UNIVERSITY OF GENEVA

ADVANCED FORMAL TOOLS
14X014

PRISM Case Study - Investor in Market

Authors: Tansen Rahman, Stephane Nguyen

E-mails: Tansen.Rahman@etu.unige.ch Stephane.Nguyen@etu.unige.ch

March 2023

UNIVERSITÉ DE GENÈVE
FACULTÉ DES SCIENCES
Département d'informatique

Contents

1 Introduction

In this report, we implement a case study for the PRISM model checking tool by extending an existing case study showcased on their website [1].

1.1 PRISM

PRISM is a probabilistic model checker, a tool for formal modelling and analysis of systems that exhibit random or probabilistic behaviour. It can be used to analyse systems from many different application domains, such as communication and multimedia protocols, randomised distributed algorithms, security protocols, and biological systems.

1.2 PRISM Usage

For learning how to use PRISM, we invite the reader to their well-documented tutorials and manual. A very useful tool from PRISM, called experiments, allows us, either from the GUI or CLI, to verify/check properties for different model constants/parameters in one go.

1.3 Futures Market Investor

We now discuss the original case study Futures Market Investor [1], found from their case studies list. The case study is based on an investor in a futures market. This example can be considered as a two player game where one player (the investor) tries to maximize their return against the other player (the market) which attempts to minimise this return. The below summarizes their description of the scenario:

"The Investor can make a single investment in 'futures': a fixed number of shares in a specific company that he can reserve on the first day of any month he chooses. Exactly one month later, the shares will be delivered and will collectively have a market value on that day – he or she can then sell the shares. The investor wants to make a reservation such that the sale has maximum value.

The market value v of the shares is a whole number of pounds between €0 and €10 inclusive; it has a probability p of going up by €1 in any month, and 1-p of going down by €1, remaining within bounds. This probability represents short-term volatility.

The probability p itself varies month-by-month in steps of 0.1 between zero and one: p rises with probability 2/3 when v is less than €5, and when v is more than €5 the probability of p falling is 2/3. When v is €5 p rises or falls with equal probability. This represents market trends, where we assume the price of a stock revolves around its "true" value, here €5.

There is a cap c on the value of v, initially €10, which has probability 1/2 of falling by €1 in any month; otherwise it remains where it is. This models a company that is in a slow decline.

If in a given month the investor does not reserve, then at the very next month the market can temporarily bar the investor from reserving. But the market cannot bar the investor in two consecutive months."– *from the Futures Market Investor case study* [1]

Note that even though the case study could be considered a two-player game, their analysis focused on a Markov Decision Process with only one single player, the investor. The market's strategy is fixed in advance. Therefore, we can assume, for the rest of the discussion, that we will be dealing with a single agent/player.

1.4 Motivations

After observing the model and their description, we have some observations on their use of terminology and description of the scenario.

Firstly, this is not a representation of a Future stock, which is a contract to buy a stock at a pre-determined time and price. Instead, the investor is simply buying shares, that he receives with some delay. However, even this does not accurately match the model, as there is no cost related to buying these shares.

Secondly, after the investor receives the stock, there is an imposition that they sell the stock immediately, when more realistically they could wait further before selling them.

A proper scenario matching their model would be as follows. An investor has a number of shares. They would like to sell the shares, but there is a time delay before the sell is executed. They want to time the sell in order to maximize the return.

The original scenario now seems fairly simplistic, with a lot of room for possible extensions. In our implementation, we explore a few differences.

In the original model, the end condition is when the investor performs the single sell of his shares but there's a possibility of never reaching the end condition. We change the end condition to a time-based one, permitting both the investor to continuously act in the market and the investor to leave the market after some time horizon. This of course would not make sense without any further changes, as the scenario has become such that the investor has access to an infinite amount of shares at his disposal. There needs to thus be a cost/limitation of some sort.

One implementation could be to add a cost (or negative reward) for performing the reservation. At first this seems to just become a lower bound for the stock value at which the investor would be willing to reserve his sell. However, if the cost is still considered even when the market bars the investor's sell that month, this variable would be interesting to observe, as it now is part of the dynamic between the two parties. However, PRISM at the moment strictly does not support costs (negative rewards). This idea is thus reserved for future work, since although for Markov Chains negative costs might be easier to implement, for Markov Decision Processes it is not as easy to handle.

