COMS 412: Final Project

Various Concurrency Experiments with Probabilistic Model Checking

Charles Dudley

Introduction

For safety critical systems, testing is simply an insufficient method of assessing correctness. Testing is a very limited method of analysis in that it can only find the bugs you look for. A robust test suite can be generated for a system which still misses some critical error in implementation. For *safety critical systems* (systems where failure is extremely costly), we cannot afford for errors to slip past our testing. Examples might include an air traffic control system, where any fault could cost hundreds lives, or the software controlling a satellite launched away into space, which could cost untold millions of dollars if it were to crash.

Formal Verification is a powerful tool used in verifying the correctness and reliability of safety critical systems which can account for the shortcomings of traditional testing. Formal Verification techniques such as *Model Checking* are a more robust method of analysis that involves the exploration of a system's entire state space. These techniques can be used to discover errors in implementation and/or specification which may easily be missed by any test generation technique. When applied effectively, model checking can provide absolute proofs of an implementation's alignment to given specifications (these determinations are referred to as *correctness*). A model checker's proof of correctness can be a very powerful safety guarantee for smaller systems, but for many systems it is very difficult (or not practical) to achieve such a guarantee. Firstly, for many systems it is simply impossible to explore the entire state space with a reasonable amount of compute and memory. Models of systems can easily contain trillions of states, in which case verification of a property over the entire system is very difficult. These systems still may be analyzed by applying simplifications to the model of the system, but the guarantees resulting from verification on the simplified model may not transfer to the real system. Additionally, many systems involve a level of uncertainty that clouds the definitive *yes/no* property satisfaction we would like to assert on systems. The behavior of a system is likely influenced by the environment it operates in. For instance, the performance of a self-driving car is affected by the accuracy of its computer vision processing (a probabilistic process itself), the conditions of the road surface, and behavior of other (potentially manually operated) vehicles. These external influences on the behavior of a system are very difficult to capture in traditional verification methods, but may be extremely valuable in assessing the safety of that system. These systems may still be analyzed by applying certain assumptions to the environment, such as modeling the road surface conditions as a markovian process.

In this report I present a study of multiple applications of probabilistic model checking to real world scenarios, such as air traffic control systems. I use PRISM model checker[**?**] and PRISM specification language to model and analyze these systems. For each system I provide quantitative analysis of safety properties.

Included is a GithHub repository containing all source code for this project. The repository includes a `.devcontainer` directory which can be used to start an editing environment inside a Docker container including all of the project dependencies. Instructions on how to start this environment can be found here.

Terminology

1. **Discrete-Time Markov Chain**:

A *Discrete-Time Markov Chain* (DTMC) is a stochastic process which models the behavior of some process external to our *agent*. DTMCs are represented as finite state machines where every transition is assigned a probability value between 0 and 1. For a DTMC, a transition from one state to another is controlled entirely by the transition probabilities of the system. Behavior of this model can be predicted and analyzed without any modification.

DTMC M_c is formally defined as $M_c : (S, \sigma, s_0, \Sigma, L)$ with

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

2. 1. **Agent**: A decision-making entity in a scenario. The agent is capable of making non-deterministic decisions, called *actions*.

3. **Markov Decision Process**:
A *Markov Decision Process* (MDP) is a stochastic process modeling the behavior of a *non-deterministic agent*. MDPs are represented as finite state machines with each transition assigned an *action* as well as a probability value between 0 and 1.
An agent (modeled with an MDP) must *non-deterministically* select an *action* with every state transition. This action may affect the probability transition function, therefore behavior cannot be predicted in isolation. Markov Decision Process M_p is formally defined as $M_p : (S, s_0, A, \sigma, \Sigma, L)$ with

- 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, resolves the non-determinism in the system. A policy is essentially a list of rules which the agent must follow, specifying an action the agent must take under certain conditions. Application of a policy to an MDP reduces it to a deterministic system which we can reason about more effectively.

5. **Parallel Composition**: A system which models the joint behavior of multiple (synchronized) independent systems.

