Technical University of Denmark DTU

Supervisor: Flemming Nielson
Student: Han Yue, s111374

Extending Stochastic Model Checking with Path Rewards

Master Thesis

August 1st 2013
M.Sc.-2013-81

DTU Informatik
Institut for Informatik og Matematisk Modellering

Abstract

Stochastic modeling is widely used for design and analysis of systems in various application domains. Stochastic model checking is an automatic procedure validating whether a stochastic model satisfies a given property which is normally expressed by some temporal logics. One of the state of art model checking tools is PRISM. In the project, *Model Checking Real Life*, conducted during the spring of 2012, a hypothetic ambulance system has been introduced and some PRISM Discrete-time Markov Chain (DTMC) models have been developed based on this system. We also attempted to evaluate some Probabilistic Computation Tree Logic (PCTL) like formulae with path reward properties over these models. However, PRISM only supports state rewards formulae which validate the expected rewards based on a certain set of paths start from a specific state. After some further studies, we believe the concept of path reward properties is also useful to the analyses of other systems.

In order to fulfill this purpose, in this paper, PCTL with state reward formulae is formally extended with path reward formulae as PCTLR. Base on the concept of dynamic programming, algorithms for model checking PCTLR probability path reward formulae over both DTMCs and Markov Decision Processes (MDP) with rewards are developed. Besides, PRISM is extended to support all the above concepts. While PCTLR is incorporated into the PRISM property specification language, we also feel the need to grant PRISM with more flexibility, in this project, an extended filter with "cust" operator is designed and implemented for the extended PRISM. Finally, the extended PRISM is tested from three aspects: correctness, performance and robustness.

Key Words: Path Reward, PCTL, DTMC, MDP and PRISM

Contents

1 Introduction 1
- 1.1 PRISM Model Checker . . . 1
- 1.2 New Requirements for Stochastic Model Checking . . . 2
- 1.3 Limitations of Prism . . . 3
- 1.4 Dynamic Programming . . . 4
- 1.5 Outline . . . 5

2 Extending DTMCs and PCTL with Rewards 6
- 2.1 Discrete-time Markov Chains with Rewards . . . 6
- 2.2 PCTL with State Reward Formulae . . . 8
- 2.3 Extended PCTL with Path Reward Formulae . . . 9
- 2.4 DTMCR Transition Rewards Elimination . . . 10

3 PCTLR Model Checking over DTMCRs 14
- 3.1 Computing $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$ . . . 14
  - 3.1.1 Top-down with Memoization . . . 14
  - 3.1.2 Bottom-up Method . . . 15
- 3.2 Computing $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{C}^{\leq k}])$ . . . 16
  - 3.2.1 To-down with Memoization . . . 16
- 3.3 Computing $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$ . . . 18
  - 3.3.1 Top-down with Memoization and Zero Strongly Connected Components . . . 19
  - 3.3.2 Optimizing Computations of $Pr_{D,s}(Z^{\Phi}_{D,s}\ \mathrm{U}\ s')$ . . . 26
  - 3.3.3 Top-down with Memoization and Zero Connected Components . . . 28
  - 3.3.4 DTMCRs with Negative Rewards . . . 32
- 3.4 Worklist Algorithm . . . 33

4 Extending MDPs and PCTL with rewards 38
- 4.1 Markov Decision Processes with Rewards . . . 38
- 4.2 PCTLR for MDPRs . . . 41
- 4.3 MDPR Action Rewards Elimination . . . 42

5 PCTLR Model Checking over MDPRs 45
- 5.1 Computing $Pr_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$ . . . 45
  - 5.1.1 Top-down with Memoization . . . 45
  - 5.1.2 Bottom-up Method . . . 46
- 5.2 Computing $Pr_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{C}^{\leq k}])$ . . . 47
  - 5.2.1 Top-down with Memoization . . . 47
- 5.3 Computing $Pr_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$ . . . 48
  - 5.3.1 Top-down with Memoization and Zero Strongly Connected Components . . . 49
  - 5.3.2 Optimizing Computations of $Pr^{\max}_{M,s}(Z^{\Phi}_{M,s}\ \mathrm{U}\ s')$ . . . 58
  - 5.3.3 Top-down with Memoization and Zero Connected Components . . . 59
  - 5.3.4 MDPRs with Negative Rewards . . . 60
- 5.4 Worklist Algorithm . . . 62

6 Extending Filters 65
- 6.1 Extended Filter . . . 65
- 6.2 Performance Optimizations for CUST Operator . . . 66

7 Extending PRISM 69
- 7.1 Modifications of Existing PRISM Classes . . . 70
- 7.2 Newly Introduced Classes . . . 72
- 7.3 User Manual of Extended PRISM . . . 73

8 Experiments **76**
8.1 Modeling DTMCRs and MDPRs in PRISM . . . 76
8.2 Extended Filters . . . 78
8.2.1 Correctness Tests . . . 78
8.2.2 Robustness Tests . . . 78
8.3 PCTLR Model Checking over DTMCRs . . . 80
8.3.1 Correctness Tests . . . 80
8.3.2 Performance Tests . . . 82
8.3.3 Robustness Tests . . . 86
8.4 PCTLR Model Checking over MDPRs . . . 86
8.4.1 Correctness Tests . . . 87
8.4.2 Robustness Tests . . . 89

9 Conclusion **90**
9.1 Related work . . . 90
9.2 Acknowledgment . . . 91

Appendix A Model Generators **iii**
A.1 Generator based on the model pattern in Figure 39 . . . iii

Appendix B Model Files **iii**
B.1 DTMCR in Figure 31 . . . iii
B.2 DTMCR in Figure 3 . . . iv
B.3 DTMCR in Figure 8 . . . iv
B.4 DTMCR in Figure 36 . . . v
B.5 DTMCR in Figure 36 . . . v
B.6 DTMCR in Figure 36 . . . vi
B.7 DTMCR in Figure 36 . . . vi
B.8 MDPR in Figure 18 . . . vii
B.9 MDPR in Figure 23 . . . vii
B.10 MDPR in Figure 41 (a) . . . vii
B.11 MDPR in Figure 41 (b) . . . viii

1 Introduction

Stochastic modeling is widely used for design and analysis of systems in various application domains. Different probabilities are used to describe the unreliable or unpredictable behaviors of these systems, e.g. the lost rate of messages in a network communication protocol and the failure rate of a specific node in a distributed system. Stochastic model checking is an automatic procedure validating whether a stochastic model satisfies a given property which is normally expressed by some temporal logics. There are many possible types of models can be used to represent distinguishing features of the same system. For example, Discrete-time Markov Chain[1] (DTMC) are good at extracting the discrete-time behaviors of a given system while Continues-time Markov Chain[2] (CTMC) is a better choice for describing continues-time behaviors, and Markov Decision Process[3] (MDP) extends DTMC with nondeterminism. To grant more expressiveness to models, a reward / cost structure is also considered together as a useful extension. In this paper, we will focus on the two discrete-time model types listed above, DTMC and MDP with reward structures.

1.1 PRISM Model Checker

PRISM[4] is a state of art probabilistic model checker and it is an open source software. A simplified model checking procedure of PRISM is shown in Figure 1. The inputs of the whole procedure are model and properties files where the model file contains both module and reward definitions. The syntax of PRISM model and properties files could be found in [5, 6]. To model check the given properties over the given model, PRISM first parses the model file into the Abstract Syntax Tree (AST) of modules and the AST of reward structure. Then it parses the properties file into the AST of properties. After that, the ASTs of modules, the reward structure and properties will be passed to the core component of PRISM, the computation engine[7], which handles the actual work of model checking and returns the result. Note that the AST of reward structure is optional and it only be needed if some of the properties are reward related.

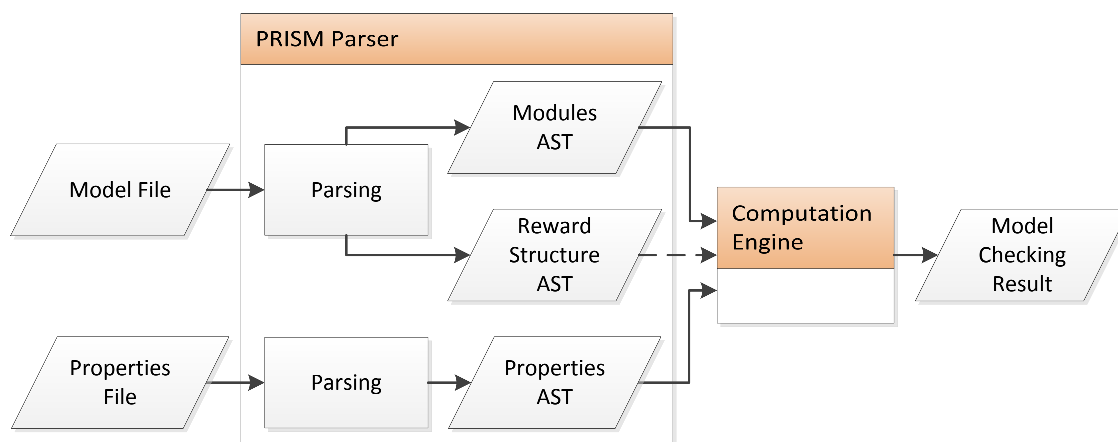


Figure 1: The simplified model checking procedure of PRISM

There are several computation engines available in PRISM, and each engine has its own strengths and weaknesses which depend on the underlying data structures and the algorithms it uses. Though the efficiencies of each engine varies toward different types of PRISM properties over different types of models, if any two engines model check the same property over the same model, the result should be much the same (they may slightly differ from each other due to the approximations during the computation). There are three mature computation engines could be used for model checking DTMCs and MDPs in the main public release (version 4.0.3) of PRISM: the *hybrid*, the *sparse* and the *MTBDD*. All these three engines are partially implemented in C for efficiency reasons while

PRISM itself is a tool implemented in JAVA. Therefore, it gives some difficulties while modifying these three engines.

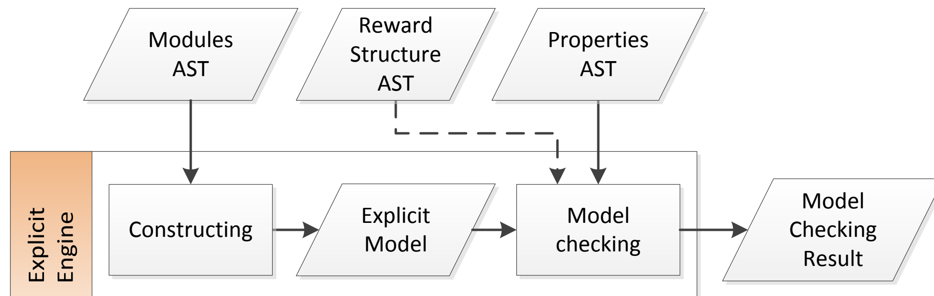


Figure 2: The simplified model checking procedure based on the explicit engine of PRISM

There is another computation engine called the *explicit* which is only published in the development version[1] (beta version 4.1). The simplified model checking procedure based on it is shown in Figure 2. The explicit engine mainly includes the data representation of the explicit model and the explicit model checker. It is implemented in pure JAVA which gives itself the ease of extending desired features. Though it still has performance problems and functionality shortages, which will be covered in the following sections, it is still the best option to test the newly developed algorithms among all these four engines. Therefore, in this paper, all the modifications of PRISM are based on the explicit engine.

1.2 New Requirements for Stochastic Model Checking

The most important motivation of this paper is from a project, *Model Checking Real Life*[8], conducted during the spring of 2012, a hypothetic ambulance system has been introduced and a PRISM DTMC model pattern[2] has been developed based on this system. We attempted to evaluate two types of properties over the models:

- the expected time to resolve an accident once it happens at a given place;
- given a specific time instance during an accident, the probability that the remaining resolution time is greater than x.

They cannot be expressed and evaluated by the conventional model checking techniques. However, these types of properties are also helpful to be evaluated for other systems:

- For a leader election protocol based communication system:
  - the expected times of communications needed to elect a new leader once there is no leader for the system;
  - the probability that the first leader will be elected with less than 10 times of communications to initialize the system.
- For a battery-powered system:
  - the expected working hours once it is fully charged;
  - given a specific time instance of the system, the probability that the energy will run out within 3 hours.

[1]A development version is a beta versions of upcoming releases. The newly introduced features on this version may not be completely implemented, tested or optimized.

[2]A model pattern is a guideline one can follow while constructing a model of a specific instance of the system. Instead of redesign and rebuild the whole model, one can easily modify the size (e.g. numbers of ambulances in an ambulance system) of the given system by adjust the existing model.

To sum it up, the first type of properties has the structure: "the expected *property* once *some conditions satisfy*", and the second type of properties has the structure: "given an instance of the system, the probability that *under some conditions* to *achieve a goal*". Therefore, if there is a way to express and evaluate properties with these structures, we could get more useful information from the analysis of a given system.

1.3 Limitations of Prism

In PRISM, Probabilistic Computation Tree Logic[9] (PCTL) with state reward formulae[3] is used for specifying properties of DTMCs and MDPs. None of the above two types of DTMC properties can be expressed and evaluated by the existing PRISM. However, they can be achieved by modifying the PRISM property syntax and introducing new algorithms.

Recall that a DTMC is a directed graph, each vertex, also called as a state, of the graph is a unique instance of the modeled system, the formal definition of DTMCs will be found in Section 2.1. One can use the following formula to evaluate "the expected *property* once *some conditions satisfy*":

$$\frac{\sum_{s \in Sat(conditions)} P_s \times property(s)}{\sum_{s \in Sat(conditions)} P_s},$$

where $Sat(conditions)$ is a set of states which satisfy the $conditions$, P_s and $property(s)$ denote the steady-state probability[10] and the evaluated property value of state s respectively.

Though PRISM supports the computation of the steady-state probabilities for all the states in the model and the evaluated property values for a set of states, it does not provide the possibility to do any further computation upon these two result sets. In *Modifying PRISM*[11], a project conducted during the autumn of 2012, an operator of PRISM filters[12] has been introduced with the form:

$$\langle filter \rangle ::= \text{filter(savg}, \langle prop \rangle, \langle states \rangle),$$

then the above formula can be expressed as:

filter(savg, property, conditions).

However, as the steady-state probabilities are not applicable to MDPs, the "savg" operator cannot be used for MDPs, and we also fell the needs to grant PRISM with more flexibilities. In this paper, a filter operator "cust" (stands for customization) will be introduced with the form:

$$\langle filter \rangle ::= \text{filter(cust}, \langle customization \rangle, \langle states \rangle),$$

then the above "savg" filter specification can be rewritten as:

filter(cust, sum(@ss * @v) / sum(@ss); v: property, conditions),

which is quite similar to the original formula. The formal definition of the "cust" operator will be given in Section 6.

In PRISM property specifications, only PCTL state reward formulae are supported. A state reward formula computes the expected rewards for a given state which is not sufficient to express the second type of properties introduced above. In an ambulance system, a state reward formula can be used to express the expected remaining resolution time, which may be much less useful than the probability that the remaining resolution time is greater than x, at a specific time instance during an accident. Assume if the accident cannot be resolved within 5 minutes, someone will die due to the lateness of emergency treatments. One can not conclude the ambulance system is satisfied only because the expected remaining resolution time is 4 minutes, as it may be the case that the accident will be

[3]The DTMCs and MDPs represented in PRISM includes both state and transition rewards, the extended PCTL for property specification only contains state reward formulae.

resolved in 2 minutes with the probability 50%, and another half chance it will be resolved in 6 minutes. On the other hand, if we can make sure the probability is 0 that the remaining resolution time is greater than 5, then the system is pretty safe based on the theoretical analysis.

In the project *Modifying PRISM*, PCTL is extended with reachability path reward formulae with the form $\mathrm{P}_{\sim p}[\mathrm{R}^{path}_{\sim' r}[\mathrm{F}\ \Phi]]$. Additionally, three different approaches (pure backward traversal, backward traversal with zero strongly connected component and backward traversal with zero connected component)of model checking have been introduced. In this paper, another two types of path reward formulae (instantaneous and cumulative) will be introduced for PCTL for DTMCs cooperate with the corresponding model checking algorithms. Also to extend this concept to PCTL for MDPs is quite straightforward. As MDPs are extended from DTMCs by adding nondeterminism, thus, instead of evaluate the probability, the second type of property becomes: "given an instance of the system, the minimal / maximal probability that *under some conditions* to *achieve a goal*". The extended PCTL and related model checking algorithms will be covered in later sections.

1.4 Dynamic Programming

Dynamic programming[13] approach solves problems by combining the solutions of subproblems. It first divides the original problem into disjoint subproblems, solves the subproblems recursively, and then uses the solutions of subproblems to solve the original problem. Dynamic programming is useful when subproblems overlap, that is some solutions of subproblems are used more than once. For instance, the original problem is divided into sub problem A and B, subproblem A can be solved based on the solutions of subproblem s_0 and s_1 and B can be solved by using the results of s_0 and s_2. If the normal divide-and-conquer[14] method is used, then subproblem s_0 will be computed twice during the computation of the original problem. In contrast, if the dynamic programming approach is applied, the result of s_0 will be cached once it is computed, then the cached value will be returned whenever the solution of s_0 is required again. By using the dynamic programming approach to implement an algorithm, one can use a reasonable amount of additional memory to improve the asymptotic running time from exponential to polynomial. Most of the model checking algorithms introduced in this paper follow the manner of dynamic programming approach and have the performance bottle-neck of the running time instead of the memory limitation, thereby an introduced algorithm in later sections will apply the dynamic programming approach if a better performance can be achieved.

There are two ways to design a dynamic programming algorithm, and they both yield the same asymptotic running time. The first approach is *top-down with memoization*[4], whose procedure is written recursively in a natural manner of the recursive definition of the original problem. Whenever a result of a subproblem is required, it first checks whether it has been solved before, then it returns the cached value if it does, otherwise it solves the subproblem and caches the solution. The second approach is the *bottom-up method*. It requires pre-sorting the subproblems based on the increasing order of their "sizes" where solving a subproblem only depends on the solutions of smaller subproblems. To apply this approach, first we find out all the subproblems need to be solved before solving the original problem, sort them by size and keep solving the current smallest unsolved subproblem until the original problem is solved.

Though both approaches yield the same asymptotic running time, usually the top-down with memoization approach has a bigger constant due to its overhead of recursive procedure calls. Hence, for a given problem, if one knows all the subproblems, which need to be solved, sorted by the order of size, then it is always better to use the bottom-up approach. Yet if all the subproblems can not be found in an easy manner, the top-down approach may be a better choice as it is always applicable and easy to use when implementing a dynamic programming algorithm. In this paper, all the algorithms

[4]As quoted from [13]: *"This is not a misspelling. The word really is* memoization, *not* memorization. Memoization *comes from* memo, *since the technique consists of recording a value so that we can look it up later."*

will be given with the top-down with memoization approach first and also the bottom-up approach if it is applicable.

1.5 Outline

This paper is structured as follows. First, in Section 2, we begin with background materials of the basic concepts of DTMCs and PCTL. Then the PCTL with state reward formulae is extended as PCTLR. Also two approaches to eliminate transition rewards of a given DTMC is included. Section 3 represents algorithms of PCTLR model checking over DTMCRs with corresponding optimizations. A worklist algorithm is introduced at the end of this section. Start from Section 4, background information of MDPs is provided with the extended PCTLR semantics supporting PCTLR model checking over MDPRs. All related algorithms can be found in Section 5. Section 6 describes the syntax, semantics and performance optimizations of the extended filter property with "cust" operator. The implementation of the extended PRISM with the concepts represented in this paper is introduced in Section 7. In Section 8, we list some test cases used to verify the extended PRISM. Section 9 concludes the paper by comparing the work done in this project with other two papers containing similar interested research areas.

2 Extending DTMCs and PCTL with Rewards

We begin with extending the PCTL with path reward formulae for DTMCs, it is based on the extension of PCTL with state reward formulae introduced in [10, 15]. For the completeness and the readability, brief introductions of the definition of DTMCs with rewards and the PCTL with state reward formulae are summarized from these two papers as Section 2.1 and Section 2.2. Though some notations are slightly modified, the basic concepts are the same.

2.1 Discrete-time Markov Chains with Rewards

In this paper, a Discrete-time Markov Chain with Rewards will be abbreviated as DTMCR[5] for simplicity. It is assumed that both of the state rewards and transition rewards for a given DTMCR are real numbers if no exception is explicitly pointed out. Let AP be a fixed, finite set of atomic propositions.

Definition 1. A DTMC is a 4-tuple $D = (S, \overline{s}, \mathbf{P}, L)$ where:

- S is a finite set of states;
- $\overline{s} \in S$ is an initial state;
- $\mathbf{P} : S \times S \to [0, 1]$ is a transition probability matrix such that for all $s \in S, \sum_{s' \in S} \mathbf{P}(s, s') = 1$;
- $L : S \to 2^{AP}$ is a labeling function mapping each state $s \in S$ to a set of atomic propositions $L(s)$ which are satisfied in that state.

A DTMCR is a 6-tuple $D = (S, \overline{s}, \mathbf{P}, L, r_s, r_t)$ where the first four elements are defined the same as DTMC and:

- $r_s : S \to \mathbb{R}$ is a state reward function;
- $r_t : S \times S \to \mathbb{R}$ is a transition reward function.

The reward a given DTMCR consumes when it moves from a state s to s' is the sum of the state reward of s and the transition reward from s to s'. Let $r_{s \to s'}$ denotes this reward, and it is defined as follows:

$$r_{s \to s'} \stackrel{def}{=} r_s(s) + r_t(s, s').$$ ■

Example 1. Figure 3 shows a DTMCR $D = (S, \overline{s}, \mathbf{P}, L, r_s, r_t)$ for a hypothetical vending machine. The machine stands idle when nobody uses it. It shows the menu once a customer C tries to buy goods on it. C may not buy anything with the probability 0.1 if nothing on the menu interests C and the machine goes back to the idle state. Otherwise C will choose from chocolate and cola both with the probability 0.5. The machine releases the chosen stuff, becomes idle again and waits for the next customer.

For graphical notations used for DTMCRs in this paper, states are drawn as circles with their names on the center, transitions are represented as arrows with associated probabilities labeled on it and the initial state is marked by an incoming arrow with no out state. Besides that, if the transition reward for a specific transition is not 0, it is added after the corresponding transition probability separated by a colon. Similarly if the state reward for a given state is not 0, it is concatenated after the state name with an additional colon.

The DTMCR of this vending machine has six states $S = (s_0, s_1, s_2, s_3, s_4, s_5)$ with the initial state

[5] As we discussed in Section 9.1, the concepts of DTMCR are roughly the same as DMRM.

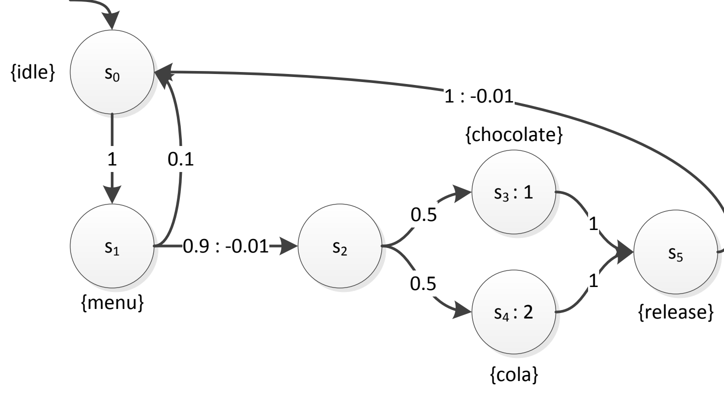


Figure 3: The DTMCR of a hypothetical vending machine

$\overline{s} = s_0$. The transition matrix $\mathbf{P}$ is given by:

$$\mathbf{P} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0.1 & 0 & 0.9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

The labeling functions L maps from S to $2^{\{idle, menu, chocolate, cola, release\}}$:

$$\begin{array}{lll} L(s_0) = \{idle\}, & L(s_1) = \{menu\}, & L(s_2) = \emptyset, \\ L(s_3) = \{chocolate\}, & L(s_4) = \{cola\}, & L(s_5) = \{release\} \end{array}.$$

The rewards denotes the profits of the vending machine. The positive state rewards represent the money earned from selling the corresponding product where for all state $s \in S$:

$$r_s(s) = \begin{cases} 1 & \text{if } s = s_3 \\ 2 & \text{if } s = s_4 \\ 0 & \text{otherwise} \end{cases},$$

and the negative transition rewards indicate the average cost (maintenance fee, electricity, etc) for executing this action , where for all state $s, s' \in S$:

$$r_t(s, s') = \begin{cases} -0.01 & \text{if } (s, s') \in \{(s_1, s_2), (s_5, s_0)\} \\ 0 & \text{otherwise} \end{cases}.$$

The reward D consumes when it transits from state s_1 to s_2 is:

$$r_{s_1 \to s_2} = r_s(s_1) + r_t(s_1, s_2) = 0 + (-0.01) = -0.01,$$

and the reward it consumes by moving from s_3 to s_5 is:

$$r_{s_3 \to s_4} = r_s(s_3) + r_t(s_3, s_4) = 1 + 0 = 1. \qquad \blacksquare$$

An infinite path ω, which represents an execution of a DTMCR $D = (S, \overline{s}, \mathbf{P}, L, r_s, r_t)$, is a non-empty sequence $s_0 s_1 s_2 \cdots$, where $s_i \in S$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 0$. For a (finite or infinite) path ω, we denote the ith state of ω by $\omega(i)$ and the length (number of transitions) of ω by $|\omega|$,

which is always ∞ for any infinite path. A prefix of an infinite path ω is a finite path $\pi = s_0 s_1 \cdots s_n$ that for all $0 \le i \le |\pi|$, $\omega(i) = \pi(i)$. For a finite path π, the last state is denoted as $last(\pi)$. In this paper, let $Path_{D,s}$ and $Path^{fin}_{D,s}$ denote the sets of all infinite and finite paths starting from state s in D, while $Path_D$ and $Path^{fin}_D$ represent the sets of all infinite and finite paths starting from any state in D.

For a finite path $\pi \in Path^{fin}_{D,s}$, the total reward consumed along π is denoted by r_π with the definition:

$$r_\pi \stackrel{def}{=} \sum_{i=0}^{|\pi|-1} r_{\pi(i)\to\pi(i+1)},$$

the probability $\mathbf{P}_D(\pi)$ is defined as follows:

$$\mathbf{P}_D(\pi) \stackrel{def}{=} \begin{cases} 1 & \text{if } |\pi| = 0 \\ \mathbf{P}(\pi(0),\pi(1)) \cdot \mathbf{P}(\pi(1),\pi(2)) \cdots \mathbf{P}(\pi(|\pi|-1), last(\pi)) & \text{otherwise} \end{cases},$$

and the probability measure for a cylinder set[16] $C(\pi) \subseteq Path_{D,s}$ is:

$$Pr_D(C(\pi)) = \mathbf{P}_D(\pi),$$

where:

$$C(\pi) \stackrel{def}{=} \{\omega \in Path_{D,s} \mid \pi \text{ is a prefix of } \omega\}.$$

For an infinite path $\omega \in Path_{D,s}$, the total reward consumed along ω, which is represented by r_ω, is always ∞, and there is no probability measure for any single infinite path.

Example 2. For the DTMCR $D = (S, \overline{s}, \mathbf{P}, L, r_s, r_t)$ shown in Figure 3, let $\pi_1 = s_3$, $\pi_2 = s_1 s_2$ and $\pi_3 = s_1 s_2 s_3$, then:

$$\begin{aligned}
&\pi_1(0) = s_3; \qquad \pi_3(2) = s_3;\\
&|\pi_1| = 0; \qquad |\pi_2| = 1; \qquad |\pi_3| = 2;\\
&Pr_D(C(\pi_1)) = \mathbf{P}_D(\pi_1) = 1;\\
&Pr_D(C(\pi_2)) = \mathbf{P}_D(\pi_2) = 0.9;\\
&Pr_D(C(\pi_2)) = \mathbf{P}_D(\pi_2) = 0.9 \times 0.5 = 0.45.\\
&r_{\pi_1} = r_s(s_3) = 0\\
&r_{\pi_3} = r_s(s_1) + r_t(s_1,s_2) + r_s(s_2) + r_t(s_2,s_3) = -0.01
\end{aligned}$$

■

2.2 PCTL with State Reward Formulae

Probabilistic Computation Tree Logic(PCTL), a probabilistic extension of the Computation Tree Logic[17] (CTL), can be used to write specifications for DTMCR models. Recall that there are two kinds of PCTL formulae, state formulae and path formulae. A PCTL formula is a state formula verifies whether a state satisfies the given condition. To consider the reward properties for a DTMCR, the PCTL is extended by introducing a new type of state formulae $\mathrm{R}^{\mathrm{state}}_{\sim r}[\varphi]$.

Definition 2. The syntax of PCTL with state reward formulae is as follows:

$$\begin{aligned}
\Phi &::= true \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathrm{P}_{\sim p}[\phi] \mid \mathrm{R}^{state}_{\sim r}[\varphi]\\
\phi &::= \mathrm{X}\ \Phi \mid \Phi\ \mathrm{U}^{\le k}\ \Phi \mid \Phi\ \mathrm{U}\ \Phi\\
\varphi &::= \mathrm{I}^{=k} \mid \mathrm{C}^{\le k} \mid \mathrm{F}\ \Phi
\end{aligned}$$

where Φ is a state formula, ϕ is a path formula, φ is a reward formula, a is an atomic proposition, $\sim \in \{<, >, \le, \ge\}$, $p \in [0,1]$, $r \in \mathbb{R}$ and $k \in \mathbb{N}$.

Note that the superscript 'state' of the $R^{state}_{\sim r}$ operator for a *state* formula is not necessarily needed, it is only added for a better clarity to distinguish from the R operator for a *path* formula which will be introduced in the later section. ■

Definition 3. Let $D = (S, \overline{s}, \mathbf{P}, L, r_s, r_t)$ be a DTMCR. For all state $s \in S$, the satisfaction relation $\models$ is defined inductively as:

$$
\begin{aligned}
s \models true &\quad\Leftrightarrow\quad s \in S \\
s \models a &\quad\Leftrightarrow\quad a \in L(s) \\
s \models \neg\Phi &\quad\Leftrightarrow\quad s \not\models \Phi \\
s \models \Phi_1 \wedge \Phi_2 &\quad\Leftrightarrow\quad s \models \Phi_1 \wedge s \models \Phi_2 \\
s \models \mathrm{P}_{\sim p}[\phi] &\quad\Leftrightarrow\quad Pr_{D,s}(\phi) \sim p \\
s \models \mathrm{R}^{state}_{\sim r}[\varphi] &\quad\Leftrightarrow\quad Exp_{D,s}(X_\varphi) \sim r,
\end{aligned}
$$

where:

$$Pr_{D,s}(\phi) \stackrel{def}{=} Pr_{D,s}\{\omega \in Path_{D,s} \mid \omega \models \phi\},$$

and $Exp_{D,s}(X_\varphi)$ denotes the expectation of the random variable $X_\varphi : Path_{D,s} \to \mathbb{R}$. For any path $\omega = s_0 s_1 s_2 \cdots \in Path_{D,s}$:

$$
\begin{aligned}
X_{\mathrm{I}^{=k}}(\omega) &\stackrel{def}{=} r_s(s_k); \\
X_{\mathrm{C}^{\leq k}}(\omega) &\stackrel{def}{=} \begin{cases} 0 & \text{if } k = 0 \\ \sum_{i=0}^{k-1} r_{s_i \to s_{i+1}} & \text{otherwise} \end{cases}; \\
X_{\mathrm{F}\Phi}(\omega) &\stackrel{def}{=} \begin{cases} 0 & \text{if } s_0 \models \Phi \\ \infty & \text{if } \forall i \in \mathbb{N}. s_i \not\models \Phi \\ \sum_{i=0}^{min\{j \mid s_j \models \Phi\}-1} r_{s_i \to s_{i+1}} & \text{otherwise} \end{cases}.
\end{aligned}
$$

The satisfaction relation $\models$ for any valid path $\omega \in Path_{D,}$ is defined as:

$$
\begin{aligned}
\omega \models \mathrm{X}\ \Phi &\quad\Leftrightarrow\quad \omega(1) \models \Phi; \\
\omega \models \Phi\ \mathrm{U}^{\leq k}\ \Psi &\quad\Leftrightarrow\quad \exists i \in \mathbb{N}.(i \leq k \wedge \omega(i) \models \Psi \wedge \forall j < i.(\omega(j) \models \Phi)); \\
\omega \models \Phi\ \mathrm{U}\ \Psi &\quad\Leftrightarrow\quad \exists i \in \mathbb{N}.(\omega(i) \models \Psi \wedge \forall j < i.(\omega(j) \models \Phi)).
\end{aligned}
$$
■

As path formulae, we allow the F operator and the quantitative form of P operator, where F Φ is equivalent to *true* U Φ, $\mathrm{F}^{\leq k}\Phi$ is equivalent to *true* $\mathrm{U}^{\leq k}$ Φ and $\mathrm{P}_{=?}[\Phi]$ is evaluated to the probability value $Pr_{D,s}(\Phi)$.

Example 3. Below are some PCTL formulae examples of the DTMCR shown in Figure 3:

- $\mathrm{P}_{>0.95}[\mathrm{F}^{<6} release]$ - the probability that the first customer buys a product is greater than 0.95;
- $\mathrm{P}_{=?}[\neg chocolate\ \mathrm{U}\ release]$ - the probability of a customer purchasing a product which is not chocolate. ■

2.3 Extended PCTL with Path Reward Formulae

To cover the aspect of the reward properties discussed in Section 1.2, the PCTL should be extended by introducing a new type of path formulae. In later sections of this paper, PCTL with both state reward formulae and path reward formulae will be called as PCTLR[6] for simplicity.

[6] As we discussed in Section 9.1, the concepts of PCTLR are roughly the same as PRCTL with different extended reward properties.

Definition 4. The syntax of the PCTLR path formulae is defined as follows and the syntax of state and reward formulae remain the same as Definition 2:

$$\phi \ ::= \ \mathrm{X}\ \Phi \mid \Phi\ \mathrm{U}^{\leq k}\ \Phi \mid \Phi\ \mathrm{U}\ \Phi \mid \mathrm{R}^{path}_{\sim r}[\varphi]$$

Given a DTMCR $D = (S, \bar{s}, \mathbf{P}, L, r_s, r_t)$, for any state $s \in S$ and any path $\omega \in Path_D$, all the definitions of satisfaction relation $\models$ in Definition 2 are applied when model checking PCTLR over DTMCRs with an additional definition for reward path formulae:

$$\omega \models \mathrm{R}^{path}_{\sim r}[\varphi] \qquad \Leftrightarrow \qquad X_{\varphi}(\omega) \sim r. \qquad \blacksquare$$

As mentioned before for the R^{state} operator, the superscript 'path' for R^{path} operator is not necessarily needed. This is because the R^{path} operator can only be used together with the P operator, on the contrary, the R^{state} operator can not be nested with the P operator. Therefore, the superscript (state or path) of the R operator will be omitted if there is no potential confusion.

Example 4. Below are some PCTLR formulae with path reward operators of the DTMCR shown in Figure 3:

- $\mathrm{P}_{=?}[\mathrm{R}_{>0}[\mathrm{I}^{=6}]]$ - the probability that a customer is just purchasing a product (on either state s_3 or s_4 whose state rewards are both above the 0) after 6 time steps;
- $\mathrm{P}_{=?}[\mathrm{R}_{>=5}[C^{\leq 10}]]$ - the probability that the total cumulative profit gained for the following 10 time step is greater equal than 5;
- $\mathrm{P}_{=?}[\mathrm{R}_{<1}[\mathrm{F}\ release]]$ - the probability that the profit gained for the first sale is less than 1. $\blacksquare$

2.4 DTMCR Transition Rewards Elimination

When model checking reward related PCTLR formulae over DTMCRs, a simpler algorithm may be derived if the model does not have any transition with non-zero transition reward, and actually we can eliminate all non-zero transition rewards for a given DTMCR $D = (S, \bar{s}, \mathbf{P}, L, r_s, r_t)$ without modifying the behaviors of the model. The procedure is described by Algorithm 1.

Algorithm 1 REPLACE-NON-ZERO-TRANSITIONS(D)

```
Input: DTMCR D
Output: D after the modification
 1: for all state s_i ∈ S do
 2:     for all s_j ∈ S where P(s_i, s_j) > 0 and r_t(s_i, s_j) ≠ 0 do
 3:         if P(s_i, s_j) = 1 then
 4:             r_s(s_i) := r_s(s_i) + r_t(s_i, s_j);
 5:             r_t(s_i, s_j) := 0;
 6:         else
 7:             D := ADD-INTERMEDIATE-STATE(D, s_i, s_j);
 8:         end if
 9:     end for
10: end for
11: return D;
```

For each transition (s_i, s_j) with a non-zero transition reward, the transition reward will be merged to the state reward $r_s(s_i)$ if $\mathbf{P}(s_i, s_j)$ is 1, otherwise the transition (s_i, s_j) will be replaced by an intermediate state s_{ij} with two new transitions (s_i, s_{ij}) and (s_{ij}, s_j) as shown in Algorithm 2, where the atomic proposition *auxState* stands for Auxiliary State and it is assumed that for all $s \in S$, $auxState \notin L(s)$.

Algorithm 2 ADD-INTERMEDIATE-STATE(D, s_i, s_j)

Input: DTMCR D, state s_i and state s_j
Output: D after the modification
1: s_{ij} := a newly created state;
2: $S := S \cup \{s_{ij}\}$;
3: $\mathbf{P}(s_i, s_{ij}) := \mathbf{P}(s_i, s_j)$; $\mathbf{P}(s_{ij}, s_j) := 1$; $\mathbf{P}(s_i, s_j) := 0$;
4: $L(s_{ij}) := \{auxState\}$;
5: $r_s(s_{ij}) := r_t(s_i, s_j)$; $r_t(s_i, s_{ij}) := 0$; $r_t(s_{ij}, s_j) := 0$; $r_t(s_i, s_j) := 0$;
6: **return** M;

Let $D' = (S', \bar{s}, \mathbf{P}', L', r'_s, r'_t)$ be the DTMCR after applying the transition reward elimination in this way, the behavior of D is retained in D'. In another words, for any finite path $\pi' \in Path^{fin}_{D'}$ where $\pi(0), last(\pi) \in S$, there is a ono-to-one mapping between π' and a finite path $\pi \in Path^{fin}_{D}$. The probability $\mathbf{P}_{D'}(\pi')$ is equal to $\mathbf{P}_D(\pi)$ and the total reward consumed along π' is also equivalent to r_π. Nevertheless, the model checking results may be affected for some PCTLR formulae. The advantages of this approach are that it requires no changes to the model checking algorithms and has the minimum impact to the model, thus it is suitable for the cases we would like to temporarily modify the model and it is known for sure that the modification will not affect the result.