Another way of implementing the limitation, one which we settled on, is simply to have a limit of shares that the investor has at hand.

Our last extension is to add a time-based reward to the sell of the shares. This is done by having the sell value accrue interest, compounded monthly. In other words, it's as if the received money is stored and grows with a fixed interest rate. This is in-line with the standard financial philosophy of "money now is worth more than money later".

The original model had 6688 states, and since we added two extra variables given at run-time, we have $6688 \cdot \text{tmax} \cdot \text{max\_stocks}$. **TODO: check this**

2 Background

We detail below some background knowledge that can be helpful. The reader can skip this section if the reader wants to directly dive into our model's case study. However, it's at least recommended to read the section 2.1.5 on our transition graph.

2.1 Models

Verifying the properties of a system requires a model, a property specification language, and property verification algorithms.

A system for which we want to verify some properties can potentially be modeled by a transition system, which is a model. A model can be a simplification of the real system in which we only focus on critical parts.

A state of the system[1] can verify some properties or atomic propositions (e.g. variables in the program taking some particular values).

The system state, under some conditions, can transition to another state. The transition could be randomly triggered or could also be caused by the actions of the users. The transition could also happen every time an instruction is executed in a program.

2.1.1 (Finite) Kripke Structure

A finite Kripke Structure is a transition system defined by $\mathcal{K} = \langle \mathcal{S}, S^0, \rightarrow, AP, \nu \rangle$ where:

- $\mathcal{S}$ finite set of states
- $S^0 \subseteq \mathcal{S}$ non-empty set of initial states
- $\rightarrow \subseteq \mathcal{S} \times \mathcal{S}$ left-total[2] binary relation on $\mathcal{S}$ representing the transitions
- AP the set of atomic propositions
- $\nu : \mathcal{S} \rightarrow \mathcal{P}(AP)$ state labelling function that labels each state with a set of atomic propositions that hold on that state

This transition system is non-deterministic as we don't necessarily know in which state we'll end up next when faced with many successor states. However, given a strategy to choose the next state when faced with many successor states (e.g. user(s) deciding which state to pick), the system becomes deterministic.

[1] or node in the transition system
[2] each state has at least one child

Relation to property specification: From a given state of the Finite Kripke Structure, we can form a tree of all the possible trajectories from that state. Each step down the tree corresponds to the one transition step in the Kripke Structure.

Properties one might want to verify could be to ask whether a set of atomic propositions $\subseteq AP$ hold on all the states in the future from a given state. We might also want to verify whether there exists at least one path/run/trajectory where a set of atomic propositions $\subseteq AP$ hold on all their states.

Specifying such properties can be done with Computation Tree Logic (CTL) formulas (syntax). Verifying a CTL formula (semantics) would give us a set of states satisfying the CTL formula.

TODO: reformulate

2.1.2 Discrete Time Markov Chain, absorbing states, extension with rewards

Intuitively, in a DTMC, if we are at a state x (with some values), we can move to some states (itself, y, z, etc) where those values can be different. Which state is chosen is determined with some probability. We also have states (terminal/absorbing) where we are guaranteed not to move away from, our values are then set in stone.

A Discrete Time Markov Chain (DTMC) [2] is more formally a memory-less discrete-time finite state space stochastic process, meaning a discrete-time finite state stochastic process (sequence of random variables $S_1, S_2, \ldots$) with states satisfying the Markov property:

- $\mathcal{S}$ finite set of states
- $s_0 \in \mathcal{S}$ the initial state
- $P[S_{t+1} = s' | S_t = s]$ the transition probability such that the states satisfy the Markov property:

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_t, \ldots, S_0]$$

If the DTMC is homogeneous, meaning that the probability does not depend on time, we can write the transition probabilities in a state transition matrix.

Absorbing states: We can also introduce the notion of absorbing states as "terminal" states of the system in which there's a probability of 1 to stay in the state and a probability of 0 to exit the state:

$$\sum_{s' \neq s_{\text{absorbing}}} P[S_{t+1} = s' | S_t = s_{\text{absorbing}}] = 0, \quad P[S_{t+1} = s_{\text{absorbing}} | S_t = s_{\text{absorbing}}] = 1$$

This is useful when we want to define a finite time horizon process as in our case study.

