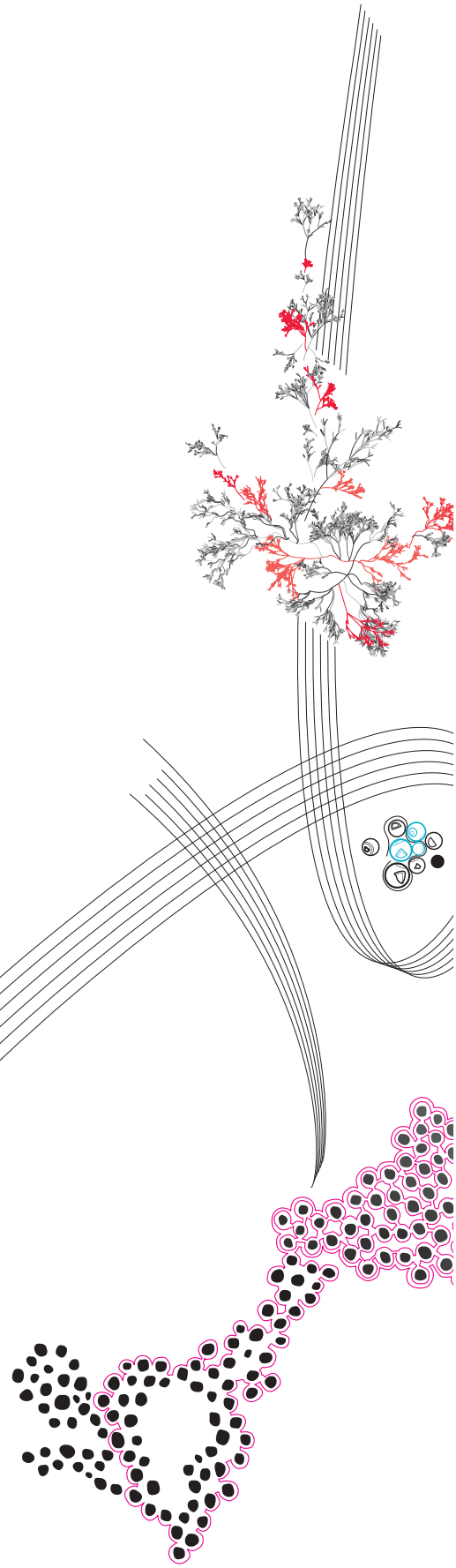


MSc Computer Science
Final Project

# The State of Multi-Objective Model Checking

Mark van Wijk

Committee:
dr. Arnd Hartmanns
dr. ir. Pieter-Tjerk de Boer

November, 2024

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

# The State of Multi-Objective Model Checking

Mark van Wijk

University of Twente, The Netherlands

**Abstract.** Probabilistic model checkers can be used to formally verify the behaviour of a system. We can analyse a quantitative property like the chance of success, the expected costs, or uptime of a system. Although traditionally only one property is considered at the same time, in most real-life processes multiple properties influence each other. For example, the highest expected uptime of a system might be disproportionately expensive. Therefore, we focus on multi-objective probabilistic model checking, which allows us to consider the trade-offs between several properties. Since model checkers are used to analyse safety-critical processes such as space travel, it is important that they are reliable. Our research aims at assessing and improving the state of probabilistic model checkers. We show that there are several mistakes in state-of-the-arts model checkers. We then show the origins of these mistakes and solve most of them. We then replicate the most common multi-objective model checking papers using the Modest Toolset to investigate the validity of these algorithms. We found several mistakes and curiosities in the replicated papers.

# Table of Contents

## 1 Introduction

Imagine sending a Mars rover on a mission with limited battery life, where each decision could mean mission success or failure. How do you ensure optimal performance while managing various risks? As is common in such cases, we start by modelling the underlying process. In order to do so, we need to determine the nondeterministic choices we can make, such as whether to drive to a crater or to stay still and collect solar power. Moreover, we need to determine the probabilities of the outcomes for each of the choices, such as the rover crashing. When we do this, the process can be modelled using a Markov decision process (MDP) [43,64].