Example 5. Figure 4 shows the result DTMCR D' after applying Algorithm 1 to the DTMCR D in Figure 3.

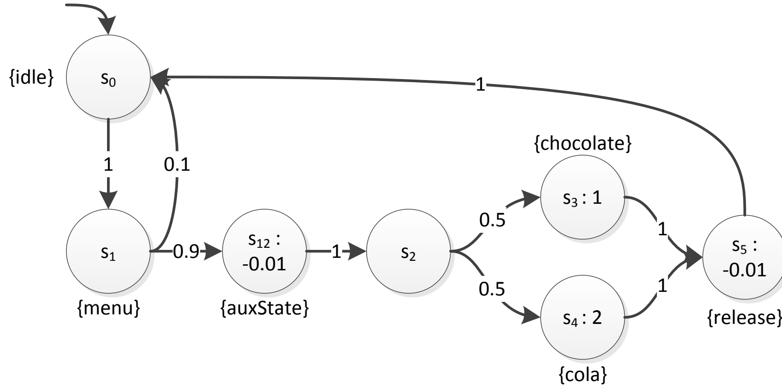


Figure 4: The simple transition rewards elimination result DTMCR

The transition reward $r_t(s_5, s_0)$ is merged to the state reward $r_s(s_5)$ as $\mathbf{P}(s_5, s_0)$ is 1, the transition (s_1, s_2) is replaced by an intermediate state s_{12}, whose state reward equals to $r_t(s_1, s_2)$, with two newly introduced transitions (s_1, s_{12}) and (s_{12}, s_2). It may affect the model checking result of the original model. Model checking $\mathrm{P}_{=?}[\mathrm{F}\ release]$ and $\mathrm{P}_{=?}[\mathrm{R}_{>1}[\mathrm{F}\ release]]$ over both DTMCRs, D and D', will achieve the same results. However, model checking $\mathrm{P}_{=?}[\mathrm{R}_{<0}[\mathrm{I}^{=2}]]$ over D will result 0 which is different from the result 0.9 for D'. ■

In order to not only retain the behaviors of the original DTMCR after eliminating all the non-zero transition rewards, but also return the same model checking result as the original DTMCR for any PCTLR formulae. We can replace all the transitions of the original DTMCR with an intermediate state and two newly introduced transitions as described in Algorithm 3.

After applying the replacements, the modified model retains the behaviors of the original model as the previous approach and all the transitions in the original model become two transitions and an intermediate state. Therefore, the semantics of PCTLR can be modified to ensure model checking any PCTLR formula will give the same result as for the original model.

Algorithm 3 REPLACE-ALL-TRANSITIONS(D)

Input: DTMCR D
Output: D after the modification
1: **for all** state $s_i, s_j \in S$ where $\mathbf{P}(s_i, s_j) > 0$ **do**
2: $\quad D :=$ ADD-INTERMEDIATE-STATE(D, s_i, s_j);
3: **end for**
4: **return** D;

Let $D = (S, \bar{s}, \mathbf{P}, L, r_s, r_t)$ be the original DTMCR and $D' = (S', \bar{s}, \mathbf{P}', L', r'_s, r'_t)$ be the modified DTMCR. First, for all state $s \in S'$, it does not satisfy any state formulae if $s \notin S$. This can be achieved by adding an additional guard to the state satisfaction relation defined before:

$$\begin{array}{rcl} s \models \Phi & \Leftrightarrow & auxState \notin L(s) \wedge s \in S \\ s \models a & \Leftrightarrow & auxState \notin L(s) \wedge a \in L(s) \\ s \models \neg\Phi & \Leftrightarrow & auxState \notin L(s) \wedge s \not\models \Phi \\ s \models \Phi_1 \wedge \Phi_2 & \Leftrightarrow & auxState \notin L(s) \wedge s \models \Phi_1 \wedge s \models \Phi_2 \\ s \models \mathrm{P}_{\sim p}[\phi] & \Leftrightarrow & auxState \notin L(s) \wedge Pr_{D,s}(\phi) \sim p \\ s \models \mathrm{R}^{state}_{\sim r}[\varphi] & \Leftrightarrow & auxState \notin L(s) \wedge Exp_{D,s}(X_\varphi) \sim r. \end{array}$$

The modification of the state satisfaction relations also filters out the need to consider the path satisfaction relation for all the paths starting from a state $s \in S' \setminus S$. This is because a path satisfaction relation will only be considered when model checking a PCTLR state formula with the P operator. Next, for all path $\omega \in Path_{D'}$, where $\omega(0) \in S$, the modified path satisfaction relation is defined as follows:

$$\begin{array}{rcl} \omega \models \mathrm{X}\ \Phi & \Leftrightarrow & \omega(2) \models \Phi; \\ \omega \models \Phi\ \mathrm{U}^{\leq k}\ \Psi & \Leftrightarrow & \exists i \in \mathbb{N}.(i \leq k\ \wedge \omega(2i) \models (\Psi)\ \wedge \forall j < i.(\omega(2j) \models (\Phi))); \\ \omega \models \Phi\ \mathrm{U}\ \Psi & \Leftrightarrow & \exists i \in \mathbb{N}.(\omega(2i) \models \Psi \wedge \forall j < i.(\omega(2j) \models \Phi)). \end{array}$$

Similarly, the modification of the state satisfaction relations filters out the need to define the random variable X_φ for all the paths starting from a state $s \in S' \setminus S$. Finally, for all path $\omega \in Path_{D'}$, where $\omega(0) \in S$, the modified definitions of X_φ is defined as follows:

$$X_{\mathrm{I}^{=k}}(\omega) \stackrel{def}{=} r_s(\omega(2k));$$

$$X_{\mathrm{C}^{\leq k}}(\omega) \stackrel{def}{=} \begin{cases} 0 & \text{if } k = 0 \\ \sum_{i=0}^{2k-1} r_s(\omega(i)) & \text{otherwise} \end{cases};$$

$$X_{\mathrm{F}\Phi}(\omega) \stackrel{def}{=} \begin{cases} 0 & \text{if } s_0 \models \Phi \\ \infty & \text{if } \forall i \in \mathbb{N}. s_i \not\models \Phi \\ \sum_{i=0}^{min\{j \mid s_j \models \Phi\}-1} r_s(\omega(i)) & \text{otherwise} \end{cases}.$$

Example 6. Figure 5 shows the result DTMCR D' after applying Algorithm 3 to the DTMCR D in Figure 3. And for any PCTLR formula, model checking it over D' with the modified semantics will give the same result as model checking it over D with the original semantics. ■

In order to cover more general cases, all the algorithms introduced in later sections will consider both state and transition rewards.

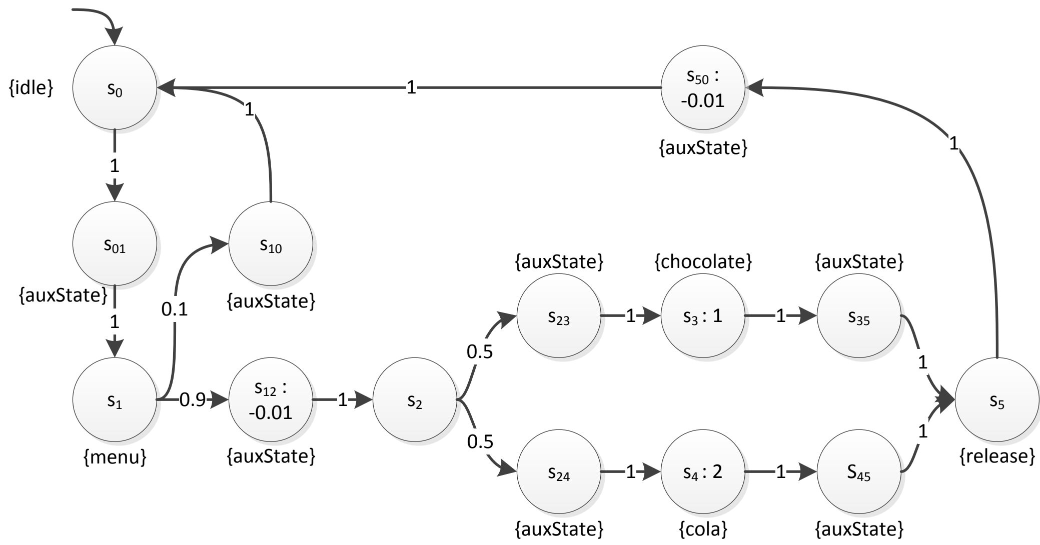


Figure 5: The general transition rewards elimination result DTMCR

3 PCTLR Model Checking over DTMCRs

The model checking algorithm for PCTL over DTMCs was first presented in [9, 18, 19]. It is extended in [10] for PCTL with state reward formulae over DTMCRs. The input of the algorithm are a DTMCR $D = (S, \bar{s}, \mathbf{P}, L, r_s, r_t)$ and a PCTL formula Φ. The output is a set of states $Sat(\Phi) = \{s \in S \mid s \models \Phi\}$. Even though in most cases, one is only interested in whether a subset of states $S' \subseteq S$ (e.g. a subset only contains the initial state $\{\bar{s}\}$) satisfies Φ, the algorithm works by checking every states in D. The summarized algorithm is given as follows:

$$\begin{aligned}
Sat(true) &= S \\
Sat(a) &= \{s | a \in L(s)\} \\
Sat(\neg\Phi) &= S \backslash Sat(\Phi) \\
Sat(\Phi_1 \wedge \Phi_2) &= Sat(\Phi_1) \cap Sat(\Phi_2) \\
Sat(\mathrm{P}_{\sim p}[\phi]) &= \{s \in S \mid Pr_{D,s}(\phi) \sim p\} \\
Sat(\mathrm{R}^{state}_{\sim r}[\varphi]) &= \{s \in S \mid Exp_{D,s}(X_\varphi) \sim r\}.
\end{aligned}$$

The detailed instruction of model checking these formulae can be found in [10] and we will not include them here. In this paper, the PCTL with state reward formulae is extended with path reward formulae as PCTLR, which only introduces a new type of path formulae. Therefore, once we can compute $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\varphi])$ for a given reward formula φ, the existing algorithm will be able to model check PCTLR over DTMCRs without any modifications. In this section, we will discuss how to compute these probabilities given a DTMCR and a path reward formula. The algorithms introduced in this section will be able to handle negative reward values and compute the accurate results if it is not particularly stated.

3.1 Computing $Pr_{D,s}(\mathbf{R}^{path}_{\sim r}[\mathbf{I}^{=k}])$

For a given DTMCR $D = (S, \bar{s}, \mathbf{P}, L, r_s, r_t)$, let $x^k_{D,s,\sim r}$ denotes $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$, the recursive definition of $x^k_{D,s,\sim r}$ is:

$$x^k_{D,s,\sim r} = \begin{cases} 1 & \text{if } k = 0 \wedge r_s(s) \sim r \\ 0 & \text{if } k = 0 \wedge \neg(r_s(s) \sim r) \\ \sum_{s' \in S} \mathbf{P}(s, s') \cdot x^{k-1}_{D,s,\sim r} & \text{otherwise} \end{cases} \tag{1}$$

3.1.1 Top-down with Memoization

One way to solve Equation (1) is applying the top-down with memoization approach of dynamic programming, the algorithm is shown in Algorithm 4 and the auxiliary function PR-I-DTMCR-AUX is shown in Algorithm 5.

Algorithm 4 MEMOIZED-PR-I-DTMCR($D, \sim r, k$)

Input: DTMCR D, reward bound $\sim r$ and step bound k
Output: $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$ for all $s \in S$

1: let p be an empty hash table whose key is composed by a state and a step value;
2: **for all** state $s \in S$ **do**
3: $x_s :=$ PR-I-DTMCR-AUX($D, s, \sim r, k, p$);
4: **end for**
5: **return** $(x_s)_{s \in S}$;

Algorithm 5 PR-I-DTMCR-AUX$(D, s, \sim r, k, p)$

Input: DTMCR D, state s, reward bound $\sim r$, step bound k and hash table p
Output: $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$

```
1: if p(s,k) is not defined then
2:     if k = 0 then
3:         p(s,k) := { 1 if r_s(s) ~ r ; 0 otherwise };
4:     else
5:         p(s,k) := Σ_{s'∈S} P(s,s') · PR-I-DTMCR-AUX(D, s', ~r, k−1, p);
6:     end if
7: end if
8: return p(s,k);
```

Example 7. Let us return to the DTMCR D in Figure 3 and compute $Pr_{D,s_0}(\mathrm{R}^{path}_{>0}[\mathrm{I}^{=3}])$. Let $x^k_{D,s,\sim r}$ denotes $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$, by applying the top-down with memoization approach, we will derive the following equations from top to bottom:

$$
\begin{aligned}
x^3_{D,s_0,>0} &= x^2_{D,s_1,>0} \\
x^2_{D,s_1,>0} &= 0.1 \times x^1_{D,s_0,>0} + 0.9 \times x^1_{D,s_2,>0} \\
x^1_{D,s_0,>0} &= x^0_{D,s_1,>0} \\
x^0_{D,s_1,>0} &= 0 && r_s(s_1) = 0 \\
x^1_{D,s_2,>0} &= 0.5 \times x^0_{D,s_3,>0} + 0.5 \times x^0_{D,s_5,>0} \\
x^0_{D,s_3,>0} &= x^0_{D,s_4,>0} = 1, && r_s(s_4) > r_s(s_3) > 0
\end{aligned}
$$

and the algorithm will solve these equation from bottom to top, we have that:

$$
\begin{aligned}
x^1_{D,s_2,>0} &= 0.5 \times 1 + 0.5 \times 1 = 1 \\
x^1_{D,s_0,>0} &= 0 \\
x^2_{D,s_1,>0} &= 0.1 \times 0 + 0.9 \times 1 = 0.9 \\
x^3_{D,s_0,>0} &= 0.9.
\end{aligned}
$$

Hence the probability $Pr_{D,s_0}(\mathrm{R}^{path}_{>0}[\mathrm{I}^{=3}])$ is 0.9. ■

3.1.2 Bottom-up Method

We can eliminate the recursive calls by applying the bottom-up method approach of dynamic programming to solve Equation (1), the algorithm is shown in Algorithm 6.

Example 8. Consider again the DTMCR D in Figure 3 and compute $Pr_{D,s}(\mathrm{R}^{path}_{>0}[\mathrm{I}^{=3}])$ for all state $s \in S$. Let $x^k_{D,s,>0}$ denotes $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$, the column vector $\underline{x}^k_{D,>0} = \{x^k_{D,s_0,>0}, \cdots, x^k_{D,s_5,>0}\}^{\mathrm{T}}$. By applying the bottom-up method approach with the base case $\underline{x}^0_{D,>0} = \{0,0,0,1,1,0\}^{\mathrm{T}}$, we have that:

$$
\begin{aligned}
\underline{x}^1_{D,>0} &= \mathbf{P} \cdot \underline{x}^0_{D,>0} = \{0,0,1,0,0,0\}^{\mathrm{T}} \\
\underline{x}^2_{D,>0} &= \mathbf{P} \cdot \underline{x}^1_{D,>0} = \{0,0.9,0,0,0,0\}^{\mathrm{T}} \\
\underline{x}^3_{D,>0} &= \mathbf{P} \cdot \underline{x}^2_{D,>0} = \{0.9,0,0,0,0,0\}^{\mathrm{T}}.
\end{aligned}
$$
■

This bottom-up method approach has the same asymptotic running time as the top-down one, the upside is it has no overheads of caching calculated values and recursive calls, where the downside is it may compute some unnecessary values.

Algorithm 6 BOTTOM-UP-PR-I-DTMCR($D, \sim r, k$)

Input: DTMCR D, reward bound $\sim r$ and step bound k
Output: $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$ for all $s \in S$
1: **for all** state $s \in S$ **do**
2: $\quad x_s := \begin{cases} 1 & \text{if } r_s(s) \sim r \\ 0 & \text{otherwise} \end{cases}$;
3: **end for**
4: **for** $i = 1 \to k$ **do**
5: $\quad$ **for all** state $s \in S$ **do**
6: $\quad\quad x'_s := \sum_{s' \in S} \mathbf{P}(s, s') \times x_s$;
7: $\quad$ **end for**
8: $\quad$ **for all** state $s \in S$ **do** $x_s := x'_s$; **end for**
9: **end for**
10: **return** $(x_s)_{s \in S}$

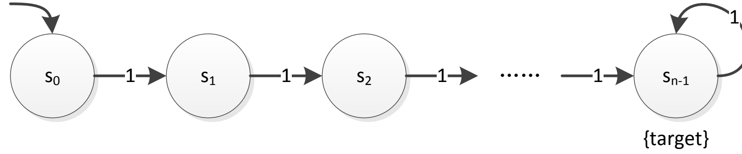


Figure 6: A linear DTMCR

Example 9. Consider the DTMCR D shown in Figure 6. It has n states, for all state s_i where $i < n-1$, $\mathbf{P}(s_i, s_{i+1}) = 1$ and $\mathbf{P}(s_{n-1}, s_{n-1}) = 1$. Assume n is 100, and one wants to compute $Pr_{D,s}(\mathrm{R}^{path}_{>0}[\mathrm{I}^{=99}])$ for all the states. By applying the top-down approach, one needs to compute:

$$\begin{aligned} &Pr_{D,s}(\mathrm{R}^{path}_{>0}[\mathrm{I}^{=99}]) \text{ for } s_0 \text{ to } s_{99} \\ &Pr_{D,s}(\mathrm{R}^{path}_{>0}[\mathrm{I}^{=98}]) \text{ for } s_1 \text{ to } s_{99} \\ &\dots \\ &Pr_{D,s}(\mathrm{R}^{path}_{>0}[\mathrm{I}^{=0}]) \text{ for } s_{99}, \end{aligned}$$

which is $100 + 98 + \cdots + 1 = 5050$ values in total. On the other hand, the bottom-up approach has to compute $100 * 100 = 10000$ values in total. ■

3.2 Computing $Pr_{D,s}(\mathbf{R}^{path}_{\sim r}[\mathbf{C}^{\leq k}])$

For a given DTMCR $D = (S, \bar{s}, \mathbf{P}, L, r_s, r_t)$, let $x^k_{D,s,\sim r}$ denotes $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{C}^{\leq k}])$, the recursive definition of $x^k_{D,s,\sim r}$ is:

$$x^k_{D,s,\sim r} = \begin{cases} 1 & \text{if } k = 0 \wedge 0 \sim r \\ 0 & \text{if } k = 0 \wedge \neg(0 \sim r) \\ \sum_{s' \in S} \mathbf{P}(s, s') \cdot x^{k-1}_{M,s',\sim(r - r_{s \to s'})} & \text{otherwise} \end{cases}. \quad (2)$$

3.2.1 To-down with Memoization

One way to solve Equation (1) is applying the top-down with memoization approach of dynamic programming, the algorithm is shown in Algorithm 7 and the auxiliary function PR-C-DTMCR-AUX is shown in Algorithm 8.

Algorithm 7 MEMOIZED-PR-C-DTMCR($D, \sim r, k$)

Input: DTMCR D, reward bound $\sim r$ and step bound k
Output: $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{C}^{\leq k}])$ for all $s \in S$

1: let p be an empty hash table whose key is composed by a state, a reward and a step value;
2: **for all** state $s \in S$ **do**
3: $x_s :=$ PR-C-DTMCR-AUX($D, s, \sim r, k, p$);
4: **end for**
5: **return** $(x_s)_{s \in S}$;

Algorithm 8 PR-C-DTMCR-AUX($D, s, \sim r, k, p$)

Input: DTMCR D, state s, reward bound $\sim r$, step bound k and hash table p
Output: $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{C}^{\leq k}])$

1: **if** $p(s, r, k)$ is not defined **then**
2: **if** $k = 0$ **then**
3: $p(s,r,k) := \begin{cases} 1 & \text{if } 0 \sim r \\ 0 & \text{otherwise} \end{cases}$;
4: **else**
5: $p(s,r,k) := \sum_{s' \in S} \mathbf{P}(s,s') \cdot$ PR-C-DTMCR-AUX($D, s', \sim (r - r_{s \to s'}), k-1, p$);
6: **end if**
7: **end if**
8: **return** $p(s, r, k)$;

Example 10. Consider the computation of $Pr_{D,s_1}(\mathrm{R}^{path}_{\geq 1}[\mathrm{C}^{\leq 3}])$ and DTMCR D in Figure 3. Let $x^k_{D,s,\sim r}$ denotes $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{C}^{\leq k}])$, by applying the top-down with memoization approach, we will derive the following equations from top to bottom:

$$\begin{aligned}
x^3_{D,s_1,\geq 1} &= 0.1 \times x^2_{D,s_0,\geq 1} + 0.9 \times x^2_{D,s_2,\geq 1.01} \\
x^2_{D,s_0,\geq 1} &= x^1_{D,s_1,\geq 1} \\
x^1_{D,s_1,\geq 1} &= 0.1 \times x^0_{D,s_0,\geq 1} + 0.9 \times x^0_{D,s_2,\geq 1.01} \\
x^0_{D,s_0,\geq 1} &= 0 && \neg(0 \geq 1) \\
x^0_{D,s_2,\geq 1.01} &= 0 && \neg(0 \geq 1.01) \\
x^2_{D,s_2,\geq 1.01} &= 0.5 \times x^1_{D,s_3,\geq 1.01} + 0.5 \times x^1_{D,s_4,\geq 1.01} \\
x^1_{D,s_3,\geq 1.01} &= x^0_{D,s_5,\geq 0.01} \\
x^0_{D,s_5,\geq 0.01} &= 0 && \neg(0 \geq 0.01) \\
x^1_{D,s_4,\geq 1.01} &= x^0_{D,s_5,\geq -0.99} \\
x^0_{D,s_5,\geq -0.99} &= 1, && 0 \geq -0.99
\end{aligned}$$

and the algorithm will solve these equation from bottom to top, we have that:

$$\begin{aligned}
x^1_{D,s_4,\geq 1.01} &= 1 \\
x^1_{D,s_3,\geq 1.01} &= 0 \\
x^2_{D,s_2,\geq 1.01} &= 0.5 \times 0 + 0.5 \times 1 = 0.5 \\
x^1_{D,s_1,\geq 1} &= 0.1 \times 0 + 0.9 \times 0 = 0 \\
x^2_{D,s_0,\geq 1} &= 0 \\
x^3_{D,s_1,\geq 1} &= 0.1 \times 0 + 0.9 \times 0.5 = 0.45.
\end{aligned}$$

Hence the probability $Pr_{D,s_0}(\mathrm{R}^{path}_{\geq 1}[\mathrm{C}^{\leq 3}])$ is 0.45. ■

3.3 Computing $Pr_{D,s}(\mathbf{R}^{path}_{\sim r}[\mathbf{F}\ \Phi])$

Three different approaches of computing this type of probabilities are introduced in [11], all of them are based on backward traversal. In this paper, we will discuss how to compute it with dynamic programming approaches.

For a given DTMCR $D = (S, \bar{s}, \mathbf{P}, L, r_s, r_t)$ and a state PCTLR formula Φ, let $x_{D,s,\sim r}$ denotes $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$, the recursive definition of $x_{D,s,\sim r}$ is:

$$x_{D,s,\sim r} = \begin{cases} 1 & \text{if } s \models \Phi \wedge 0 \sim r \\ 0 & \text{if } s \models \Phi \wedge \neg(0 \sim r) \\ \sum_{s' \in S} \mathbf{P}(s,s') \cdot x_{D,s',\sim(r - r_{s \to s'})} & \text{otherwise} \end{cases} . \tag{3}$$

The problem of Equation (3) is that it cannot always be solved by a dynamic programming algorithm. This is because after unfolding the recursive definition of any given $x_{D,s,\sim r}$, the algorithm requires that there are finite number unfolded equations and no mutual dependences[7] among them, which is not guaranteed by Equation (3). For instance, let $x_{D,s,\sim r}$ denotes $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ target])$ and it is assumed that $0 \sim r_1$ is satisfied. By unfolding the recursive definition of $x_{D,s_1,\sim r_1}$ based on the DTMCR shown in Figure 7 (a), we will have:

$$\begin{aligned} x_{D,s_1,\sim r_1} &= x_{D,s_2,\sim r_1} \\ x_{D,s_2,\sim r_1} &= 1 \end{aligned}$$

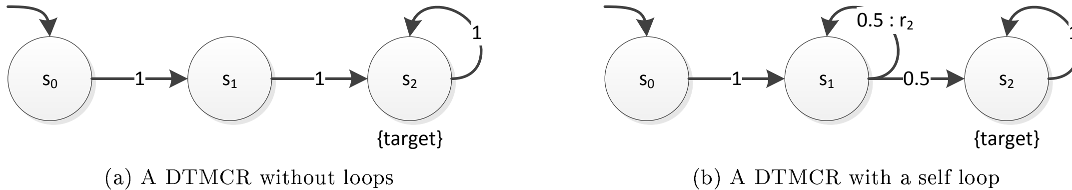
then we can solve the above equations from bottom to top. However, by unfolding the definition based on the DTMCR shown in Figure 7 (b), if $r_2 = 0$, the unfolded equations are:

$$\begin{aligned} x_{D,s_1,\sim r_1} &= 0.5 \times x_{D,s_1,\sim r_1} + 0.5 \times x_{D,s_2,\sim r_1} \\ x_{D,s_2,\sim r_1} &= 1 \end{aligned}$$

where there is a self dependence of $x_{D,s_1,\sim r_1}$, and if $r_2 \neq 0$:

$$\begin{aligned} x_{D,s_1,\sim r_1} &= 0.5 \times x_{D,s_1,\sim(r_1 - r_2)} + 0.5 \times x_{D,s_2,\sim r_1} \\ x_{D,s_1,\sim(r_1 - r_2)} &= 0.5 \times x_{D,s_1,\sim(r_1 - 2r_2)} + 0.5 \times x_{D,s_2,\sim(r_1 - r_2)} \\ x_{D,s_1,\sim(r_1 - 2r_2)} &= 0.5 \times x_{D,s_1,\sim(r_1 - 3r_2)} + 0.5 \times x_{D,s_2,\sim(r_1 - 2r_2)} \\ &\dots \end{aligned}$$

where it ends up with infinitely many unfolded equations. All these two scenarios will prevent the algorithm to define $x_{D,s_1,\sim r_1}$ with only the base cases $x_{D,s_2,\sim r'}$.



(a) A DTMCR without loops (b) A DTMCR with a self loop

Figure 7: Potential problems for computing Equation (3)

In this section, we will discuss how to refine Equation (3) in order to apply dynamic programming algorithms to compute the probability $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$. As we have not found a general approach handling a DTMCR, which has both positive and negative rewards, therefore, it is assumed that the DTMCRs considered in this section only have non-negative rewards.

[7] A mutual dependence exists between x_i and x_j if and only if the value of x_j is needed to compute the value x_i and vice versa. It also includes the case that x_i and x_j are the same.

3.3.1 Top-down with Memoization and Zero Strongly Connected Components

To refine Equation (3), a set of states $S^0_\Phi \subseteq S$ will be introduced first, where $Pr_{D,s}(\mathrm{F}\ \Phi) = 0$ for all state $s \in S^0_\Phi$. Then it can be derived from the definition of the random variable $X_{\mathrm{F}\Phi}$ that $X_{\mathrm{F}\Phi}(\omega) = \infty$ for all path $\omega \in Path_{D,s}$, where $s \in S^0_\Phi$. Therefore, two more base cases can be introduced for the recursive definition of $x_{D,s,\sim r}$:

$$x_{D,s,\sim r} = \begin{cases} 1 & \text{if } s \in S^0_\Phi \wedge \infty \sim r \\ 0 & \text{if } s \in S^0_\Phi \wedge \neg(\infty \sim r) \end{cases}.$$

Algorithm 9 shows one way to find the set S^0_Φ for a given DTMCR D and a PCTLR state formula Φ. It is similar to apply a breadth-first traversal on the transposed graph of D. First, the initial set R contains only the states satisfy Φ. Then the algorithm adds all the states in $S\backslash R$, which can reach a state in R, to R and repeats this procedure until there are no new states can be added to R. Finally, R contains all the states which can reach a state satisfies Φ, and the algorithm will return $S\backslash R$ as S^0_Φ.

Algorithm 9 COMPUTE-S0-DTMCR(D, Φ)

Input: DTMCR D and PCTLR state formula Φ
Output: $S^0_\Phi = \{s \in S \mid Pr_{D,s}(\mathrm{F}\ \Phi) = 0\}$

```
1: R := Sat(Φ);
2: repeat
3:     R' := R;
4:     R := R ∪ {s ∈ S\R | P(s', s) > 0};
5: until R = R'
6: return S\R;
```

Example 11. Let us compute S^0_{idle} for the DTMCR shown in Figure 3. The set R is initialized as $\{s_0\}$. After the first iteration $R = \{s_0, s_1, s_5\}$, then $R = \{s_0, s_1, s_3, s_4, s_5\}$ and $R = \{s_0, s_1, s_2, s_3, s_4, s_5\}$ are the results of the second and the third iteration, respectively. As there are no more states can be added to R, therefore, we have $S^0_{idle} = S\backslash R = \emptyset$. ■

Next, we will discuss how to resolve the problems of having infinitely many unfolded equations and with mutual dependences among them. Both of these two types of problems are due to loops contain only non-base states [8] in a DTMCR. A loop of a given DTMCR $D = (S, \overline{s}, \mathbf{P}, L, r_s, r_t)$ is a finite path π, where $\pi(0) = last(\pi)$ and for all $i, j \in [0, |\pi|), i \neq j \Rightarrow \pi(i) \neq (j)$. For instance, the DTMCR shows in Figure 7 (b) has one self loop $\pi = s_1$ contains only non-base states where the base state is s_2 when computing $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ target])$. There are two types of loops contain only non-base states, $r_\pi > 0$ for one type of loops π and $r_{\pi'} = 0$ for another type of loops π'. The former one will cause the unfolding procedure never ends and the later one will introduce mutual dependences among the unfolded equations.

The infinitely many unfolded equations problem can be resolved with some additional base cases. As there are no negative rewards exist in a DTMCR, thus a probability $x_{D,s,\sim r}$ can only be defined with some probabilities $x_{D,s',\sim r'}$ where $r' \leq r$, and for each fixed value r' there are at most $|S|$ of different $x_{D,s',\sim r'}$. If there are finitely many possible values of r', then there are only finitely many $x_{D,s',\sim r'}$ needed to define $x_{D,s,\sim r}$ which means there are finitely many result equations of unfolding the recursive definition of $x_{D,s,\sim r}$. By the definition of the random variable $X_{\mathrm{F}\Phi}$, for all path $\omega \in Path_{D,s}$, $X_{\mathrm{F}\Phi}(\omega) >= 0$. As a result, we can stop unfolding the recursive definition when

[8] The base states of DTMCR D, which is based on the computation of $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$, are all states belong to $Sat(\Phi) \cup S^0_\Phi$.

facing the following cases based on different reward bound operators:

$$\begin{aligned} x_{D,s,>r} &= 1 && \text{if } r < 0 \\ x_{D,s,\geq r} &= 1 && \text{if } r \leq 0 \\ x_{D,s,<r} &= 0 && \text{if } r \leq 0 \\ x_{D,s,\leq r} &= 0 && \text{if } r < 0. \end{aligned}$$

Because there are finite possible reward values in a DTMCR and there are finite combinations of different reward values. Therefore, there are finitely many possible values $r' \in [0, r]$ of probabilities $x_{D,s',\sim r'}$ used to define $x_{D,s,\sim r}$.

Before move to the solution of the mutual dependences problem, we have to introduce a new concept of a connected component of a DTMCR.

Definition 5. Let $D = (S, \overline{s}, \mathbf{P}, L, r_s, r_t)$ be a DTMCR. A Zero Strongly Connected Component (ZSCC) of D based on a state reward formula Φ is a subset of states $Z_D \subseteq S$ satisfying the following conditions:

- for all state $s \in Z_D$, $s \not\models \Phi$ and $r_s(s) = 0$;
- for every pair of states $s_i, s_j \in Z_D$, where s_i can be the same as s_j, there is a finite path π from s_i to s_j, where $\forall i \in [0, |\pi|].(\pi(i) \in Z_D)$ and $r_\pi = 0$;
- if $|Z_D| = 1$, then the only state s in Z_D must satisfies $\mathbf{P}(s, s) > 0$ and $r_t(s, s) = 0$, this ensures the algorithm will not use the relatively more complicated way to handle states which do not have to;
- Z_D is a maximal subcomponent of D, i.e. there is no distinct ZSCC Z'_D such that if $s \in Z_D$, then $s \in Z'_D$.

In this paper, $D.ZSCC(\Phi)$ denotes a set of all the ZSCCs of D based on Φ and $Z^{\Phi}_{D,s}$ represents the ZSCC contains s where $Z^{\Phi}_{D,s} \in D.ZSCC(\Phi)$. To simplify the formulae introduced in the later part of this paper, a state s belongs to a ZSCC in $D.ZSCC(\Phi)$ is denoted as follows:

$$s \in D.ZSCC(\Phi) \quad \Leftrightarrow \quad \exists Z_D \in D.ZSCC(\Phi).(s \in Z_D). \qquad \blacksquare$$

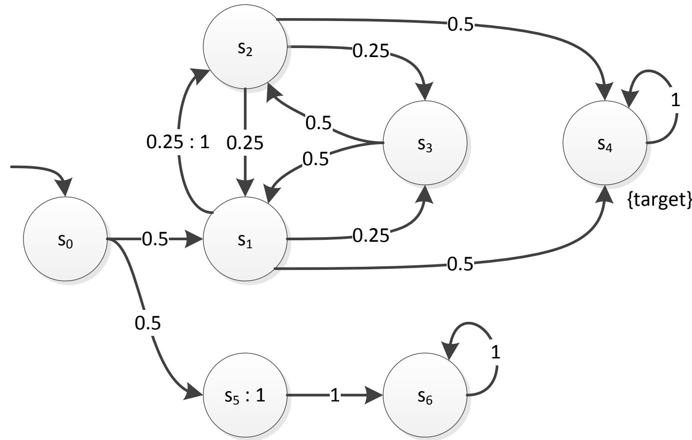


Figure 8: A DTMCR with ZSCCs

Example 12. Based on the PCTLR state formula $\Phi = target$, the DTMCR shown in Figure 8 has two ZSCCs: $\{s_1, s_2, s_3\}$ and $\{s_6\}$. $\blacksquare$

Algorithm 10 DETECT-ZSCC-DTMCR(D, Φ)

Input: DTMCR D and PCTL state formula Φ
Output: $D.ZSCC(\Phi)$

1: Construct the directed graph $G = (V, E)$, where:
- $V := \{s \mid s \not\models \Phi \wedge r_s(s) = 0\}$;
- $E := \{(s_i, s_j) \mid s_i, s_j \in V \wedge \mathbf{P}(s_i, s_j) > 0 \wedge r_t(s_i, s_j) = 0\}$;

2: $D.ZSCC(\Phi) := \emptyset$;
3: **for all** SCC found by calling STRONGLY-CONNECTED-COMPONENT(G) **do**
4:     **if** it is not the case that $|SCC| = 1$ and the only state in SCC does not have a self loop **then**
5:         $D.ZSCC(\Phi) := D.ZSCC(\Phi) \cup \{SCC\}$;
6:     **end if**
7: **end for**
8: **return** $D.ZSCC(\Phi)$

Algorithm 10 provides one way to calculate $D.ZSCC(\Phi)$ based on a given DTMCR D and a PCTLR state formula Φ. This algorithms is based on the STRONGLY-CONNECTED-COMPONENT algorithm from [20]. The algorithm first constructs the directed reduced graph of D based on Φ and pass it as a parameter of STRONG-CONNECTED-COMPONENT. For all the returned strongly connected components, it selects the ones satisfy the definition of ZSCC for DTMCRs and added them into the result set $D.ZSCC(\Phi)$.

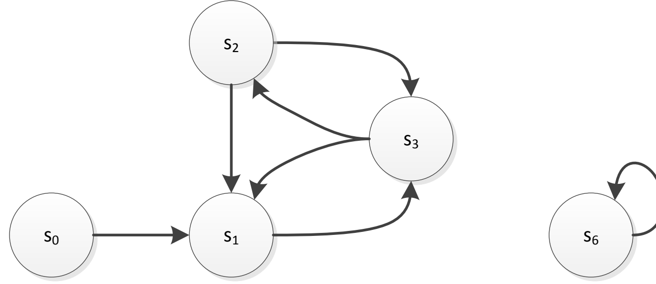


Figure 9: The directed reduced graph of the DTMCR shown in Figure 8

Example 13. To calculate $D.ZSCC(target)$ for the DTMCR D in Figure 8 by calling DETECT-ZSCC-DTMCR($D, target$), the algorithm first constructs the directed reduced graph G which is shown in Figure 9. The state s_4 is not included as it is the target state, s_5 is not included due to its non-zero state reward and the edge (s_1, s_2) is not included since $r_t(s_1, s_2) = 1 \neq 0$. Then we call STRONGLY-CONNECTED-COMPONENT(G) which will return three SCCs: $\{s_0\}$, $\{s_1, s_2, s_3\}$ and $\{s_6\}$. $\{s_0\}$ is not a ZSCC because s_0 is the only state of the SCC and it does not have a self loop in G. Therefore, the result set $D.ZSCC(target) = \{\{s_1, s_2, s_3\}, \{s_6\}\}$. ■

For a given DTMCR $D = (S, \bar{s}, \mathbf{P}, L, r_s, r_t)$ and a ZSCC Z_D based on a PCTLR state formula Φ, let $Z_D.Out$ be a set of states which can be reached directly from Z_D:

$$Z_D.Out \overset{def}{=} \{s \in S \mid \exists s' \in Z_D.((\mathbf{P}(s', s) > 0) \wedge (r_t(s', s) \neq 0 \vee s \notin Z_D))\}.$$

In able to ensure that:

- $Z_D.Out \cap Z_D = \emptyset$;
- $\forall s \in Z_D. \forall \omega \in Path_{D,s}.(\omega \models \mathrm{F}\ \Phi' \Rightarrow X_{\mathrm{F}\Phi'}(\omega) = 0)$, where $s \in \Phi' \Leftrightarrow s \in Z_D.Out$,

we have to replace all the outgoing transitions from some states in Z_M, which have a non-zero reward, with an intermediate state and two new transitions. This procedure is shown in Algorithm 11. Note that this procedure only modifies $Z_D.Out$ but not Z_D itself.