Relation to property specification: In addition to our definition, we can define a set of atomic propositions AP and a state labelling function $v : \mathcal{S} \to \mathcal{P}(AP)$ just like in Finite Kripke structures.

Finite Markov Reward Process To make things more interesting, we can add the notion of rewards to Markov Processes (in our case DTMC). A reward can be attached to a state or to a transition.

So our DTMC with rewards can be defined as:

- $\mathcal{S}$ finite set of states
- $s_0 \in \mathcal{S}$ the initial state
- $P[S_{t+1} = s', R_{t+1} = r' | S_t = s]$[3] the probability to end up in a given state s' from state s and receive a particular reward r'.

We can for example estimate how "good" it is to be in a given state (see value functions in reinforcement learning) but it's still impossible to take any action in this environment.

Rewards can also be used to measure the expected number of steps from the initial state to an absorbing state. Note that we will only give a reward of 0 once we reach the absorbing state in order to not create infinitely positive or negative cumulative reward.

[3] Up to our best knowledge, PRISM only supports special cases with deterministic rewards and not stochastic rewards

2.1.3 Finite Markov Decision Process

In a DTMC with or without rewards, all transitions are determined through probability. If that were not the case, however, then some actor (or player) must choose one transition. These transitions are then called actions. A Finite Markov Decision Process is then a DTMC with rewards where some actors must take decisions.

A finite MDP can then be defined with the following:

- $\mathscr{S}$ finite set of states
- $\mathscr{A}$ finite set of actions
- $s_0 \in \mathscr{S}$ the initial state
- $P[S_{t+1} = s', R_{t+1} = r' | S_t = s, A_t = a]$[4] the probability to end up in s' and receive a particular reward r' after taking action a from state s.

Actions introduce non-determinism. To resolve non-determinism, we need to give a policy/strategy $\pi : \mathscr{S} \times \mathscr{A} \to [0,1]$ saying what is the probability to pick an action in a given state. Given a policy, the model falls back to a Markov Reward Process which is deterministic. The goal of an agent is to find a policy/strategy that might maximize expected cumulative future rewards, rewards given by the environment.

One question we might ask is what is the maximum we can earn from the initial state ?

Relation to Futures Market Investor case study: The Futures Market Investor case study was defined as a Markov Decision Process, containing two players, the investor and the market. However, the market was given a fixed strategy in the PRISM model, therefore, removing its non-determinism and merging the actions of the market agent into the environment dynamics. We still have a Markov Decision Process, with the investor being an actor.

Relation to property specification: In addition to our definition, we can define just like before, a set of atomic propositions AP and a state labelling function $v : \mathscr{S} \to \mathscr{P}(AP)$.

2.1.4 Turned-based stochastic game

Turned-based stochastic games are extensions of MDP with multiple agents. They are essentially MDP in which states are attached to a finite number of players. Only one player/agent can act from a given state.

Therefore, a TSG is an MDP with a partition of the state-space $\mathscr{S}$ describing which agent can act from which state.

- $\langle \mathscr{S}, \mathscr{A}, s_0, P[S_{t+1} = s', R_{t+1} = r' | S_t = s, A_t = a] \rangle$ an MDP
- N agents/players and $P = \{\mathscr{S}_i\}_{i=1}^{N}$ a partition of $\mathscr{S}$

The goal for each agent is to learn a good policy $\pi_i : \mathscr{S}_i \times \mathscr{A} \to [0,1]$ where $\mathscr{S}_i$ represents the set of states in which the agent i can act.

This type of model could be used for future works but it also requires a more general property specification language that deals with TSGs [3] as well as a new tool, the PRISM-games tool.

2.1.5 Our transition graph

We will represent, later on, our PRISM model graphically for many reasons such as these:

- Number of states can be extremely huge for our case study even though the PRISM code isn't huge (see Appendix)
- Clarify details one might miss
- Better understand the PRISM model (code)
- Help for extending the model without creating or destroying properties unintentionally

Therefore, it requires some intermediate level of details, sitting between the code (using its PRISM variables) and between a high level view (removing some unnecessary details).

TODO, add explanation on the different arrows, colors in my transition graph

[4] PRISM only supports, up to our best knowledge, some sub-case of $P[S_{t+1} = s', R_{t+1} = r' | S_t = s, A_t = a]$, for example with positive deterministic rewards

2.2 Property Specification

2.2.1 PCTL

Specifying properties of DTMC can be done via PCTL.