A parallel composition of an MDP, $M_p : (S_p, s_{0p}, A, \sigma_p, \Sigma_p, L_p)$ and an DTMC, $M_c : (S_c, \sigma_c, s_{0c}, \Sigma_c, L_c)$, is defined as $M_{||} : (S, s_{00}, A, \sigma, \Sigma, L)$ with

- Set of states, $S = S_p \times S_c$
  - State $(s_p, s_c) \in S$ may be abbreviated to s_{pc}
- Initial state, $s_{00} = (s_{0p}, s_{0c}) \in S$
- Finite set of actions, A
- Probability transition function, $\sigma : S \times A \times S \to [0, 1]$
  - $\sigma(s_{ix}, a, s_{jy}) = \sigma_p(s_i, a, s_j) \times \sigma_c(s_x, s_y)$
- Set of atomic propositions, $\Sigma = \Sigma_p \cup \Sigma_c$
- Labeling function, $L : S \to \Sigma$

A parallel composition of an MDP, $M_x : (S_x, s_x, A_x, \sigma_x, \Sigma_x, L_x)$ and another MDP, $M_y : (S_y, s_y, A_y, \sigma_y, \Sigma_y, L_y)$, is defined as $M_{||} : (S, s_{xy}, A, \sigma, \Sigma, L)$ with

- Set of states, $S = S_x \times S_y$
  - State $(s_x, s_y) \in S$ may be abbreviated to s_{pc}
- Initial state, $s_{xy} = (s_x, s_y) \in S$
- Finite set of actions, $A = A_x \cup A_y$
- Probability transition function, $\sigma : S \times A_x \times A_y \times S \to [0, 1]$
  - $\sigma(s_{im}, a_x, a_y, s_{jn}) = \sigma_x(s_i, a_x, s_j) \times \sigma_y(s_m, a_y, s_n)$
- Set of atomic propositions, $\Sigma = \Sigma_p \cup \Sigma_c$
- Labeling function, $L : S \to \Sigma$

Tools

PRISM

PRISM is both a *specification language* and a *probabilistic model checker*. Both forms of PRISM are used in this project to conduct probabilistic analysis on a series of probabilistic systems.

PRISM (specification language) provides simple and concise syntax for defining deterministic and non-deterministic systems, such as DTMCs and MDPs. A system can be defined by its state variables, actions (if necessary), probability transition function. This is how all models are defined for this project. Statements defining the probability transition function will include *action labels*, *guard clauses*, and *transition bodies*. The *action label* is a string which restricts a transition to be made only when a particular action is selected. The *guard clause* is a boolean expression which defines the conditions under which a transition is enabled. The *transition body* of this statement defines the probabilities for making certain transitions; An entire transition definition might look like:

```
[go] a != 5 & a != 8 ->
            .85: (a'=a+1) +
            .15: (a'=a)   ;
```

PRISM enables automatic parallel composition of multiple independent systems, regardless of their deterministic/non-deterministic nature. This is a very powerful feature for models which include interaction between several entities and is a large reason why PRISM was selected as the modeling language.

PRISM also enables analysis of systems using PCTL/PLTL specifications. Specifications used in this research take the form of $P_{max} =?[\phi]$, for some LTL expression ϕ. This specification can be used to compute the maximum probability that a system may satisfy a specification; In this computation, PRISM (model checker) can generate an optimal policy for a non-deterministic system. This optimal policy allows for reasoning about the maximum (or minimum) safety guarantee of a system with respect to ϕ.

For a PRISM model implemented in `example.pm` and a specification file implemented in `example.specs`, PRISM model checking can be performed over this system using the following command:

```
$ prism example.pm example.specs
```

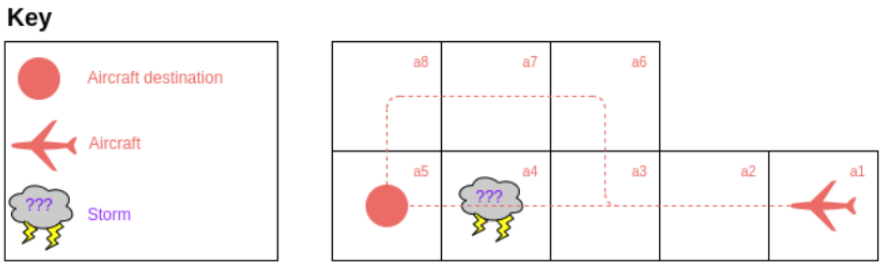
Optionally, PRISM can be configured to output an optimal policy with command line option `--exportstrat stdout`, though this forces PRISM to use *explicit-state model checking*, which drastically increases the memory required to perform verification.

Overview of Scenarios

Scenario 1: Around the Storm

For this scenario we will consider a single non-deterministic agent and its interaction with a probabilistic environment.

