned action w.

Analysis

Car-Pedestrian System

Extensive results have been collected on the **Car-Pedestrian** system. Firstly, our manually generated policy was examined, as we visualized with the two heatmaps above. These results show that the system provides a safety guarantee for collision avoidance with probability 0.88. This value is sub-optimal, but reasonable given for little computational effort. Additional policies and their respective specifications have also been outlined above.

The automatically generated policy **Car-Pedestrian - Optimal Policy Attempt 1** provides an optimal policy for our collision-avoidance specification. The safety guarantee is nearly identical to that of our manual policy, but marginally better. Important to notice in this policy is the tendency to brake when collision becomes impossible. This is not expected behavior for our system, as we should also strive to reach the end state. Therefore we generated an additional policy with a modified specification, **Car-Pedestrian - Optimal Policy Attempt 2** (also shown above). This is an optimal policy for our collision-avoidance and goal-reaching specification. The safety guarantee is identical to the previous policy, however we can see the tendency to brake after avoiding a collision has been replaced with a tendency to accelerate.

The automatically generated policy **Car-Pedestrian - Worst Case** provides an optimal policy for our collision-assurance specification. This experiment is intended to consider the worst possible behavior of the system, potentially executed if the system is compromised by an attacker and a dangerous policy is put in place. With this policy we can see a collision can be guaranteed with probability 0.632.

We did not design our system to maximize or minimize any of these safety guarantees; rather we attempted to model a real system and measure the inherent properties emergent from its probabilistic behavior.

Aircraft Storm Avoidance System

Similar analysis has been conducted on the **Aircraft Storm Avoidance** scenario. We provide one generated policy for this case in **Aircraft Storm Avoidance - Optimal Policy** above. This policy provides a 0.776 probability of satisfying our storm and collision avoidance spec, meaning even with the optimal policy, this system can only navigate its course safely 77.6% of the time. This safety guarantee speaks to the design of the system and reliability of its components. In a real system, this value would be used to refine the systems that carry out execution of actions `go`, `slow`, `accelerate`, and `turn` to improve this guarantee.

Challenges and Future Work

We encountered many challenges during work on this project. The most challenging aspect of this project was the creation of systems which are reasonable to ask probabilistic queries about. Creating our **Car-Pedestrian** and **Aircraft Storm Avoidance** systems was not a trivial process and required extensive research of both tools and applications.

The **Aircraft Collision Avoidance** system was particularly difficult to create due to its greater complexity. We originally considered an even more complex (and interesting) model in which the existence of a storm was a probabilistic element of the system. In this case, the presence of the storm would dynamically change the trajectory of Aircraft A, allowing it to follow its planned trajectory if no storm is present, initiating a safe `turn` maneuver if a storm is present and no collision is eminent, or proceeding into the storm if a storm is present and a `turn` will likely cause a collision. This model would have allowed us to explore the possibility of entering a potentially dangerous state (storm) to avoid a certainly dangerous state (collision). For this model we prepared the following English specifications:

- Aircraft A and B never collide.
- Aircraft A does not enter the storm unless a collision with Aircraft B is eminent.

We were unable to correctly translate these specifications into PCTL. We made many attempts at implementation of these specifications, some of which are visible in `turn_storm_specs.scratch`. Analysis was conducted on each of these specifications, however inspection of the policies generated in each case showed that desired behavior was not present in the the most interesting states. Eventually we reduced the complexity of this model by removing the uncertainty of the storm and conducted analysis on the **Aircraft Storm Avoidance** system discussed previously. Further exploration of this system is a very interesting opportunity to extend this work in the future.

Additional challenges were encountered in learning to use the various tools, including StormPy and PRISM to model and verify our systems. The documentation for StormPy in particular is lacking, and left us struggling to make progress for some time. It was not until we changed our focus to the PRISM model checker that we were able to make greater progress with both of our systems, automatically conducting parallel compositions and creating optimal policies.

Bibliography

1. M. Kwiatkowska and G. Norman and D. Parker, 2011, *PRISM 4.0: Verification of Probabilistic Real-time Systems*

2. T. Wongpiromsarn and R. D. Murray, 2020, Course Material for EECI_IGSC-2020

3. Schäffeler, Maximilian and Abdulaziz, Mohammad, June 2023, *Formally Verified Solution Methods for Markov Decision Processes*

4. Hansson, Hans and Jonsson, Bengt, 1994, *A logic for reasoning about time and reliability*

5. Tingting Han and Katoen, J.-P. and Berteun, D., 2009 *Counterexample generation in probabilistic model checking*

6. Lacerda, Bruno and Parker, David and Hawes, Nick, 2014, *Optimal and dynamic planning for Markov decision processes with co-safe Properties*