TODO

2.2.2 TODO

3 Implementation

3.1 Model

In the following, we represent our PRISM model using a graphical representation of the three main modules (month, investor, market), textual descriptions of other modules (cap, value etc.) and textual description of the reward structures.

The model is parametrized by:

- `p_bar` : the probability of the market to bar
- `interest` : the interest rate from placing the money earned somewhere
- `v_init` : the initial market price
- `tmax` : the number of months
- `max_stocks` : the number of available stocks to sell

Changing these parameters can lead to very different quantitative properties as well as number of states of the MDP.

We can describe most of the model's behavior via our three main modules, explanation on how the other modules work and what are the different reward structures.

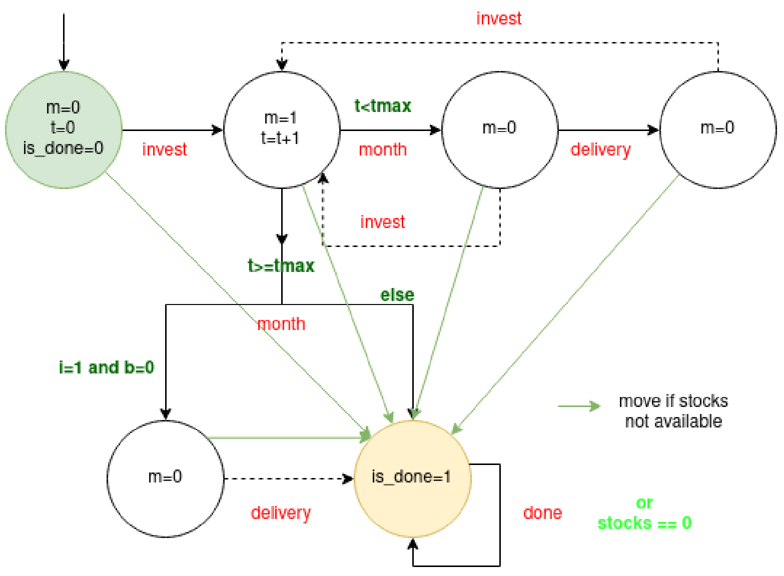
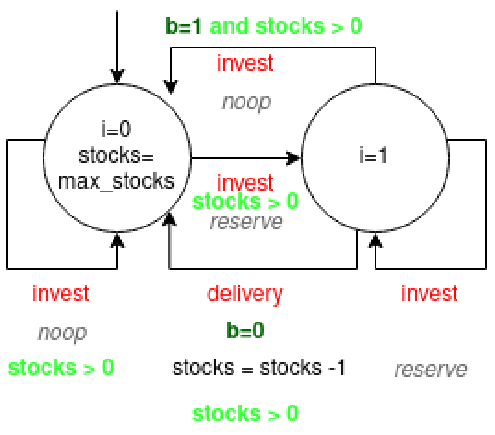


Figure 1: Month Module

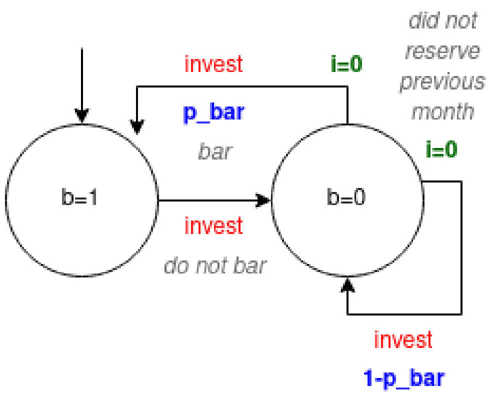
The month module is the backbone module that the investor and barred/market modules will synchronize their actions with. It represents the passing of time with transitions occurring at the start, during, and end of each month. The variable `m` represents this, with `m=1` being the start of the month, `m=0` being the end, so the transition between the two is the "during". We have synchronized actions `[invest]` , `[delivery]` , and `[month]` for the aforementioned time of the month respectively. These will be discussed with their respective modules. The end condition is also monitored in this module, moving to the absorbing state with value `is_done=1` when the investor is out of stocks or when we reach the time limit.