Visualization:



Description:
An *aircraft* is travelling along a planned trajectory at a constant rate. The planned trajectory, T, follows along locations $[a_1, a_2, a_3, a_4, a_5]$ (without making any maneuvers), which is the optimal path from initial location, a_1, to destination, a_5.

There is a certain probability that a *storm* will occur along the planned trajectory (in location a_3), making the planned trajectory dangerous for air travel. In this case, the aircraft should maneuver to follow a safe trajectory, T'. This alternate trajectory traverses along locations $[a_1, a_2, a_3, a_6, a_7, a_8, a_5]$. The aircraft may fail to execute this maneuver.

In order to minimize cost, trajectory T is preferred. However, a maneuver should be executed by the aircraft if it is on course for dangerous airspace (the storm).

Aside: Although very simple, this model can provide interesting insights into expected behavior of a probabilistic system. Results and practices from this example may be expanded upon to model much more interesting scenarios, as will be explored in later sections.

Specifications:
In order to conduct analysis over this system, specifications of desired/undesired behavior must be drafted. A few examples of English specifications:

- The aircraft always eventually reaches its destination.
- The aircraft reaches its destination without entering the storm.
- The aircraft reaches its destination without entering the storm while only changing course to avoid a storm.
- The aircraft enters the storm.

For probabilistic analysis, these specifications must be revised to the following form:

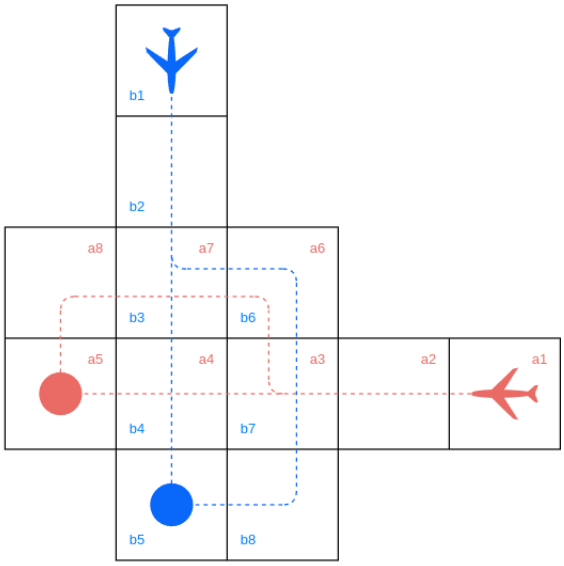
- What is the maximum probability the aircraft always eventually reaches its destination?
- What is the maximum probability the aircraft can reach its destination without entering the storm?

- What is the maximum probability the aircraft can reach its destination without entering the storm, given it never changes course except to avoid a storm?
- What is the maximum probability the aircraft enters the storm?

Scenario 2: Air Traffic Control

For this scenario we will consider the interaction between two non-deterministic agents.

Visualization:



Description:
An aircraft, *pink*, is travelling along a planned trajectory at a constant rate. The planned trajectory, T_p, follows along locations $[a_1, a_2, a_3, a_4, a_5]$ (without making any maneuvers), which is the optimal path from initial location, a_1, to destination, a_5.

Another similar aircraft, *blue*, is also travelling along its planned trajectory at the same rate. This planned trajectory, T_b, follows along locations $[b_1, b_2, b_3, b_4, b_5]$ (without making any maneuvers), which is the optimal path from initial location, b_1, to destination, b_5.

Trajectories T_p and T_b intersect in a_4/b_4, creating a potential collision. This collision is highly likely due to the aircraft's initial speeds and distances from a_4/b_4. In order to avoid a potential collision, either aircraft may maneuver into their alternate trajectory. For *pink* this alternate trajectory, T'_p, is $[a_1, a_2, a_3, a_6, a_7, a_8, a_5]$. For *blue* this alternate trajectory, T'_b, is $[b_1, b_2, b_3, b_6, b_7, b_8, b_5]$. Either aircraft may fail to execute their maneuver.

Similarly to the Storm Avoidance scenario, each aircraft should prefer their initial trajectory in navigating to their destination to minimize cost. However, a maneuver must be executed if a collision is imminent.

Specifications:
Some interesting specifications for analysis of this system:

- Each aircraft reaches its respective destination.