Algorithm 11 ADD-AUX-STATE-DTMCR($D, D.ZSCC(\Phi)$)

Input: DTMCR D and set $D.ZSCC(\Phi)$
Output: D after the modification

```
1: for all Z_D ∈ D.ZSCC(Φ) do
2:     for all state s_i ∈ Z_D do
3:         for all state s_j ∈ S where P(s_i, s_j) > 0 and r_t(s_i, s_j) ≠ 0 do
4:             D := ADD-INTERMEDIATE-STATE(D, s_i, s_j);
5:         end for
6:     end for
7: end for
8: return D;
```

Example 14. Figure 10 shows the model in Figure 8 after replacing the transition (s_1, s_2) with an intermediate state s_{12} and two new transitions (s_1, s_{12}) and (s_{12}, s_2). ■

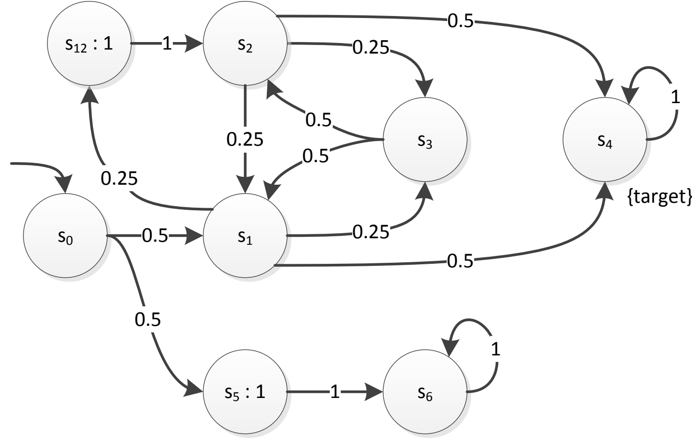


Figure 10: The modified DTMCR of Figure 8

To sum all the above concepts up, for a given DTMCR D and a PCTLR state formula Φ, first we compute $T = Sat(\Phi)$, S^0_Φ and $D.ZSCC(\Phi)$, then the necessary auxiliary states are added, finally, the recursive definition of $x_{D,s,\sim r}$ is refined as follows:

$$x_{D,s,\sim r} = \begin{cases} 1 & \text{if } \begin{pmatrix} (s \in T \wedge 0 \sim r) \\ \vee (s \in S^0_\Phi \wedge \infty \sim r) \\ \vee (\sim \text{is} > \wedge\, r < 0) \\ \vee (\sim \text{is} \geq \wedge\, r \leq 0) \end{pmatrix} & \text{(4a)} \\ 0 & \text{if } \begin{pmatrix} (s \in T \wedge \neg(0 \sim r)) \\ \vee (s \in S^0_\Phi \wedge \neg(\infty \sim r)) \\ \vee (\sim \text{is} < \wedge\, r \leq 0) \\ \vee (\sim \text{is} \leq \wedge\, r < 0) \end{pmatrix} & \text{(4b)} \\ \sum\limits_{s' \in Z^\Phi_{D,s}.Out} Pr_{D,s}(Z^\Phi_{D,s} \text{ U } s') \cdot x_{D,s',\sim r} & \text{if } s \in D.ZSCC(\Phi) & \text{(4c)} \\ \sum\limits_{s' \in S} \mathbf{P}(s,s') \cdot x_{D,s',\sim(r - r_{s \to s'})} & \text{otherwise} & \text{(4d)} \end{cases}$$

where:

$$Pr_{D,s}(Z^\Phi_{D,s} \text{ U } s') \stackrel{def}{=} Pr_{D,s}\{\Phi_1 \text{ U } \Phi_2\},$$

where for any state $s'' \in S$:

$$
\begin{aligned}
s'' \models \Phi_1 \quad &\Leftrightarrow \quad s'' \in Z^{\Phi}_{D,s} \\
s'' \models \Phi_2 \quad &\Leftrightarrow \quad s'' = s'.
\end{aligned}
$$

Let us take a detailed look at the above recursive definition:

- Case (4a) and (4b) are the base cases where $x_{D,s,\sim r}$ can be defined with the fixed value 1 or 0.
- Case (4c) is the case when the subscript s of $x_{D,s,\sim r}$ belongs to a ZSCC. It ensures that for a given $x_{D,s,\sim r}$ where $s \in Z^{\Phi}_{D,s}$, it does not need any $x_{D,s',\sim r}$, where $s' \in Z^{\Phi}_{D,s}$, to define it. If it does, then $Z^{\Phi}_{D,s}$ is not a maximal subcomponent as there is a loop π, where $r_{\pi} = 0$ and $\exists i \in [0, |\pi|].(\pi(i) \notin Z^{\Phi}_{D,s})$ and we can construct a bigger ZSCC by adding all those states appeared on π to the existing ZSCC $Z^{\Phi}_{D,s}$. This is contradictory to the definition of ZSCCs. Therefore, this case excludes the possibility of mutual dependence problems.
- Case (4d) is the same as the recursive case of Equation (3).
- Note that there are some cases have intersections. Even though take any of the intersected cases will not affect the final result, consider the cases with the order from (4a) to (4d) will give a better performance. For instance, $x_{D,s,\sim r}$ can satisfy ($\sim$ is $< \wedge\ r \leq 0$) and $s \in D.ZSCC(\Phi)$ at the same time, and taking the former case requires no further computation while the later one does.

With the purpose of simplifying the calculations of Equation (4c). All the ZSCCs of a DTMCR based on a PCTLR state formula can be further split into several disjoint subsets and different approaches can be applied to each subset for better performances.

Definition 6. For all the ZSCCs $Z_D \in D.ZSCC(\Phi)$ of a given DTMCR D and a state formula Φ, they can be split into following disjoint subsets:

- if $|Z_D.Out| = 1$, then $Z_D \in D.ZSCC^{1o}(\Phi)$ where $1o$ stands for one out state;
- else if $|Z_D| = 1$, then $Z_D \in D.ZSCC^{1s}(\Phi)$ where $1s$ stands for the ZSCC contains only one state;
- the rest ZSCCs belong to $D.ZSCC^{n}(\Phi)$.

A ZSCC set with multiple superscripts separated by commas represents the union of the corresponding subsets. For instance, $D.ZSCC^{1o,n}(\Phi) = D.ZSCC^{1o}(\Phi) \cup D.ZSCC^{n}(\Phi)$. ■

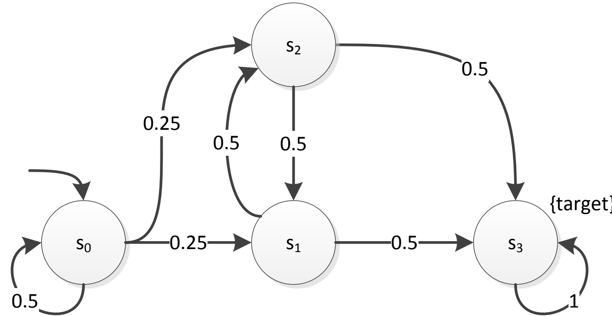


Figure 11: DTMCRs contains special types ZSCCs

Example 15. Consider the PCTLR state formula $\Phi = target$, for the MDPR M in Figure 11:

$$M.ZSCC^{1o}(\Phi) = \{\{s_1, s_2\}\}$$
$$M.ZSCC^{1s}(\Phi) = \{\{s_0\}\},$$

where $|Z^{\Phi}_{D,s_1}.Out| = |\{s_3\}| = 1$ and $|Z^{\Phi}_{D,s_0}| = \{s_0\} = 1$. ■

For a given DTMCR D and a ZSCC $Z^{\Phi}_{D,s} \in D.ZSCC(\Phi)$:

- If $Z^{\Phi}_{D,s} \in D.ZSCC^{1o}(\Phi)$, let $\{s'\} = Z^{\Phi}_{D,s}.Out$. As whenever the model goes into $Z^{\Phi}_{D,s}$, it will always leave $Z^{\Phi}_{D,s}$ to s' with probability 1, therefore, $x_{D,s,\sim r}$ can be defined with the following formula:
$$x_{D,s,\sim r} = Pr_{D,s}(Z^{\Phi}_{D,s} \text{ U } s') \cdot x_{D,s',\sim r} = x_{D,s',\sim r}.$$

- If $Z^{\Phi}_{D,s} \in D.ZSCC^{1s}(\Phi)$, once the model enters the ZSCC, it will leave the ZSCC through one of the state $s' \in Z^{\Phi}_{D,s}.Out$ with the probability:
$$\frac{\mathbf{P}(s, s')}{\sum_{s' \in Z^{\Phi}_{D,s}.Out} \mathbf{P}(s, s')},$$
thus $x_{D,s,\sim r}$ can be redefined in this case as follows:
$$x_{D,s,\sim r} = \frac{\sum_{s' \in Z^{\Phi}_{D,s}.Out} \mathbf{P}(s, s') \cdot x_{D,s',\sim r}}{\sum_{s' \in Z^{\Phi}_{D,s}.Out} \mathbf{P}(s, s')}.$$

For the above two cases, they both eliminate all computations of $Pr_{D,s}(Z^{\Phi}_{D,s} \text{ U } s')$ for all $s' \in Z^{\Phi}_{D,s}.Out$. The former one introduces no new computations and the only new computations the latter one introduces are some additions and one division which is faster than using the original definition.

Now we can apply top-down with memoization approach of dynamic programming to solve Equation (4), the algorithm is shown in Algorithm 12 and the auxiliary function Pr-F-DTMCR-Aux is shown in Algorithm 13.

Algorithm 12 Memoized-Pr-F-DTMCR($D, \sim r, \Phi$)

Input: DTMC D, reward bound $\sim r$ and PCTLR state formula Φ
Output: $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$ for all $s \in S$

1: $T := Sat(\Phi)$;
2: S^0_{Φ} := Compute-S0-DTMCR(D, Φ);
3: $D.ZSCC(\Phi)$:= Detect-ZSCC-DTMCR(D, Φ);
4: D := Add-Aux-State-DTMCR($D, D.ZSCC(\Phi)$);
5: let p be an empty hash table whose key is composed by a state and a reward value;
6: **for all** state $s \in S$ **do**
7: x_s := Pr-F-DTMCR-Aux($D, S^0_{\Phi}, s, \sim r, T, p$);
8: **end for**
9: **return** $(x_s)_{s \in S}$

Example 16. To compute $Pr_{D,s_0}(\mathrm{R}^{path}_{<1}[\mathrm{F}\ target])$ for the DTMCR D in Figure 10, we have:

$$T = Sat(\Phi) = \{s_4\}$$
$$S^0_{target} = \{s_5, s_6\}$$
$$D.ZSCC(\Phi) = \{\{s_1, s_2, s_3\}, \{s_6\}\}.$$

Algorithm 13 PR-F-DTMCR-AUX$(D, S^0_\Phi, s, \sim r, T, p)$

Input: DTMCR D, set of states S^0_Φ, state s, reward bound $\sim r$, set of target states T and hash table p
Output: $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$

1: **if** $p(s,r)$ is not defined **then**
2: **if** $s \models \Phi$ **then**
3: $p(s,r) := \begin{cases} 1 & \text{if } 0 \sim r \\ 0 & \text{otherwise} \end{cases}$;
4: **else if** $s \in S^0_\Phi$ **then**
5: $p(s,r) := \begin{cases} 1 & \text{if } \infty \sim r \\ 0 & \text{otherwise} \end{cases}$;
6: **else if** $(\sim \in \{>, \geq\} \wedge\ r \leq 0)$ **then**
7: $p(s,r) := 1$;
8: **else if** $(\sim \text{ is } < \wedge\ r \leq 0) \vee (\sim \text{ is } \leq \wedge\ r < 0)$ **then**
9: $p(s,r) := 0$;
10: **else if** $s \in D.ZSCC(\Phi)$ **then**
11: **if** $s \in D.ZSCC^{1o}(\Phi)$ and let $\{s'\} = Z^\Phi_{D,s}.Out$ **then**
12: $p(s,r) :=$ PR-F-DTMCR-AUX$(D, S^0_\Phi, s', \sim r, T, p)$;
13: **else if** $s \in D.ZSCC^{1s}(\Phi)$ **then**
14: $p(s,r) := \frac{\sum_{s' \in Z^\Phi_{D,s}.Out} \mathbf{P}(s,s') \cdot \text{PR-F-DTMCR-AUX}(D, S^0_\Phi, s', \sim r, T, p)}{\sum_{s' \in Z^\Phi_{D,s}.Out} \mathbf{P}(s,s')}$.;
15: **else**
16: $p(s,r) := \sum_{s' \in Z^\Phi_{D,s}.Out} Pr_{D,s}(Z^\Phi_{D,s}\ \mathrm{U}\ s') \cdot$ PR-F-DTMCR-AUX$(D, S^0_\Phi, s', \sim r, T, p)$;
17: **end if**
18: **else**
19: $p(s,r) := \sum_{s' \in S} \mathbf{P}(s,s') \cdot$ PR-F-DTMCR-AUX$(D, S^0_\Phi, s', \sim (r - r_{s \to s'}), T, p)$;
20: **end if**
21: **end if**
22: **return** $p(s,r)$;

Let $x_{D,s,\sim r}$ denotes $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ target])$, by applying the top-down with memoization approach, we will derive the following equations from top to bottom:

$$
\begin{aligned}
x_{D,s_0,<1} &= \frac{1}{2}x_{D,s_1,<1} + \frac{1}{2}x_{D,s_5,<1} \\
x_{D,s_1,<1} &= Pr_{D,s_1}(Z^{target}_{D,s_1}\ \mathrm{U}\ s_{12}) \times x_{D,s_{12},<1} + Pr_{D,s_1}(Z^{target}_{D,s_1}\ \mathrm{U}\ s_4) \times x_{D,s_4,<1} && s_1 \in Z^{target}_{D,s_1} \\
&= \frac{7}{23}x_{D,s_{12},<1} + \frac{16}{23}x_{D,s_4,<1} \\
x_{D,s_{12},<1} &= x_{D,s_2,<0} \\
x_{D,s_2,<0} &= 0 && \sim \text{is} < \wedge\ 0 \le 0 \\
x_{D,s_4,<1} &= 1 && s_4 \in T \wedge 0 < 1 \\
x_{D,s_5,<1} &= 0 && s_5 \in S^0_{target} \wedge \neg(\infty < 1)
\end{aligned}
$$

and the algorithm will solve these equation from bottom to top, we have that:

$$
\begin{aligned}
x_{D,s_{12},<1} &= 0 \\
x_{D,s_1,<1} &= \frac{7}{23} \times 0 + \frac{16}{23} \times 1 = \frac{16}{23} \\
x_{D,s_0,<1} &= \frac{1}{2} \times \frac{16}{23} + \frac{1}{2} \times 0 = \frac{8}{23}.
\end{aligned}
$$

Hence the probability $Pr_{D,s_0}(\mathrm{R}^{path}_{<1}[\mathrm{F}\ target])$ is $\frac{8}{23}$. ■

3.3.2 Optimizing Computations of $Pr_{D,s}(Z^{\Phi}_{D,s}\ \mathbf{U}\ s')$

To compute $Pr_{D,s}(Z^{\Phi}_{D,s}\ \mathrm{U}\ s')$ for a given DTMCR $D = (S, \bar{s}, \mathbf{P}, L, r_s, r_t)$, we can reuse the algorithms for computations of $Pr_{D,s}(\Phi_1\ \mathrm{U}\ \Phi_2)$ based on the definition of $Pr_{D,s}(Z^{\Phi}_{D,s}\ \mathrm{U}\ s')$. Normally, the probabilities $Pr_{D,s}(\Phi_1\ \mathrm{U}\ \Phi_2)$ for all state $s \in S$ are obtained as the unique solution of the following linear equation system:

$$
Pr_{D,s}(\Phi_1\ \mathrm{U}\ \Phi_2) = \begin{cases} 0 & \text{if } s \in S^0_{\Phi_1} \\ 1 & \text{if } s \models \Phi_2 \\ \sum_{s' \in S} \mathbf{P}(s,s') \cdot Pr_{D,s'}(\Phi_1\ \mathrm{U}\ \Phi_2) & \text{otherwise} \end{cases}.
$$

One can either solve it by a direct method (e.g. Gaussian elimination) or an iterative method (e.g. Jacobi). In practice, an iterative method is more likely to be used as it has a much better scalability and the accuracy is sufficient in most cases. However, because a ZSCC Z_D is normally much smaller than S, i.e. $|Z_D| << |S|$ and only probabilities of states within Z_D may change through each iteration. Thus an iterative method which updates the probabilities for all states through each iteration, particularly Gauss-Seidel used by PRISM, will end up doing a lot of unnecessary computations. Therefore, we introduce a faster iterative method which only updates the state whose probability will be changed to optimize the performances of computing $Pr_{D,s}(Z^{\Phi}_{D,s}\ \mathrm{U}\ s')$.

The algorithm is shown in Algorithm 14. It maintains a FIFO queue contains all the states whose corresponding probabilities should be updated. In each iteration, the algorithm pops a state from the head of the queue, and updates its corresponding probability. If the value is converged, the algorithm moves to the next iteration directly, otherwise, all the states whose probabilities are based on the current value will be added to the queue first. The algorithm keeps running in this manner until the queue is empty.

Algorithm 14 COMPUTE-UNTIL-PROBS-DTMCR(D, Z_D, s_o, ϵ)

Input: DTMCR D, ZSCC Z_D, state $s_o \in Z_D.Out$ and convergence criterion ϵ
Output: approximate $Pr_{D,s}(Z_D \text{ U } s_0)$ for all states $s \in Z_D$

1: **for all** state $s \in Z_D \cup Z_D.Out$ **do** $x_s := 0$; **end for**
2: $x_{s_o} := 1$;
3: Let Q be a FIFO queue which disregards insertions of existing elements;
4: Push all the states in Z_D into queue Q;
5: **while** Q is not empty **do**
6: $s :=$ pop the head of Q;
7: $x'_s := \sum_{s' \in Z_D \cup Z_D.Out} \mathbf{P}(s, s') \cdot x_{s'}$;
8: **if** $|x_s - x'_s| > \epsilon$ **then**
9: Push all the states $s' \in Z_D$ where $\mathbf{P}(s', s) > 0$ into queue Q;
10: **end if**
11: $x_s := x'_s$;
12: **end while**
13: **return** $(x_s)_{s \in Z_D}$

Example 17. Assume we are going to solve the following linear equation system with convergence criterion $\epsilon = 0.01$:

$$x_0 = 0 \qquad x_1 = x_2 \qquad x_2 = x_3$$
$$x_3 = 0.5x_4 + 0.5x_5 \qquad x_4 = 0.5x_1 + 0.5x_6$$
$$x_5 = 1 \qquad x_6 = 0.$$

The values of each variables through each iteration until all values are converged by using both Gauss-Seidel method and the newly introduced method are the same, which is shown in Table 1. Note that the result is based on the assumption that the FIFO queue for the newly introduced method is initialized as $\{x_3\}$. This assumption is reasonable as no matter how to initialize the FIFO queue, it will at most have 6 more iterations, which do not update any variables, than the result shown in the table.

Table 1: Solving the above linear equation system via iterative methods

Variables	Values through each iteration										
	0	1	2	3	4	5	$\cdots$	13	14	15	16
x_0	0	0	0	0	0	0	$\cdots$	0	0	0	0
x_1	0	0	0	$\frac{1}{2}^*$	$\frac{1}{2}$	$\frac{1}{2}$	$\cdots$	$\frac{21}{32}$	$\frac{21}{32}$	$\frac{85}{128}^*$	$\frac{85}{128}$
x_2	0	0	$\frac{1}{2}^*$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\cdots$	$\frac{21}{32}$	$\frac{85}{128}^*$	$\frac{85}{128}$	$\frac{85}{128}$
x_3	0	$\frac{1}{2}^*$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{5}{8}^*$	$\cdots$	$\frac{85}{128}^*$	$\frac{85}{128}$	$\frac{85}{128}$	$\frac{85}{128}$
x_4	0	0	0	0	$\frac{1}{4}^*$	$\frac{1}{4}$	$\cdots$	$\frac{21}{64}$	$\frac{21}{64}$	$\frac{21}{64}$	$\frac{85}{256}^*$
x_5	1	1	1	1	1	1	$\cdots$	1	1	1	1
x_6	0	0	0	0	0	0	$\cdots$	0	0	0	0

Though they have the same number of iterations, Gauss-Seidel method recomputes and / or updates each variable for each iteration, while the newly introduced method only computes and updates one variable for each iteration. The updated variable of each iteration by applying the newly introduced method is marked with the superscript "$*$". ■

3.3.3 Top-down with Memoization and Zero Connected Components

Besides using the concept of ZSCCs, we can use another type of connected components to compute the probability $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$.

Definition 7. Let $D = (S, \overline{s}, \mathbf{P}, L, r_s, r_t)$ be a DTMCR. A Zero Connected Component (ZCC) of D based on a state reward formula Φ is a subset of states $Z_D \subseteq S$ satisfying the following conditions:

- for all state $s \in Z_D$, $s \not\models \Phi$, $s \notin S^0_\Phi$ and $r_s(s) = 0$;
- for every pair of distinct states s_i and s_j, there is a finite path π either from s_i to s_j or from s_j to s_i, where for any state s appears on π, $s \in Z_M$ and $r_\pi = 0$;
- if $|Z_D| = 1$, then the only state s in Z_D must satisfies $\mathbf{P}(s, s) > 0$ and $r_t(s, s) = 0$;
- Z_D is a maximal subcomponent of D, i.e. there is no distinct ZCC Z'_D such that if $s \in Z_D$, then $s \in Z'_D$.

The notations of ZSCCs can be applied to ZCCs by replacing all the ZSCCs with ZCCs. ■

One way to compute $D.ZCC(\Phi)$ based on a given DTMCR D and a PCTLR state formula Φ is shown in Algorithm 15. The algorithm first constructs the undirected reduced graph. Then for each state s which does not belong to any ZCC, it finds all the states can reach from s and the set contains all these states is a new ZCC. The algorithm repeats this procedure until there are no remaining states.

Algorithm 15 DETECT-ZCC-DTMCR(D, S^0_Φ, Φ)

Input: DTMCR D, set of states S^0_Φ and PCTL state formula Φ
Output: $D.ZCC(\Phi)$
1: Construct the undirected graph $G = (V, E)$, where:
  - $V := \{s \in S \mid s \not\models \Phi \wedge s \notin S^0_\Phi \wedge r_s(s) = 0\}$;
  - $E := \{(s_i, s_j) \mid s_i, s_j \in V \wedge \mathbf{P}(s_i, s_j) > 0 \wedge r_t(s_i, s_j) = 0\}$;

2: $D.ZCC(\Phi) := \emptyset$;
3: **for all** state $s \in V$ **do**
4:   **if** $s \notin D.ZCC(\Phi)$ **then**
5:     $Z_D := \{s' \in V \mid s'$ is reachable from s and $s' \notin D.ZCC(\Phi)\}$;
6:   **end if**
7:   $D.ZCC(\Phi) := D.ZCC(\Phi) \cup \{Z_D\}$;
8: **end for**
9: **return** $D.ZCC(\Phi)$;

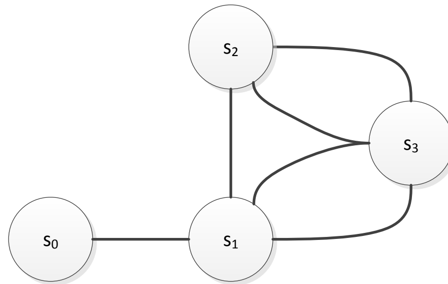


Figure 12: The undirected reduced graph of the DTMCR shown in Figure 8

Example 18. To compute $D.ZCC(target)$ for the DTMCR D in Figure 8 by calling DETECT-ZCC-DTMCR$(D, S^0_{target}, target)$, the algorithm first constructs the undirected reduced graph G as shown in Figure 12. Then it picks up one state, assume it is s_0. All the states reachable from s_0

are s_1, s_2 and s_3 which gives us a ZCC $Z_D = \{s_0, s_1, s_2, s_3\}$. The algorithm terminates from here as there are no remaining states left. Therefore, the result set $D.ZCC(target) = \{\{s_0, s_1, s_2, s_3\}\}$. ■

For a given DTMCR D and a PCTLR formula Φ, once $D.ZCC(\Phi)$ is computed, we can directly apply Algorithm 12 to compute $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$ by replacing all the ZSCCs with ZCCs. Note that the definition of ZCC excludes all states in S^0_Φ to be part of any ZCCs. This ensures the correctness of the algorithm.

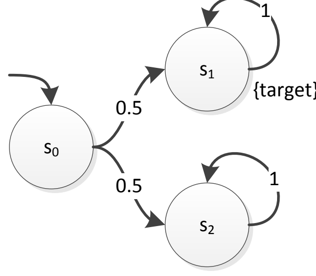


Figure 13: The problem of including states $s \in S^0_{target}$ in ZCCs

Example 19. For the DTMCR D in Figure 13, in order to compute $Pr_{D,s_0}(\mathrm{R}^{path}_{>0}[\mathrm{F}\ target])$, we have:

$$T = Sat(target) = \{s_1\}$$
$$S^0_{target} = \{s_2\}.$$

Let $x_{D,s,\sim r}$ denotes $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ target])$, and the top-down with memoization approach will be applied. If all states in S^0_{target} are not excluded from ZCCs, we have $D.ZSCC(\Phi) = \{\{s_0, s_2\}\}$ and the following equation will be derived:

$$x_{D,s_0,>0} = x_{D,s_1,>0} \qquad s_0 \in D.ZCC^{1o}(target)$$

which is obviously incorrect. Even though we apply the derivation rule for general ZCCs (assume $s_0 \in D.ZCC^n(target)$) here:

$$\begin{aligned} x_{D,s_0,>0} &= Pr_{D,s_0}(Z^{target}_{D,s_0}\ \mathrm{U}\ s_1) \times x_{D,s_1,>0} && s_0 \in Z^{target}_{D,s_0} \\ &= \frac{1}{2} x_{D,s_1,>0} \\ x_{D,s_1,>0} &= 0 && s_1 \in T \wedge \neg(0 > 0) \end{aligned}$$

where it is concluded that:

$$x_{D,s_0,>0} = \frac{1}{2} \times 0 = 0,$$

which is still not the correct result. On the other hand, if we exclude all states in S^0_{target} from ZCCs, we have $D.ZSCC(\Phi) = \emptyset$ and:

$$\begin{aligned} x_{D,s_0,>0} &= \mathbf{P}(s_0, s_1) \times x_{D,s_1,>0} + \mathbf{P}(s_0, s_2) \times x_{D,s_2,>0} \\ x_{D,s_1,>0} &= 0 && s_1 \in T \wedge \neg(0 > 0) \\ x_{D,s_2,>0} &= 1 && s_2 \in S^0_{target} \wedge \infty > 0 \end{aligned}$$

which gives us the correct result:

$$x_{D,s_0,>0} = \frac{1}{2} \times 0 + \frac{1}{2} \times 1 = \frac{1}{2}.$$ ■

Though both approaches with ZSCCs and ZCCs have the similar structure, the performance varies in different cases. A ZCC can be consisted of some ZSCCs only, some single states (which does not belong to any ZSCCs) only or both ZSCCs and single states.

- If a ZCC is consisted of single states only, the performance is no better than considering those single states separately, and this is the downside of the ZCC approach.
- If a ZCC Z is consisted of a set of ZSCCs $ZSCCSet$ and some single states:
  - If $|Z.Out| = 1$, then consider these states as a single ZCC is better than considering these ZSCCs and single states separately.
  - If $|Z.Out| < \sum_{Z' \in ZSCCSet} |Z'.Out|$, then which approach is better relies on each specific case.
  - If $|Z.Out| \geq \sum_{Z' \in ZSCCSet} |Z'.Out|$, then consider these ZSCCs and single states separately yield a better performance.

Let $x_{D,s,<r} = Pr_{D,s}(\mathrm{R}^{path}_{<r}[\mathrm{F}\ target])$. The following examples will provide some cases illustrating the above performance issues.

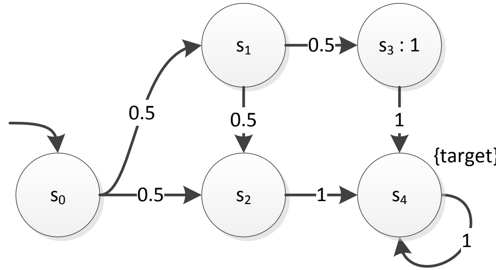


Figure 14: The downside of the ZCC approach

Example 20. First we will show you a case where a ZCC is consisted with single states only and provides a worse performance than considering single states separately. In the DTMCR D in Figure 14, $D.ZSCC(target) = \emptyset$ and $D.ZCC(target) = \{\{s_0, s_1, s_2\}\}$. To compute $x_{D,s_0,<1}$ with ZSCCs, we have:

$$\begin{aligned}
x_{D,s_0,<2} &= \mathbf{P}(s_0, s_1) \times x_{D,s_1,<2} + \mathbf{P}(s_0, s_2) \times x_{D,s_2,<2} \\
x_{D,s_1,<2} &= \mathbf{P}(s_1, s_2) \times x_{D,s_2,<2} + \mathbf{P}(s_1, s_3) \times x_{D,s_3,<2} \\
x_{D,s_2,<2} &= \mathbf{P}(s_2, s_4) \times x_{D,s_4,<2}.
\end{aligned}$$

Once $x_{D,s_3,<2}$ and $x_{D,s_4,<2}$ are computed, only three more equations, $x_{D,s_2,<2}$, $x_{D,s_1,<2}$ and $x_{D,s_0,<2}$, need to be computed in order to achieve the final result. However, if the computation of $x_{D,s_0,<2}$ is done with ZCCs, we have

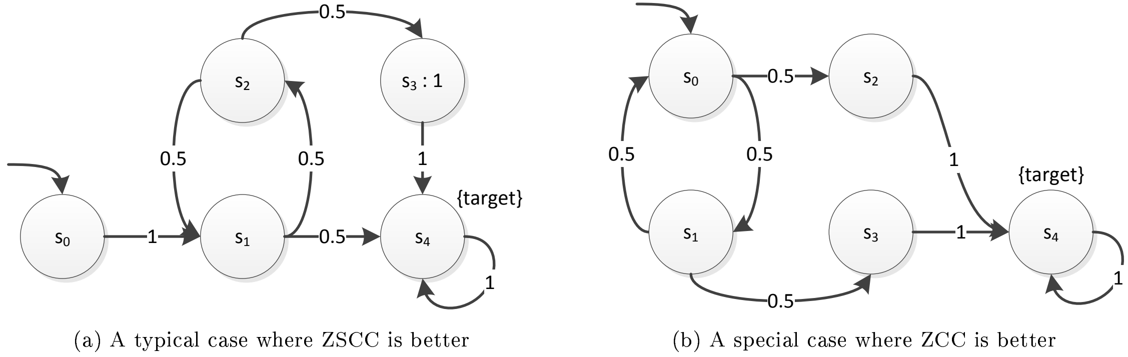
$$\begin{aligned}
x_{D,s_0,<2} &= Pr_{D,s_0}(Z^{target}_{D,s_0}\ \mathrm{U}\ s_3) \times x_{D,s_3,<2} + Pr_{D,s_0}(Z^{target}_{D,s_0}\ \mathrm{U}\ s_4) \times x_{D,s_4,<2} \\
x_{D,s_3,<2} &= x_{D,s_4,<2} = 1. && \text{assume they are computed in advance}
\end{aligned}$$

Algorithm 14 is applied to compute the until probability. First we push all the three states in Z^{target}_{D,s_0} into the queue with the order $\{s_0, s_1, s_2\}$ and initialize $x_{D,s_0,<2}$, $x_{D,s_1,<2}$ and $x_{D,s_2,<2}$ with value 0.

The algorithm updates these variables in the following order with the convergence criteria $\epsilon = 0.01$:

$$x_{D,s_0,<2} = 0 \qquad \text{queue} => \{s_1, s_2\}$$
$$x_{D,s_1,<2} = 0.5 \qquad \text{queue} => \{s_2, s_0\}$$
$$x_{D,s_2,<2} = 1 \qquad \text{queue} => \{s_0, s_1\}$$
$$x_{D,s_0,<2} = 0.75 \qquad \text{queue} => \{s_1\}$$
$$x_{D,s_1,<2} = 1 \qquad \text{queue} => \{s_0\}$$
$$x_{D,s_0,<2} = 1, \qquad \text{queue} => \{\}$$

this requires computations for 6 equations which is 3 more than the previous approach. ■



(a) A typical case where ZSCC is better

(b) A special case where ZCC is better

Figure 15: Performance comparisons between ZSCCs and ZCCs

Example 21. Then two more cases will be shown where a ZCC is consisted with both single states and ZSCCs. Let us start by considering a typical case where using ZSCCs provides a better performance. In the DTMCR D_1 in Figure 15 (a), $D_1.ZSCC(target) = \{Z_{D_1} = \{s_1, s_2\}\}$ and $D_1.ZCC(target) = \{Z'_{D_1} = \{s_0, s_1, s_2\}\}$. To compute $x_{D_1,s_0,<1}$ with ZSCCs, we have:

$$x_{D_1,s_0,<1} = x_{D_1,s_1,<1}$$
$$x_{D_1,s_1,<1} = Pr_{D_1,s_1}(Z_{D_1} \text{ U } s_3) \times x_{D_1,s_3,<1} + Pr_{D_1,s_1}(Z_{D_1} \text{ U } s_4) \times x_{D_1,s_4,<1}.$$

While by using the concept of ZCCs, we have:

$$x_{D_1,s_0,<1} = Pr_{D_1,s_0}(Z'_{D_1} \text{ U } s_3) \times x_{D_1,s_3,<1} + Pr_{D_1,s_1}(Z'_{D_1} \text{ U } s_4) \times x_{D_1,s_4,<1}.$$

Assume both $x_{D_1,s_3,<1}$ and $x_{D_1,s_4,<1}$ are computed and Algorithm 14 is used to compute $Pr_{D,s}(Z_D \text{ U } s')$. It is easy to see that computing $Pr_{D_1,s_1}(Z_{D_1} \text{ U } s_3)$ requires less iterations than $Pr_{D_1,s_0}(Z'_{D_1} \text{ U } s_3)$ as in the latter case, whenever $Pr_{D_1,s_1}(Z'_{D_1} \text{ U } s_3)$ is updated through iterations, $Pr_{D_1,s_0}(Z'_{D_1} \text{ U } s_3)$ will be updated afterwards. Therefore, the approach with ZCCs is slower than the one with ZSCCs in this case.

Next let us move to a special case where using ZCCs is better. In the DTMCR D_2 in Figure 15 (b), $D_2.ZSCC(target) = \{\{s_0, s_1\}\}$ and $D_2.ZCC(target) = \{\{s_0, s_1, s_2, s_3\}\}$. To compute $x_{D_1,s_0,<1}$ with ZSCCs, we have:

$$x_{D_2,s_0,<1} = Pr_{D_2,s_0}(Z^{target}_{D_2,s_0} \text{ U } s_2) \times x_{D_2,s_2,<1} + Pr_{D_2,s_0}(Z^{target}_{D_2,s_0} \text{ U } s_3) \times x_{D_2,s_3,<1}$$
$$x_{D_2,s_2,<1} = x_{D_2,s_4,<1}$$
$$x_{D_2,s_3,<1} = x_{D_2,s_4,<1}.$$

While by using the concept of ZCCs, we have:

$$x_{D_2,s_0,<1} = x_{D_2,s_4,<1}. \qquad s_0 \in D_2.ZCC^{1o}(target)$$

In the former case, after $x_{D_2,s_4,<1}$ is calculated, further computations are required. On the contrary, the value of $x_{D_2,s_4,<1}$ can be assigned directly to $x_{D_2,s_0,<1}$ to finish the computation in the later case. ■

As for a given DTMCR and a PCTLR formula, some ZCCs may provide a better performance and others do not, so it is hard to say which approach is definitely better than the other before we know the actual conditions. Due to the fact that the discussion of this is beyond the scope of this paper, we will not go any further and instead will give some test cases in Section 8.3.2 covering two cases shown in Example 21.

3.3.4 DTMCRs with Negative Rewards

As mentioned before, the algorithm introduced above for computing $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$ only supports non-negative rewards. In this section, we will discuss to what extent we can modify it to support DTMCRs with negative rewards. For a DTMCR D has only non-positive rewards, we can simply call Algorithm 12 with parameters $D', \neg\sim -r$ and Φ to compute $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$, where:

- D' is the modified DTMCR by replacing both the state and transition rewards in D with their absolute values, e.g. a reward -5 will be replaced by the value 5;
- $\neg > \Leftrightarrow <, \neg \geq \Leftrightarrow \leq, \neg < \Leftrightarrow >$ and $\neg \leq \Leftrightarrow \geq$.

Next, let us consider DTMCRs with both positive and negative rewards. For a given DTMCR D and a PCTLR formula Φ, by the definition of the random variable $X_{\mathrm{F}\Phi}$, for all path $\omega \in Path_{D,s}$, $X_{\mathrm{F}\Phi}(\omega) >= 0$ is no longer always true due to the newly introduced negative rewards. However, the rest properties still hold. Therefore, in order to support these DTMCRs we have to and only have to modify the following four cases in Equation (4) by replacing all the four 0 with $r^{\min}_{\mathrm{F}\Phi}(s)$:

- $\sim$ is $> \wedge\ r < 0$
- $\sim$ is $\geq \wedge\ r \leq 0$
- $\sim$ is $< \wedge\ r \leq 0$
- $\sim$ is $\leq \wedge\ r < 0$,

where $r^{\min}_{\mathrm{F}\Phi}(s)$ is the total rewards consumed along the shortest path start from state s to any state $s' \in Sat(\Phi)$. The formal definition of it is as follows:

$$r^{\min}_{\mathrm{F}\Phi}(s) \stackrel{def}{=} \min_{\omega \in Path_{D,s}} X_{\mathrm{F}\Phi}(\omega).$$

The rest of the introduced algorithm for computing $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$ will be kept the same as before. To compute $r^{\min}_{\mathrm{F}\Phi}(s)$, the converted graph of the given DTMCR is constructed first.