The investor module represents the actions the investor takes, which is to reserve the sell at the beginning of the month, at synchronized action `[invest]` , and to perform the sell at the end of the month, at synchronized action `[delivery]` . Delivery also decrements the number of shares available to the investor, as they were just sold.

The barred/market module represents the actions the market takes, which is to either bar the investor from reserving the sell or not. This is notably independent to the action the investor decides to take at the same time, since they are synchronized on `[invest]` , but the market can only bar if the investor did not invest the previous month. Note however, that the model is a Markov Decision Process, with one single player as the strategy of the market is fixed.



(a) Investor Module



(b) Barred Module (Market module)

The remaining modules, for the value of the share, the probability of the value increasing/decreasing and for the cap[5], update their values as described in subsection 1.3. The evolution of the value of this stock can be represented with a time series. An example is presented below.

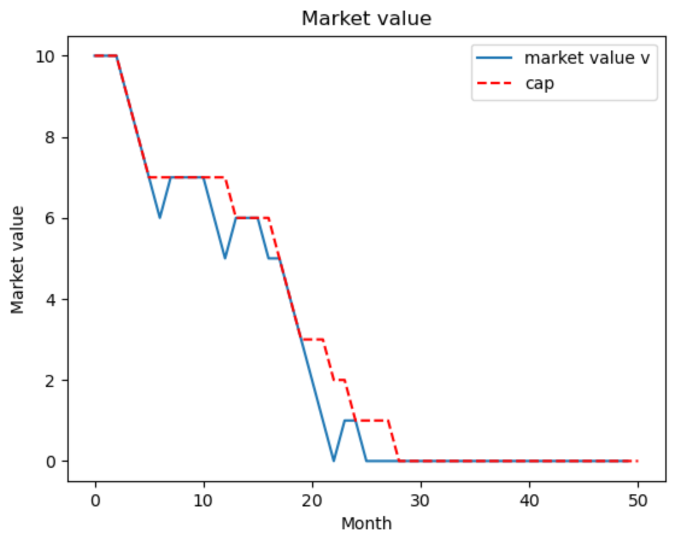


Figure 3: Example Time Series of the Stock Value. Note that the market value is not directly given as reward due to the `interest` parameter

Rewards: The model would be incomplete without reward structures. We define two reward structures:

1. Giving a reward of 1 for every transition except for transitions from absorbing states[6].
2. Giving a reward of $v \cdot (1 + \text{interest})^{\text{tmax}-\text{time}}$ every time a stock is delivered (transition `delivery`) where v is the price of the stock for which the investor sold.

The first reward structure is used to count the number of steps or transitions until reaching an absorbing state. The second reward structure is used to see how much the investor can earn given some policy.

TODO verify explanation formula on accrued interest + reward structures

[5]ceiling of the value of the share that can decrease over time
[6]Transitions from absorbing states come back to absorbing states

4 Value of the Game

TODO

5 Property Verification

In Figure 4a, we verify 4 quantitative properties within `tmax=12` months with no reinvestment into a bank or whatsoever (meaning `interest=0`) and `max_stocks=12` just to make sure we're never out of stocks to sell within the year (even though we need less):

- The first property (top left) shows the maximum cumulative reward/money the investor can get. We can see that the less likely the market bars, the higher the maximum reward was expected.
- The second property (top right) shows the minimum cumulative reward/money the investor can get. It's 0 because the investor can just do `noop` all the time. In other words, the investor never reserves. Note that it's independant of `v_init` and `p_bar`.
- The third property (bottom left) shows the maximum number of steps that the investor can get. We can see that it's independant of `v_init` but decreases as the `p_bar` increases. The maximum number of steps is always when the investor always reserves.
  In particular;
  - When the market never bars, the maximum number of steps is $3 \cdot \text{tmax} = 3 \cdot 12 = 36$.
  - When the market always bars when it can, the maximum number of steps is $2 \cdot \text{tmax}/2 + 3 \cdot \text{tmax}/2 = 12 + 18 = 30$ because the market bars only at second, fourth, sixth, ..., 12 month. Each time the market bars, it removes the delivery action/step.
- The fourth property (bottom right) shows the minimum number of steps that the investor can get. We can see that it's independant of both `v_init` and `p_bar`. The minimum number of steps is when the investor never reserves and the value is $2\text{tmax} = 2 \cdot 12 = 24$. We can see some weird colors due to the slight deviation from 24 and this might be due to the iterative algorithm.