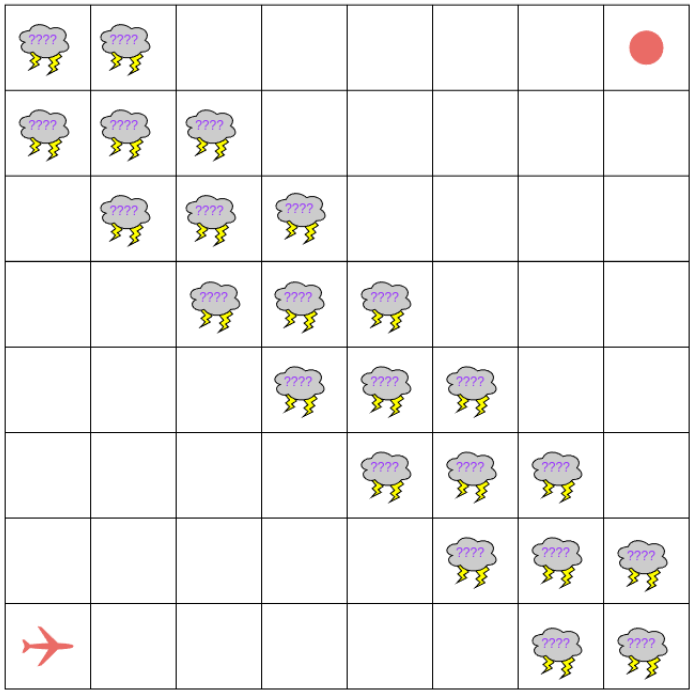
- Each aircraft reaches its respective destination without a collision.
- Each aircraft reaches its respective destination without a collision, executing a maneuver only to avoid an imminent collision.
- A collision eventually occurs in some specific cell $a_x b_y$.
- A collision occurs in any cell.

Similarly to the **Around the Storm** scenario, these specifications must be rewritten into an inquisitive form (*What is the maximum probability of X?*) to allow for probabilistic analysis of these properties.

Scenario 3: Through The Storm

For this scenario we will, again, consider the interaction between a single non-deterministic agent and its deterministic environment.

Visualization:



Description:
An *aircraft* is initially travelling East from position (0,0). In order to reach its destination, the aircraft must navigate from its initial position to its destination, (8,8). This aircraft is capable of turning to travel Northward or Eastward, or continuing in its current direction, but it is forbidden from turning in any other direction.

In order to reach its destination, the aircraft must pass through a *storm* which spans the entire airspace. Due to the severity of the storm, the aircraft may only sustain itself for a limited period of time inside of the storm. Therefore it is critical that the aircraft navigate through the dangerous airspace while minimizing its time spent inside the storm.

Specifications:
Some interesting probabilistic specifications for analysis of this system:

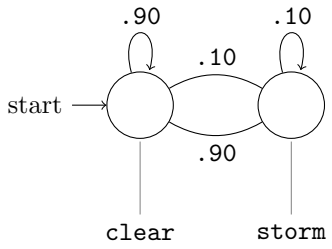
- The aircraft eventually reaches its destination.
- The aircraft eventually reaches its destination without entering a storm.
- The aircraft eventually reaches its destination while minimizing contact with storms below some threshold, ψ.

Similarly to previous scenarios, these specifications must be rewritten into an inquisitive form (*What is the maximum probability of X?*) to allow for probabilistic analysis of these properties.

Modeling the Scenarios

Scenario 1: Around The Storm

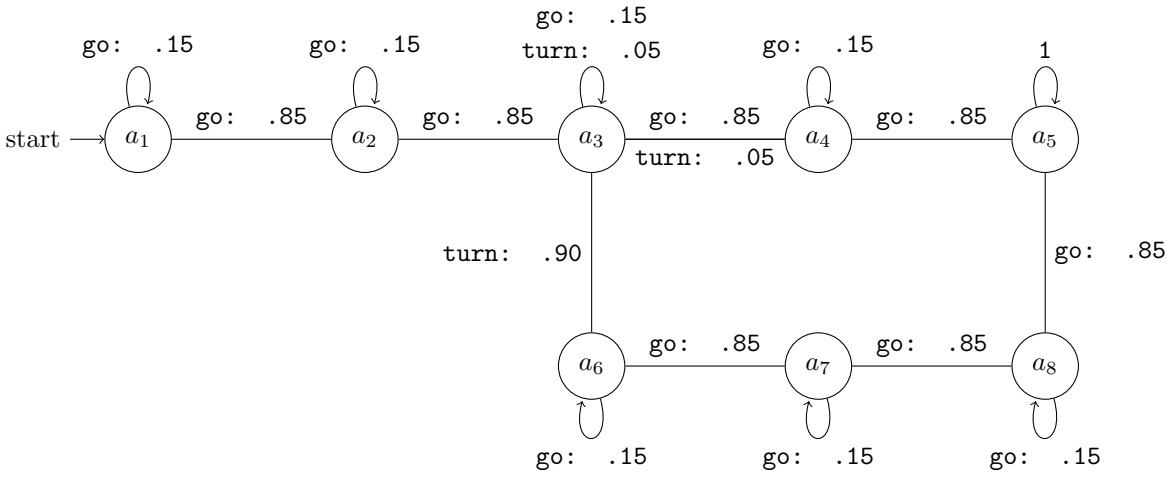
The behavior of the ***storm*** will be modeled by a DTMC, M_c as follows:



This DTMC has two states, corresponding to the status of the airspace in a_4: *clear* or *stormy* skies. The skies are initially *clear* and tend to change state with low probability (0.10).

Aside: This is clearly a gross oversimplification of the behavior of a real storm, however simplifications of this magnitude must be applied in order to create a reasonably sized model a system (in fact, even an entity as simple as this one can create state space explosion, as will be detailed in a later section). Simplifications of this sort will be applied to entities in all scenarios, including both *storms* and *aircraft*.

The behavior of the ***aircraft*** will be modeled by an MDP, M_p as follows:



This MDP has 8 states, representing each of the a_i locations the aircraft may travel through.

The two *non-deterministic actions* available to this agent are `go` and `turn`. `go` is an action whose execution results in high probability (0.85) that the aircraft will move directly from its current state a_i, to the next state a_{i+1}, and a low probability (0.15) that the aircraft will remain in a_i. `go` is enabled in all states except the destination, a_5. `turn` is an action whose execution results in high probability (0.90) of the aircraft altering its trajectory, and a low probability of either remaining in its current position or moving ahead (0.50 for either case). This action is defined only in a_3. Execution of a specific action still leads to a probabilistic state transition (i.e. when executing `turn`, the aircraft may fail to alter its trajectory).

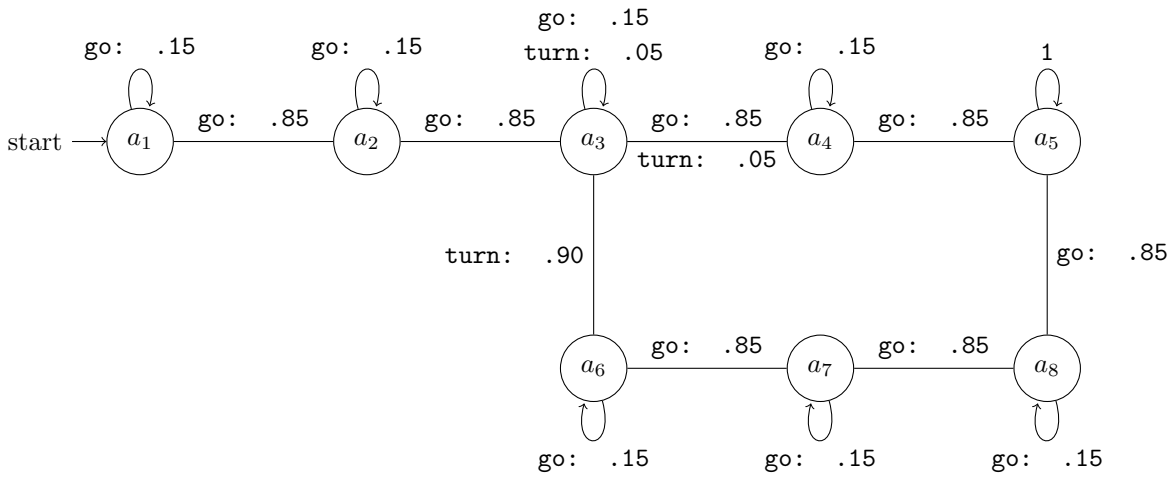
The ***joint behavior*** of the two independent systems, M_c and M_p, can be modeled using a *parallel composition* of the two systems. This parallel composition will be defined as an MDP, M, constructed according to the definition given above in **Terminology**.

For the sake of brevity, a depiction of this parallel composition is not provided.