Definition 8. The converted graph of a given DTMCR $D = (S, \overline{s}, \mathbf{P}, L, r_s, r_t)$ is a 3-tuple $G = (V, E, w)$ where:

- V is a set of vertices where $V = S$;
- E is a set of edges where $E = \{(u, v) \mid u, v \in S \wedge \mathbf{P}(u, v) > 0\}$;
- $w : V \times V \to \mathbb{R}$ is a weight function where for any $u, v \in V$, $w(u, v) = r_{u \to v}$. ■

Once the converted graph is constructed, a shortest path algorithm, which supports negative weights, can be applied to compute the shortest path from one specific state to all the target states. Let $SD(u, v)$ denotes the shortest distance between vertex u and v in the converted graph, then:

$$r^{\min}_{\mathrm{F}\Phi}(s) = \min_{s' \in Sat(\Phi)} SD(s, s').$$

One well-known shortest path algorithm capable for this job is the Bellman-Ford algorithm[21], also Shortest Path Faster Algorithm (SPFA)[22] is a nice choice as for sparse graphs it has a better performance than Bellman-Ford. Because if a graph contains negative cycles, the shortest path for some pairs of vertices are unable to be defined as there is always a path which is shorter than any given path between these two vertices. Thus, this improved algorithm fails if the input DTMCR contains negative loops (i.e. its converted graph contains negative cycles). In fact, the improved algorithm can also be applied to DTMCRs with only non-negative rewards, though it is not recommended due to the non-negligible running time consumed by the shortest paths computations.

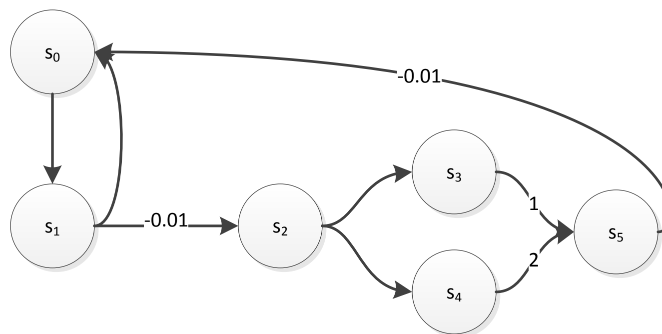


Figure 16: The converted graph of the DTMCR in Figure 3

Example 22. Figure 16 is the converted graph of the DTMCR in Figure 3 where the weights with value 0 are omitted, and:

$$r^{\min}_{\mathrm{F}\ release}(s_0) = 0.99.$$

■

3.4 Worklist Algorithm

Reconsider the algorithms introduced in this section which use the top-down approach of dynamic programming, all of them compute the probabilities for all the states in the given DTMCR. However, not all the value are always needed in practice, we can thus apply a simple performance optimization, which only computing the probabilities for a subset of all the states. Furthermore, a well-known performance issue of the algorithms using the top-down approach of dynamic programming is the overhead (both time and memory consuming) of recursive procedure calls. If we can eliminate them, even though the asymptotic running time of the algorithms will not change, their performances in practice might be improved. Note that whether the performances will be improved or not after eliminating the recursive procedure calls depends on the actual implementation of the modified algorithms and the original implementation of recursive procedure calls for a specific programming language. In this section, we will introduce a worklist algorithm with no recursive procedure calls and its performance (both running time and memory usage) will be analyzed in Section 8.3.2.

To begin with, let us take a closer look at how those algorithms with top-down approaches work:

- a list of values are requested to be computed for a given algorithm first;
- for any value v needs to be computed, if all its dependent values (those used to compute v) are precomputed, then v will be computed directly. Otherwise, the computation of v will be postponed and the algorithm will compute all its dependent values which are not precomputed;
- once all the dependent values of v are computed, the algorithm will go back to compute v.

For a specific run of an algorithm, we have a dependent graph of all the values computed during the whole procedure. For instance, assume an algorithm are requested to compute v_0, v_0 are dependent on v_1, v_2 and v_3, v_1 is dependent on v_2, v_2 is dependent on v_3 and v_3 can be computed directly. Then we will have a dependent graph of these values as shown in Figure 17. Notice that there should

not be any loops in a dependent graph, otherwise, the recursive definition an algorithm is based on must have the potential mutual dependences problem.

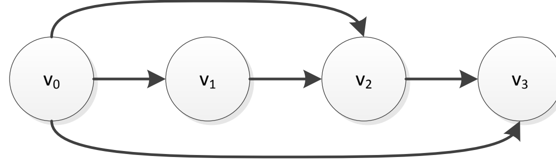


Figure 17: The dependent graph of some values

Therefore, if we first find all the values need to be computed, do a topological sort on them and work on them in the post topological order. Then there will be no recursive calls any more. To achieve this, we can use a doubly linked list, add all the requested values in the list. Then start from the beginning of the list, for each value:

- if it can be computed directly (the base case), nothing happens;
- otherwise, for all the dependent values of the current value:
  - if it does not exist in the list, it will be added at the end of the list,
  - otherwise, it will be moved to the end from where it is.

Once we reach the end of the list, we have all the values need to be computed in the list in topological order and they can be computed backward with no recursive calls.

Example 23. Consider the problem described by Figure 17. We start with a list contains only v_0:

$$\{v_0\}.$$

As v_0 is dependent on v_1, v_2 and v_3 and none of them exists in the list, so they will be added to the end of the list. Assume the insertion order is v_2, v_1 and v_3, then the list becomes:

$$\{\boxed{v_0}, v_2, v_1, v_3\}.$$

Next value to be considered is v_2 which is dependent on v_3 which is already at the end of the list, nothing happens. We move to v_1 which is dependent on v_2 which is in the middle of the list, we thus move v_2 to the end. We have:

$$\{v_0, \boxed{v_1}, v_3, v_2\}.$$

Because v_3 is the base case, hence the next value changes the list is v_2 which moves v_3 to the end:

$$\{v_0, v_1, \boxed{v_2}, v_3\}.$$

Again no changes of the list for v_3 and the end of the list is reached, therefore, in order to compute v_0 without any recursive procedure calls, one should compute values in the order v_3, v_2, v_1, v_0. ■

Note that in the previous example, v_2 and v_3 have been moved several times, to reduce this type of movements, we can use the doubly linked list as a stack. Then start from adding all the requested values in the stack, for each value at the top of the stack:

- if it is visited or it is the base case, compute it directly;
- otherwise, mark it as visited and for all the uncomputed dependent values of the current value:
  - if it does not exist in the stack, add it to the top of the stack;
  - otherwise, it will be moved to the top from where it is.

Once the stack is empty, all the values are computed with no recursive calls. Notice that if a value is marked as visited, it will never be moved to the top of the stack otherwise it is a sign of existing loops in the dependence graph.

Example 24. Let us retry the problem described by Figure 17 with the new method. We start with a stack contains only v_0 marked as unvisited (F):

$$\{(v_0, F)\}^9.$$

v_0, the unvisited top element, is marked as visited (T) and its dependent values are added into the stack in the order of v_2, v_1 and v_3, the stack becomes:

$$\{(v_0, T), (v_2, F), (v_1, F), (v_3, F)\}.$$

Because the top element v_3 is the base case, it is computed directly. Then comes v_1, its dependent value v_2 is moved to the top:

$$\{(v_0, T), (v_1, T), (v_2, F)\}.$$

From here, all the elements can be computed directly from the top to the bottom of the stack which yields less steps compared to the previous method. ■

To ensure all the operations of the stack take constant time, a data structure called Linked Hash Stack will be introduced first.

Definition 9. A Linked Hash Stack (denoted by LHS<K, P>) is a stack coupled with a hash table and can be customized with K, a type of keys, and P, a type of parameters. The stack is implemented by a doubly linked list which ensures that it takes constant time to move an element from the middle of the stack to the top. Each element in the stack is a key-parameter pair with the form (K, P) and the hash table maps a key to the pointer which points to the element with the same key in a linked list. Some functions are defined for a given LHS<K, P> as shown in Algorithm 16, where *stack* and *ht* are the stack and the hash table of LHS.

Algorithm 16 Functions defined for a given LHS<K, P>

1: **function** ISEMPTY( ) **return** whether *ht* is empty; **end function**

1: **function** PUSH(k, p)
2:     **if** $ht(k)$ is defined **then**
3:         $ptr := ht(k)$;
4:         move the element *ptr* points to the the top of *stack*;
5:     **else**
6:         push (k, p) into *stack* and let *ptr* points to it;
7:         $ht(k) := ptr$;
8:     **end if**
9: **end function**

1: **function** PEAK( ) **return** the top element (k, p) of *stack* **end function**

1: **function** POP( )
2:     remove the mapping of k from *ht*;
3:     **return** the top element (k, p) of *stack* and pop it from *stack*;
4: **end function**

1: **function** UPDATE(p') for the top element (k, p) of *stack*, $p := p'$; **end function**

[9] Assume that the right side is the top of the stack.

It is obvious that isEmpty, peak, pop and update take only constant time. For push, it also takes constant time as check an existing key in the hash table takes constant time, move an element from the middle of a doubly linked list to the end takes constant time and so does add a key-value pair into the hash table. ■

With the newly introduced linked hash stack and the performance improvements mentioned above, a worklist algorithm is developed as shown in Algorithm 17. It only computes the probabilities for states in the subset $S' \subseteq S$ of DTMCR D. By specifying different key type K, tuple of constants C and details of method Initial-Key, Dependent-Keys and Pr-R-DTMCR, it can compute the probabilities for all three types of path reward formulae. The worklist specifications for instantaneous reward formula $\mathrm{I}^{=k}$, cumulative reward formula $\mathrm{C}^{\leq k}$ and reachability reward formula F Φ are shown in Algorithms 18 to 20, respectively.

Algorithm 17 Worklist-Pr-DTMCR($D, S', \sim r, C$)

Input: DTMCR D, set of states $S' \subseteq S$, reward bound $\sim r$ and tuple of constants C
Output: $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\varphi])$ for all $s \in S'$

```
1: let p be an empty hash table maps a key of type K to a probability value;
2: let W be an empty LHS<K, Boolean>;
3: for all state s ∈ S' do W.push(Initial-Key(s, ∼ r, C), false); end for
4: while ¬W.isEmpty() do
5:     (key, canCompute) := W.peak();
6:     if canCompute then
7:         W.pop();
8:         Pr-R-DTMCR(D, key, ∼ r, C, p);
9:     else
10:        W.update(true);
11:        for all key' ∈ Dependent-Keys(D, key, ∼ r, C) do
12:            if p(key') is not defined then W.push(key', false); end if
13:        end for
14:    end if
15: end while
16: return (p(Initial-Key(s, ∼ r, C)))_{s∈S'};
```

Algorithm 18 Worklist specifications for $\mathrm{I}^{=k}$

C is a 1-tuple (k) where k is a step bound;
A key of type K is a 2-tuple (s, k) where s is a state and k is a step value;

```
1: function Initial-Key(s, ∼ r, C) return (s, C.k); end function

1: function Dependent-Keys(D, key, ∼ r, C)
2:     KeySet := ∅;
3:     if key.k > 0 then
4:         for all s ∈ S where P(key.s, s) > 0 do
5:             KeySet := KeySet ∪ {(s, key.k − 1)};
6:         end for
7:     end if
8:     return KeySet;
9: end function

1: function Pr-R-DTMCR(D, key, ∼ r, C, p)
2:     Pr-I-DTMCR-Aux(D, key.s, ∼ r, key.k, p);
3: end function
```

Algorithm 19 Worklist specifications for $C^{\leq k}$

C is a 1-tuple (k) where k is a step bound;
A key of type K is a 3-tuple (s, r, k) where s is a state, r is a reward value and k is a step value;

```
1: function Initial-Key(s, ~ r, C) return (s, r, C.k); end function

1: function Dependent-Keys(D, key, ~ r, C)
2:     KeySet := ∅;
3:     if key.k > 0 then
4:         for all s ∈ S where P(key.s, s) > 0 do
5:             KeySet := KeySet ∪ {(s, key.r − r_{key.s→s}, key.k − 1)};
6:         end for
7:     end if
8:     return KeySet;
9: end function

1: function Pr-R-DTMCR(D, key, ~ r, C, p)
2:     Pr-C-DTMCR-Aux(D, key.s, ~ key.r, key.k, p);
3: end function
```

Algorithm 20 Worklist specifications for F Φ

C is a 2-tuple (S^0_Φ, T) where Φ and T are sets of states;
A key of type K is a 2-tuple (s, r) where s is a state and r is a reward value;

```
1: function Initial-Key(s, ~ r, C) return (s, r); end function

1: function Dependent-Keys(D, key, ~ r, C)
2:     KeySet := ∅;
3:     if x_{D,key.s,~key.r} does not satisfy any base case conditions of its recursive definition then
4:         if key.s ∈ D.ZSCC(Φ) then
5:             for all s ∈ Z^Φ_{D,key.s}.Out do
6:                 KeySet := KeySet ∪ {(s, key.r)};
7:             end for
8:         else
9:             for all s ∈ S where P(key.s, s) > 0 do
10:                KeySet := KeySet ∪ {(s, key.r − r_{key.s→s})};
11:            end for
12:        end if
13:    end if
14:    return KeySet;
15: end function

1: function Pr-R-DTMCR(D, key, ~ r, C, p)
2:     Pr-F-DTMCR-Aux(D, C.S^0_Φ, key.s, ~ key.r, C.T, p);
3: end function
```

4 Extending MDPs and PCTL with rewards

Next we will extend the PCTL with path reward formulae for MDPs, it is based on the extension of PCTL with state reward formulae introduced in [15, 23]. Brief introductions of the definition of MDPs with rewards and the PCTL with state reward formulae are summarized from these two papers as Section 4.1 and Section 4.2 for the completeness and the readability. Some notations are slightly modified for the consistency of notations used in this paper.

4.1 Markov Decision Processes with Rewards

For simplicity, a Markov Decision Process with Rewards will be written as MDPR. It is assumed that both of the state and action rewards for a given MDPR are real numbers if it is not stated explicitly. Let AP be a fixed, finite set of atomic propositions.

Definition 10. An MDP is a 5-tuple $M = (S, \overline{s}, Act, Steps, L)$ where:

- S is a finite set of states;
- $\overline{s} \in S$ is an initial state;
- Act is a finite set of actions;
- $Steps : S \times Act \rightarrow Dist_S$ is a probabilistic transition function, where $Dist_S : S \rightarrow [0, 1]$ is a probability distribution, i.e. the probability that a transition from s to s' by taking action a occurs is $Steps(s, a)(s')$;
- $L : S \rightarrow 2^{AP}$ is a labeling function mapping each state to a set of atomic propositions the state satisfies.

An MDPR is a 7-tuple $M = (S, \overline{s}, Act, Steps, L, r_s, r_a)$ where the first five elements are defined the same as MDP and:

- $r_s : S \rightarrow \mathbb{R}$ is a state reward function;
- $r_a : S \times Act \rightarrow \mathbb{R}$ is an action reward function.

The reward a given MDPR consumes, when it takes action a at state s, is the sum of the state reward of s and the action reward of a at s. Let $r_{s \rightarrow a}$ denotes this reward, and it is defined as follows:

$$r_{s \rightarrow a} \overset{def}{=} r_s(s) + r_a(s, a). \qquad \blacksquare$$

For a given MDPR, if an action $a \in Act$ is available at $s \in S$, then $Step(s, a)$ is defined and $\sum_{s' \in S} Steps(s, a)(s') = 1$, otherwise $Step(s, a)$ is not defined. The set of available actions for a given state s is denoted by $A(s) = \{a \in ACT \mid Steps(s, a)\}$, and $A(s) \neq \emptyset$ as deadlocks are disallowed.

Example 25. Figure 18 shows an MDPR $M = (S, \overline{s}, Act, Steps, L, r_s, r_a)$ for a hypothetical vending machine which is similar to the one used in Example 1. The only difference is that after it shows the menu to the customer C, C will choose between cancel the purchase or select a product nondeterministically instead of based on certain probabilities.

For graphical notations used of MDPs in this paper, states are drawn as circles with their names on the center, actions are denoted as a black dot linking to the state it belongs to by a solid line labeled with the action name. transitions are represented as arrows with associated probabilities on it and the initial state is marked by an incoming arrow with no out state. Besides that, if the action reward for a specific action is not 0, it is added after the corresponding action name separated by a colon. Similarly if the state reward for a given state is not 0, it is concatenated after the state name with an additional colon.

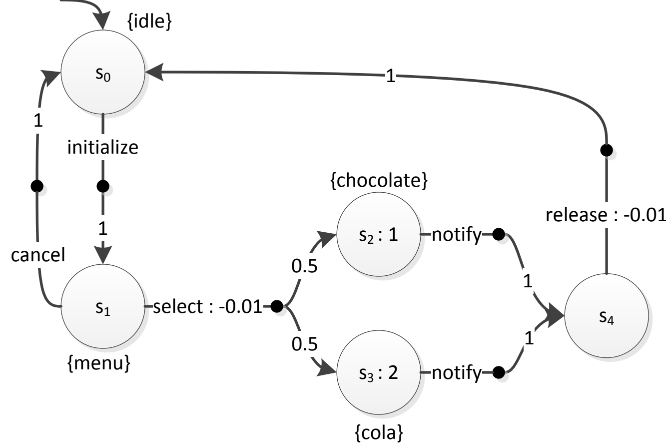


Figure 18: The MDPR for a hypothetical vending machine

The MDPR of this vending machine has five states $S = (s_0, s_1, s_2, s_3, s_4)$ with the initial state $\overline{s} = s_0$ and the alphabet of actions $Act = \{initialize, cancel, select, nofity, release\}$. The probabilistic transition function *Steps* is given by:

$$\begin{aligned}
Steps(s_0) &= \{(initialize, [s_1 \mapsto 1])\};\\
Steps(s_1) &= \{(cancel, [s_0 \mapsto 1]), (select, [s_2 \mapsto 0.5, s_3 \mapsto 0.5])\};\\
Steps(s_2) &= \{(notify, [s_4 \mapsto 1])\};\\
Steps(s_3) &= \{(notify, [s_4 \mapsto 1])\};\\
Steps(s_4) &= \{(release, [s_0 \mapsto 1])\}.
\end{aligned}$$

The labeling functions L maps from S to $2^{\{idle, menu, chocolate, cola\}}$:

$$L(s_0) = \{idle\}, L(s_1) = \{menu\}, L(s_2) = \{chocolate\}, L(s_3) = \{cola\} \text{ and } L(s_4) = \emptyset$$

The rewards denotes the profits of the vending machine. The positive state rewards represent the money earned from selling the corresponding product where $\forall s \in S$:

$$r_s(s) = \begin{cases} 1 & \text{if } s = s_2 \\ 2 & \text{if } s = s_3 \\ 0 & \text{otherwise} \end{cases},$$

and the negative action rewards indicate the average cost (maintenance fee, electricity, etc) for executing this action , where $\forall s \in S, a \in Act$:

$$r_a(s,a) = \begin{cases} -0.01 & \text{if } (s,a) \in \{(s_1, select), (s_4, release)\} \\ 0 & \text{otherwise} \end{cases}.$$

The reward D consumes when it choses the action *select* at s_1 is:

$$r_{s_1 \to select} = r_s(s_1) + r_t(s_1, select) = 0 + (-0.01) = -0.01,$$

and the reward it consumes by choosing the action *notify* at s_3 is:

$$r_{s_3 \to notify} = r_s(s_3) + r_a(s_3, notify) = 1 + 0 = 1.$$ ■

We can achieve a set of paths by unfolding an MDPR $M = (S, \overline{s}, Act, Steps, L, r_s, r_a)$. An infinite path ω through M is a non-empty sequence $s_0 a_0 s_1 a_1 \cdots$, where $s_i \in S$, $a_i \in A(s_i)$ and $Steps(s_i, a_i)(s_{i+1}) > 0$ for all $i \geq 0$. A prefix of a infinite path is a finite path $\pi = s_0 a_0 s_1 \cdots s_n$ ending in a state. For a (finite or infinite) path ω, $\omega^s(i)$ and $\omega^a(i)$ represent the ith state and the ith action of a path ω respectively and the length (number of actions) of ω is denoted by $|\omega|$, which is always ∞ for any infinite path. The last state of a finite path π is denoted by $last(\pi)$. In this paper, let $Path_{M,s}$ and $Path^{fin}_{M,s}$ represent the sets of all infinite and finite paths starting from state s in M, while $Path_M$ and $Path_M$ denote the sets of all infinite and finite paths starting from any state in M.

For a finite path $\pi \in Path^{fin}_{M,s}$, the total reward consumed along it is denoted by r_π, which is defined as follows:

$$r_\pi \stackrel{def}{=} \sum_{i=0}^{|\pi|-1} r_{\pi_s(i) \to \pi_a(i)},$$

and on the other hand, the total reward consumed along an infinite path $\omega \in Path_{D,s}$, which is represented by r_ω, is always ∞.

Example 26. For the MDPR $M = (S, \overline{s}, Act, Steps, L, r_s, r_a)$ shown in Figure 18, let $\pi_1 = s_2$, $\pi_2 = s_1$ *select* s_2 and $\pi_3 = s_1$ *select* s_3 *notify* s_4, then:

$$\begin{array}{lll} |\pi_1| = 0; & |\pi_2| = 1; & |\pi_3| = 2; \\ \pi_1^s(0) = s_1; & \pi_3^s(1) = s_3; & \\ \pi_2^a(0) = select; & \pi_3^a(1) = notify; & \end{array}$$

$$r_{\pi_1} = r_s(s_2) = 0;$$
$$r_{\pi_2} = r_s(s_1) + r_a(s_1, select) = -0.01;$$
$$r_{\pi_3} = r_s(s_1) + r_a(s_1, select) + r_s(s_3) + r_a(s_3, notify) = -0.01 + 2 = 1.99.$$ ■

To model check probabilistic properties over MDPRs, a probability space[24] over infinite paths needs to be built. However, we have to resolve all the nondeterminism before constructing the probability space. A possible resolution of nondeterminism is denoted by an adversary, also called a policy, which chooses an action at each state based on the history of its execution.

Definition 11. An adversary of a given MDPR $M = (S, \overline{s}, Act, Steps, L, r_s, r_a)$ is a function $\sigma : Path^{fin} \to Dist_{Act}$, where $Dist_{Act} : Act \to [0, 1]$ is a probability distribution:

- $\sum_{a \in A(last(\pi))} \sigma(\pi)(a) = 1$;
- and $\sigma(\pi)(a) = 0$ for all action $a \notin A(last(\pi))$.

An adversary σ is memoryless if the distribution $\sigma(\pi)$ only depends on $last(\pi)$ and it is deterministic if for all $\pi \in Path^{fin}_M$, $\exists a \in A(last(\pi)).(\sigma(\pi)(a) = 1)$. ■

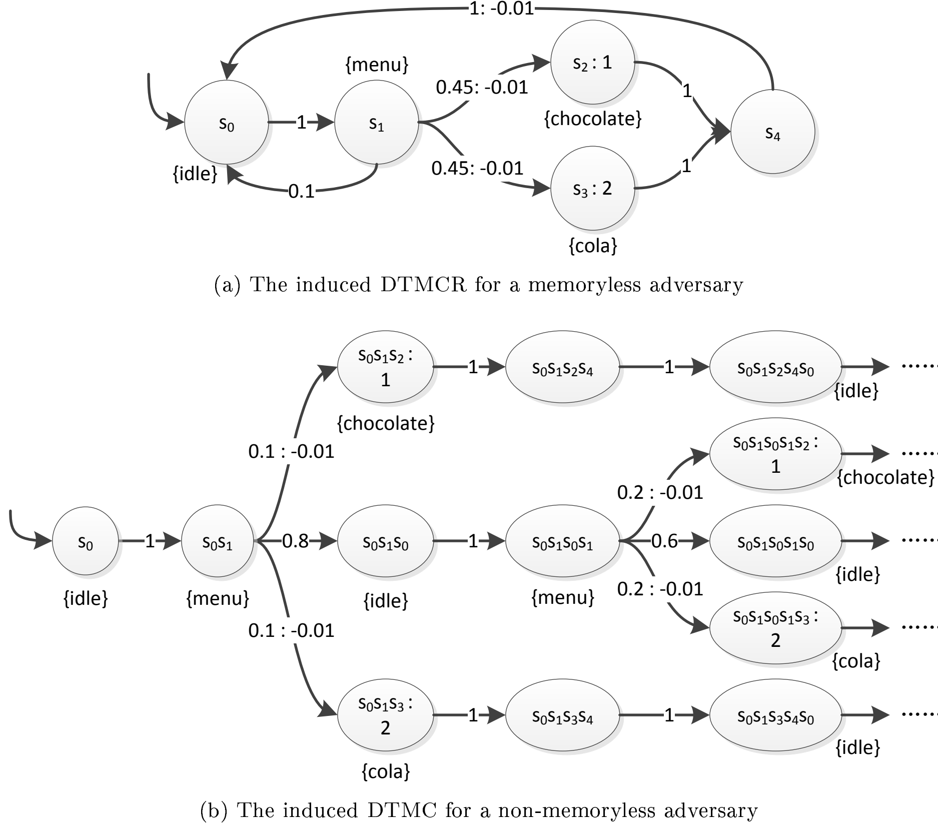
Let Adv represents the set of all possible adversaries of a given MDPR M. Once an adversary $\sigma \in Adv$ is applied to M, the behavior of M can be captured by an induced DTMCR, whose each state is a finite path of M.

Definition 12. The induced DTMCR for an MDPR $M = (S, \overline{s}, Act, Steps, L, r_s, r_a)$ and an adversary σ is $M^\sigma = (Path^{fin}_M, \overline{s}, \mathbf{P}, L', r'_s, r_t)$ where for all $\pi, \pi' \in Path^{fin}_M$:

- $\mathbf{P}(\pi, \pi') = \begin{cases} \sigma(\pi)(a) \cdot Steps(last(\pi), a)(s) & \text{if } \pi' = \pi a s \\ 0 & \text{otherwise} \end{cases}$;
- $L'(\pi) = L(last(\pi))$;
- $r'_s(\pi) = r_s(last(\pi))$;

- $r_t(\pi, \pi') = \begin{cases} r_a(last(\pi), a) & \text{if } \pi' = \pi as \\ 0 & \text{otherwise} \end{cases}$.

Note that there is a one-to-one mapping between $Path_M$ and $Path_{M^\sigma}$, which means from the start state $\overline{s}$, the induced DTMCR M^σ yields a probability space, denoted by $Pr^\sigma_{M,s}$, over all the infinite paths in $Path_M$. Though the induced DTMCR constructed in this way has infinite number of states, if an adversary σ' is memoryless, then the induced DTMCR $M_{\sigma'}$ can be reduced to an $|S|$-state DTMCR. ■



(a) The induced DTMCR for a memoryless adversary

(b) The induced DTMC for a non-memoryless adversary

Figure 19: Two induced DTMCRs of the MDPR shown in Figure 18

Example 27. For the MDPR introduced in Example 25 (shown in Figure 18) and a memoryless adversary σ_1 which will choose the action *cancel* with probability 0.1 and *select* with probability 0.9 at state s_1. The induced DTMCR can be reduced in an $|S|$-state DTMCR which is shown in Figure 19 (a). For another non-memoryless adversary σ_2 which will choose the action *cancel* with probability $1 - k \% 5 \times 0.2$ and select with probability $k \% 5 \times 0.2$ at state s_1, where k is the number of times (includes the current one) it reaches state s_1. The induced DTMCR is partialy shown in Figure 19 (b). ■

4.2 PCTLR for MDPRs

The syntax of PCTLR for MDPRs are exactly the same as for DTMCRs introduced in Section 2, the main differences are:

- Definitions of the random variable X_φ where transition rewards are placed by the action

rewards. For any path $\omega = s_0 a_0 s_1 a_1 \cdots \in Path_{M,s}$:

$$X_{\mathrm{I}^{=k}}(\omega) \stackrel{def}{=} r_s(s_k);$$

$$X_{\mathrm{C}^{\leq k}}(\omega) \stackrel{def}{=} \begin{cases} 0 & \text{if } k = 0 \\ \sum_{i=0}^{k-1} r_{s_i \to a_i} & \text{otherwise} \end{cases};$$

$$X_{\mathrm{F}\Phi}(\omega) \stackrel{def}{=} \begin{cases} 0 & \text{if } s_0 \models \Phi \\ \infty & \text{if } \forall i \in \mathbb{N}. s_i \not\models \Phi \\ \sum_{i=0}^{min\{j \mid s_j \models \Phi\}-1} r_{s_i \to a_i} & \text{otherwise} \end{cases}.$$

- Semantics of the probabilistic operator and the reward operator which require a certain property should be satisfied for all possible adversaries. Let:

$$Pr^{\min}_{M,s}(\phi) \stackrel{def}{=} \min_{\sigma \in Adv} \{Pr^{\sigma}_{M,s}(\phi)\}$$

$$Pr^{\max}_{M,s}(\phi) \stackrel{def}{=} \max_{\sigma \in Adv} \{Pr^{\sigma}_{M,s}(\phi)\}$$

$$Exp^{\min}_{M,s}(X_\varphi) \stackrel{def}{=} \min_{\sigma \in Adv} \{Exp^{\sigma}_{M,s}(X_\varphi)\}$$

$$Exp^{\max}_{M,s}(X_\varphi) \stackrel{def}{=} \max_{\sigma \in Adv} \{Exp^{\sigma}_{M,s}(X_\varphi)\},$$

where $Pr^{\sigma}_{M,s}(\phi)$ is the probability measure and $Exp^{\sigma}_{M,s}(X_\varphi)$ is the reward measure, when σ is applied to M. The satisfaction relation $\models$ is defined for these two operators as follows:

$$s \models \mathrm{P}_{\sim p}[\phi] \quad \Leftrightarrow \quad \begin{cases} Pr^{\min}_{M,s}(\phi) \sim p & \sim \in \{>, \geq\} \\ Pr^{\max}_{M,s}(\phi) \sim p & \sim \in \{<, \leq\} \end{cases}$$

$$s \models \mathrm{R}^{state}_{\sim r}[\varphi] \quad \Leftrightarrow \quad \begin{cases} Exp^{\min}_{M,s}(X_\varphi) \sim r & \sim \in \{>, \geq\} \\ Exp^{\max}_{M,s}(X_\varphi) \sim r & \sim \in \{<, \leq\} \end{cases}.$$

Note that it is not as the PCTLR for DTMCRs, the quantitative form of the probabilistic operator $\mathrm{P}_{=?}[\phi]$ can not be evaluated to a specific value, instead, we should use the minimum and maximum quantitative forms of the P operator, $\mathrm{P}^{\min}_{=?}[\phi]$ and $\mathrm{P}^{\max}_{=?}[\phi]$.

Example 28. Below are some PCTLR formulae with path reward formulae of the MDPR shown in Figure 18:

- $\mathrm{P}_{<0.5}[\mathrm{R}_{>0}[\mathrm{I}^{=6}]]$ - the probability that a customer is just purchasing a product (on either state s_2 or s_3, the only two states whose rewards are greater than 0) after 6 time steps is less than 0.5;
- $\mathrm{P}^{\min}_{=?}[\mathrm{R}_{>=5}[C^{\leq 10}]]$ - the minimum probability that the total cumulative profit gained for the following 10 time step is greater equal than 5;
- $\mathrm{P}^{\max}_{=?}[\mathrm{R}_{<1}[\mathrm{F}\ release]]$ the maximum probability that the profit gained for the first sale is less than 1. ■

4.3 MDPR Action Rewards Elimination

Similar to the transition rewards elimination, the non-zero action rewards can be eliminated for a given MDPR. There are also two approaches, the first approach retains the behaviors of the original MDPR but may give different results when model checking some PCTLR formulae over the modified MDPR. The procedure of eliminating the non-zero action rewards in this way is described by Algorithm 21, where the action name *auxAct* stands for Auxiliary Action.

Algorithm 21 REPLACE-NON-ZERO-ACTIONS(M)

Input: MDPR M
Output: M after the modification

1: $Act := Act \cup \{auxAction\}$; ▷ Assume that $auxAction \notin Act$.
2: **for all** state $s \in S$ **do**
3:     **for all** action $a \in A(s)$ where $r_a(s,a) \neq 0$ **do**
4:         **if** $A(s) = \{a\}$ **then**
5:             $r_s(s) := r_s(s) + r_a(s,a)$;
6:             $r_a(s,a) := 0$;
7:         **else**
8:             $M :=$ ADD-INTERMEDIATE-STATE(M, s, a);
9:         **end if**
10:     **end for**
11: **end for**
12: **return** M;

For each action (s,a) with a non-zero action reward, the action reward will be merged to the state reward $r_s(s)$ if $A(s)$ is $\{a\}$, otherwise, the action (s,a) will be replaced by an intermediate state s_a with the modified action (s,a) and the newly introduced action $(s_a, auxAct)$ as illustrated in Algorithm 22, where the atomic proposition *auxState* stands for Auxiliary State and it is assumed that for all $s \in S$, $auxState \notin L(s)$.

Algorithm 22 ADD-INTERMEDIATE-STATE(M, s, a)

Input: MDPR M, state s and action a
Output: M after the modification

1: $s_a :=$ a newly created state;
2: $S := S \cup \{s_a\}$;
3: $Steps(s_a, auxAct) := Steps(s,a)$; $Steps(s,a) := [s_a \to 1]$;
4: $L(s_a) = \{auxState\}$;
5: $r_s(s_a) := r_a(s,a)$; $r_a(s,a) := 0$; $r_a(s_a, auxAct) := 0$;
6: **return** M;

Example 29. Figure 20 shows the result MDPR M' after applying Algorithm 21 to the MDPR M in Figure 18.

The action reward $r_a(s_4, release)$ is merged to the state reward $r_s(s_4)$ as *release* is the only option for s_4, the action $(s_1, select)$ is replaced by an intermediate state s_{123}, whose state reward equals to $r_a(s_1, select)$, with the modified action $(s_1, select)$ and the newly introduced action $(s_{124}, auxAct)$. The modification may affect the model checking result compared to the original model. Model checking $\mathrm{P}^{\min}_{=?}[\mathrm{F}\ release]$ and $\mathrm{P}^{\max}_{=?}[\mathrm{R}_{>1}[\mathrm{F}\ release]]$ over both MDPRs, M and M', will achieve the same results. However, model checking $\mathrm{P}^{\max}_{=?}[\mathrm{R}_{<0}[\mathrm{I}^{=2}]]$ over M will result 0 which is different from the result 1 for M'. ■

The second approach is applying Algorithm 22 to all the actions of all the states in the original MDPR as described by Algorithm 23. The satisfaction relation $\models$ defined for the P and R^{state} operators are modified as follows, for all state $s \in S$:

$$s \models \mathrm{P}_{\sim p}[\phi] \quad \Leftrightarrow \quad auxState \notin L(s) \land \forall \sigma \in Adv.(Pr^{\sigma}_{M,s}(\phi) \sim p)$$
$$s \models \mathrm{R}^{state}_{\sim r}[\varphi] \quad \Leftrightarrow \quad auxState \notin L(s) \land \forall \sigma \in Adv.(Exp^{\sigma}_{M,s}(X_{\varphi}) \sim r)$$

and the rest semantics of PCTLR is modified as for DTMCRs. This approach will ensure the modified model not only retains the behaviors of the original model, but also returns the same result when model checking any PCTLR formula over it.

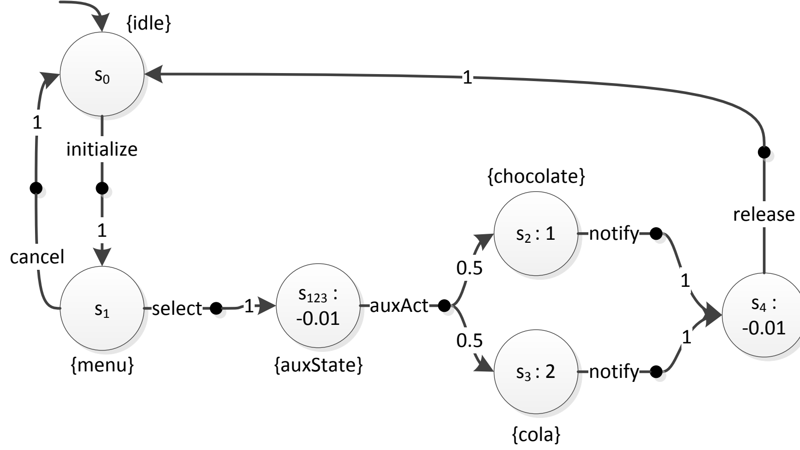


Figure 20: The simple action rewards elimination result MDPR

Algorithm 23 Replace-All-Actions(M)

Input: MDPR M
Output: M after the modification
1: **for all** state $s \in S$ **do**
2: **for all** action $a \in A(s)$ **do**
3: $M :=$ Add-Intermediate-State(M, s, a);
4: **end for**
5: **end for**
6: **return** M;

Example 30. Figure 21 shows the result MDPR M' after applying Algorithm 23 to the MDPR M in Figure 18. And for any PCTLR formula, model checking it over M' with the modified semantics will give the same result as model checking it over M with the original semantics. ■

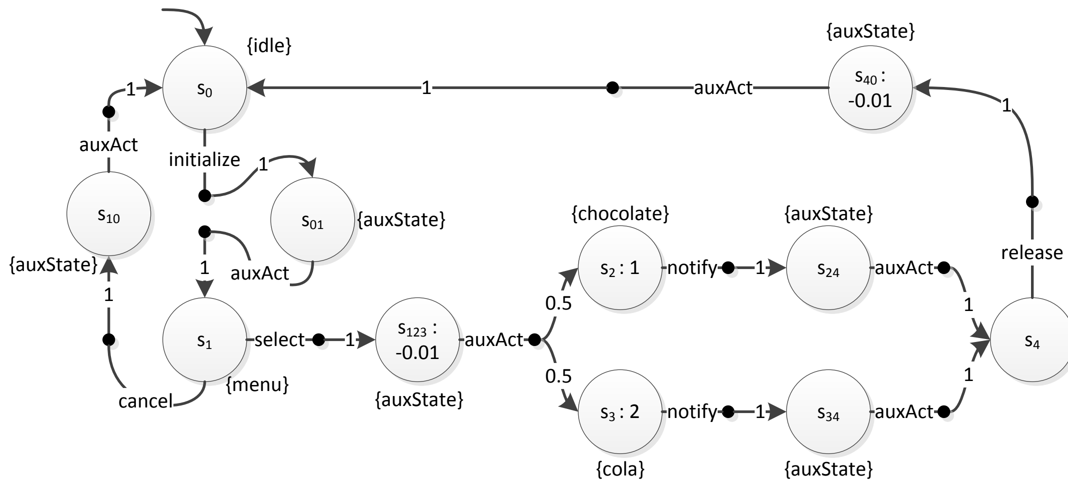


Figure 21: The general action rewards elimination result MDPR

In order to cover more general cases, all the algorithms introduced in later sections will consider both state and action rewards.