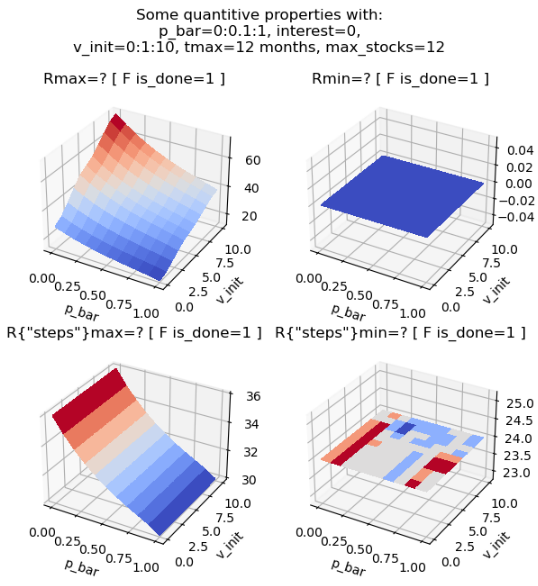
See our graphical representation of the module `month` in Figure 1 for more intuition.
TODO explanation of second graph on the right

6 Conclusion

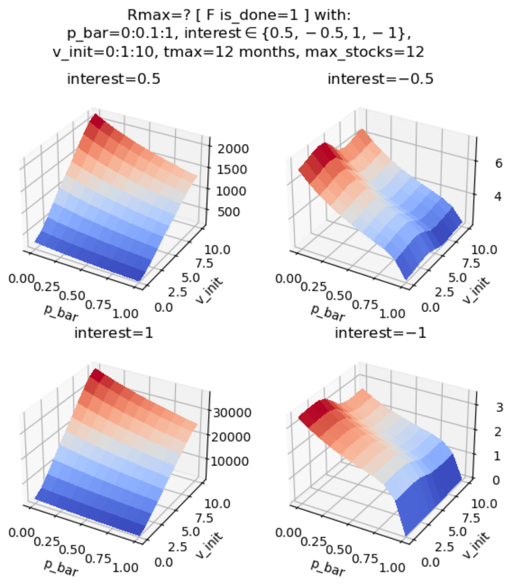
TODO

7 Future Work

TODO



(a)



(b)

Figure 4: Verifying 4 quantitative properties for different parameters of the model. Parameters of the model include: `p_bar` : the probability of the market to bar, `interest` : the interest rate from placing the money earned somewhere, `v_init` : the initial market price, `tmax` : the number of months and `max_stocks` : the number of available stocks to sell

8 PRISM limitations

We show below a non-exhaustive list of some PRISM limitations we encountered:

- No for loops, no lists or compact way to specify many similar transitions. See Bluetooth case study:

```
        [reply]  receiver=2 & y1=0 -> 1/(maxr+1) : (receiver'=3) & (y1'=0) // reply and make random choice
                     + 1/(maxr+1) : (receiver'=3) & (y1'=2*1)
                     + 1/(maxr+1) : (receiver'=3) & (y1'=2*2)
                     + 1/(maxr+1) : (receiver'=3) & (y1'=2*3)
                     + 1/(maxr+1) : (receiver'=3) & (y1'=2*4)
                     + 1/(maxr+1) : (receiver'=3) & (y1'=2*5)
                     + 1/(maxr+1) : (receiver'=3) & (y1'=2*6)
                     + 1/(maxr+1) : (receiver'=3) & (y1'=2*7)
                     [...]
                     + 1/(maxr+1) : (receiver'=3) & (y1'=2*122)
                     + 1/(maxr+1) : (receiver'=3) & (y1'=2*123)
                     + 1/(maxr+1) : (receiver'=3) & (y1'=2*124)
                     + 1/(maxr+1) : (receiver'=3) & (y1'=2*125)
                     + 1/(maxr+1) : (receiver'=3) & (y1'=2*126)
                     + 1/(maxr+1) : (receiver'=3) & (y1'=2*127);
```

- No way to search/find elements (ctrl + f)
- No negative rewards possible for MDP
- No direct way to specify/check a property on a transition based on a transition label: e.g where we only know the transition label from which we can reach a particular state but we don't know the properties that the state should satisfy.