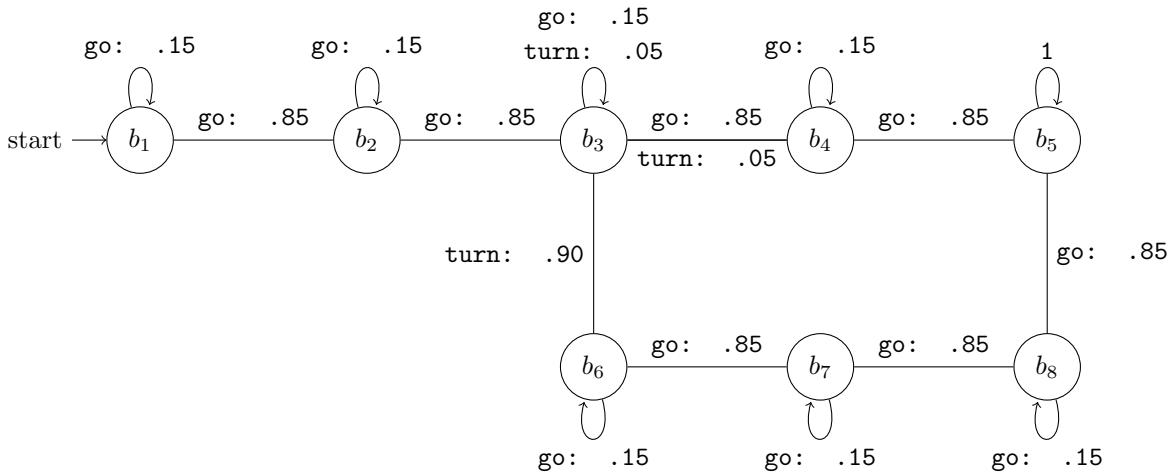
A PRISM implementation of this model is included in `around_the_storm/around_the_storm.pm`. In this model, the *aircraft* uses state variable $a \in [1..8]$ to store its location, and the storm uses state variable $p \in [0..1]$ to encode its presence/absence.

Scenario 2: Air Traffic Control

The behavior of ***pink*** will be modeled by an MDP as follows:



The behavior of ***blue*** will be modeled by an identical MDP as follows:



The *non-deterministic actions* available to ***pink*** and ***blue*** are identical to those of the ***aircraft*** from **Around The Storm** above.

The ***joint behavior*** of these two independent systems can also be modeled using a *parallel composition*. This resulting system will contain 64 states (8 states in *pink* $\times$ 8 states in *blue*). Additionally, the actions available to this composed system will be pairs of actions selected in each subsystem, ex. `pink_go_blue_go`, `pink_turn_blue_go`, etc. (we might think of this composed system as coordinating decision making between the two independent agents). Details of the parallel composition are defined in **Terminology**.

Again, for the sake of brevity the parallel composition is not depicted for this system. A PRISM implementation of this model is included in `air_traffic_control/air_traffic_control.pm`. In this model, each aircraft uses a single state variable, $p \in [1..8]$ for `pink`, $b \in [1..8]$ for `blue`, to encode its location. The full details of the probability transition function is defined in the source file given above, although each is identical to the aircraft of **Around the Storm**.

Scenario 3: Through The Storm

The behavior of the ***storm*** will be modeled by a parallel composition of many independent ***storm cells***. Each cell is modeled by a DTMC, M_i, where each M_i is defined identically to that of the storm in **Around The Storm**. The entire ***storm*** is defined as a parallel composition of its 22 *storm cells*, denoted as $M_0||M_1||\cdots||M_{21}$. The resulting state space, S_{storm}, has $|S_{storm}| = 2^{22}$.

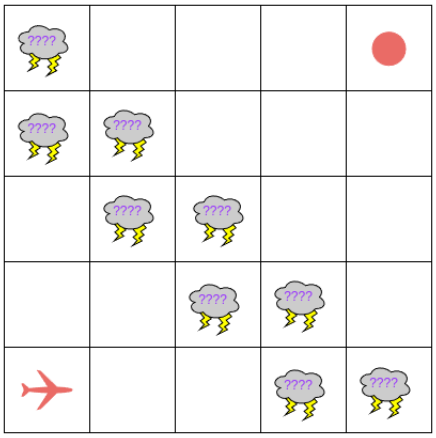
The behavior of the ***aircraft*** will be modeled by an MDP defined as follows:

Each state for the aircraft will be defined by its three state variables, x, its position relative to the x-axis, y, its position relative to the y-axis, and *heading*, the direction the aircraft is facing. Therefore $s \in S_{aircraft}$ takes the form $(x, y, heading)$. For this state space we have $|S_{aircraft}| = 128$.