5 PCTLR Model Checking over MDPRs

The detailed model checking algorithm for PCTL over MDPs can be found in [25] and it is extended in [10] for PCTL with state reward formulae over MDPRs. The later algorithm is almost identical with the model checking algorithm for PCTLR over DTMCRs. The key difference is how it handles the P and R^{state} operators. For a given MDPR $M = (S, \overline{s}, Act, Steps, L, r_s, r_a)$, a set of states satisfy the probability state formula $\mathrm{P}_{\sim p}[\phi]$ is defined as:

$$Sat(\mathrm{P}_{\sim p}[\phi]) = \begin{cases} \{s \in S \mid Pr^{\min}_{M,s}(\phi) \sim p\} & \sim \in \{>, \geq\} \\ \{s \in S \mid Pr^{\max}_{M,s}(\phi) \sim p\} & \sim \in \{<, \leq\} \end{cases}$$

$$Sat(\mathrm{R}^{state}_{\sim r}[\varphi]) = \begin{cases} \{s \in S \mid Exp^{\min}_{D,s}(X_\varphi) \sim r\} & \sim \in \{>, \geq\} \\ \{s \in S \mid Exp^{\max}_{D,s}(X_\varphi) \sim r\} & \sim \in \{<, \leq\} \end{cases}.$$

The detailed illustrations of the algorithm are also included in the above two papers and we will not cover them here. Similar to the situation of model checking the newly introduced PCTLR over DTMCRs, PCTLR also only brings in one new type of path formulae for MDPRs. Therefore, once both $Pr^{\min}_{M,s}(\mathrm{R}^{path}_{\sim r}[\varphi])$ and $Pr^{\min}_{M,s}(\mathrm{R}^{path}_{\sim r}[\varphi])$ can be computed for a given reward formula φ, the existing algorithm will be able to model check PCTLR over MDPRs without any modifications. In this section, we will discuss how to compute these probabilities given an MDPR and a path reward formula. As before, the algorithms introduced in this section will be able to handle negative reward values and compute accurate results if no exception is explicitly pointed out.

5.1 Computing $Pr_{M,s}(\mathbf{R}^{path}_{\sim r}[\mathbf{I}^{=k}])$

For a given MDPR $M = (S, \overline{s}, Act, Steps, L, r_s, r_a)$, let $x^{\min,k}_{M,s,\sim r}$ and $x^{\max,k}_{M,s,\sim r}$ denote $Pr^{\min}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$ and $Pr^{\max}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$ respectively. Then the recursive definitions of $x^{\min,k}_{M,s,\sim r}$ and $x^{\max,k}_{M,s,\sim r}$ are:

$$x^{\min,k}_{M,s,\sim r} = \begin{cases} 1 & \text{if } k = 0 \wedge r_s(s) \sim r \\ 0 & \text{if } k = 0 \wedge \neg(r_s(s) \sim r) \ , \\ \min_{a \in A(s)}\{\sum_{s' \in S} Steps(s,a)(s') \cdot x^{\min,k-1}_{D,s',\sim r}\} & \text{otherwise} \end{cases} \quad (5)$$

$$x^{\max,k}_{M,s,\sim r} = \begin{cases} 1 & \text{if } k = 0 \wedge r_s(s) \sim r \\ 0 & \text{if } k = 0 \wedge \neg(r_s(s) \sim r) \ . \\ \max_{a \in A(s)}\{\sum_{s' \in S} Steps(s,a)(s') \cdot x^{\max,k-1}_{D,s',\sim r}\} & \text{otherwise} \end{cases} \quad (6)$$

5.1.1 Top-down with Memoization

One way to solve Equations (5) and (6) is applying the top-down with memoization approach of dynamic programming, the algorithm is shown in Algorithm 24 and the auxiliary function PR-I-MDPR-AUX is shown in Algorithm 25.

Algorithm 24 MEMOIZED-PR-I-MDPR($M, \sim r, k, func$)

Input: MDPR M, reward bound $\sim r$, step bound k and function $func \in \{\min, \max\}$
Output: $Pr^{func}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$ for all $s \in S$

1: let p be an empty hash table whose key is composed by a state and a step value;
2: **for all** state $s \in S$ **do**
3: $x_s :=$ PR-I-MDPR-AUX($M, s, \sim r, k, func, p$);
4: **end for**
5: **return** $(x_s)_{s \in S}$;

Algorithm 25 Pr-I-MDPR-Aux$(M, s, \sim r, k, func, p)$

Input: MDPR M, state s, reward bound $\sim r$, step bound k, function $func \in \{\min, \max\}$ and hash table p
Output: $Pr^{func}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$

1: **if** $p(s,k)$ is not defined **then**
2: **if** $k = 0$ **then**
3: $p(s,k) := \begin{cases} 1 & \text{if } r_s(s) \sim r \\ 0 & \text{otherwise} \end{cases};$
4: **else**
5: $k' := k - 1;$
6: $p(s,k) := func_{a\in A(s)}\{\sum_{s'\in S} Steps(s,a)(s') \cdot \textsc{Pr-I-MDPR-Aux}(M, s', \sim r, k', func, p)\};$
7: **end if**
8: **end if**
9: **return** $p(s,k)$;

Example 31. To compute $Pr^{\max}_{M,s_0}(\mathrm{R}^{path}_{>1}[\mathrm{I}^{=2}])$ for the MDPR M in Figure 18. Let $x^{\max,k}_{M,s,\sim r}$ denotes $Pr^{\max}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$, by applying the top-down with memoization approach, we will derive the following equations from top to bottom:

$$\begin{aligned}
x^{\max,2}_{M,s_0,>1} &= \max\{1 \times x^{\max,1}_{M,s_1,>1}\} \\
x^{\max,1}_{M,s_1,>1} &= \max\{1 \times x^{\max,0}_{M,s_0,>1}, 0.5 \times x^{\max,0}_{M,s_2,>1} + 0.5 \times x^{\max,0}_{M,s_3,>1}\} \\
x^{\max,0}_{M,s_0,>1} &= 0 && r_s(s_0) = 0 \\
x^{\max,0}_{M,s_2,>1} &= 0 && r_s(s_2) = 1 \\
x^{\max,0}_{M,s_3,>1} &= 1, && r_s(s_4) = 2 > 1
\end{aligned}$$

and the algorithm will solve these equation from bottom to top, we have that:

$$\begin{aligned}
x^{\max,1}_{M,s_1,>1} &= \max\{1 \times 0, 0.5 \times 0 + 0.5 \times 1\} = \max\{0, 0.5\} = 0.5 \\
x^{\max,2}_{M,s_0,>1} &= \max\{1 \times 0.5\} = 0.5.
\end{aligned}$$

Hence the probability $Pr^{\max}_{M,s_0}(\mathrm{R}^{path}_{>1}[\mathrm{I}^{=2}])$ is 0.5. ■

5.1.2 Bottom-up Method

We can eliminate the recursive calls by applying the bottom-up method approach of dynamic programming to solve Equations (5) and (6), the algorithm is shown in Algorithm 26.

Example 32. Return to the MDPR M in Figure 18. Let $x^{\max,k}_{M,s,\sim r}$ denotes $Pr^{\max}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$, the row vector $\underline{x}^{\max,k}_{M,\sim r} = \{x^{\max,k}_{M,s_0,\sim r}, \cdots, x^{\max,k}_{M,s_4,\sim r}\}$. To compute $Pr^{\max}_{M,s}(\mathrm{R}^{path}_{>1}[\mathrm{I}^{=2}])$ for all state $s \in S$, we can apply the bottom-up method approach with the base case:

$$\underline{x}^{\max,0}_{M,>1} = \{0, 0, 0, 1, 0\},$$

then by using the Equation (6) for each state $s \in S$, we have:

$$\underline{x}^{\max,1}_{M,>1} = \{0, 0.5, 0, 0, 0\},$$

finally, the desired probabilities can be computed by reapplying the Equation (6):

$$\underline{x}^{\max,2}_{M,>1} = \{0.5, 0, 0, 0, 0\}.$$ ■

Algorithm 26 BOTTOM-UP-PR-I-MDPR($D, \sim r, k, func$)

Input: MDPR M, reward bound $\sim r$, step bound k and function $func \in \{\min, \max\}$
Output: $Pr^{func}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$ for all $s \in S$

1: **for all** state $s \in S$ **do**
2: $\quad x_s := \begin{cases} 1 & \text{if } r_s(s) \sim r \\ 0 & \text{otherwise} \end{cases};$
3: **end for**
4: **for** $i = 1 \to k$ **do**
5: $\quad$ **for all** state $s \in S$ **do**
6: $\quad\quad x'_s := func_{a \in A(s)}\{\sum_{s' \in S} Steps(s,a)(s') \cdot x_{s'}\};$
7: $\quad$ **end for**
8: $\quad$ **for all** state $s \in S$ **do** $x_s := x'_s$; **end for**
9: **end for**
10: **return** $(x_s)_{s \in S}$

Similar to the performance comparison between the top-down and bottom-up approaches for computing $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$ over DTMCRs. Both top-down and bottom-up approaches for MDPRs have the same asymptotic running time, while the top-down method has overheads of caching calculated values and recursive procedure calls and the bottom-up method may compute some unnecessary values.

5.2 Computing $Pr_{M,s}(\mathbf{R}^{path}_{\sim r}[\mathbf{C}^{\leq k}])$

For a given MDPR $M = (S, \overline{s}, Act, Steps, L, r_s, r_a)$, let $x^{\min,k}_{M,s,\sim r}$ and $x^{\max,k}_{M,s,\sim r}$ denote the probability $Pr^{\min}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{C}^{\leq k}])$ and $Pr^{\max}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{C}^{\leq k}])$ respectively. Then the recursive definitions of $x^{\min,k}_{M,s,\sim r}$ and $x^{\max,k}_{M,s,\sim r}$ are:

$$x^{\min,k}_{M,s,\sim r} = \begin{cases} 1 & \text{if } k = 0 \wedge 0 \sim r \\ 0 & \text{if } k = 0 \wedge \neg(0 \sim r) \\ \min_{a \in A(s)}\{\sum_{s' \in S} Steps(s,a)(s') \cdot x^{\min,k-1}_{M,s',\sim(r - r_{s \to a})}\} & \text{otherwise} \end{cases}, \quad (7)$$

$$x^{\max,k}_{M,s,\sim r} = \begin{cases} 1 & \text{if } k = 0 \wedge 0 \sim r \\ 0 & \text{if } k = 0 \wedge \neg(0 \sim r) \\ \max_{a \in A(s)}\{\sum_{s' \in S} Steps(s,a)(s') \cdot x^{\max,k-1}_{M,s',\sim(r - r_{s \to a})}\} & \text{otherwise} \end{cases}. \quad (8)$$

5.2.1 Top-down with Memoization

Algorithm 27 MEMOIZED-PR-C-MDPR($M, \sim r, k, func$)

Input: MDPR M, reward bound $\sim r$, step bound k and function $func \in \{\min, \max\}$
Output: $Pr^{func}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{C}^{\leq k}])$ for all $s \in S$

1: let p be an empty hash table whose key is composed by a state, a reward and a step value;
2: **for all** state $s \in S$ **do**
3: $\quad x_s :=$ PR-C-MDPR-AUX($M, s, \sim r, k, func, p$);
4: **end for**
5: **return** $(x_s)_{s \in S}$

One way to solve Equations (7) and (8) is applying the top-down with memoization approach of dynamic programming, the algorithm is shown in Algorithm 27 and the auxiliary function PR-C-MDPR-AUX is shown in Algorithm 28.

Algorithm 28 $\textsc{Pr-C-MDPR-Aux}(M, s, \sim r, k, func, p)$

Input: MDPR M, reward bound $\sim r$, step bound k, function $func \in \{\min, \max\}$ and hash table p
Output: $Pr^{func}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{C}^{\leq k}])$

1: **if** $p(s, r, k)$ is not defined **then**
2:     **if** $k = 0$ **then**
3:         $p(s, r, k) := \begin{cases} 1 & \text{if } 0 \sim r \\ 0 & \text{otherwise} \end{cases}$;
4:     **else**
5:         $k' := k - 1$;
6:         **for all** action $a \in A(s)$ **do**
7:             $r' := r - r_{s \to s'}$;
8:             $v_a := \sum_{s' \in S} Steps(s, a)(s') \cdot \textsc{Pr-C-MDPR-Aux}(M, s', \sim r', k', func, p)$;
9:         **end for**
10:        $p(s, r, k) := func_{a \in A(s)}\{v_a\}$;
11:    **end if**
12: **end if**
13: **return** $p(s, r, k)$;

Example 33. Let us go back to the MDPR M in Figure 18 and computate $Pr^{\max}_{M,s_0}(\mathrm{R}^{path}_{\geq 1}[\mathrm{C}^{\leq 3}])$. Let $x^{\max,k}_{M,s,\sim r}$ denotes $Pr^{\max}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{C}^{\leq k}])$, by applying the top-down with memoization approach, we will derive the following equations from top to bottom:

$$\begin{aligned}
x^{\max,3}_{M,s_0,\geq 1} &= \max\{x^{\max,2}_{M,s_1,\geq 1}\} \\
x^{\max,2}_{M,s_1,\geq 1} &= \max\{x^{\max,1}_{M,s_0,\geq 1}, 0.5 \times x^{\max,1}_{M,s_2,\geq 1.01} + 0.5 \times x^{\max,1}_{M,s_3,\geq 1.01}\} \\
x^{\max,1}_{M,s_0,\geq 1} &= \max\{x^{\max,0}_{M,s_1,\geq 1}\} \\
x^{\max,0}_{M,s_1,\geq 1} &= 0 && \neg(0 \geq 1) \\
x^{\max,1}_{M,s_2,\geq 1.01} &= \max\{x^{\max,0}_{M,s_4,\geq 0.01}\} \\
x^{\max,0}_{M,s_4,\geq 0.01} &= 0 && \neg(0 \geq 0.01) \\
x^{\max,1}_{M,s_3,\geq 1.01} &= \max\{x^{\max,0}_{M,s_4,\geq -0.99}\} \\
x^{\max,0}_{M,s_4,\geq -0.99} &= 1, && 0 \geq -0.99
\end{aligned}$$

and the algorithm will solve these equations from bottom to top, we have that:

$$\begin{aligned}
x^{\max,1}_{M,s_3,\geq 1.01} &= \max\{1\} = 1 \\
x^{\max,1}_{M,s_2,\geq 1.01} &= \max\{0\} = 0 \\
x^{\max,2}_{M,s_1,\geq 1} &= \max\{0, 0.5 \times 0 + 0.5 \times 1\} = 0.5 \\
x^{\max,3}_{M,s_0,\geq 1} &= \max\{0.5\} = 0.5.
\end{aligned}$$

Hence the probability $Pr^{\max}_{M,s_0}(\mathrm{R}^{path}_{\geq 1}[\mathrm{C}^{\leq 3}])$ is 0.5. ■

5.3 Computing $Pr_{M,s}(\mathbf{R}^{path}_{\sim r}[\mathbf{F}\ \Phi])$

For a given MDPR $M = (S, \overline{s}, Act, Steps, L, r_s, r_a)$, let $x^{\min}_{M,s,\sim r}$ and $x^{\max}_{M,s,\sim r}$ denote the probability $Pr^{\min}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$ and $Pr^{\max}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$ respectively. Then the recursive definitions of $x^{\min}_{M,s,\sim r}$

and $x^{\max}_{M,s,\sim r}$ are:

$$x^{\min}_{M,s,\sim r} = \begin{cases} 1 & \text{if } s \models \Phi \wedge 0 \sim r \\ 0 & \text{if } s \models \Phi \wedge \neg(0 \sim r) \ , \\ \min_{a \in A(s)} \{\sum_{s' \in S} Steps(s,a)(s') \cdot x^{\min}_{M,s',\sim(r - r_{s \to a})}\} & \text{otherwise} \end{cases} \quad (9)$$

$$x^{\max}_{M,s,\sim r} = \begin{cases} 1 & \text{if } s \models \Phi \wedge 0 \sim r \\ 0 & \text{if } s \models \Phi \wedge \neg(0 \sim r) \ . \\ \max_{a \in A(s)} \{\sum_{s' \in S} Steps(s,a)(s') \cdot x^{\max}_{M,s',\sim(r - r_{s \to a})}\} & \text{otherwise} \end{cases} \quad (10)$$

Similar to the recursive definition Equation (3) for DTMCRs, Equations (9) and (10) also have the potential problems of infinitely many unfolded equations and mutual dependences. For instance, assume that $0 \sim r_1$ is satisfied, by unfolding the recursive definition of $x^{\min}_{M,s_0,\sim r_1}$ based on the MDPR shown in Figure 22, if $r_2 = 0$, we will have:

$$\begin{aligned} x^{\min}_{M,s_0,\sim r_1} &= \min\{x^{\min}_{M,s_0,\sim r_1}, x^{\min}_{M,s_1,\sim r_1}\} \\ x^{\min}_{M,s_1,\sim r_1} &= 1, \end{aligned}$$

where there is a self dependence of $x^{\min}_{M,s_0,\sim r_1}$, and if $r_2 \neq 0$:

$$\begin{aligned} x^{\min}_{M,s_0,\sim r_1} &= \min\{x^{\min}_{M,s_0,\sim(r_1 - r_2)}, x^{\min}_{M,s_1,\sim r_1}\} \\ x^{\min}_{M,s_0,\sim(r_1 - r_2)} &= \min\{x^{\min}_{M,s_0,\sim(r_1 - 2r_2)}, x^{\min}_{M,s_1,\sim(r_1 - r_2)}\} \\ x^{\min}_{M,s_0,\sim(r_1 - 2r_2)} &= \min\{x^{\min}_{M,s_0,\sim(r_1 - 3r_2)}, x^{\min}_{M,s_1,\sim(r_1 - 2r_2)}\} \\ &\cdots \end{aligned}$$

where it ends up with infinitely many unfolded equations.

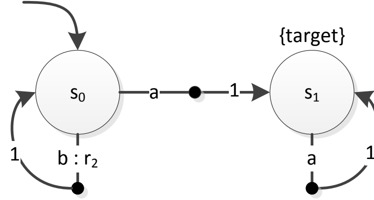


Figure 22: Potential problems for computing Equations (9) and (10)

To resolve theses problem, we can refine Equations (9) and (10) in the similar ways as we did for Equation (3) with some modifications due to the nondeterministic properties of MDPRS. Like the assumption we made for DTMCRs, the MDPRs considered in this section only have non-negative rewards.

5.3.1 Top-down with Memoization and Zero Strongly Connected Components

First we have to introduced two sets of states:

$$\begin{aligned} S^{\min,0}_{\Phi} &= \{s \in S \mid Pr^{\min}_{M,s}(\text{F } \Phi) = 0\} \\ S^{\max,0}_{\Phi} &= \{s \in S \mid Pr^{\max}_{M,s}(\text{F } \Phi) = 0\}, \end{aligned}$$

where $S^{\min,0}_{\Phi}$ contains all the states, which cannot reach any state satisfies Φ, for *some certain* adversaries and $S^{\max,0}_{\Phi}$ contains all the states, which cannot reach any state satisfies Φ, for *all possible* adversaries. As for any state s, if $Pr^{\max}_{M,s}(\text{F } \Phi) = 0$ then $Pr^{\min}_{M,s}(\text{F } \Phi) = 0$, hence $S^{\max,0}_{\Phi} \subseteq S^{\min,0}_{\Phi}$.

Algorithm 29 can be used to find either of these sets, it is the combination of the Algorithm 1 and 3 in [23].

Algorithm 29 COMPUTE-S0-MDPR($M, \Phi, func$)

Input: MDPR M, PCTLR state formula Φ and function $func \in \{\min, \max\}$
Output: $S^0_\Phi = \{s \in S \mid Pr^{func}_{M,s}(\mathrm{F}\ \Phi) = 0\}$

```
1: R := Sat(Φ);
2: repeat
3:     R' := R;
4:     if func = min then
5:         R := R ∪ {s ∈ S\R | ∀a ∈ A(s).(∃s' ∈ R.(Steps(s,a)(s') > 0))};
6:     else
7:         R := R ∪ {s ∈ S\R | ∃a ∈ A(s).(∃s' ∈ R.(Steps(s,a)(s') > 0))};
8:     end if
9: until R = R'
10: return S\R;
```

Once we have these two sets, two more base cases can be introduced for each of the recursive definition of $x^{\min}_{M,s,\sim r}$ and $x^{\max}_{M,s,\sim r}$:

$$x^{\min}_{M,s,\sim r} = \begin{cases} 1 & \text{if } s \in S^{\max,0}_\Phi \wedge \infty \sim r \\ 0 & \text{if } s \in S^{\min,0}_\Phi \wedge \neg(\infty \sim r) \end{cases},$$

$$x^{\max}_{M,s,\sim r} = \begin{cases} 1 & \text{if } s \in S^{\min,0}_\Phi \wedge \infty \sim r \\ 0 & \text{if } s \in S^{\max,0}_\Phi \wedge \neg(\infty \sim r) \end{cases}.$$

Please pay attention to the ways how $S^{\min,0}_\Phi$ and $S^{\max,0}_\Phi$ are used above. As for all state $s \in S^{\max,0}_\Phi$, by the definition of the random variable $X_{\mathrm{F}\Phi}$, $\forall \omega \in Path_{M,s}.(X_{\mathrm{F}\Phi}(\omega) = \infty)$, thus both $x^{\min}_{M,s,\sim r}$ and $x^{\max}_{M,s,\sim r}$ can be defined with a fixed value based on the truth value of $\infty \sim r$. On the other hand, for all state $s \in S^{\min,0}_\Phi$:

$$\exists \sigma \in Adv.\forall \omega \in Path_{M^\sigma,s}.(X_{\mathrm{F}\Phi}(\omega) = \infty),$$

where:

$$Path_{M^\sigma,s} \stackrel{def}{=} \bigcup_{\substack{\pi \in Path^{fin}_M \\ last(\pi) = s}} Path_{M^\sigma,\pi}.$$

Also both the possible values of a given $x^{\min}_{M,s,\sim r}$ and a given $x^{\max}_{M,s,\sim r}$ are within the interval $[0,1]$ and they should be defined with the minimum and the maximum of all possible values respectively. If we stop unfolding the recursive definition of $x^{\min}_{M,s,\sim r}$ and $x^{\max}_{M,s,\sim r}$ once $s \in S^{\min,0}_\Phi$, they will be defined with a fixed value of either 0 or 1. Otherwise, they will be defined with a value within the interval $[0,1]$ which might differ from 0 or 1. Therefore, for a given $x^{\min}_{M,s,\sim r}$ where $s \in S^{\min,0}_\Phi$, it should be defined as 0 if $\neg(\infty \sim r)$ is satisfied as 0 is the minimum possible value it can be, otherwise the recursive definition should be applied to cover the possible values which is smaller than 1. The case for a given $x^{\min}_{M,s,\sim r}$ where $s \in S^{\min,0}_\Phi$ is similar, it should be defined as 1 directly if $\infty \sim r$ is satisfied, otherwise the recursive definition should be applied.

Next, we define the Zero Strongly Connected Component of a MDPR as follows.

Definition 13. Let $M = (S, \overline{s}, Act, Steps, L, r_s, r_a)$ be an MDPR. A Zero Strongly Connected Component (ZSCC) of M based on a state reward formula Φ is a subset of states $Z_M \subseteq S$ satisfying the following conditions:

- for all state $s \in Z_M$, $s \not\models \Phi$ and $r_s(s) = 0$;

- for every pair of states $s_i, s_j \in Z_M$, where s_i can be the same as s_j, there is a finite path π from s_i to s_j, where for any state s appears on this path, $s \in Z_M$ and $r_\pi = 0$;
- if $|Z_M| = 1$, then the only state s in Z_M must satisfies:

$$\exists a \in A(s).(Steps(s,a)(s) > 0 \wedge r_a(s,a) = 0),$$

  this ensures the algorithm will not use the relatively more complicated way to handle states which do not have to;
- Z_M is a maximal subcomponent of M.

Like the ZSCC related concepts of DTMCRs, $M.ZSCC(\Phi)$ denotes a set of all the ZSCCs of M based on Φ and $Z^{\Phi}_{M,s}$ represents the ZSCC contains s where $Z^{\Phi}_{M,s} \in M.ZSCC(\Phi)$. To simplify the formulae introduced in the later part of this paper, a state s belongs to a ZSCC in $M.ZSCC(\Phi)$ is denoted as follows:

$$s \in M.ZSCC(\Phi) \quad \Leftrightarrow \quad \exists Z_M \in M.ZSCC(\Phi).(s \in Z_M). \qquad \blacksquare$$

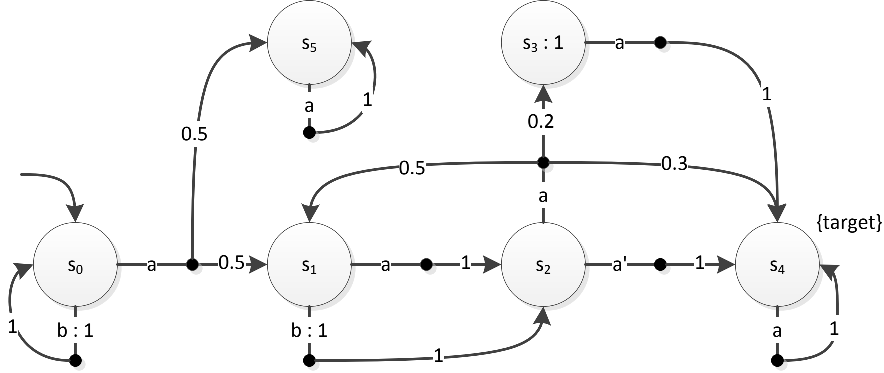


Figure 23: A MDPR with a ZSCC

Example 34. Based on the PCTLR state formula $\Phi = target$, the MDPR shown in Figure 23 has two ZSCCs: $\{s_1, s_2\}$ and $\{s_5\}$. ■

Algorithm 30 DETECT-ZSCC-MDPR(M, Φ)

Input: MDPR M and PCTL state formula Φ
Output: $M.ZSCC(\Phi)$
1: Construct the direct graph $G = (V, E)$, where:
  - $V := \{s \mid s \notin \Phi \wedge r_s(s) = 0\}$;
  - $E := \{(s_i, s_j) \mid s_i \in V \wedge s_j \in V \wedge \exists a \in Act(s_i).(r_a(s_i, a) = 0 \wedge Steps(s_i, a)(s_j) > 0)\}$;

2: **for all** SCC found by calling STRONGLY-CONNECTED-COMPONENT(G) **do**
3:     **if** it is not the case that $|SCC| = 1$ and the only state in SCC does not have a self loop **then**
4:         $M.ZSCC(\Phi) := M.ZSCC(\Phi) \cup \{SCC\}$;
5:     **end if**
6: **end for**
7: **return** $M.ZSCC(\Phi)$

Algorithm 30, which is based on the STRONGLY-CONNECTED-COMPONENT algorithm from [20], provides one way to calculate $M.ZSCC(\Phi)$ for a given MDPR M and a given PCTLR state formula Φ. It is similar to Algorithm 10 and they only differ from how they construct the edges of the reduced graphs. This algorithm considers actions and the one before considers transitions.

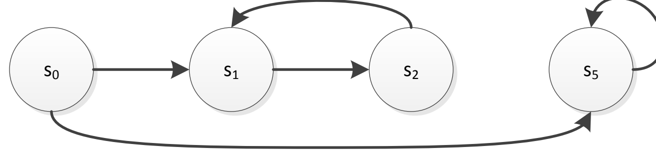


Figure 24: The reduced graph of the MDPR shown in Figure 23

Example 35. To find all the ZSCCs of the MDPR M shown in Figure 23 based on the state formula $\Phi = target$. First the reduced graph of M is constructed as shown in Figure 24. Then it is passed to STRONGLY-CONNECTED-COMPONENT as the parameter. The SCC algorithm will return three SCCs: $\{s_0\}$, $\{s_1, s_2\}$ and s_5. $\{s_0\}$ is not a ZSCC as it only contains one state which does not have a self loop in the reduced graph. Therefore, the result set $M.ZSCC(\Phi) = \{\{s_1, s_2\}, \{s_5\}\}$. ■

For a given MDPR $M = (S, \overline{s}, Act, Steps, L, r_s, r_a)$, let $Z_M \in M.ZSCC(\Phi)$, for a PCTLR state formula Φ:

$$Z_M.Out \stackrel{def}{=} \{s \in S \mid \exists s' \in Z_M.\exists a \in A(s).((Steps(s', a)(s) > 0) \wedge (r_a(s', a) \neq 0 \vee s \notin Z_M))\},$$

and we have to modify the MDPR M by adding some auxiliary states to ensure:

- $\forall s \in Z_M.Out.(s \notin Z_M)$;
- $\forall s \in Z_M.\forall \omega \in Path_{M,s}.(\omega(i) \models \Phi' \Rightarrow X_{\mathrm{F}\Phi'}(\omega) = 0)$, where $s \models \Phi' \Leftrightarrow s \in Z_M.Out$.

The algorithm to add these auxiliary states for Z_M is shown in Algorithm 31. Note that this procedure only modifies $Z_M.Out$ but not Z_M itself.

Algorithm 31 ADD-AUX-STATE-MDPR($M, M.ZSCC(\Phi)$)

Input: MDPR M and set $M.ZSCC(\Phi)$
Output: M after the modification

1: **for all** $Z_M \in M.ZSCC(\Phi)$ **do**
2:     **for all** state $s_i \in Z_M$ **do**
3:         **for all** action $a \in A(s_i)$ where $r_a(s_i, a) \neq 0$ **do**
4:             $M :=$ ADD-INTERMEDIATE-STATE(M, s, a);
5:         **end for**
6:     **end for**
7: **end for**
8: **return** M;

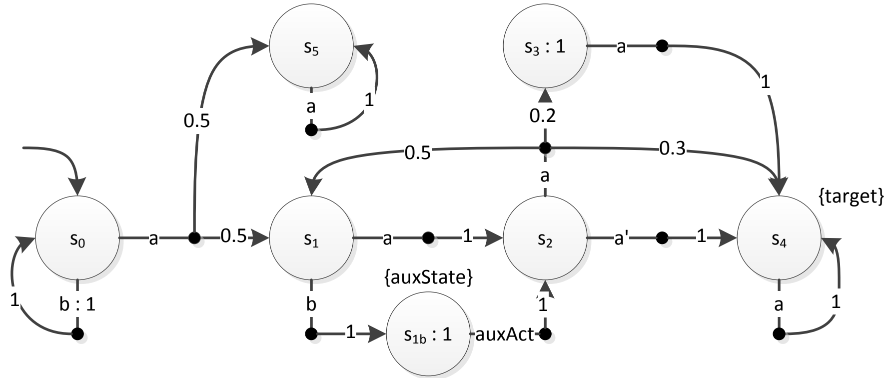


Figure 25: The modified result of the MDPR in Figure 23

Example 36. Figure 25 shows the model in Figure 23 after insert the auxiliary state s_{1b} for action b at state s_1. ■

To sum all the above concepts up, the recursive definition of $x^{\max}_{M,s,\sim r}$ is refined as follows after compute $T = Sat(\Phi)$, $S^{min,0}_{\Phi}$, $S^{max,0}_{\Phi}$ and $M.ZSCC(\Phi)$ and add necessary auxiliary sates for a given MDPR M:

$$x^{\min}_{M,s,\sim r} = \begin{cases} 1 & \text{if } \begin{pmatrix} (s \in T \wedge 0 \sim r) \\ \vee (s \in S^{\max,0}_{\Phi} \wedge \infty \sim r) \\ \vee (\sim \text{is} > \wedge\, r < 0) \\ \vee (\sim \text{is} \geq \wedge\, r \leq 0) \end{pmatrix} & \text{(11a)} \\ 0 & \text{if } \begin{pmatrix} (s \in T \wedge \neg(0 \sim r)) \\ \vee (s \in S^{\min,0}_{\Phi} \wedge \neg(\infty \sim r)) \\ \vee (\sim \text{is} < \wedge\, r \leq 0) \\ \vee (\sim \text{is} \leq \wedge\, r < 0) \end{pmatrix} & \text{(11b)} \\ \min\limits_{\sigma \in Adv} \{ \sum\limits_{s' \in Z^{\Phi}_{M,s}.Out} Pr^{\sigma}_{M,s}(Z^{\Phi}_{M,s} \text{ U } s') \cdot x^{\min}_{M,s',\sim r} \} & \text{if } s \in M.ZSCC(\Phi) & \text{(11c)} \\ \min\limits_{a \in A(s)} \{ \sum\limits_{s' \in S} Steps(s,a)(s') \cdot x^{\min}_{M,s',\sim(r - r_{s \to a})} \} & \text{otherwise} & \text{(11d)} \end{cases}$$

where:

$$Pr^{\sigma}_{M,s}(Z^{\Phi}_{M,s} \text{ U } s') \stackrel{def}{=} Pr^{\sigma}_{M,s}(\Phi_1 \text{ U } \Phi_2),$$

where for any state $s'' \in S$:

$$\begin{array}{ccc} s'' \models \Phi_1 & \Leftrightarrow & s'' \in Z^{\Phi}_{M,s} \\ s'' \models \Phi_2 & \Leftrightarrow & s'' = s' \end{array}$$

Here are some further explanations of the above recursive definition:

- Case (11a) and (11b) are the base cases where $x^{\min}_{M,s,\sim r}$ can be defined with the fixed value 1 or 0 where the four conditions ($\sim$ is $> \wedge\, r < 0$), ($\sim$ is $\geq \wedge\, r \leq 0$), ($\sim$ is $< \wedge\, r \leq 0$) and ($\sim$ is $\leq \wedge\, r < 0$) are introduced to prevent the infinitely many unfolded equations problem as for DTMCRs.
- Case (11c) is taken when the subscript s of $x^{\min}_{M,s,\sim r}$ belongs to a ZSCC $Z^{\Phi}_{M,s}$. It ensures that there is no mutual dependences problems among the unfolded equations. Once for all the state $s' \in Z^{\Phi}_{M,s}.Out$, $x^{\min}_{M,s',\sim r}$ is computed, we can compute the formula

  $$\min_{\sigma \in Adv} \{ \sum_{s' \in Z_M.Out} Pr^{\sigma}_{M,s}(Z^{\Phi}_{M,s} \text{ U } s') \cdot x^{\min}_{M,s',\sim r} \}$$

  by either solving a Linear Programming (LP) problem to get a theoretically accurate result (the actual result may still be approximate due to the inaccuracy of real number computations in practice) or using a value iteration algorithm to get an approximate result. For the former approach, the following LP problem needs to be solved:

  $$\text{maximize} \sum_{s \in Z^{\Phi}_{M,s}} x^{\min}_{M,s,\sim r} \text{ subject to the constraints:}$$

  $$x^{\min}_{M,s,\sim r} = \text{ the precomputed value} \qquad \text{for all } s \in Z^{\Phi}_{M,s}.Out$$

  $$x^{\min}_{M,s,\sim r} \leq \sum_{s' \in Z^{\Phi}_{M,s} \cup Z^{\Phi}_{M,s}.Out} Steps(s,a)(s') \cdot x^{\min}_{M,s',\sim r} \qquad \text{for all } s \in Z^{\Phi}_{M,s} \text{ and } a \in Act(s)$$

and it is demonstrated in [26] that it has a unique solution. For the latter approach, let us introduce variables y_s^n for all state $s \in Z^{\Phi}_{M,s} \cup Z^{\Phi}_{M,s}.Out$, $n \in \mathbb{N}$ and equations:

$$y_s^n = \begin{cases} \text{the precomputed value} & \text{if } s \in Z^{\Phi}_{M,s}.Out \\ 0 & \text{if } s \in Z^{\Phi}_{M,s} \wedge n = 0 \\ \min_{a \in Act(a)} \sum_{s' \in Z_M \cup Z_M.Out} Steps(s,a)(s') \cdot y_{s'}^{n-1} & \text{otherwise} \end{cases}$$

It is shown in [27] that $\lim_{n\to\infty} y_s^n = \min_{\sigma \in Adv}\{\sum_{s' \in Z_M.Out} Pr^{\sigma}_{M,s}(Z^{\Phi}_{M,s} \text{ U } s') \cdot x^{\min}_{M,s',\sim r}\}$, hence we can approximate the $\lim_{n\to\infty} y_s^n$ for a sufficiently large n. Algorithm 32 provides one way doing the value iteration based on the above equations to compute the desired value. Notice that instead of updating all the variables for each iteration, a variable is only updated once the values of its dependent variables (those used to compute it) are changed. Though solve the LP problem will give a relatively more accurate solution, the value iteration approach offers a better scalability and the approximation is good enough in practice. In this paper, we will go for the value iteration approach.

- Case (11d) is the same as the recursive case of Equation (9).
- Note that there are some cases intersect with each other. Though the final result will be the same by taking any of the intersected cases, consider the cases with the order from (11a) to (11d) will yield a better performance and some optimizations introduced in the later part also based on this order.

By replacing all the appearances of min with max and max with min in Equation (11), the refined recursive definition of $x^{\max}_{M,s,\sim r}$, after computing $T = Sat(\Phi)$, $S_{\Phi}^{min,0}$, $S_{\Phi}^{max,0}$ and $M.ZSCC(\Phi)$ and adding necessary auxiliary sates for a given MDPR M, can be derived as follows:

$$x^{\max}_{M,s,\sim r} = \begin{cases} 1 & \text{if } \begin{pmatrix} (s \in T \wedge 0 \sim r) \\ \vee (s \in S_{\Phi}^{\min,0} \wedge \infty \sim r) \\ \vee (\sim \text{is} > \wedge\ r < 0) \\ \vee (\sim \text{is} \geq \wedge\ r \leq 0) \end{pmatrix} & \text{(12a)} \\ 0 & \text{if } \begin{pmatrix} (s \in T \wedge \neg(0 \sim r)) \\ \vee (s \in S_{\Phi}^{\max,0} \wedge \neg(\infty \sim r)) \\ \vee (\sim \text{is} < \wedge\ r \leq 0) \\ \vee (\sim \text{is} \leq \wedge\ r < 0) \end{pmatrix} & \text{(12b)} \\ \max\limits_{\sigma \in Adv}\{\sum\limits_{s' \in Z^{\Phi}_{M,s}.Out} Pr^{\sigma}_{M,s}(Z^{\Phi}_{M,s} \text{ U } s') \cdot x^{\max}_{M,s',\sim r}\} & \text{if } s \in M.ZSCC(\Phi) & \text{(12c)} \\ \max\limits_{a \in A(s)}\{\sum\limits_{s' \in S} Steps(s,a)(s') \cdot x^{\max}_{M,s',\sim(r - r_{s\to a})}\} & \text{otherwise} & \text{(12d)} \end{cases}$$

where Case (12c) can also be handled either by solving a LP problem:

$$\text{minimize} \sum_{s \in Z^{\Phi}_{M,s}} x^{\max}_{M,s,\sim r} \text{ subject to the constraints:}$$

$$x^{\max}_{M,s,\sim r} = \text{ the precomputed value} \qquad \text{for all } s \in Z^{\Phi}_{M,s}.Out$$

$$x^{\max}_{M,s,\sim r} \geq \sum_{s' \in Z^{\Phi}_{M,s} \cup Z^{\Phi}_{M,s}.Out} Steps(s,a)(s') \cdot x^{\max}_{M,s',\sim r} \qquad \text{for all } s \in Z^{\Phi}_{M,s} \text{ and } a \in Act(s)$$

or by Algorithm 32 using the value iteration approach. In this paper, we will go for the latter one.