Inherent to the models, state-space explosion is an issue which can be further explored.
Fix references with websites and check for plagiarism

9 Appendix

Below we give the PRISM code for this model.

```
// EXTENDED "INVESTING IN THE FUTURES MARKET" from McIver and Morgan 03
// To finite horizon, multiple investments by Stephane NGUYEN and Tansen RAHMAN

mdp

const double p_bar;
const double interest;
const int v_init;
const int tmax;
const int max_stocks; // number of stocks investor has at the start

// module used to synchronize transitions (time)
module month

    m : [0..1];
    time : [0..tmax] init 0;
    is_done : [0..1] init 0;

    // Increment time, and perform transitions for the start of a new month (invest/bar).
    [invest] (m=0) & (time < tmax) -> (m'=1) & (time'=time+1);

    // Perform transitions that occur during the month (time series).
    [month] (m=1) & (time < tmax) & (is_done=0) -> (m'=0);

    // If last month, go to absorbing state, cashing in the shares if invested previous month
    [month] ((i=0)|((i=1) &(b=1))) & (m=1) & (time >= tmax) & (is_done=0) -> (is_done'=1);
    [month] (i=1) & (b=0) & (m=1) & (time >= tmax) & (is_done=0) -> (m'=0);

    // cash in shares
    [delivery] (m=0) & (time < tmax) & (is_done=0) -> (m'=0);
    [delivery] (m=0) & (time >= tmax) & (is_done=0) -> (is_done'=1);

    // two end conditions
    [done] (is_done=1) -> (is_done'=1);
    [done] (stocks=0) -> (is_done'=1);
endmodule

// the investor
module investor

    stocks: [0..max_stocks] init max_stocks; // number of stocks available to investor
    i : [0..1]; // i=0 no reservation and i=1 made reservation

    [invest] (i=0) -> (i'=0); // do nothing
    [invest] (i=0) & (stocks>0) -> (i'=1); // make reservation
    [invest] (i=1) & (b=1) -> (i'=0); // barred previous month: try again and do nothing
    [invest] (i=1) & (b=1) & (stocks>0) -> (i'=1); // barred previous month: make reservation
    [delivery] (i=1) & (b=0) -> (i'=0) & (stocks'=stocks-1); // cash in shares

endmodule

// market barring
module barred

    // b=0 - not barred and b=1 - barred, initially cannot bar
    b : [0..1] init 1;

    // do not bar this month
    [invest] (b=1) -> (b'=0);
    // bar this month (cannot have barred the previous month), only when investor did not invest last month
    [invest] (b=0) & (i=0) -> p_bar: (b'=1) + (1-p_bar): (b'=0);
    // case of b=0 and i=1 never happens because delivery would update i=0

endmodule

// value of the shares
module value

    v : [0..10] init v_init;

    [month] true -> p/10 : (v'=min(v+1,c)) + (1-p/10) : (v'=min(max(v-1,0),c));

endmodule

// probability of shares going up/down
module probability

    p : [0..10] init 5; // probabilitity is p/10 and initially the probability is 1/2

    [month] (v<5) -> 2/3 : (p'=min(p+1,10)) + 1/3 : (p'=max(p-1,0));
    [month] (v=5) -> 1/2 : (p'=min(p+1,10)) + 1/2 : (p'=max(p-1,0));
    [month] (v>5) -> 1/3 : (p'=min(p+1,10)) + 2/3 : (p'=max(p-1,0));

endmodule

// cap on the value of the shares
module cap

    c : [0..10] init 10; // cap on the shares
    // probability 1/2 the cap decreases
    [month] true -> 1/2 : (c'=max(c-1,0)) + 1/2 : (c'=c);

endmodule

rewards
    // reward from transition [delivery], accrued monthly interest
    [delivery] true : v * pow(1 + interest, tmax - time);
endrewards

rewards "steps"
    true : 1;
endrewards
```

References

[1] PRISM - Case Studies - Futures Market Investor.

[2] Markov chain, May 2023. Page Version ID: 1154313977.

[3] Kwiatkowska, M., Norman, G., and Parker, D. Probabilistic Model Checking and Autonomy. *Annual Review of Control, Robotics, and Autonomous Systems 5*, 1 (2022), 385–410. _eprint: https://doi.org/10.1146/annurev-control-042820-010947.