The aircraft will be capable of executing actions [`go`, `turn_left`, `turn_right`] to navigate through the environment. `go` allows the aircraft to continue in its current direction. `turn_left` allows the aircraft to change from a Eastward trajectory to an Northbound one, and `turn_right` allows the opposite trajectory change. The aircraft will only be capable of travelling right (positive x) or up (positive y) until it reaches the boundary of the environment. Similar to previous examples, execution of a specific action still leads to a probabilistic state transition (i.e. when executing `turn_left`, the aircraft may fail to change its heading).

Problem: The parallel composition required to model the *storm* alone contains 2^{22} states; the final parallel composition that models the joint behavior of the *storm* with the *aircraft* contains 2^{29} states (that's 536,870,912 states!). This is far too many states for my resources, so a **state space reduction** is necessary.

In order to reduce the state space of this system, the space available to the *aircraft* will be reduced to a 5x5 grid with 9 *storm cells* as follows:



With unchanged behavior for the *aircraft* and each *storm cell*, the state space of the final parallel composition is reduced to $2^9 \times 50$ states (down to 25,600).

The PRISM implementations for the original system and its restricted version are included in `through_the_storm/through_the_storm.pm` and `through_the_storm/restricted.pm`, respectively. In each implementation, the aircraft uses two state variables to record its position, x and y, and another to record the direction it is facing, *heading*. Each storm requires only one boolean state variable, $present_{xy}$ representing whether the storm at (x, y) is present. These PRISM files contain the full details of the probability transition function for this system.

Analyzing The Models

Scenario 1: Around The Storm

For **Around The Storm**, some example specifications were mentioned in **Overview of Scenarios**. Those can be translated to LTL expressions as follows:

- The aircraft always eventually reaches its destination
  $\phi_1 = \Box\Diamond(a = 5)$
- The aircraft reaches its destination without entering the storm
  $\phi_2 = \Box(\Diamond(a = 5) \wedge \neg(a = 4 \wedge p = 1))$
- The aircraft reaches its destination without entering the storm while only changing course to avoid a storm
  $\phi_3 = \Box(\Diamond(a = 5) \wedge \neg(a = 4 \wedge p = 1) \wedge (\mathcal{X}(a = 6) \implies (p = 1)))$
- The aircraft enters the storm
  $\phi_4 = \Diamond(a = 4 \wedge p = 1)$

Each of the LTL specifications given above can be used to analyze the system in PRISM model checker via the P_{max} =? specification operator. These specifications can be rewritten as P_{max} =?$[\phi_i]$ to compute the policy that maximizes probability of satisfying the specification. These specifications are given in `around_the_storm/around_the_storm.specs`.

Results from PRISM model checking over the system with these specifications are as follows:

- P_{max} =?$[\phi_1] = 1.0$
- P_{max} =?$[\phi_2] = 0.9726029506462466$
- P_{max} =?$[\phi_3] = 0.8674681337547496$
- P_{max} =?$[\phi_4] = 0.2819704755353184$

PRISM output from these computations is included in `around_the_storm/out.log`.

Scenario 2: Air Traffic Control

For **Air Traffic Control**, some English specifications were mentioned in **Overview of Scenarios**. Those can be translated to LTL expressions as follows:

- Each aircraft reaches its respective destination.
  $\phi_1 = \Diamond(p = 5 \wedge b = 5)$
- Each aircraft reaches its respective destination without a collision.
  $\phi_2 = \phi_1 \wedge \Box\neg((p = 3 \wedge b = 7) \vee (p = 4 \wedge b = 4) \vee (p = 6 \wedge b = 6) \vee p(= 7 \wedge b = 3))$
- Each aircraft reaches its respective destination without a collision, executing a maneuver only to avoid an imminent collision.
  $\phi_3 = \phi_2 \wedge (\mathcal{X}(a = 6) \implies \mathcal{X}(b = 4))$
- A collision eventually occurs in some specific cell $a_x b_y$.
  $\phi_4 = \Diamond(p = x \wedge b = y)$
- A collision occurs in any cell.
  $\phi_5 = \Diamond((p = 3 \wedge b = 7) \vee (p = 4 \wedge b = 4) \vee (p = 6 \wedge b = 6) \vee p(= 7 \wedge b = 3))$

Similarly to the previous scenario, these LTL expressions can be translated into PRISM probabilistic specification form with use of the P_{max} =? operator. These specifications are given in `air_traffic_control/air_traffic_control.specs`.