In order to simplify the calculations of Equation (11c) and (12c). All the ZSCCs of MDPR based on a PCTLR state formula can be further split into several disjoint subsets and different approaches can be applied to each subset for better performances.

Algorithm 32 VALUE-ITERATION-ZSCC^n-CASE$(M, Z_M, (x^{func}_{M,s,\sim r})_{s \in Z_M.Out}, func, \epsilon)$

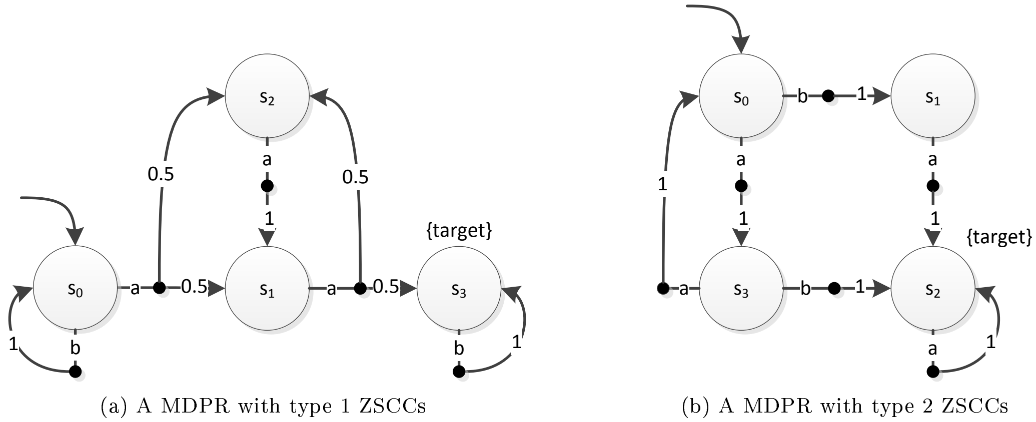
Input: MDPR M, reward bound $\sim r$, precomputed $x^{func}_{M,s,\sim r}$ for all the states $s \in Z_M.Out$, function $func \in \{\min, \max\}$ and convergence criterion ϵ
Output: approximate $x^{func}_{M,s,\sim r}$ for all the states $s \in Z_M$

1: **for all** state $s \in Z_M.Out$ **do** $x_s := x^{func}_{M,s,\sim r}$; **end for**
2: **for all** state $s \in Z_M$ **do** $x_s := 0$; **end for**
3: Let Q be a FIFO queue which disregards the insertion of existing elements in it;
4: Push all the states in Z_M into queue Q;
5: **while** Q is not empty **do**
6: $s :=$ pop the head of Q;
7: $x'_s := func_{a \in A(s)} \sum_{s' \in Z_M \cup Z_M.Out} Steps(s,a)(s') \cdot x_{s'}$;
8: **if** $|x_s - x'_s| > \epsilon$ **then**
9: Push all the states $s' \in Z_M$ where $\exists a \in A(s').(Steps(s',a)(s) > 0)$ into queue Q;
10: **end if**
11: $x_s := x'_s$;
12: **end while**
13: **return** $(x_s)_{s \in Z_M}$

Definition 14. For all the ZSCCs $Z_M \in M.ZSCC(\Phi)$ of a given MDPR M and a state formula Φ, they can be split into following disjoint subsets:

- if $|Z_M| = 1$ or $|Z_M.Out| = 1$, then $Z_M \in M.ZSCC^1(\Phi)$;
- else if $|Z_M.Out| = 2$, then $Z_M \in M.ZSCC^2(\Phi)$;
- the rest ZSCCs belong to $M.ZSCC^n(\Phi)$.

A ZSCC set with multiple superscripts separated by commas represents the union of the corresponding subsets. For instance, $M.ZSCC^{1,n}(\Phi) = M.ZSCC^1(\Phi) \cup M.ZSCC^n(\Phi)$. ■



(a) A MDPR with type 1 ZSCCs (b) A MDPR with type 2 ZSCCs

Figure 26: MDPRs contain special types ZSCCs

Example 37. Consider the PCTLR state formula $\Phi = target$, for the MDPR M_1 in Figure 26 (a):

$$M_1.ZSCC^1(\Phi) = \{\{s_0\}, \{s_1, s_2\}\},$$

where $|Z^{\Phi}_{M_1,s_0}| = 1$ and $|Z^{\Phi}_{M_1,s_1}.Out| = |\{s_3\}| = 1$. For the MDPR M_2 in Figure 26 (b):

$$M_2.ZSCC^2(\Phi) = \{\{s_0, s_3\}\},$$

where $|Z^{\Phi}_{M_2,s_0}.Out| = |\{s_1, s_2\}| = 2|$. ■

For a given $x^{func}_{M,s,\sim r}$ where $func \in \{\min, \max\}$ and a PCTLR state formula Φ, based on the order we consider the recursive definition cases, the case where $s \in M.ZSCC(\Phi)$ will only be considered if $s \notin S^{\max,0}_{\Phi}$. Also when $s \in S^{\min,0}_{\Phi}$, the ZSCC case will only be considered if some adversaries, which ensure M will finally reach a target state (leave $Z^{\Phi}_{M,s}$), might define $x^{func}_{M,s,\sim r}$ with a better value than not reaching any target state forever. Therefore, for the ZSCC case, only the adversaries ensure M leave $Z^{\Phi}_{M,s}$ need to be considered.

When $Z^{\Phi}_{M,s} \in M.ZSCC^1(\Phi)$:

- if $|Z^{\Phi}_{M,s}.Out| = 1$ and let $\{s'\} = Z^{\Phi}_{M,s}.Out$, as M finally will leave $Z^{\Phi}_{M,s}$ and s' is the only state it will go with probability 1, therefore, we can define $x^{func}_{M,s,\sim r}$ directly with $x^{func}_{M,s',\sim r}$;
- if $|Z^{\Phi}_{M,s}| = 1$, M will finally leave $Z^{\Phi}_{M,s}$ through one of the action $a \in Act(s)$, therefore, we can ensure $x^{func}_{M,s,\sim r}$ is defined with the minimum / maximum value by choosing the adversary, which guarantees M will leave with the action provides the minimum / maximum value with probability 1.

The following equation covers both the above scenarios:

$$func_{a \in Act(s)}\{ \sum_{s' \in Z^{\Phi}_{M,s}.Out} \frac{Steps(s,a)(s')}{\sum_{s' \in Z^{\Phi}_{M,s}.Out} Steps(s,a)(s')} \times x_{s'} \},$$

and it will be used as the special approach to handle states $s \in M.ZSCC^1(\Phi)$.

When $Z^{\Phi}_{M,s} \in M.ZSCC^2$, let $\{s_1, s_2\} = Z^{\Phi}_{M,s}$ and assume that $x^{func}_{M,s_1,\sim r} \geq x^{func}_{M,s_2,\sim r}$. Because M finally will leave $Z^{\Phi}_{M,s}$ through either s_1 or s_2 and for each adversary σ:

$$Pr^{\sigma}_{M,s}(Z^{\Phi}_{M,s} \text{ U } s_1) + Pr^{\sigma}_{M,s}(Z^{\Phi}_{M,s} \text{ U } s_2) = 1.$$

Therefore, we can maximize $Pr^{\sigma}_{M,s}(Z^{\Phi}_{M,s} \text{ U } s_1)$ or $Pr^{\sigma}_{M,s}(Z^{\Phi}_{M,s} \text{ U } s_2)$ to achieve the maximum / minimum value of $x^{func}_{M,s,\sim r}$. Let $Pr^{\max}_{M,s_1}(Z^{\Phi}_{M,s} \text{ U } s_2)$ denotes the maximum probabilities of $Pr^{\sigma}_{M,s_1}(Z^{\Phi}_{M,s} \text{ U } s_2)$ over all adversaries and it is defined as follows:

$$Pr^{\max}_{M,s_1}(Z^{\Phi}_{M,s} \text{ U } s_2) \stackrel{def}{=} \max_{\sigma \in Adv} Pr^{\sigma}_{M,s_1}(Z^{\Phi}_{M,s} \text{ U } s_2),$$

the algorithm used to handle states $s \in M.ZSCC^2(\Phi)$ is shown in Algorithm 33. Note that we can only maximize either probabilities to achieve the minimum / maximum value instead of minimum the other probability. This is because for the case $s \in S^{\min,0}_{\Phi}$, an adversary σ, which gives the minimum value of one probability, will make both probability $Pr^{\sigma}_{M,s}(\text{F } s_1)$ and $Pr^{\sigma}_{M,s}(\text{F } s_2)$ to be 0 at the same time.

Algorithm 33 COMPUTE-ZSCC^2-CASE$(M, s, Z_M, (x^{func}_{M,s',\sim r})_{s' \in Z_M.Out}, func)$

Input: MDPR M, state s, ZSCC Z_M, precomputed $x^{func}_{M,s',\sim r}$ for all the states $s' \in Z_M.Out$ and function $func \in \{\min, \max\}$

Output: $x^{func}_{M,s,\sim r}$

1: Let $\{s_1, s_2\}$ be $Z_M.Out$, where $x^{func}_{M,s_1,\sim r} \geq x^{func}_{M,s_2,\sim r}$;
2: **if** $func = \min$ **then**
3: $x := Pr^{\max}_{M,s}(Z_M \text{ U } s_2) \cdot x^{func}_{M,s_2,\sim r} + (1 - Pr^{\max}_{M,s}(Z_M \text{ U } s_2)) \cdot x^{func}_{M,s_1,\sim r}$;
4: **else**
5: $x := Pr^{\max}_{M,s}(Z_M \text{ U } s_1) \cdot x^{func}_{M,s_1,\sim r} + (1 - Pr^{\max}_{M,s}(Z_M \text{ U } s_1)) \cdot x^{func}_{M,s_2,\sim r}$;
6: **end if**
7: **return** x

Algorithm 34 MEMOIZED-PR-F-MDPR($M, \sim r, \Phi, func, \epsilon$)

Input: MDPR M, reward bound $\sim r$, PCTL state formula Φ, function $func \in \{\min, \max\}$ and convergence criterion ϵ
Output: $Pr^{func}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$ for all $s \in S$

1: $T := Sat(\Phi)$;
2: $S^{\min,0}_{\Phi} :=$ COMPUTE-S0-MDPR($M, \Phi, \min$);
3: $S^{\max,0}_{\Phi} :=$ COMPUTE-S0-MDPR($M, \Phi, \max$);
4: $M.ZSCC(\Phi) :=$ DETECT-ZSCC-MDPR(M, Φ);
5: $M :=$ ADD-AUX-STATE-MDPR($M, M.ZSCC(\Phi)$);
6: let p be an empty hash table whose key is composed by a state and a reward value;
7: **for all** state $s \in S$ **do**
8: $\quad x_s :=$ PR-F-MDPR-AUX($M, S^{\min,0}_{\Phi}, S^{\max,0}_{\Phi}, s, \sim r, T, func, p, \epsilon$);
9: **end for**
10: **return** $(x_s)_{s \in S}$

Algorithm 35 PR-F-MDPR-AUX($M, S^{\min,0}_{\Phi}, S^{\max,0}_{\Phi}, s, \sim r, T, func, p, \epsilon$)

Input: MDPR M, set of states $S^{\min,0}_{\Phi}$ and $S^{\max,0}_{\Phi}$, reward bound $\sim r$, set of target states T, function $func \in \{\min, \max\}$, hash table p and convergence criterion ϵ
Output: $Pr^{func}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$

1: **if** $p(s, r)$ is not defined **then**
2: $\quad$ **if** $\begin{pmatrix} (s \in T \wedge 0 \sim r) \\ \vee (s \in S^{\neg func,0}_{\Phi} \wedge \infty \sim r) \\ \vee (\sim \text{is} > \wedge\ r < 0) \\ \vee (\sim \text{is} \geq \wedge\ r \leq 0) \end{pmatrix}$ **then** $\quad \triangleright\ \neg\min = \max,\ \neg\max = \min$
3: $\quad\quad p(s, r) := 1$;
4: $\quad$ **else if** $\begin{pmatrix} (s \in T \wedge \neg(0 \sim r)) \\ \vee (s \in S^{func,0}_{\Phi} \wedge \neg(\infty \sim r)) \\ \vee (\sim \text{is} < \wedge\ r \leq 0) \\ \vee (\sim \text{is} \leq \wedge\ r < 0) \end{pmatrix}$ **then**
5: $\quad\quad p(s, r) := 0$;
6: $\quad$ **else if** $s \in M.ZSCC(\Phi)$ **then**
7: $\quad\quad (x_{s'})_{s' \in Z^{\Phi}_{M,s}.Out} :=$ (PR-F-MDPR-AUX($M, S^{\min,0}_{\Phi}, S^{\max,0}_{\Phi}, s', \sim r, T, func, p, \epsilon$))$_{s' \in Z^{\Phi}_{M,s}}$;
8: $\quad\quad$ **if** $s \in M.ZSCC^1(\Phi)$ **then**
9: $\quad\quad\quad p(s, r) := func_{a \in Act(s)}\{\sum_{s' \in Z^{\Phi}_{M,s}.Out} \frac{Steps(s,a)(s')}{\sum_{s' \in Z^{\Phi}_{M,s}.Out} Steps(s,a)(s')} \times x_{s'}\}$;
10: $\quad\quad$ **else if** $s \in M.ZSCC^2(\Phi)$ **then**
11: $\quad\quad\quad p(s, r) :=$ COMPUTE-ZSCC^2-CASE($M, s, Z^{\Phi}_{M,s}, (x_{s'})_{s' \in Z^{\Phi}_{M,s}.Out}, func$);
12: $\quad\quad$ **else**
13: $\quad\quad\quad (p(s', r))_{s' \in Z_M} :=$ VALUE-ITERATION-ZSCC^n-CASE($M, \sim r, (x'_s)_{s' \in Z^{\Phi}_{M,s}.Out}, func, \epsilon$);
14: $\quad\quad$ **end if**
15: $\quad$ **else**
16: $\quad\quad$ **for all** adversary $a \in A(s)$ **do**
17: $\quad\quad\quad r' := r - r_s(s) - r_a(s, a)$;
18: $\quad\quad\quad v_a := \sum_{s' \in S} Steps(s, a)(s') \cdot$ PR-F-MDPR-AUX($M, s', \sim r', \Phi, func, p$);
19: $\quad\quad$ **end for**
20: $\quad\quad p(s, r) = func_{a \in A(s)}\{v_a\}$;
21: $\quad$ **end if**
22: **end if**
23: **return** $p(s, r)$;

Now the top-down with momoization approach of dynamic programming can be applied to solve Equations (11) and (12), the algorithm is shown in Algorithm 34 and the auxiliary function PR-F-MDPR-AUX is introduced by Algorithm 35.

Example 38. To compute $Pr^{\min}_{M,s_0}(\mathrm{R}^{path}_{\geq 1}[\mathrm{F}\ target])$ for the MDPR M in Figure 25, we have:

$$\begin{aligned} T &= Sat(\Phi) = \{s_4\} \\ S^{\min,0}_{target} &= \{s_0, s_5\} \\ S^{\max,0}_{target} &= \{s_5\} \\ M.ZSCC(\Phi) &= \{\{s_1, s_2, s_3\}, \{s_5\}\} \end{aligned}$$

Let $x^{\min}_{M,s,\sim r}$ denotes $Pr^{\min}_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ target])$, by applying the top-down with memoization approach, we will derive the following equations from top to bottom:

$$\begin{aligned} x^{\min}_{M,s_0,\geq 1} &= \min\{x^{\min}_{M,s_0,\geq 0}, 0.5 \times x^{\min}_{M,s_1,\geq 1} + 0.5 \times x^{\min}_{M,s_5,\geq 1}\} \\ x^{\min}_{M,s_0,\geq 0} &= 1 && \sim \text{is} \geq \wedge\ r = 0 \leq 0 \\ x^{\min}_{M,s_1,\geq 1} &= \min_{\sigma \in Adv} \left\{ \begin{array}{r} Pr^{\sigma}_{M,s_1}(Z^{target}_{M,s_1}\ \mathrm{U}\ s_{1b}) \cdot x^{\min}_{M,s_{1b},\geq 1} \\ +Pr^{\sigma}_{M,s_1}(Z^{target}_{M,s_1}\ \mathrm{U}\ s_3) \cdot x^{\min}_{M,s_3,\geq 1} \\ +Pr^{\sigma}_{M,s_1}(Z^{target}_{M,s_1}\ \mathrm{U}\ s_4) \cdot x^{\min}_{M,s_4,\geq 1} \end{array} \right\} \\ x^{\min}_{M,s_{1b},\geq 1} &= \min\{x^{\min}_{M,s_2,\geq 0}\} \\ x^{\min}_{M,s_2,\geq 0} &= 1 && \sim \text{is} \geq \wedge\ r = 0 \leq 0 \\ x^{\min}_{M,s_3,\geq 1} &= \min\{x^{\min}_{M,s_4,\geq 0}\} \\ x^{\min}_{M,s_4,\geq 0} &= 1 && s_4 \in T \wedge 0 \geq 0 \\ x^{\min}_{M,s_4,\geq 1} &= 0 && s_4 \in T \wedge \neg(0 \geq 1) \\ x^{\min}_{M,s_5,\geq 1} &= 1 && s_5 \in S^{\max,0}_{target} \wedge \infty \geq 1 \end{aligned}$$

and the algorithm will solve these equation from bottom to top. We have that:

$$\begin{aligned} x^{\min}_{M,s_3,\geq 1} &= \min\{1\} = 1 \\ x^{\min}_{M,s_{1b},\geq 1} &= \min\{1\} = 1. \end{aligned}$$

Then VALUE-ITERATION-ZSCC^n-CASE is called to compute $x^{\min}_{M,s_1,\geq 1}$ with the convergence criterion $\epsilon = 10^{-6}$, let y_s denotes $x^{\min}_{M,s,\geq 1}$. The algorithm starts with the initial values: $y_{s_1} = 0$, $y_{s_2} = 0$, $y_{s_{1b}} = 1$, $y_{s_3} = 1$ and $y_{s_4} = 0$. After the first iteration: $y_{s_1} = 0$ and $y_{s_2} = 0$. The algorithm stops as no values are changed, i.e. the convergence criterion is satisfied. Therefore:

$$\begin{aligned} x^{\min}_{M,s_1,\geq 1} &= y_{s_1} = 0 \\ x^{\min}_{M,s_0,\geq 1} &= \min\{1, 0.5 \times 0 + 0.5 \times 1\} = 0.5. \end{aligned}$$

Thus the probability $Pr^{\min}_{M,s_0}(\mathrm{R}^{path}_{\geq 1}[\mathrm{F}\ target])$ is 0.5. ■

5.3.2 Optimizing Computations of $Pr^{\max}_{M,s}(Z^{\Phi}_{M,s}\ \mathrm{U}\ s')$

To compute $Pr^{\max}_{M,s}(Z^{\Phi}_{M,s}\ \mathrm{U}\ s')$ for a given MDPR $M = (S, \overline{s}, Act, Steps, L, r_s, r_a)$, we can reuse the algorithms for computations of $Pr^{\max}_{M,s}(\Phi_1\ \mathrm{U}\ \Phi_2)$ based on the definition of $Pr^{\max}_{M,s}(Z^{\Phi}_{M,s}\ \mathrm{U}\ s')$. In normal cases, $Pr^{\max}_{M,s}(\Phi_1\ \mathrm{U}\ \Phi_2)$ can be computed by either solving a linear programming problem or applying an iterative method, particularly Gauss-Seidel used by PRISM. Similar to the downsides for computing until probabilities for DTMCRs, the existing value iteration method updates too

Algorithm 36 COMPUTE-UNTIL-PROBS-MDPR($M, Z_M, s_0, func, \epsilon$)

Input: MDPR M, ZSCC Z_M, state $s_0 \in Z_M.Out$, function $func \in \{\min, \max\}$ and convergence criterion ϵ
Output: approximate $Pr^{func}_{M,s}(Z_M \text{ U } s')$ for all the states $s \in Z_M$

1: **for all** state $s \in Z_M \cup Z_M.Out$ **do** $x_s := 0$; **end for**
2: $x_{s_0} := 1$;
3: Let Q be a FIFO queue which disregards insertions of existing elements;
4: Push all the states in Z_M into queue Q;
5: **while** Q is not empty **do**
6: $\quad s :=$ pop the head of Q;
7: $\quad x'_s := func_{a \in A(s)} \sum_{s' \in Z_M \cup Z_M.Out} Steps(s,a)(s') \cdot x_{s'}$;
8: $\quad$ **if** $|x_s - x'_s| > \epsilon$ **then**
9: $\quad\quad$ Push all the states $s' \in Z_M$ where $\exists a \in A(s').(Steps(s',a)(s) > 0)$ into queue Q;
10: $\quad$ **end if**
11: $\quad x_s := x'_s$;
12: **end while**
13: **return** $(x_s)_{s \in Z_M}$

many unnecessary values (of those states $s' \notin Z^{\Phi}_{M,s}$). Therefore, a new iteration method, which only update values (of those states $s' \in Z^{\Phi}_{M,s}$) when it is necessary, is introduced in Algorithm 36. $Pr^{\max}_{M,s}(Z^{\Phi}_{M,s} \text{ U } s')$ for a given MDPR M with the convergence criterion ϵ can be computed by calling COMPUTE-UNTIL-PROBS-MDPR($M, Z^{\Phi}_{M,s}, s', max, \epsilon$).

The performance gain of the newly introduced algorithm can be analyzed with the similar case shown in Example 17.

5.3.3 Top-down with Memoization and Zero Connected Components

Instead of using the concept of ZSCCs, we can also use another type of connected components to compute the probability $Pr_{M,s}(\text{R}^{path}_{\sim r}[\text{F } \Phi])$.

Definition 15. Let $M = (S, \overline{s}, Act, Steps, L, r_s, r_a)$ be an MDPR. A Zero Connected Component (ZCC) of M based on the computation of $Pr^{func}_{M,s}(\text{R}^{path}_{\sim r}[\text{F } \Phi])$ is a subset of states $Z_M \subseteq S$ satisfying the following conditions:

- for all state $s \in Z_M$, $r_s(s) = 0$, $s \not\models \Phi$ and:
  - if $(func = \min \wedge \sim \in \{>, \geq\}) \vee (func = \max \wedge \sim \in \{<, \leq\})$, $s \notin S^{\max,0}_{\Phi}$,
  - otherwise, $s \notin S^{\min,0}_{\Phi}$;
- for every pair of distinct states s_i and s_j, there is a finite path π either from s_i to s_j or from s_j to s_i, any state appears on this path $s \in Z_M$ and $r_\pi = 0$;
- if $|Z_D| = 1$, then the only state $s \in Z_D$ must satisfies:

$$\exists a \in A(s).(Steps(s,a)(s) > 0 \wedge r_a(s,a) = 0);$$

- Z_M is a maximal subcomponent of M, i.e. there is no distinct ZCC Z'_M such that if $s \in Z_M$, then $s \in Z'_M$.

The notations of ZSCCs can be applied to ZCCs by replacing all the ZSCCs with ZCCs. One exception is that because ZCCs of a given MDPR do not only depend on a PCTLR state formula Φ, but also depend on a function $func \in \{\min, \max\}$ and a relational operator $\sim \in \{<, \leq, >, \geq\}$. Therefore, $M.ZSCC(\Phi)$ will be replaced by $M.ZCC(func, \sim, \Phi)$. ■

The algorithm used to compute $M.ZCC(func, \sim, \Phi)$ is almost identical with Algorithm 15. By applying the new exclusion rules for states and considering actions instead of transitions when constructing the reduced graph, we will achieve the new algorithm as shown in Algorithm 37.

Algorithm 37 DETECT-ZCC-MDPR$(M, S_\Phi^{\min,0}, S_\Phi^{\max,0}, \sim, \Phi, func)$

Input: MDPR M, set of states $S_\Phi^{\min,0}$ and $S_\Phi^{\max,0}$, relational operator $\sim \in \{<, \leq, >, \geq\}$, PCTL state formula Φ and function $func \in \{\min, \max\}$
Output: $M.ZCC(func, \sim, \Phi)$

```
 1: V := {s ∈ S | s ⊭ Φ ∧ r_s(s) = 0};
 2: if (func = min ∧ ~∈ {>, ≥}) ∨ (func = max ∧ ~∈ {<, ≤}) then
 3:     V := V\S_Φ^{max,0};
 4: else
 5:     V := V\S_Φ^{min,0};
 6: end if
 7: Construct the undirected graph G = (V, E), where:
        E := {(s_i, s_j) | s_i, s_j ∈ V ∧ ∃a ∈ A(s_i).(Steps(s_i, a)(s_j) > 0 ∧ r_a(s_i, a) = 0)};
 8: M.ZCC(func, ~, Φ) := ∅;
 9: for all state s ∈ V do
10:     if s ∉ M.ZCC(func, ~, Φ) then
11:         Z_D := {s' ∈ V | s' is reachable from s and s' ∉ M.ZCC(func, ~, Φ)};
12:     end if
13:     M.ZCC(func, ~, Φ) := M.ZCC(func, ~, Φ) ∪ {Z_D};
14: end for
15: return M.ZCC(func, ~, Φ);
```

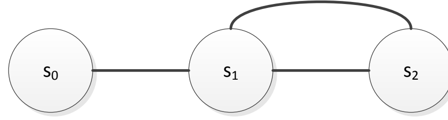


Figure 27: The undirected reduced graph of the MDPR shown in Figure 23

Example 39. To compute $M.ZCC(\min, >, target)$ for the MDPR M in Figure 23 by calling Algorithm 37, the algorithm first constructs the undirected reduced graph G as shown in Figure 27. The reason why the reduced graph does not include state s_5 is that it belongs to $S_{target}^{\max,0}$. Then the algorithm picks up one state, assume it is s_0. All the states reachable from s_0 are s_1 and s_2 which gives us a ZCC $Z_D = \{s_0, s_1, s_2\}$. The algorithm terminates from here as there are no remaining states left. Therefore, the result set $M.ZCC(\min, >, target) = \{\{s_0, s_1, s_2\}\}$. ■

Once $M.ZCC(func, \sim, \Phi)$ is found, we can replace all the ZSCCs of Algorithm 34 with ZCCs to compute $Pr_{M,s}(\mathrm{R}_{\sim r}^{path}[\mathrm{F}\ \Phi])$. Note that similar to the additional exclusion rules for ZCCs of DTMCRs, the definition of ZCCs for MDPRs also excludes some additional states which ensures the correctness of the algorithm. One example to show it can be constructed with the same concept used in Example 19. The performance comparisons between ZSCCs and ZCCs of MDPRs are the same as those of DTMCRs and they will not be repeated here.

5.3.4 MDPRs with Negative Rewards

As mentioned earlier, the above algorithm for computing $Pr_{M,s}(\mathrm{R}_{\sim r}^{path}[\mathrm{F}\ \Phi])$ over MDPRs only supports non-negative rewards. The way how to modify it to support negative rewards is similar as we did for DTMCRs. For a MDPR M with only non-positive rewards, Algorithm 34 can be called with parameters $M', \neg\sim -r, \Phi, func$ and ϵ to compute $Pr_{M,s}^{func}(\mathrm{R}_{\sim r}^{path}[\mathrm{F}\ \Phi])$, where:

- M' is the modified MDPR by replacing both the state and transition rewards in M with their absolute values;
- $\neg > \Leftrightarrow <$, $\neg \geq \Leftrightarrow \leq$, $\neg < \Leftrightarrow >$ and $\neg \leq \Leftrightarrow \geq$.

To modify the algorithm supporting MDPRs with both positive and negative rewards, a approach almost the same as for DTMCRs can be applied. The only differences the definition of the converted graph of a given MDPR and modifying the following four cases in both Equation (9) and Equation (10) by replacing all the four 0 with $r^{\min}_{\mathrm{F}\Phi}(s)$:

- $\sim$ is $> \wedge\, r < 0$
- $\sim$ is $\geq \wedge\, r \leq 0$
- $\sim$ is $< \wedge\, r \leq 0$
- $\sim$ is $\leq \wedge\, r < 0$,

where $r^{\min}_{\mathrm{F}\Phi}(s)$ is the total rewards consumed along the shortest path start from state s to any state $s' \in Sat(\Phi)$. The formal definition of it is as follows:

$$r^{\min}_{\mathrm{F}\Phi}(s) \stackrel{def}{=} \min_{\omega \in Path_{M,s}} X_{\mathrm{F}\Phi}(\omega).$$

The rest of the introduced algorithm for computing $Pr_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$ will be kept the same as before. To compute $r^{\min}_{\mathrm{F}\Phi}(s)$, the converted graph of the given MDPR is constructed first.

Definition 16. The converted graph of a given MDPR $M = (S, \bar{s}, Act, Steps, L, r_s, r_a)$ is a 3-tuple $G = (V, E, w)$ where:

- V is a set of vertices where $V = S$;
- E is a set of edges where $E = \{(u, v) \mid u, v \in S \wedge \exists a \in A(u).Steps(u, a)(v) > 0\}$;
- $w : V \times V \to \mathbb{R}$ is a weight function where for any $u, v \in V$:

$$w(u, v) = \min_{\substack{a \in A(u) \\ Steps(u,a)(v) > 0}} r_{u \to a}. \qquad \blacksquare$$

Once the converted graph is constructed, a shortest path algorithm, as introduced for DTMCRs, can be applied to compute the shortest path from one specific state to all the target states.

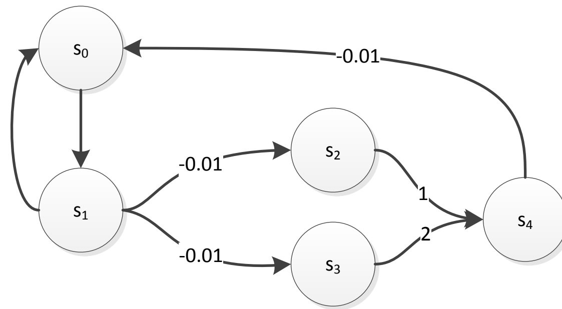


Figure 28: The converted graph of the MDPR in Figure 18

Example 40. Figure 28 is the converted graph of the MDPR in Figure 18 where the weights with value 0 are omitted, and:

$$r^{\min}_{\mathrm{F}\ idle}(s_1) = 0. \qquad \blacksquare$$

Same as the improved algorithm for DTMCRs, the improved algorithm for MDPRs can also be applied to MDPRs with only non-negative rewards with a worse performance due to the non-negligible running time consumed by shortest paths computations and it does not support MDPRs with negative cycles.

5.4 Worklist Algorithm

Similar to the concepts introduced for the worklist algorithm for DTMCRs in Section 3.4, a worklist algorithm for MDPRs is developed and shown in Algorithm 38. The algorithm is almost identical to Algorithm 17 with an additional parameter $func$ indicates whether the algorithm computes the minimum or the maximum probability. The specifications for all three types of reward reward formulae are covered in Algorithms 39 to 41.

Algorithm 38 WORKLIST-PR-MDPR($M, S', \sim r, C, func$)

Input: MDPR M, set of states $S' \subseteq S$, reward bound $\sim r$ tuple of parameters C and function $func \in \{\min, \max\}$
Output: $Pr^{func}_{M,s}(\mathrm{R}^{path}_{\sim r}[\varphi])$ for all $s \in S'$

1: let p be an empty hash table maps a key of type K to a probability value;
2: let W be an empty LHS<K, Boolean>;
3: **for all** state $s \in S'$ **do** W.PUSH(INITIAL-KEY($s, \sim r, C$), $false$); **end for**
4: **while** $\neg W$.ISEMPTY() **do**
5: ($key, canCompute$) := W.PEAK();
6: **if** $canCompute$ **then**
7: W.POP();
8: PR-R-MDPR($M, key, \sim r, C, func, p$);
9: **else**
10: W.UPDATE($true$);
11: **for all** $key' \in$ DEPENDENT-KEYS($M, key, \sim r, C, func, p$) **do**
12: **if** $p(key')$ is not defined **then** W.PUSH($key', false$); **end if**
13: **end for**
14: **end if**
15: **end while**
16: **return** $(p(\text{INITIAL-KEY}(s, \sim r, C)))_{s \in S'}$;

Algorithm 39 Worklist specifications for $\mathrm{I}^{=k}$

```
   C is a 1-tuple (k) where k is a step bound;
   A key of type K is a 2-tuple (s, k) where s is a state and k is a step value;

1: function INITIAL-KEY(s, ~ r, C) return (s, C.k); end function

1: function DEPENDENT-KEYS(D, key, ~ r, C, func, p)
2:     KeySet := ∅;
3:     if key.k > 0 then
4:         for all state s ∈ S where ∃a ∈ Act(Key.s).(Steps(key.s, a)(s) > 0) do
5:             KeySet := KeySet ∪ {(s, key.k − 1)};
6:         end for
7:     end if
8:     return KeySet;
9: end function

1: function PR-R-MDPR(M, key, ~ r, C, func, p)
2:     return PR-I-MDPR-AUX(M, key.s, ~ r, key.k, func, p);
3: end function
```

Algorithm 40 Worklist specifications for $\mathrm{C}^{\leq k}$

```
   C is a 1-tuple (k) where k is a step bound;
   A key of type K is a 3-tuple (s, r, k) where s is a state, r is a reward value and k is a step value;

1: function INITIAL-KEY(s, ~ r, C) return (s, r, C.k); end function

1: function DEPENDENT-KEYS(D, key, ~ r, C, func, p)
2:     KeySet := ∅;
3:     if key.k > 0 then
4:         for all state s ∈ S and all action a ∈ Act(key.s) where Steps(key.s, a)(s) > 0 do
5:             KeySet := KeySet ∪ {(s, key.r − r_{key.s→a}, key.k − 1)};
6:         end for
7:     end if
8:     return KeySet;
9: end function

1: function PR-R-MDPR(M, key, ~ r, C, func, p)
2:     return PR-C-MDPR-AUX(M, key.s, ~ key.r, key.k, func, p);
3: end function
```

Algorithm 41 Worklist specifications for F Φ

C is a 4-tuple $(S_\Phi^{\min,0}, S_\Phi^{\max,0}, T, \epsilon)$ where $S_\Phi^{\min,0}, S_\Phi^{\max,0}, T$ are sets of states and ϵ is a convergence criterion;
A key of type K is a 2-tuple (s, r) where s is a state and r is a reward value;

1: **function** INITIAL-KEY($s, \sim r, C$) **return** (s, r); **end function**

1: **function** DEPENDENT-KEYS($D, key, \sim r, C, func, p$)
2: $KeySet := \emptyset$;
3: **if** $x^{func}_{M,key.s,\sim key.r}$ does not satisfy any base case conditions of its recursive definition **then**
4: **if** $key.s \in M.ZSCC(\Phi)$ **then**
5: **for all** $s \in Z^{\Phi}_{M,key.s}.Out$ **do**
6: $KeySet := KeySet \cup \{(s, key.r)\}$;
7: **end for**
8: **else**
9: **for all** state $s \in S$ and all action $a \in Act(key.s)$ where $Steps(key.s, a)(s) > 0$ **do**
10: $KeySet := KeySet \cup \{(s, key.r - r_{key.s \to a})\}$;
11: **end for**
12: **end if**
13: **end if**
14: **return** $KeySet$;
15: **end function**

1: **function** PR-R-MDPR($M, key, \sim r, C, func, p$)
2: **return** PR-F-MDPR-AUX($M, C.S_\Phi^{\min,0}, C.S_\Phi^{\max,0}, key.s, \sim key.r, C.T, func, p, C.\epsilon$);
3: **end function**

6 Extending Filters

The syntax of the original filter is as follows:

$$\langle filter\rangle ::= \text{filter}(\langle op\rangle, \langle prop\rangle, \langle states\rangle)$$

where $\langle op\rangle$ is a possible operator for the filter, $\langle prop\rangle$ is a PRISM property and $\langle states\rangle$ is a boolean expression used to filter a set of state applying the operator based on the property. The detailed definition, which is omitted here for simplicity, could be found in [12]. In this section, a new operator "cust" (stands for customization) will be introduced to grant filters with better flexibilities.

6.1 Extended Filter

Definition 17. The syntax of the extended filter is defined as follows:

$$\begin{aligned}
\langle filter\rangle &::= \text{filter}(\langle parameter\rangle, \langle states\rangle)\\
\langle parameter\rangle &::= \langle op\rangle, \langle prop\rangle \mid \text{cust}, \langle customization\rangle\\
\langle customization\rangle &::= \langle formula\rangle\langle definition\rangle
\end{aligned}$$

where $\langle formula\rangle$ is a user defined arithmetic expression which has the following syntax:

$$\begin{aligned}
\langle formula\rangle &::= \langle term\rangle \mid \langle formula\rangle + \langle term\rangle \mid \langle formula\rangle - \langle term\rangle\\
\langle term\rangle &::= \langle factor\rangle \mid \langle term\rangle * \langle factor\rangle \mid \langle term\rangle / \langle factor\rangle\\
\langle factor\rangle &::= \langle operand\rangle \mid -\langle operand\rangle\\
\langle operand\rangle &::= \langle literal\rangle \mid @\langle cust\text{-}variable\rangle \mid \langle identifier\rangle \mid \langle func\rangle \mid (\langle formula\rangle)\\
\langle cust\text{-}variable\rangle &::= \langle reserved\rangle \mid \langle identifier\rangle\\
\langle reserved\rangle &::= \text{ss}\\
\langle func\rangle &::= \text{count}() \mid \langle fname\rangle(\langle formula\rangle)\\
\langle fname\rangle &::= \text{min} \mid \text{max} \mid \text{sum} \mid \text{avg} \mid \text{first}
\end{aligned}$$

where $\langle literal\rangle$ is a number (integer or double) and $\langle identifier\rangle$ is a user-defined identifier which could represent a variable, a constant or a formula defined in the model or properties file, all of them follow the PRISM syntaxes of numbers and identifiers, and $\langle definition\rangle$ is a list of cust-variable definitions whose syntax is:

$$\langle definition\rangle ::= \epsilon \mid ;\langle identifier\rangle : \langle prop\rangle\langle definition\rangle. \qquad \blacksquare$$

A "cust" filter property with the form:

$$\text{filter}(\text{cust}, \langle formula\rangle\langle definition\rangle, \langle states\rangle),$$

can be translated to a formula which gives a specific value for a given model and looks similar to $\langle formula\rangle$. The translation of the property is the translation of $\langle formula\rangle$ which is a recursively defined procedure and can be summarized as follows:

- For $@\langle cust-variable\rangle$, if it is not in the scope of a $\langle func\rangle$, an error will be reported, otherwise:
  - for @ss, if the property is verified over a MDPR,an error will be given, otherwise, it will be translated to P_s which denotes the steady-state probability of state s;
  - for $@\langle identifier\rangle$, an error will be given if it is not defined in $\langle definition\rangle$, otherwise, assume it is defined with a PRISM property *prop*, it will be translated to *prop*(s) which represents the value of *prop* over the state s.

- For $\langle identifier \rangle$ not following with an @, if it is not defined in the model file or the properties file, an error will be reported, otherwise:
  - in the case it represents a variable, an error will be given if it is not in the scope of a $\langle func \rangle$, otherwise, it will be translated to $\langle identifier \rangle(s)$ which represents the value of the variable at the state s.
  - in the case it represents a constant or a formula, no further translation needs to be done.
- count() will be translated to $|Sat(\langle states \rangle)|$.
- For $\langle fname \rangle(\langle formula' \rangle)$:
  - min will be translated to $\min_{s \in Sat(\langle states \rangle)} \{\langle formula' \rangle\}$;
  - max will be translated to $\min_{s \in Sat(\langle states \rangle)} \{\langle formula' \rangle\}$;
  - sum will be translated to $\sum_{s \in Sat(\langle states \rangle)} \{\langle formula' \rangle\}$;
  - avg will be translated to $\frac{\sum_{s \in Sat(\langle states \rangle)} \{\langle formula' \rangle\}}{|Sat(\langle states \rangle)|}$;
  - let s_f be the first (lowest-indexed) state in $Sat(\langle states \rangle)$, first will be translated to $evaluate(s_f, \langle formula' \rangle)$ where *evaluate* is a function replacing all the appearance of s in the translation of $\langle formula' \rangle$ with s_f.
- The rest of $\langle formula \rangle$ will be kept as it is in the property.

Example 41. The filter property:

filter(cust, sum(@ss * @v) / sum(@ss); v: prop, states),

will be translated to the formula:

$$\frac{\sum_{s \in Sat(states)} \{P_s \times prop(s)\}}{\sum_{s \in Sat(states)} P_s}. \qquad \blacksquare$$

6.2 Performance Optimizations for CUST Operator

As we mentioned in Section 1.1, all properties will be parsed to property Abstract Syntax Trees (AST). Then the property AST is passed to the computation engine and the model checking result is computed. If the passed property is a "cust" filer property with the form:

filter(cust, $\langle formula \rangle \langle definition \rangle, \langle states \rangle$).

The computation engine will first compute $Sat(\langle states \rangle)$, then it will verify all the properties listed in the $\langle definition \rangle$, finally, based on these results, it evaluates the $\langle formula \rangle$. In this section, we will introduce some optimizations could be applied before and during the computation in order to improve the performance. As all of these performance optimizations are based on simple AST modifications and value cache mechanisms, thus, for the simplicity of this paper we will be focusing on the ideas themselves instead of providing detailed algorithms.

Before the property AST is passed to the computation engine, following modifications can be done to improve the computation performances:

- If a cust-variable defined in $\langle definition \rangle$ has not been used in the $\langle formula \rangle$, then the property it represents does not need to be verified. Therefore, it can be remove from the AST.

  E.g. In the following property:

filter(cust, sum(@v); v: prop; v: prop′, states),

there are two cust-variables v and v' defined in $\langle definition \rangle$. v is used in $\langle formula \rangle$ while v' is not. After the optimization, the sub-AST of v: prop′ is removed and the modified AST represents the following property instead:

filter(cust, sum(@v); v: prop, states).

- If some sub-formulae of $\langle formula \rangle$ can be precomputed, we can simplify the $\langle formula \rangle$ by replacing the sub-formula with a single literal. This may improve the performance in practice.

  E.g. In the following property:

  filter(cust, sum(1 + 2 + @v); v: prop, states),

  where the sub-formula 1+2 can be precomputed. After the optimization, the sub-AST of 1+2 will be replaced with the literal 3 and the modified AST passed to the computation engine represents the following property:

  filter(cust, sum(3 + @v); v: prop, states).

  This optimization will give a better performance because the computation engine will compute sum(3 + @v) in the following manner:

  1: $sum := 0$;
  2: **for all** state $s \in Sat(states)$ **do**
  3: $\quad sum := sum + 3 + prop(s)$;
  4: **end for**

  Before the AST modification, the engine needs to compute 1 + 2 in each iteration which is saved after the optimization. Also this concept can be extended to sub-formulae not only includes literals but also constants.

  E.g. For the following property:

  filter(cust, sum(a + 1 + 2 + @v); v: prop, states),

  where a is a constant assigned as 3. After the optimization, then modified AST used to do the computation will represent the following property:

  filter(cust, sum(6 + @v); v: prop, states).

During the computation, following two cache mechanisms can be applied to avoid redundant computations:

- Once the result of a filter customization function is computed, the result will be cached for later usage.

  E.g. After the optimization, the avg(@v) in the following property will only be computed once for the whole calculation:

  filter(cust, avg(@v - avg(@v)); v: prop, states).

  This optimization will give a better performance because the computation engine will compute avg(@v - avg(@v)) in the following manner:

  1: $avg := 0$;
  2: **for all** state $s \in Sat(states)$ **do**
  3: $\quad avg := prop(s) + compute(avg(@v))$;
  4: **end for**
  5: $avg := avg/|Sat(states)|$;

  where $compute(avg(@v))$ is a procedure call returns the computation result of avg(@v). Before the optimization, the engine computes avg(@v) in each iteration. If the result is cached after it is computed first time, then the computations needed for the rest iterations are saved.

- The previous optimization can be further improved by not only caching the calculated result for the filter customization function itself, but also other filter customization functions with the same structure.

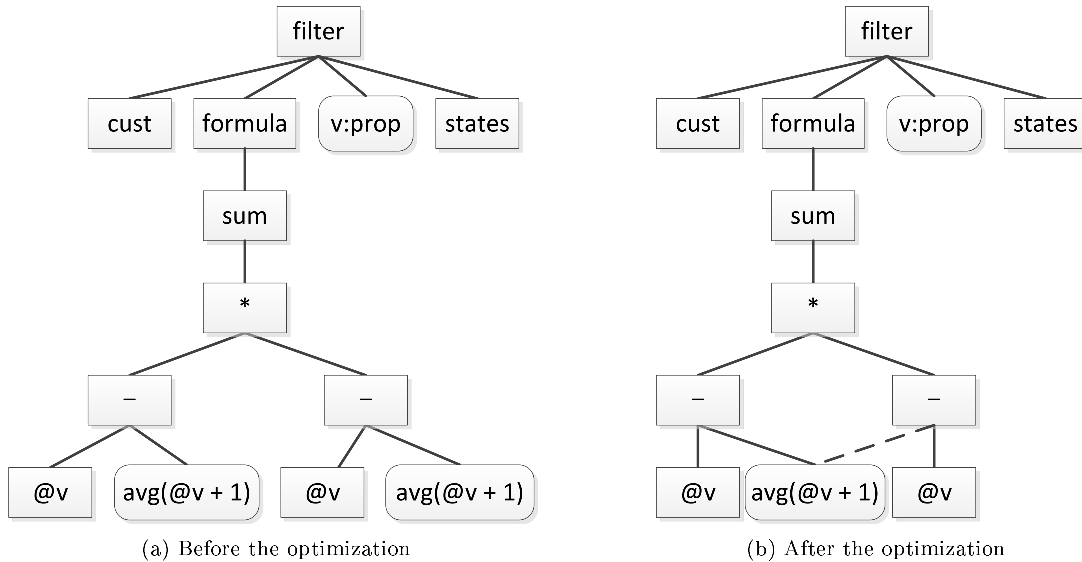
  E.g. After the optimization, the both appearances of avg(@v + 1) in the following property will only be computed once for the whole calculation:

  filter(cust, sum((@v - avg(@v + 1)) * (@v - avg(@v + 1))); v: prop, states).

  This can be achieved by either:

  – Having a hash table maps all pre-calculated filter customization functions to its cached result. When the algorithm retrieves the result for a specific filter customization function from the hash table, if it is not defined in the table, it will be calculated and put into the table. Otherwise, the cached value will be returned.

  – Or instead of having a hash table, we can make all the references of the sub-ASTs with the same structure point to a same instance. This is the approach used in the actual implementation of this project. To make it more clear, the AST before the optimization of the above property is shown in Figure 29 (a), where the rectangles with round corners are unfolded sub-ASTs. After the optimization, the AST becomes the one shown in Figure 29 (b). Note that even though from the data structure point of view, there is only one instance of the sub-AST of avg(@v + 1), the algorithm uses the modified AST treats it the same as the original one. From here, we can apply the previous optimization to cache the result once it is computed for later usage.



(a) Before the optimization (b) After the optimization

Figure 29: The ASTs of filter(cust, sum((@v - avg(@v + 1)) * (@v - avg(@v + 1))); v: prop, states)

7 Extending PRISM

Based on the existing PRISM source code, following functionalities are further extended for PRISM:

- Added the support of transition rewards for explicit engine. Transition rewards were supported for other computation engines but explicit. However, for the explicit engine of the extended PRISM, transition rewards can only be used for the newly introduced algorithms included in this paper yet.
- Added the support of expressing PCTLR formulae with PRISM properties.
- Implemented the following model checking algorithms:
  - Top-down with memoization and Worklist approach for:
    - Computing $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$ over DTMCRs;
    - Computing $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{C}^{\leq k}])$ over DTMCRs;
    - Computing $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$ with either ZSCCs or ZCCs over DTMCRs with non-negative rewards only;
    - Computing $Pr_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$ over MDPRs;
    - Computing $Pr_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{C}^{\leq k}])$ over MDPRs;
    - Computing $Pr_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$ with either ZSCCs or ZCCs over MDPRs with non-negative rewards only.
  - Bottom-up method approach for:
    - Computing $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$ over DTMCRs;
    - Computing $Pr_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$ over MDPRs.
- Applied the following model checking algorithm optimizations for both DTMCRs and MDPRs:
  - Optimized the algorithm for probability until operator computations;
  - Computing probabilities of path reward operator over a subset of all states only.
- Realized the extended filters with "cust" operator with all performance optimizations presented in Section 6.
- Added an option tap to ease the setups of the newly introduced model checking algorithms.
- Added the reports of error messages when a user mis-uses the newly introduced functionalities.

Table 2: Modification Statistics

Language	files	comment	code
HTML(All Same)	1	0	3
make(All Same)	16	144	431
Bourne Shell(All Same)	6	82	168
C++(All Same)	93	6,613	19,445
Java			
same	553	41,484	123,994
modified	39	3	705
added	39	167	5,428
removed	0	0	124

In this section, we will show that where these changes are taking place in the existing PRISM source code. Once the source code is downloaded from the PRISM website, several folders can be found under the root folder. All actual source code is located under a folder called "src" which is the only folder has been modified. Table 2 shows the modification statistics of the "src" folder. For all changed folders under it, we will go through them one by one.

7.1 Modifications of Existing PRISM Classes

First let us take a look at modifications of the existing PRISM classes. Note that for the simplicity reason, not all the modified classes are included. Those classes which are modified merely due to the effect of structure renovations but remain the same functionalities are omitted.

explicit is the folder contains all classes related to the explicit computation engine.

- *ConstructModel* is the model constructor which constructs the model data structure based on the model file AST. It is modified to add transition rewards to the model when it constructs a DTMC or CTMC. The action rewards for MDPs and CTMDPs are already supported.
- *DTMCModelChecker* which extends ProbModelChecker is the model checker used for DTMC models. Algorithm 14, the optimized probability until computation algorithm for DTMCRs, is added to this class.
- *DTMCSimple* is the actual data structure of the DTMCR model used in the extended PRISM. Transitions are stored as a list of hash tables which maps a state index to a probability value. Each hash table on the ith position of the list represents transitions from the ith state of the model. It is modified to support the transition rewards.
- *MDPModelChecker* which extends ProbModelChecker is the model checker used for MDP models. Algorithm 36, the optimized probability until computation algorithm for MDPRs, is added to this class.
- *MDPSimple* is the actual data structure of the MDPR model used in the extended PRISM. Transitions are stored as a list of lists of hash tables. The first layer list maps a state index to its action lists and the action list maps an action index to the actual transitions. A method instantiate MDPSimple from an instance of MDPSparse is added.
- *MDPSparse* is another implementation of the MDP model used in the original PRISM. It is a sparse matrix (non-mutable) explicit-state representation of the MDP model. It is smaller and faster if the modification of the constructed model is not required. This class is unchanged.
- *ProbModelChecker*, which extends StateModelChecker contains model checking algorithms for state formulae. The support of model checking probability path reward formulae is added here.
- *StateModelChecker* is the super class for explicit-state model checkers. The framework of model checking PRISM properties are included here. It is modified to support model checking a sub-AST of ExpressionFilterCust (a class of a property AST element which will be introduced later).

explicit/rewards is the folder contains all classes related to the reward data structures of explicit computation engine.

- *ConstructRewards* is the reward structure constructor. It is modified to support transition rewards for DTMCRs.

- *MCRewards* is the interface provides accesses to explicit-state rewards for both DTMCs and CTMCs. Two additional API is added, one is used to get transition rewards and the other one is used to return whether there are negative rewards for the current model.
- *MDPRewards* is the interface provides accesses to explicit-state rewards for MDPs. One additional API is added to return whether there are negative rewards for the current model.
- *MDPRewardsSimple*, which implements MDPRewards, is the actual reward data structure used for MDPRs which supports both state and action rewards. It is modified to be able to return a boolean value indicates whether the current model contains negative rewards.

parser is the folder contains classes related to the PRISM parsers for both model and properties files. It is also accompanied with related AST data structures and JavaCC (Java Compiler Compiler, a parser generator for java) parser files.

- *PrismParser* is the auto-generated class by JavaCC based on PrismParser.jj. It is the parser class for both model and properties files.
- *PrismParserConstants* is the auto-generated class by JavaCC based on PrismParser.jj. It contains all the constants of the parser.
- *PrismParserTokenManager* is also the auto-generated class by JavaCC based on PrismParser.jj. It is the token manager class for prism parser.
- *PrismParser.jj* is the JavaCC file whom the prism parser is auto-generated based on. It is modified to support the newly introduced syntaxes of PCTLR and filter properties.

parser/ast is the folder contains all AST element classes of model and properties file ASTs.

- *ExpressionFilter* is the AST element class of the filter property. It is modified to support the newly introduced "cust" operator.

parser/visitor All operations toward to model and properties ASTs are implemented by using the Visitor design pattern[28]. This is the folder contains all visitor classes.

- *ASTTraverse*, which implements ASTVisitor, is the visitor class performances a depth-first traversal of a given AST without modifying any elements of it. It is extended to support the newly introduced filter properties.
- *ASTTraverseModify*, which implements ASTVistor, is the visitor class performances a depth-first traversal of a given AST with the ability to modify the traversed elements. It is extended to support the newly introduced filter properties.
- *ASTVistor* is the interface of the PRISM AST visitor framework.
- *SemanticCheck*, which extends ASTTraverse, is the visitor class checks the semantic correctness of a given AST. It is modified to support the semantic checking of the newly introduced filter properties.
- *Simplify*, which extends ASTTraverseModify, is the visitor class simplifies the given AST. Performance optimizations of the extended filter ASTs before passing to the computation engine are implemented here.
- *TypeCheck*, which extends ASTTraverse, is the visitor class checks the type correctness of a given AST, e.g. whether the customization function count() takes no arguments, whether all operands of the customization function sum() are integers or doubles. It is modified to support the type correctness checking of the newly introduced filter properties.

prism is the folder contains classes of the main PRISM API, the command-line tool, etc.

- *PrismSettings* is the class of the option dialog. It is modified to add an additional tab for setups of newly introduced algorithms.

7.2 Newly Introduced Classes

Then the newly introduced classes are introduced as follows.

explicit/rewards is added with one new class.

- *MCRewardsSimple*, which implements MCRewards, is the actual reward data structure used for DTMCRs which supports both state and transition rewards. It also can return a boolean value indicates whether there are negative rewards for the current model.

parser/ast is added with three new classes.

- *ExpressionFilterCust* is the AST element class of the "cust" filter property itself.
- *ExpressionFilterCustFunc* is the AST element class of customization functions of the "cust" filter property.
- *ExpressionFilterCustVar* is the AST element class of customization variables of the "cust" filter property.

explicit/probpathrewardformulae is a newly introduced folder contains all classes related to the PCTLR probability path reward formulae.

- *DTMCSimpleAdv*, which extends DTMCSimple is the data structure of the DTMCR used when the concepts of ZSCCs or ZCCs are needed. It extends DTMCSimple with informations of connected components and all the related algorithms are implemented here.
- *LinkedHashStack* is the actual implementation of the data structure introduced in Definition 9.
- *MDPSimpleAdv*, which extends MDPSimple is the data structure of the MDPR used when the concepts of ZSCCs or ZCCs are needed. It extends MDPSimple with informations of connected components and all the related algorithms are implemented here.
- *ProbPathRewardsFormulaeChecker* is the model checker for PCTLR probability path reward formulae. Note that in the actual implementation, all top-down with memoization approaches included in this paper are implemented as a worklist algorithm with recursive procedure calls while the worklist approaches introduced in this paper are implemented as a worklist algorithm without recursive procedure calls. The Algorithms 6 and 26 and the optimization only computing probabilities for a subset of the total states of a given model are implemented here.
- *ProbPathRewardsFormulaeSettings* is the data contract class contains all informations of the setup tab of model checking probability path reward formulae.

explicit/probpathrewardformulae/multikey is a newly added folder contains all classes of multi-key data structures used as a unique key of hash tables used in the model checking PCTLR probability path reward formulae algorithms.

- *MultiKey* is the interface of all multi-key classes.
- *MultiKeyC*, which implements MultiKey, is the multi-key class used for computations of cumulative path reward formulae.

- *MultiKeyI*, which implements MultiKey, is the multi-key class used for computations of instantaneous path reward formulae.
- *MultiKeyF*, which implements MultiKey, is the multi-key class used for computations of reachability path reward formulae.

explicit/probpathrewardformulae/specification is a newly introduced folder contains all specification classes.

- *DTMCSpecification*, which implements SpecificationMemoized and SpecificationWorklist, is the base specification class for DTMCRs.
- *DTMCSpecificationC*, which extends DTMCSpecification, is the specification class as introduced in Algorithm 19, it also includes the implementation of Algorithm 8.
- *DTMCSpecificationF*, which extends DTMCSpecification, is the specification class as introduced in Algorithm 20, it also includes the implementation of Algorithm 13.
- *DTMCSpecificationI*, which extends DTMCSpecification, is the specification class as introduced in Algorithm 18, it also includes the implementation of Algorithm 5.
- *MDPSpecification*, which implements SpecificationMemoized and SpecificationWorklist, is the base specification class for MDPRs.
- *MDPSpecificationC*, which extends MDPSpecification, is the specification class as introduced in Algorithm 40, it also includes the implementation of Algorithm 28.
- *MDPSpecificationF*, which extends MDPSpecification, is the specification class as introduced in Algorithm 41, it also includes the implementation of Algorithm 35.
- *MDPSpecificationI*, which extends MDPSpecification, is the specification class as introduced in Algorithm 39, it also includes the implementation of Algorithms 25, 32 and 33.
- *Specification* is the base interface of all specification classes.
- *SpecificationMemoized*, which extends Specification, is the base interface of specification classes for top-down with memoization approaches.
- *SpecificationWorklist*, which extends Specification, is the base interface of specification classes for worklist approaches.

7.3 User Manual of Extended PRISM

Detailed user manual of the original PRISM can be found at PRISM Manual version 4.0.3 included with the PRISM source code package. In this section, we will illustrated how to use the newly introduced functionalities of the extended PRISM.

Probability path reward formulae are supported by the extended PRISM property syntax with the following form:

P operator [R operator [rewardprop]],

where both P and R operators follow the PRISM property syntax introduced in the manual . Note that there is no significant distinction between the state R operator and the path R operator. When the R operator used separately, it means the state R operator, while it is nested with a P operator, it becomes the path R operator. The only difference is that the path R operator can only be used with *bound* ($< r$, etc.) instead of *query* (=?).

Example 42. The formula $Pr^{\min}(\mathrm{R}^{path}_{\geq 1}[\mathrm{F}\ target])$, where $target$ is a boolean variable, can be expressed by the PRISM property Pmin=? [R>=1 [F target]]. ■

Extended filter properties with "cust" operator can be used by following the syntax introduced in Section 6.

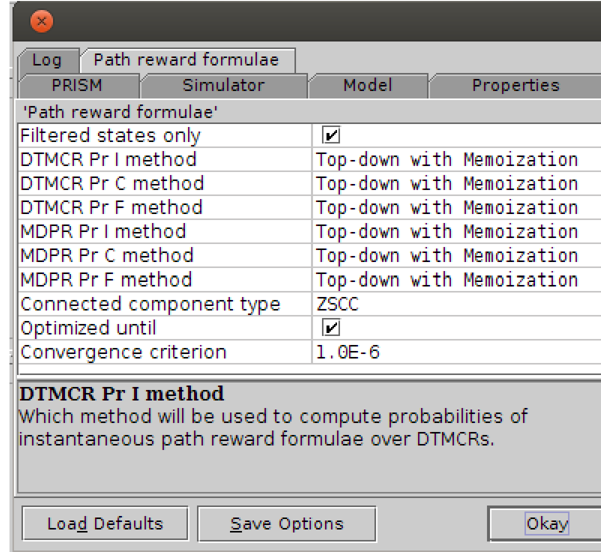


Figure 30: The snapshot of the setup tap for probability path reward formulae algorithms

Path reward formulae setup tab is newly added to the option dialog which can be accessed via the PRISM menu Options/Options. A snapshot of it is shown in Figure 30.

- *Filtered states only* indicates whether the probabilities will be computed for all the states or only the states selected by the filter properties.
  Example 43. The PRISM property:

  filter(print, P=? [R<1[F s=5]], s<2),

  will only computes $Pr(\mathrm{R}_{<1}[\mathrm{F}\ s = 5])$ for all states where $s < 2$ are satisfied when "Filtered states only" is on, otherwise, the probabilities will be computed for all states of the given model. The property:

  P=? [R<1[F s=5]],

  will only computes $Pr(\mathrm{R}_{<1}[\mathrm{F}\ s = 5])$ for initial states when the parameter is on. ■

- *DTMCR Pr I method* indicates which approach will be used to compute probabilities of instantaneous path reward formulae over DTMCRs and it has three options: "Top-down with Memoization", "Bottom-up Method" and "Worklist".

- *DTMCR Pr C method* indicates which approach will be used to compute probabilities of cumulative path reward formulae over DTMCRs and it has two options: "Top-down with Memoization" and "Worklist".

- *DTMCR Pr F method* indicates which approach will be used to compute probabilities of reachability path reward formulae over DTMCRs and it has two options: "Top-down with Memoization" and "Worklist".

- *MDPR Pr I method* indicates which approach will be used to compute probabilities of instantaneous path reward formulae over MDPRs and it has three options: "Top-down with Memoization", "Bottom-up Method" and "Worklist".
- *MDPR Pr C method* indicates which approach will be used to compute probabilities of cumulative path reward formulae over MDPRs and it has two options: "Top-down with Memoization" and "Worklist".
- *MDPR Pr F method* indicates which approach will be used to compute probabilities of reachability path reward formulae over MDPRs and it has two options: "Top-down with Memoization" and "Worklist".
- *Connected component type* indicates which type of connected components will be used to compute probabilities of reachability path reward formulae and it has two options: "ZSCC" and "ZCC".
- *Optimized until* indicates whether Algorithms 14 and 36 will be applied over DTMCRs and MDPRs, respectively.
- *Convergence criterion* is the convergence criterion value used for the newly introduced algorithms.

8 Experiments

In this section, we will test the newly introduced functionalities of the extended PRISM. The test cases are divided into three groups: test cases for extended filters, PCTLR model checking over DTMCRs and PCTLR model checking over MDPRs. Test cases in each group are further split into three categories, correctness, performance and robustness, if applicable. Let us start by taking a look at how to construct model files in PRISM.

8.1 Modeling DTMCRs and MDPRs in PRISM

[5] provides a good introduction about the PRISM language. To write the model file of a given DTMCR or MDPR in PRISM, we can first construct the model without assigning rewards to any states, transitions (DTMCR) or actions (MDPR).

Example 44. Let us have a look at two examples of model files of a DTMCR model and a MDPR model. The DTMCR shown in Figure 3 can be described by using the code as follows:

```
1  dtmc
2
3  module VendingMachine
4
5      s : [0..5] init 0;
6
7      [] s=0 -> (s'=1);
8      [] s=1 -> 0.1 : (s'=0) + 0.9 : (s'=2);
9      [] s=2 -> 0.5 : (s'=3) + 0.5 : (s'=4);
10     [] s=3 -> (s'=5);
11     [] s=4 -> (s'=5);
12     [fee] s=5 -> (s'=0);
13
14 endmodule
```

and the MDPR shown in Figure 18 can be expressed by using the following code:

```
1  mdp
2
3  module VendingMachine
4
5      s : [0..4] init 0;
6
7      [initialize] s=0 -> (s'=1);
8      [cancel] s=1 -> (s'=0);
9      [select] s=1 -> 0.5 : (s'=2) + 0.5 : (s'=3);
10     [notify] s=2 -> (s'=4);
11     [notify] s=3 -> (s'=4);
12     [release] s=4 -> (s'=0);
13
14 endmodule
```

Note that the above two model files only include the DTMCR and the MDPR without rewards. ■

Once the model is constructed, we can assign the rewards to the corresponding states, transitions or actions. The way to express state rewards is the same to both DTMCR and MDPR models, and describing action rewards for MDPR models is also quite straightforward.

Example 45. The state rewards and action rewards of the MDPR in Figure 18 can be described by the following reward structure:

```
1 rewards
2     s=2 : 1;
3     s=3 : 2;
4     [select] true : -0.01;
5     [release] true : -0.01;
6 endrewards
```

and for the DTMCR in Figure 3, we can apply the similar approach to express the state rewards and the reward of the transition from s_5 to s_0 with the reward structure as follows:

```
1 rewards
2     s=3 : 1;
3     s=4 : 2;
4     [fee] true : -0.01;
5 endrewards
```

However, this will not assign any reward value to the transition from s_1 to s_2 and we cannot fix this problem by simply labeling transitions from s_1 with "fee" as follows:

```
1     [fee] s=1 -> 0.1 : (s'=0) + 0.9 : (s'=2);
```

as this will assign transitions from s_1 to both s_0 and s_2 with reward -0.01 which is not the same as shown by the DTMCR in Figure 3. Also PRISM does not provide a way to give a special label to a specific transition in DTMCRs. ■

Next, let us discuss how to resolve the above issue. One way is applying the transition rewards elimination algorithm introduced in Section 2.3 with the pros and cons stated previously. The other way is modeling each transition with separate commands based on the support of local nondeterminism in PRISM. For a DTMCR model, when there are more than one command whose guard is satisfied at the same time, the probability to choose each command is the same and the total probability is 1. Hence, we can model the transitions from s_1 in DTMCR shown in Figure 3 with following code instead:

```
1     [] s=1 -> (s'=0);
2     [fee] s=1 -> (s'=2);
3     [fee] s=1 -> (s'=2);
4     [fee] s=1 -> (s'=2);
5     [fee] s=1 -> (s'=2);
6     [fee] s=1 -> (s'=2);
7     [fee] s=1 -> (s'=2);
8     [fee] s=1 -> (s'=2);
9     [fee] s=1 -> (s'=2);
10    [fee] s=1 -> (s'=2);
```

Though it seems that it creates 9 transitions from s_1 to s_2 each with probability 0.1, all "parallel" transitions will be merged to a single transition in PRISM. So by using the above code, the transitions from s_1 will end up as one to s_0 with probability 0.1 and one to s_2 with probability 0.9 which is the same as desired. From here, we can label all commands describing the transition from s_1 to s_2 with "fee" as shown above to assign the corresponding transition reward.

To generalize this approach for a given DTMCR $D = (S, \bar{s}, \mathbf{P}, L, r_s, r_t)$, in order to express transitions

from a given state s in separate commands, one needs to find the smallest[10] integer n, where:

$$\forall s' \in S.\exists x_{s'} \in \mathbb{N}.(\frac{x_{s'}}{n} = \mathbf{P}(s, s')).$$

Then $\forall s' \in S$, write $n \times \mathbf{P}(s, s')$ commands with the following form:

[ ] <At state s> -> (<At state s'>);

Once we have the model file with transitions described by separating commands, we can label commands for each transition with different names and assign each transition with different reward values when needed. If n does not exist, then this approach is not applicable to that specific model.

Note that the second approach is a workaround based on the gray side of PRISM language specification. PRISM treats the appearance of local nondeterminism as the result of a user error and a warning will be given if there is one and the model file will become almost impossible to understand when model becomes bigger and more complicate. Therefore, this approach is not recommended for big and complex models and the approach with transition reward elimination should be considered instead. On the other hand, this approach works fine with small and simple models as its ease to apply.

8.2 Extended Filters

In this section, several test cases will be given covering the correctness and robustness aspects of the implementation of the extended filters. As the performance gains of the optimizations introduced in this paper are straightforward and the implementation does not provide the interfaces to control whether they are applied or not. Therefore, the performance gains of these optimizations will not be covered here.

8.2.1 Correctness Tests

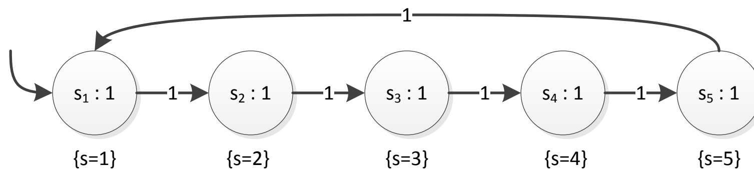


Figure 31: A DTMCR for testing extended filters

Test Case 1. Verify the correctness of the implementation of extended filters

Model: DTMCR in Figure 31, the model file can be found at Section B.1.

Conditions: c in a declared integer constant which is assigned as 3.

Result: The test results are shown in Table 3. All the properties can be easily computed by hand to verify the results are all correct and the computation details are omitted. ■

8.2.2 Robustness Tests

The test cases in this section will cover the situations when a user mis-uses the "cust" filter properties where an error message containing the problem information should be shown to the user.

[10] This is not a necessary requirement, but it will significantly reduce the code length.

Table 3: Experiment results of Test Case 1

Properties	Results
filter(cust, 1+2/3)	1.6666666666666665
filter(cust, (1+2)/3)	1.0
filter(cust, (c+3)/4)	1.5
filter(cust, count(), s>1&s<5)	3
filter(cust, min(s))	1
filter(cust, max(s))	5
filter(cust, sum(s))	15
filter(cust, avg(s))	3.0
filter(cust, sum(s+1)/count())	4.0
filter(cust, max(@v); v: R=? [F s=5])	4.0
filter(cust, sum(@v*@ss); v: R=? [F s=5], s>1)	1.2
filter(cust, sum(@v*@ss); v: R=? [F s=5], s<5)	2.0

Test Case 2. A variable is used out of the scope of a customization function

Model: Same as Test Case 1.

Property: filter(cust, s)

Result: An error message is given as shown in Figure 32. ■

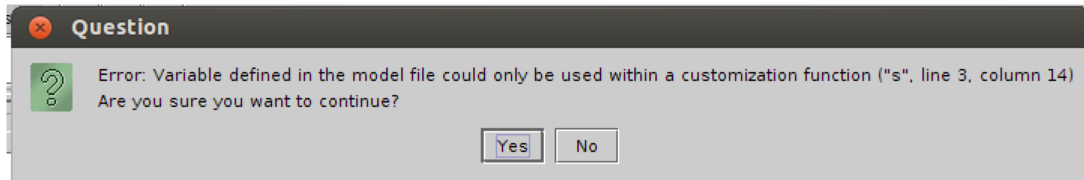


Figure 32: The error message of Test Case 2

Test Case 3. An undefined identifier is used

Model: Same as Test Case 1.

Property: filter(cust, avg(s0))

Result: An error message is given as shown in Figure 33. ■

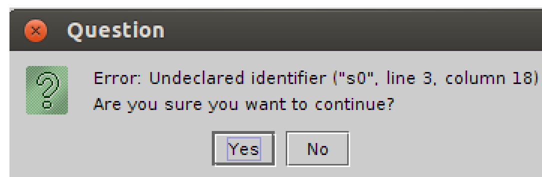


Figure 33: The error message of Test Case 3

Test Case 4. A cust-variable is used out of the scope of a customization function

Model: Same as Test Case 1.

Property: filter(cust, @v; v: R=? [F s=5])

Result: An error message is given as shown in Figure 34. ■

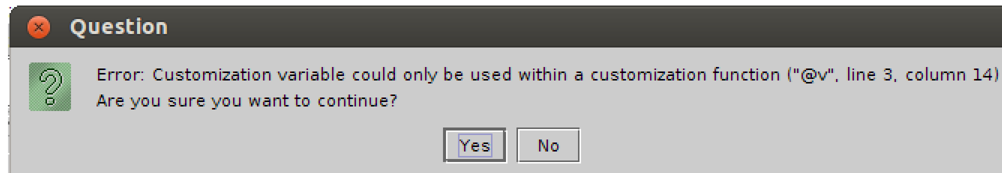


Figure 34: The error message of Test Case 4

Test Case 5. An undefined cust-variable is used

Model: Same as Test Case 1.

Property: filter(cust, avg(@v))

Result: An error message is given as shown in Figure 35. ■

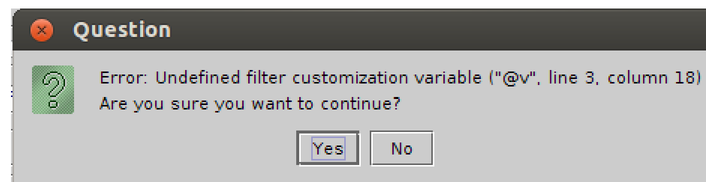


Figure 35: The error message of Test Case 5

8.3 PCTLR Model Checking over DTMCRs

The functionality of the extended PRISM model checking PCTLR formulae over DTMCRs will be tested in this section.

8.3.1 Correctness Tests

First let us test the correctness of the extended PRISM model checking PCTLR formulae over DTMCRs by computing the four cases in Examples 7, 8, 10 and 16. The results computed by the extended PRISM should almost be the same as the results we did manually. There might be some minor differences due to the known precision lost problem of double value computations in JAVA.

Test Case 6. Verify the result in Example 7

Model: DTMCR D in Figure 3, the model file can be found at Section B.2

Property: P=? [R>0 [I=3]]

Conditions: All four combinations of:

- whether using the "Top-down with Memoization" or the "Worklist" approach;
- whether "Filtered states only" is on or off.

Result: 0.8999999999999999 for all above four conditions which is slightly different from the result 0.9 we got from the manual computation. This is due to the precision lost problem of JAVA, when execute the following JAVA code snippet:

```
1 double i = 1.0 / 10, sum = 0;
2 for (int j = 0; j < 9; j++) { sum += i; }
3 System.out.println(sum);
```

the actual output (the final value of sum) is 0.8999999999999999 while the expected result is 0.9. The above JAVA code snippet mimics the way how PRISM computes the probability of the transition of D from s_1 to s_2 with the model file we provide. Thus the value of $\mathbf{P}(s_1, s_2)$ in the PRISM DTMCR model of D is 0.8999999999999999. Therefore, the extended PRISM gives the result as we expected. ■

Test Case 7. Verify the result in Example 8

Model: Same as Test Case 6.

Property: filter(print, P=? [R>0 [I=3]])

Conditions: Bottom-up Method.

Result: 0.8999999999999999 for state s_0 and 0 for the rest states which is the same as the result we got by the manual computation regardless the precision lost problem. ■

Test Case 8. Verify the result in Example 10

Model: Same as Test Case 6.

Property: filter(state, P=? [R>=1 [C<=3]], s=1)

Conditions: Same as Test Case 6.

Result: 0.44999999999999996 for all four conditions which is the same as the result in Example 10 despite of the precision lost problem. ■

Test Case 9. Verify the result in Example 16

Model: DTMCR in Figure 8, the model file can be found at Section B.3.

Property: P=? [R<1 [F s=4]]

Conditions: All eight combinations of:

- whether using the "Top-down with Memoization" or the "Worklist" approach;
- whether "Optimized until" is on or off;
- whether "Filtered states only" is on or off.

Result: The results with different values of the convergence criterion ϵ for using both the "ZSCC" and the "ZCC" approach is shown in Table 4, the results for each approach under the same convergence criterion value are the same for all eight combinations. We can see easily from the table that the smaller the convergence criterion value is, the more precise the result is ($\frac{8}{23} \approx$ 0.3478260869565217391304347826087). ■

Table 4: Experiment results of Test Case 9

ϵ	Results for ZSCCs	Results for ZCCs
10^{-6}	0.34782600175006134	0.34782578400021813
10^{-7}	0.3478260802165576	0.34782606299220475
10^{-8}	0.34782608642338003	0.3478260802165576
10^{-9}	0.3478260868065757	0.34782608642338003
10^{-10}	0.3478260869446608	0.3478260869143494
10^{-15}	0.3478260869565216	0.3478260869565213
10^{-20}	0.34782608695652173	0.34782608695652173

Next, we modified a dice model included in the PRISM source code to continue the correctness tests of the extended PRISM model checking PCTLR formulae over DTMCRs. The modified model is shown in Figure 36 and the modified parts are marked as dark yellow.