Results from PRISM model checking over the system with these specifications are as follows:

- P_{max} =?$[\phi_1] = 1.0$
- P_{max} =?$[\phi_2] = 0.9313078531522638$
- P_{max} =?$[\phi_3] = 0.9126234014317556$
- P_{max} =?$[\phi_4] = 0.0613130758058098$, for $x = 7$, $y = 3$
- P_{max} =?$[\phi_5] = 0.5761349858270032$

PRISM output from these computations is included in `aair_traffic_control/out.log`.

Scenario 3: Through The Storm

For **Through the Storm**, English specifications are provided in **Overview of Scenarios**. Those specifications can be translated to LTL expressions as follows:

- The aircraft eventually reaches its destination.
  $\phi_1 = \Diamond(x = 5 \wedge y = 5)$
- The aircraft eventually reaches its destination without entering a storm.
  $\phi_2 = \phi_1 \wedge \neg$`damage`,
  for `damage` defined as $((x = p) \wedge (y = q) \wedge (present_{pq}))$, $(p, q) \in \{(0,4),(0,3),(1,3),(1,2),(2,2),(2,1),(3,1),(3,0),(4,0)\}$
  (Reminder: $present_{pq}$ is the boolean state variable of a storm at (p, q) representing the state of the storm)
- The aircraft eventually reaches its destination while minimizing contact with storms below some threshold, $\psi = 2$.
  $\phi_3 = \phi_1 \wedge \neg($`damage` $\wedge (\mathcal{X}(\mathcal{F}$ `damage`$))$,
  for `damage` defined as $((x = p) \wedge (y = q) \wedge (present_{pq}))$, $(p, q) \in \{(0,4),(0,3),(1,3),(1,2),(2,2),(2,1),(3,1),(3,0),(4,0)\}$
  (Reminder: $present_{pq}$ is the boolean state variable of a storm at (p, q) representing the state of the storm

Similarly to the previous scenarios, these LTL expressions can be translated into PRISM probabilistic specification form with use of the $P_{max} =?$ operator. These specifications are given in `through_the_storm/through_the_storm.specs`.

Results from PRISM model checking over the system with these specifications are as follows:

- $P_{max} =?[\phi_1] = 1.0$
- $P_{max} =?[\phi_2] = 0.4692561170484501$
- $P_{max} =?[\phi_3] = 0.8267056300097458$

PRISM output from these computations is included in `through_the_storm/out.log`.

Results

For each of the models, **Around the Storm**, **Air Traffic Control**, and **Through the Storm**, I have compiled a series of specifications and their maximum probability of satisfaction in their respective models. Each result gives insight into the properties of each system and allows for reasoning about how effective an agent may behave. Of course, all of these models are "toys" in that they bare little connection to real systems. However, the practices used in these analyses to measure the probabilistic properties of each of these models serves as a proof of concept which could be applied to real, more complex, systems.

Interesting results to note include analysis of ϕ_1 in each model. ϕ_1 represented the specification of "agent reaches its destination" in each model. Every analysis of these specifications resulted in probability 1.0. This is due to the implementation of the agents; I did not implement trap states for "crashes" which could cause an agent to fail to reach its goal. Additionally, **Air Traffic Control's** ϕ_5 represents the worst case behavior of this system. This case could represent a malicious actor who has taken control of our air traffic control system, and details exactly how likely they are to succeed in their attack.

Future Work

Extra Experimentation

To extend my research and learning in probabilistic analysis, I also took on another, somewhat unrelated, problem to solve. There is a dice game which is well liked among my family, called "6-5-4" or "Ship-Captain-Crew" for which I was interested in computing an optimal policy. The rules for this game are defined in detail here.

I made an effort to model this game as an MDP which I could analyze with respect to specifications such as "What is the maximum probability of ending with the of score 27?" or "What is the maximum probability of ending with a score greater than 20?" (these PRISM specifications are included in `654/654.specs`). I have included (what I believe to be) a working implementation of this game in `654/654.pm`, however I have not been able to validate my implementation. Additionally, the policy generated by PRISM is not as useful as I had initially hoped, likely because I poorly chose a game which does not involve many player-actions.

Output from PRISM computation over this model for the specifications mentioned just above is contained in `654/out.log`.