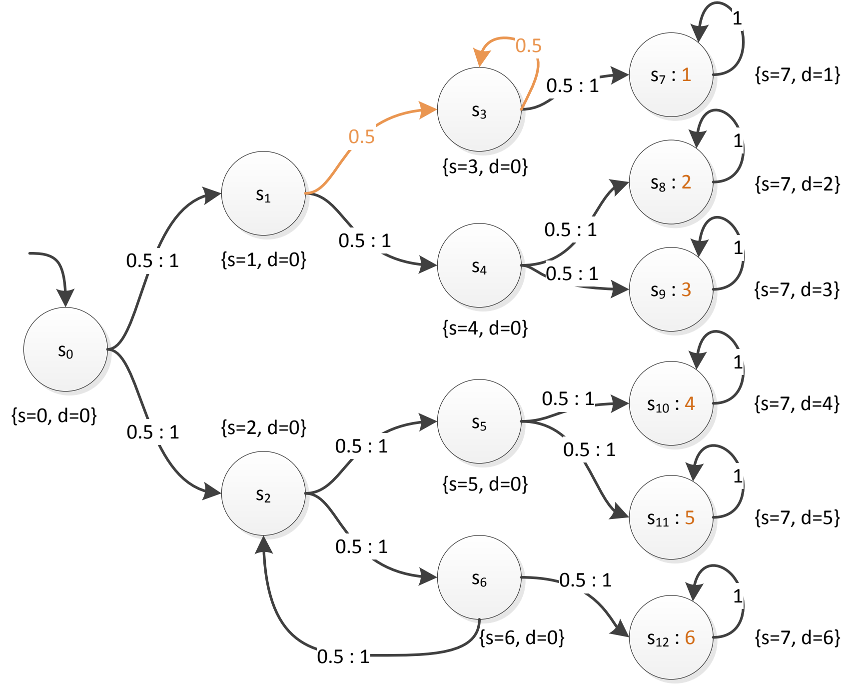


Figure 36: A DTMCR of the modified dice model

Test Case 10. Verify properties over the modified dice model

Model: DTMCR in Figure 36, the model file can be found at Section B.4.

Conditions: The options are set as follows:

- using the "Worklist" approach;
- "Optimized until" is on;
- "Filtered states only" is on;
- "ZSCC" is the used connected component type;
- convergence criterion value is 10^{-6}.

Result: The results can be found at Table 5. As all the properties can be easily computed by hand to verify that they are all correct, thus the computation details are omitted. ■

8.3.2 Performance Tests

Test cases in this section cover the performance aspects we talked in this paper.

Test Case 11. Performance comparisons between approaches of computing $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{I}^{=k}])$

Model: DTMCR in Figure 6, the model file can be found at Section B.5.

Property: P=? [R<=0 [I=n]]

Conditions: The property are verified under the following conditions:

Table 5: Experiment results of Test Case 10

Properties	Result
P=? [R<=0 [I=3]]	0.25
P=? [R>3 [I=3]]	0.375
P=? [R>=6 [I=3]]	0.125
P=? [R<2 [C<=3]]	0.125
P=? [R<3 [C<=5]]	0.0625
P=? [R>10 [C<=5]]	0.375
P=? [R>12[C<=5]]	0.25
P=? [R>13 [C<=5]]	0.125
P=? [R<=3 [F s=7 & d=1]]	0.25

- C1: "Top-down with Memoization" and "Filtered states only" is off;
- C2: "Bottom-up Method";
- C3: "Top-down with Memoization" and "Filtered states only" is on.

Result: The results can be found at Table 6. From the results, it is clear that the "Top-down with Memoization" approach has quite some overheads of caching computed values and recursive procedure calls. Therefore, even though it computes less values than the "Bottom-up Method", it is still much slower than the "Bottom-up Method". However, the "Top-down with Memoization" can apply the "Filtered states only" optimization which significantly improves the performance if the set of states, which the property should be verified on them, is much smaller than the set of all states in the model. In this case, only the property verification result of the initial state is required. ■

Table 6: Experiment results of Test Case 11

n	Running times		
	C1	C2	C3
100	0.006s	0.001s	<0.001s
200	0.039s	0.012s	<0.001s
300	0.087s	0.026s	<0.001s
500	0.211s	0.042s	0.001s
1000	0.953s	0.152s	0.002s
2000	6.625s	0.525s	0.003s

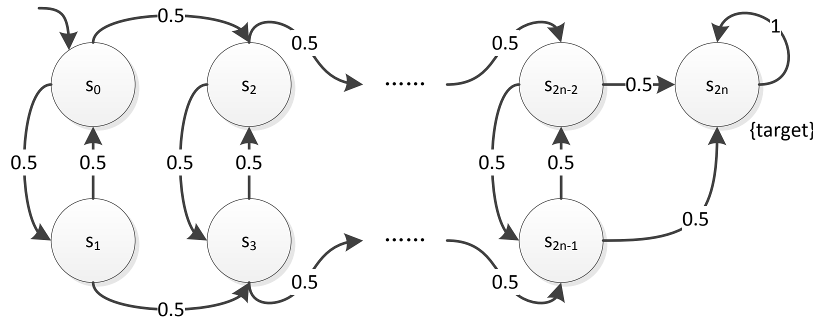


Figure 37: A DTMCR for probability until computation performance tests

Test Case 12. Test performance improvement of Algorithm 14

Model: DTMCR in Figure 37, the model file can be found at Section B.6.

Property: P=? [R<=0 [F s=2*n]]

Conditions: The property are verified under the following conditions:

- C1: "Optimized until" is off;
- C2: "Optimized until" is on.

The following options are set the same for both conditions:

- "Filtered states only" is on;
- using "Top-down with Memoization" approach;
- "ZSCC" is the used connected component type;
- convergence criterion value is 10^{-6}.

Result: The results can be found at Table 7. We can see that the newly introduced algorithm runs faster than the existing one with a bit precision lost. ■

Table 7: Experiment results of Test Case 12

n	Running times		Results	
	C1	C2	C1	C2
100	0.048s	0.028s	0.9999822081707235	0.9999055906544994
200	0.108s	0.065s	0.9999643272542411	0.9998102367278698
300	0.161s	0.091s	0.9999464466574863	0.999714891894469
400	0.334s	0.133s	0.9999285663804587	0.9996195561534309
500	0.422s	0.239s	0.9999106864231516	0.9995242295038884

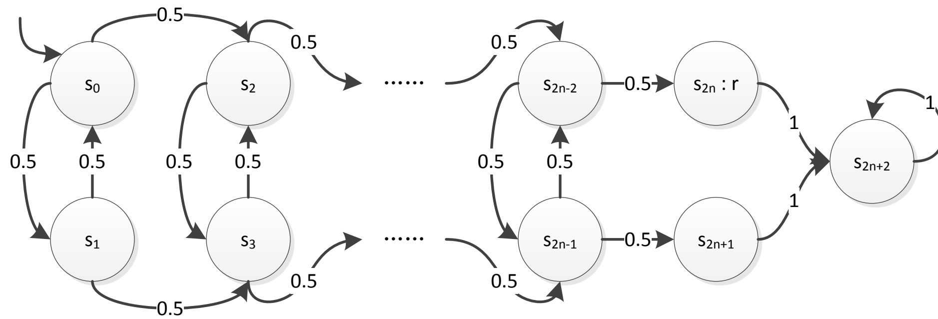


Figure 38: A DTMCR for connected component performance tests

Test Case 13. Performance comparisons between approaches based on ZSCCs and ZCCs

Model: DTMCR in Figure 38, the model file can be found at Section B.7.

Property: P=? [R<=0 [F s=2*n+2]]

Conditions: The property are verified under the following conditions:

- C1: "ZSCC" is the used connected component type;
- C2: "ZCC" is the used connected component type.

The following options are set the same for both conditions:

- "Filtered states only" is on;
- using "Top-down with Memoization" approach;
- "Optimized until" is on;
- convergence criterion value is 10^{-6}.

Result: The results can be found at Table 8. When r=1, the model is the similar case as the DTMCR in Figure 15 (a) and r=0 covers the case as the DTMCR in Figure 15 (b). The result running times are the same as expected. ■

Table 8: Experiment results of Test Case 13

n	Running times			
	r = 1		r = 0	
	C1	C2	C1	C2
100	0.038s	0.048s	0.045s	0.029s
200	0.64s	0.098s	0.068s	0.04s
300	0.107s	0.16s	0.1s	0.073s
500	0.223s	0.361s	0.259s	0.164s
750	0.416s	0.614s	0.424s	0.321s
1000	0.632s	0.959s	0.673s	0.497s

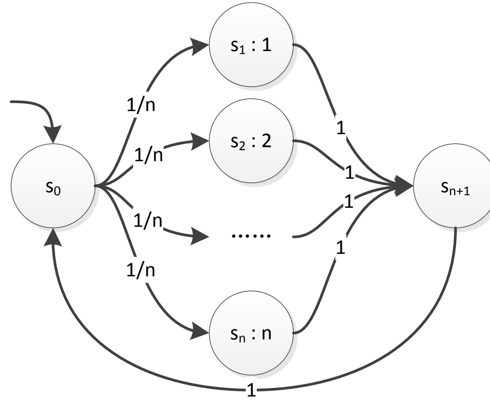


Figure 39: A DTMCR for performance tests of the worklist approach

Test Case 14. Performance comparisons between the "Top-down with Memoization" and "Worklist" approaches

Model: DTMCR in Figure 39 with n=1000, the model generator (a JAVA script generates the model file) can be found at Section A.1. The reason of using this model is that it has more transitions from a single state than a normal DTMCR model, which gives a bad situation of the worklist approach as it needs to push all the following states into the stack when it mimics the recursive procedure call.

Property: P=? [R<=300 [I=step]], where step is an integer constant declared in the properties file.

Conditions: The property are verified under the following conditions with "Filtered states only" is on:

- C1: using "Top-down with Memoization" approach;
- C2: using "Worklist" approach.

Result: The results can be found at Table 9, where OOM stands for "Out Of Memory". The results table shows that the worklist approach has more overheads than the existing recursive procedure calls. This may be due to the additional time spend on instantiating classes, accessing data from a class instance, queue and stack operations. However, it has a better memory performance, which means it can handle more complicate problems. ■

Table 9: Experiment results of Test Case 14

step	Running times	
	C1	C2
151	0.122s	0.215s
301	0.261s	0.372s
451	0.471s	0.561s
601	0.563s	0.937s
1501	1.339s	2.691s
3001	OOM	6.753s
4501	OOM	21.373s

The tests of the performance comparison between the algorithms introduced in this paper for computing $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$ and the algorithms introduced in [11] are also conducted. As the algorithms introduced in [11] are similar to the bottom-up method of the approaches included in this paper. Therefore, the performance comparison results are the same as Test Case 11. The approaches introduced in this paper are slower, while the "Filtered states only" optimization can be applied to significantly improve the performance in certain situations.

8.3.3 Robustness Tests

Finally, the robustness aspect is covered in this section.

Test Case 15. Compute $Pr_{D,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$ over DTMCRs with negative rewards

Model: Same as Test Case 9 by replacing all the reward values r with $-r$.

Property: Same as Test Case 9

Result: An error message is given as shown in Figure 40. ■

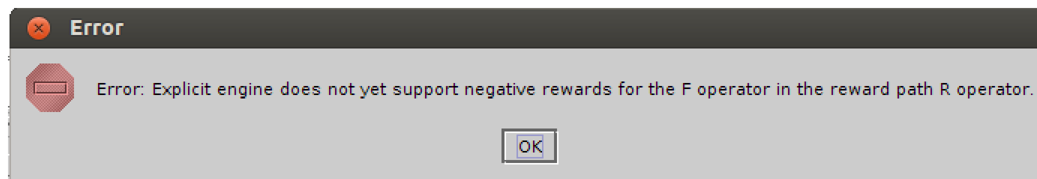


Figure 40: The error message of Test Case 15

8.4 PCTLR Model Checking over MDPRs

In this section, the functionality of the extended PRISM model checking PCTLR formulae over MDPRs will be tested. Because the performance comparison results between different approaches of

MDPRs are the same as DTMCRs, hence only the correctness and robustness aspects are covered.

8.4.1 Correctness Tests

Similar to the test cases of DTMCRs, we will test the correctness of the extended PRISM model checking PCTLR formulae over MDPRs by computing the four cases in Examples 31 to 33 and 38.

Test Case 16. Verify the result in Example 31

Model: MDPR in Figure 18, the model file can be found at Section B.8

Property: Pmax=? [R>1 [I=2]]

Conditions: All four combinations of:

- whether using the "Top-down with Memoization" or the "Worklist" approach;
- whether "Filtered states only" is on or off.

Result: 0.5 for all above four conditions which is the same as the manually computed result. ■

Test Case 17. Verify the result in Example 32

Model: Same as Test Case 16.

Property: filter(print, Pmax=? [R>1 [I=2]])

Conditions: Bottom-up Method.

Result: 0.5 for state s_0 and 0 for the rest states, and it is the same as we computed in Example 32. ■

Test Case 18. Verify the result in Example 33

Model: Same as Test Case 16.

Property: Pmax=? [R>=1 [C<=3]]

Conditions: Same as Test Case 16.

Result: 0.5 for all four conditions which is the same as the result shown in Example 33. ■

Test Case 19. Verify the result in Example 38

Model: MDPR in Figure 23, the model file can be found at Section B.9.

Property: Pmin=? [R>=1 [F s=4]]

Conditions: All sixteen combinations of:

- whether using the "Top-down with Memoization" or the "Worklist" approach;
- whether using the "ZSCC" or the "ZCC" approach;
- whether "Filtered states only" is on or off;
- whether "Optimized until" is on or off.

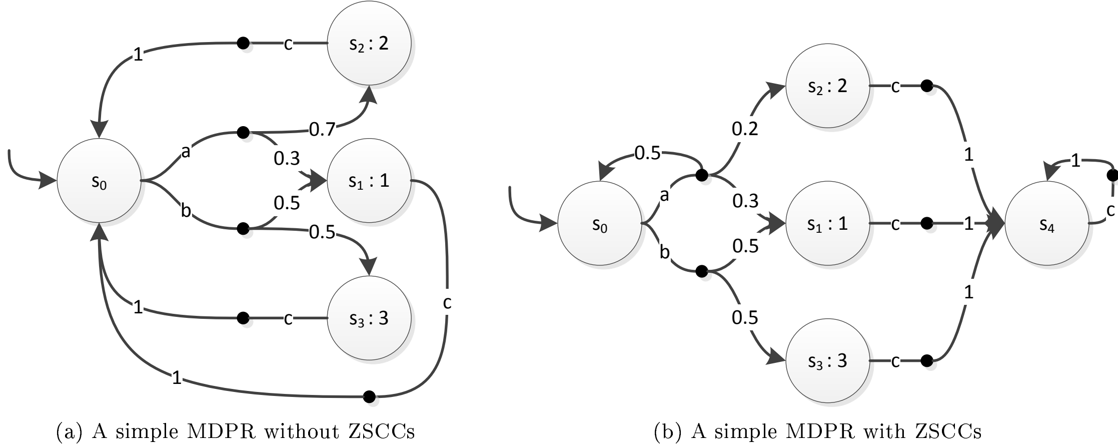
Result: 0.5 for all above sixteen conditions as we expect. ■

Then two simple MDPRs are created to further test the correctness of the extended PRISM model checking PCTLR formulae over MDPRs. The models are shown in Figure 41.

Test Case 20. Verify properties over a simple MDPR without ZSCCs

Model: MDPR in Figure 41 (a), the model file can be found at Section B.10.

Conditions: Same as Test Case 10



(a) A simple MDPR without ZSCCs

(b) A simple MDPR with ZSCCs

Figure 41: Two simple MDPRs

Result: The results can be found at Table 10. As all the properties can be easily computed by hand to verify that they are all correct, thus the computation details are omitted. ■

Table 10: Experiment results of Test Case 20

Properties	Result
Pmin=? [R<=1 [I=3]]	0.3
Pmax=? [R<=1 [I=3]]	0.5
Pmin=? [R<=1 [I=5]]	0.3
Pmax=? [R<=1 [I=5]]	0.5
Pmin=? [R<=5 [C<=4]]	0.75
Pmax=? [R<=5 [C<=4]]	1.0
Pmin=? [R<=2 [C<=4]]	0.09
Pmax=? [R<=2 [C<=4]]	0.25
Pmin=? [R>1 [I=3]]	0.5
Pmax=? [R>1 [I=3]]	0.7
Pmin=? [R>1 [I=7]]	0.5
Pmax=? [R>1 [I=7]]	0.7
Pmin=? [R>=5 [C<=4]]	0
Pmax=? [R>=5 [C<=4]]	0.35
Pmin=? [R>=2 [C<=4]]	1.0
Pmax=? [R>=2 [C<=4]]	1.0

Test Case 21. Verify properties over a simple MDPR with ZSCCs

Model: MDPR in Figure 41 (b), the model file can be found at Section B.11.

Conditions: Same as Test Case 10

Result: The results are shown at Table 11. As all the properties can be easily computed by hand to verify that they are all correct, thus the computation details are omitted. ■

Table 11: Experiment results of Test Case 21

Properties	Result
Pmin=? [R>1 [F s=4]]	0.4
Pmax=? [R>1 [F s=4]]	0.5
Pmin=? [R>2 [F s=4]]	0.0
Pmax=? [R>2 [F s=4]]	0.5
Pmin=? [R<2 [F s=4]]	0.5
Pmax=? [R<2 [F s=4]]	0.6

8.4.2 Robustness Tests

At last, two more test cases will cover the situations when a user mis-uses the functionalities of the extended PRISM model checking PCTLR formulae over MDPRs.

Test Case 22. Compute $Pr_{M,s}(\mathrm{R}^{path}_{\sim r}[\mathrm{F}\ \Phi])$ over MDPRs with negative rewards

Model: Same as Test Case 19 by replacing all the reward values r with $-r$.

Property: Same as Test Case 19

Result: An error message is given as shown in Figure 42. ■

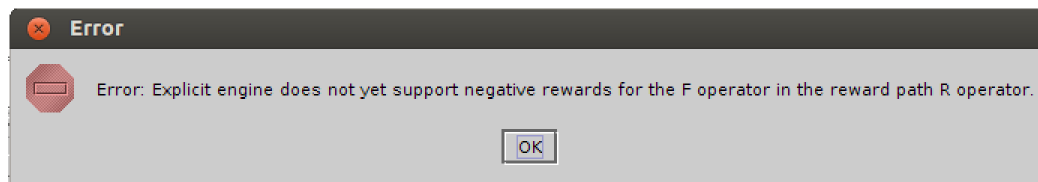


Figure 42: The error message of Test Case 22

Test Case 23. Verify a “cust” filter property with @ss over MDPRs

Model: Same as Test Case 19.

Property: filter(cust, sum(@ss*@v); v: Pmin=? [R>=1 [F s=4]])

Result: An error message is given as shown in Figure 43. ■

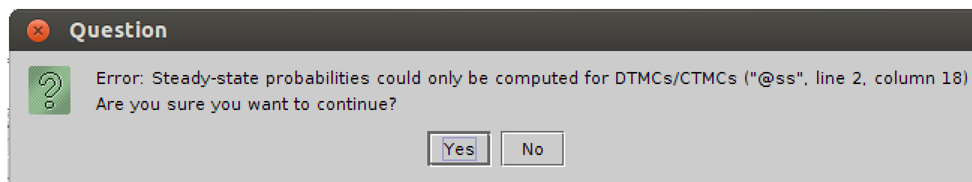


Figure 43: The error message of Test Case 23

9 Conclusion

In this paper, we extended both the syntax and semantics of the PCTL with state reward formulae. The extended logic is called PCTLR which includes instantaneous, cumulative and reachability path reward formulae. The PCTLR can be used to describe properties for both DTMCRs and MDPRs. The PCTLR model checking algorithms over DTMCRs and MDPRs have been discussed and developed. We also provided some performance optimizations towards these algorithms.

Meanwhile, we extended the syntax and semantics of the existing PRISM filter property with an additional operator "cust" which grants PRISM properties with more flexibilities. Several performance optimizations have been applied to the computations of the newly introduced property type.

Besides the theoretical work mentioned above, we also implemented these concepts as extensions of PRISM which is implemented in Java. The extended PRISM has been tested from three aspects: correctness, performance and robustness and the results are satisfying as expected.

9.1 Related work

After this project is done, some papers with the related topic have been found and studied. Two papers with the algorithms similar to the ones presented in this paper are [29, 30]. [29] introduces a model checking approach for analyzing Discrete-time Markov Reward Models (DMRM). A DMRM with a single reward structure is almost defined the same as the DTMCR introduced in this paper, the only difference is that a DMRM does not have transition rewards. In [29], the PCTL is extended as the Probabilistic Reward CTL (PRCTL) with some reward constraints. One type of PRCTL state formulae it introduces is:

$$\mathrm{P}_{\sim p}[\Phi\ \mathrm{U}_J^N\ \Phi],$$

where Φ is a state formula, $\sim \in \{<, >, \leq, \geq\}$, $p \in [0, 1]$, $N \subseteq \mathbb{N} \cup \{\infty\}$ and $J \subseteq \mathbb{R}_{\geq 0}$. This type of PRCTL state formulae can express the same meaning of the following type PCTLR formulae introduced in this paper:

$$\mathrm{P}_{\sim p}[\mathrm{R}_{\sim r}^{path}[\mathrm{F}\ \Phi],$$

by letting $N = \{\infty\}$, $J = \{j \in \mathbb{R}_{\geq 0} \mid j \sim r\}$, then the following formula is equal to the above one:

$$\mathrm{P}_{\sim p}[true\ \mathrm{U}_J^N\ \Phi].$$

The paper introduces an efficient algorithm which is capable to model checking such formulae. The algorithm unfolds the given DMRM step by step from some initial states and stops if some conditions are satisfied. For the unbounded time case, it also uses the concept of the ZSCCs. As at each step, it only keeps unfolding the model along those paths satisfying the reward conditions. Therefore, the sum of total probabilities of the paths will be unfolded is monotonous decreasing when step increases. This grants the correctness for the algorithm to stop once the sum is smaller than p when $\sim \in \{>, \geq\}$. The support of transition rewards should be easy to be extended for this algorithm. However, as the potential problem it may cause as we discussed earlier in this paper, the concept of adding intermediate states for non-zero reward transitions go out of a ZSCC should also be applied.

To sum it up, the algorithm introduced in [29] has the similar approach as introduced in this paper to solve the same problem, while the former one follows a breadth-first traversal of the unfolded path forest of a given DMRM and the later one follows a depth-first traversal of the unfolded path forest of a given DTMCR. They both have the same running complexity.

[30] presents an approach to model check quantile queries for until properties in MDP with non-negative rewards on states. PRCTL supports the type of path formulae $\Phi\ \mathrm{U}_{\leq r}\ \Phi$ is introduced, where Φ is a state formula and $r \in \mathbb{N}$. The sub-problem the paper solves to achieve its final goal is computing the probability $Pr_{M,s}(\Phi'\ \mathrm{U}_{\leq r}\ \Phi)$, which means the same as $Pr_{M,s}(\mathrm{R}_{\leq r}^{path}[\mathrm{F}\ \Phi])$ introduced in this paper when $\Phi' = true$. The way the paper uses to compute $Pr_{M,s}(\Phi'\ \mathrm{U}_{\leq r}\ \Phi)$ is solving a

linear program. For instance, given an MDP with non-negative integer state reward M, let $x_{M,s,i}$ be $Pr^{max}_{M,s}(true \ \mathrm{U}_{\leq i} \ \Phi)$. To compute $Pr^{max}_{M,s}(true \ \mathrm{U}_{\leq r} \ \Phi)$ for all state $s \in S$, the following linear program will be solved:

$$\begin{aligned}
&\text{minimize } \sum x_{M,s,i} \text{ subject to the constraints:} \\
&x_{M,s,i} \geq 0 && \text{for all } s \in S \text{ and } i \leq r \\
&x_{M,s,i} = 1 && \text{for all } s \in Sat(\Phi) \text{ and } i \leq r \\
&x_{M,s,i} \geq \sum_{s' \in S} Steps(s,a)(s') \cdot x_{M,s',i-r_s(s')}. && \text{for all } s \in S \text{ and } a \in A(s) \text{ and } r_s(s') \leq i \leq r
\end{aligned}$$

The running complexity to solve this linear program is polynomial to the number of equations, which is greater than r multiplies |S|. To compute $Pr^{max}_{M,s}(true \ \mathrm{U}_{\leq r} \ \Phi)$ with the approach introduced in this paper, we need at most r multiplies |S| equations and each equation can be computed in linear time of the operations included in it if the equation is not used to define a state belongs to a ZSCC. Therefore, our approach yields a generally better running complexity than the approach introduced in [30] and in the worst case when the whole model is a $ZSCC^n$, our approach regresses to the same running complexity as solving the above LP problem.

9.2 Acknowledgment

At the end of this paper, I would like to thank my supervisor Flemming Nielson for helpful discussions and valuable remarks and suggestions. David Parker is thanked for his nicely introduction and patient guidance of modifying PRISM. Also I would appreciate the enormous support I got from my family.

References

[1] C. Baier and J.-P. Katoen, *Principles of Model Checking*, ch. 10.1 Markov Chains, p. 747. The MIT Press, 2008.

[2] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Verifying continuous time Markov chains," in *Proc. 8th International Conference on Computer Aided Verification (CAV'96)* (R. Alur and T. Henzinger, eds.), vol. 1102 of *LNCS*, pp. 269–276, Springer, 1996.

[3] C. Baier and J.-P. Katoen, *Principles of Model Checking*, ch. 10.6 Markov Decision Processes, p. 833. The MIT Press, 2008.

[4] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)* (G. Gopalakrishnan and S. Qadeer, eds.), vol. 6806 of *LNCS*, pp. 585–591, Springer, 2011.

[5] D. Parker, G. Norman, and M. Kwiatkowska, *PRISM Manual version 4.0.3*, ch. The PRISM Language. University of Oxford.

[6] D. Parker, G. Norman, and M. Kwiatkowska, *PRISM Manual version 4.0.3*, ch. Property Specification. University of Oxford.

[7] D. Parker, G. Norman, and M. Kwiatkowska, *PRISM Manual version 4.0.3*, ch. Configuring PRISM / Computation Engines. University of Oxford.

[8] H. Yue, "Model checking real life." The final report of the autumn special course, 2012.

[9] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, pp. 102–111, 1994.

[10] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *Proceedings of the 7th international conference on Formal methods for performance evaluation*, SFM'07, (Berlin, Heidelberg), pp. 220–270, Springer-Verlag, 2007.

[11] H. Yue, "Modifying prism." The final report of the autumn special course, 2012.

[12] D. Parker, G. Norman, and M. Kwiatkowska, *PRISM Manual version 4.0.3*, ch. Property Specification / Filters. University of Oxford.

[13] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, ch. 15 Dynamic Programming, p. 359. McGraw-Hill Higher Education, 2nd ed., 2001.

[14] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, ch. 4 Divide-and-Conquer, p. 65. McGraw-Hill Higher Education, 2nd ed., 2001.

[15] M. Kwiatkowska and D. Parker, "Advances in probabilistic model checking," in *Software Safety and Security - Tools for Analysis and Verification* (T. Nipkow, O. Grumberg, and B. Hauptmann, eds.), vol. 33 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pp. 126–151, IOS Press, 2012.

[16] R. A. Minlos, *Encyclopedia of Mathematics*, ch. Cylinder Set. Springer, 2001.

[17] C. Baier and J.-P. Katoen, *Principles of Model Checking*, ch. 6.2 Computation Tree Logic, p. 317. The MIT Press, 2008.

[18] C. Courcoubetis and M. Yannakakis, "Verifying temporal properties of finite state probabilistic programs," in *Proc. 29th Annual Symposium on Foundations of Computer Science (FOCS'88)*, pp. 338–345, IEEE Computer Society Press, 1988.

[19] C. Courcoubetis and M. Yannakakis, "The complexity of probabilistic verification," *Journal of the ACM*, vol. 42, no. 4, pp. 857–907, 1995.

[20] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, ch. 22.5 Strongly connected components, p. 615. McGraw-Hill Higher Education, 2nd ed., 2001.

[21] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, ch. 24.1 The Bellman-Ford algorithm, p. 651. McGraw-Hill Higher Education, 2nd ed., 2001.

[22] F. Duan, "A faster algorithm for shortest-path–spfa," pp. 207–212, Computer Center, Southwest Jiaotong University, Chengdu 610031, China, 1994.

[23] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker, "Automated verification techniques for probabilistic systems," in *Formal Methods for Eternal Networked Software Systems (SFM'11)* (M. Bernardo and V. Issarny, eds.), vol. 6659 of *LNCS*, pp. 53–113, Springer, 2011.

[24] V. Sazonov, *Encyclopedia of Mathematics*, ch. Probability Space. Springer, 2001.

[25] A. Bianco and L. D. Alfaro, "Model checking of probabilistic and nondeterministic systems," pp. 499–513, Springer-Verlag, 1995.

[26] C. Costas and Y. Mihalis, "Markov decision processes and regular events," in *Automata, Languages and Programming* (M. Paterson, ed.), vol. 443 of *Lecture Notes in Computer Science*, pp. 336–349, Springer Berlin Heidelberg, 1990.

[27] C. Baier, "On algorithmic verification methods for probabilistic systems." Habilitation thesis, Fakultät für Mathematik & Informatik, Universität Mannheim, 1998.

[28] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*, ch. Vistor, p. 525. Prentice Hall, 1st ed., 2002.

[29] S. Andova, H. Hermanns, and J.-P. Katoen, "Discrete-time rewards model-checked," in *Formal Modeling and Analysis of Timed Systems* (K. Larsen and P. Niebert, eds.), vol. 2791 of *Lecture Notes in Computer Science*, pp. 88–104, Springer Berlin Heidelberg, 2004.

[30] M. Ummels and C. Baier, "Computing quantiles in markov reward models," in *Foundations of Software Science and Computation Structures* (F. Pfenning, ed.), vol. 7794 of *Lecture Notes in Computer Science*, pp. 353–368, Springer Berlin Heidelberg, 2013.

A Model Generators

A.1 Generator based on the model pattern in Figure 39

```java
1  import java.io.BufferedWriter;
2  import java.io.File;
3  import java.io.FileWriter;
4  import java.io.IOException;
5
6  /**
7   *
8   * author Han Yue
9   */
10 public class DTMCRPerfStackLoad {
11
12     /**
13      * param args the command line arguments
14      */
15     public static void main(String[] args) throws IOException {
16         int size = 10000;
17
18         File file = new File("DTMCRPerfStackLoad.pm");
19         BufferedWriter output = new BufferedWriter(new FileWriter(file))
               ;
20         output.write("dtmc\r\n");
21         output.write("\r\n");
22
23         output.write("module model\r\n");
24         output.write("\ts : [0.." + (size + 2) + "] init 0;\r\n");
25
26         for (int i = 1; i <= size; i++) {
27             output.write("\t[] s=0 -> (s'=" + i + ");\r\n");
28             output.write("\t[] s=" + i + " -> (s'=" + (size + 2) + ");\r
                   \n");
29         }
30         output.write("\t[] s=" + (size + 2) + " -> (s'=0);\r\n");
31         output.write("endmodule\r\n");
32         output.write("\r\n");
33
34         output.write("rewards\r\n");
35         output.write("\ttrue : s;\r\n");
36         output.write("endrewards");
37         output.close();
38     }
39 }
```

B Model Files

B.1 DTMCR in Figure 31

```
1 dtmc
2
3 module FilterTester
```

```
4
5       s : [1..5] init 1;
6
7       [] s<5 -> (s'=s+1);
8       [] s=5 -> (s'=1);
9
10  endmodule
11
12  rewards
13      true : 1;
14  endrewards
```

B.2 DTMCR in Figure 3

```
1   dtmc
2
3   module VendingMachine
4
5       s : [0..5] init 0;
6
7       [] s=0 -> (s'=1);
8       [] s=1 -> (s'=0);
9       [fee] s=1 -> (s'=2);
10      [fee] s=1 -> (s'=2);
11      [fee] s=1 -> (s'=2);
12      [fee] s=1 -> (s'=2);
13      [fee] s=1 -> (s'=2);
14      [fee] s=1 -> (s'=2);
15      [fee] s=1 -> (s'=2);
16      [fee] s=1 -> (s'=2);
17      [fee] s=1 -> (s'=2);
18      [] s=2 -> 0.5 : (s'=3) + 0.5 : (s'=4);
19      [] s=3 -> (s'=5);
20      [] s=4 -> (s'=5);
21      [fee] s=5 -> (s'=0);
22
23  endmodule
24
25  rewards
26      s=3 : 1;
27      s=4 : 2;
28      [fee] true : -0.01;
29  endrewards
```

B.3 DTMCR in Figure 8

```
1   dtmc
2
3   module ZSCC
4
5       s : [0..6] init 0;
6
7       [] s=0 -> 0.5 : (s'=1) + 0.5 : (s'=5);
8       [cost] s=1 -> (s'=2);
```

```
9       [] s=1 -> (s'=3);
10      [] s=1 -> (s'=4);
11      [] s=1 -> (s'=4);
12      [] s=2 -> 0.25 : (s'=1) + 0.25 : (s'=3) + 0.5 : (s'=4);
13      [] s=3 -> 0.5 : (s'=1) + 0.5 : (s'=2);
14      [] s=4 -> (s'=4);
15      [] s=5 -> (s'=6);
16      [] s=6 -> (s'=6);
17
18  endmodule
19
20  rewards
21      s=5 : 0;
22      [cost] true : 1;
23  endrewards
```

B.4 DTMCR in Figure 36

```
1   dtmc
2
3   module dice
4
5       // local state
6       s : [0..7] init 0;
7       // value of the dice
8       d : [0..6] init 0;
9
10      [] s=0 -> 0.5 : (s'=1) + 0.5 : (s'=2);
11      [disturb] s=1 -> (s'=3);
12      [] s=1 -> (s'=4);
13      [] s=2 -> 0.5 : (s'=5) + 0.5 : (s'=6);
14      [disturb] s=3 -> (s'=3);
15      [] s=3 -> (s'=7) & (d'=1);
16      [] s=4 -> 0.5 : (s'=7) & (d'=2) + 0.5 : (s'=7) & (d'=3);
17      [] s=5 -> 0.5 : (s'=7) & (d'=4) + 0.5 : (s'=7) & (d'=5);
18      [] s=6 -> 0.5 : (s'=2) + 0.5 : (s'=7) & (d'=6);
19      [] s=7 -> (s'=7);
20
21  endmodule
22
23  rewards
24      [] s<7 : 1;
25      d = 1 : 1;
26      d = 2 : 2;
27      d = 3 : 3;
28      d = 4 : 4;
29      d = 5 : 5;
30      d = 6 : 6;
31  endrewards
```

B.5 DTMCR in Figure 36

```
1   dtmc
2
```

```
3  const int n = 10;
4
5  module DTMCRPrIPerfTester
6
7      s : [0..n] init 0;
8
9      [] s<n -> (s'=s+1);
10     [] s=n -> (s'=n);
11
12 endmodule
13
14 rewards
15     true : 0;
16 endrewards
```

B.6 DTMCR in Figure 36

```
1  dtmc
2
3  const int n = 10;
4
5  module DTMCROptimizedUntilTester
6
7      s : [0..2*n] init 0;
8
9      [] s<2*n & mod(s, 2)=0 -> 0.5 : (s'=s+1) + 0.5 : (s'=s+2);
10     [] s<2*n-1 & mod(s, 2)=1 -> 0.5 : (s'=s-1) + 0.5 : (s'=s+2);
11     [] s=2*n-1 -> 0.5 : (s'=s-1) + 0.5 : (s' = s+1);
12     [] s=2*n -> true;
13
14 endmodule
15
16 rewards
17     true : 0;
18 endrewards
```

B.7 DTMCR in Figure 36

```
1  dtmc
2
3  const int n = 500;
4  const double r = 0;
5
6  module DTMCRCCPerfTester
7
8      s : [0..2*n+2] init 0;
9
10     [] s<2*n & mod(s, 2)=0 -> 0.5 : (s'=s+1) + 0.5 : (s'=s+2);
11     [] s<2*n & mod(s, 2)=1 -> 0.5 : (s'=s-1) + 0.5 : (s'=s+2);
12     [] s>=2*n -> (s'=2*n+2);
13
14 endmodule
15
16 rewards
```

```
17      s=2*n : r;
18  endrewards
```

B.8 MDPR in Figure 18

```
1   mdp
2
3   module VendingMachine
4
5       s : [0..4] init 0;
6
7       [initialize] s=0 -> (s'=1);
8       [cancel] s=1 -> (s'=0);
9       [select] s=1 -> 0.5 : (s'=2) + 0.5 : (s'=3);
10      [notify] s=2 -> (s'=4);
11      [notify] s=3 -> (s'=4);
12      [release] s=4 -> (s'=0);
13
14  endmodule
15
16  rewards
17      s=2 : 1;
18      s=3 : 2;
19      [select] true : -0.01;
20      [release] true : -0.01;
21  endrewards
```

B.9 MDPR in Figure 23

```
1   mdp
2
3   module ZSCC
4
5       s : [0..5] init 0;
6
7       [a] s=0 -> 0.5 : (s'=1) + 0.5 : (s'=5);
8       [b] s=0 -> true;
9       [a] s=1 -> (s'=2);
10      [b] s=1 -> (s'=2);
11      [a] s=2 -> 0.5 : (s'=1) + 0.2 : (s'=3) + 0.3 : (s'=4);
12      [a2] s=2 -> (s'=4);
13      [a] s=3 -> (s'=4);
14      [a] s=4 -> (s'=4);
15      [a] s=5 -> (s'=5);
16
17  endmodule
18
19  rewards
20      s=3 : 1;
21      [b] true : 1;
22  endrewards
```

B.10 MDPR in Figure 41 (a)

```
1  mdp
2
3  module SimpleMDPR1
4
5      s : [0..3] init 0;
6
7      [] s=0 -> 0.3 : (s'=1) + 0.7 : (s'=2);
8      [] s=0 -> 0.5 : (s'=1) + 0.5 : (s'=3);
9      [] s>0 -> (s'=0);
10
11 endmodule
12
13 rewards
14     s=1 : 1;
15     s=2 : 2;
16     s=3 : 3;
17 endrewards
```

B.11 MDPR in Figure 41 (b)

```
1  mdp
2
3  module SimpleMDPR2
4
5      s : [0..4] init 0;
6
7      [] s=0 -> 0.5 : true + 0.3 : (s'=1) + 0.2 : (s'=2);
8      [] s=0 -> 0.5 : (s'=1) + 0.5 : (s'=3);
9      [] s>0 -> (s'=4);
10
11 endmodule
12
13 rewards
14     s=1 : 1;
15     s=2 : 2;
16     s=3 : 3;
17 endrewards
```