By modelling processes as MDPs, we can analyse the behaviour of the modelled process. We can address quantitative questions such as "what is the probability that a robot will successfully hand over an item?" [71] or "what is the minimum cost of an attack on a smart grid?" [9]. Quantitative questions are so fundamental in probabilistic model checking that they have their own name. We will from now on refer to quantitative questions as properties.

There are several ways to evaluate properties. In this research, we will focus on probabilistic model checking. In probabilistic model checking, the entire model is explored to compute exact results. However, when MDPs grow too large to fully explore, it is possible to use partial exploration [51], statistical model checking [41,74] or reinforcement learning [46]. These approaches can typically handle much larger models, at the expense of losing some accuracy.

In addition to several ways to evaluate, we also distinguish several types of properties. First, there are probabilistic reachability properties. These properties express the probability of reaching a certain state, such as "what is the probability of a robot successfully handing over an item?". The other two categories of properties use rewards. A reward is a value given when a positive or negative event occurs. We want to maximise a positive reward, such as the uptime of a system, while we want to minimise a negative reward such as the costs. We consider two types of properties: expected properties and strict properties. Expected properties calculate the expected reward. Strict properties on the other hand, consider the highest or lowest value a reward might have. If we had the choice of taking €90 or a probability of 0.99 to get €100, the maximum expected reward would be €99, but the maximum strict reward would be €90. Both types of properties have important applications. For a plane, low expected fuel consumption would be beneficial for monetary reasons, but it is more critical that fuel does not run out for safety reasons. On the other hand, a company might be more interested in getting a high expected return on investment and less in the maximum return it could theoretically achieve. Although strict properties have important applications, we consider only expected properties in this paper.

So far, we have mainly considered properties with one variable. These are single-objective properties and have been thoroughly investigated. In practice, however, engineers often must find a balance between several variables. For example, we might want a robot to complete as many tasks as possible, but we also need to ensure that its battery does not run out [69]. Conventional

single-objective approaches cannot capture such constraints. This is where multi-objective model checking comes in. Multi-objective model checking allows us to find the trade-offs between several variables.

Multi-objective model checking has received less research attention than single-objective approaches. However, several algorithms [10,28,30,36] and tools [6,31,40,52,53] are available to evaluate multi-objective properties.

It is important that the results provided by these tools are actually correct. Errors in model checkers are problematic, as users need to be able to trust the results. Model checkers are used in safety-critical applications, such as space travel [70] or medical devices [20] to formally verify their systems. Failures in these domains can cost billions of dollars and even lives. Therefore, it is crucial that when model checkers are used in these domains, they are accurate and reliable.

In an attempt to guarantee the correctness of these tools, some algorithms have been proven correct [29], but for some only parts have been proven [30]. This is important because this means that there might be mistakes in these unproven algorithms. For example, one of the most prevalent algorithms in probabilistic non-multi-objective model checking was only shown to not have a reliable upper-bound in 2014 [34]. Even if these proofs are given, classical proofs such as [30] are error prone because they are susceptible to human error. Moreover, the actual implementations of the algorithms have not been proven either.

Proving the correctness of probabilistic model checking algorithms using interactive theorem provers [60] is more reliable [73]. However, this is still an active area of research [39]. This means that most implementations have not yet been proven in this way, and thus there may still be errors in the implementations.

### 1.1 Related Work

A benchmark of multi-objective properties on several multi-objective model checkers was performed in [2]. This research assessed whether the speed at which tools determined whether it was possible to achieve certain combinations of values for the given variables. However, it did not verify the accuracy of the results of the properties. Only the speed of the tools to provide any answer was considered. In fact, some tools provided different results for the same models. However, the research did not investigate which answers were correct.

For single-objective properties, the accuracy of probabilistic model checking tools has been examined [37]. It was shown that there is a large difference in the reliability and performance of different implementations.

In addition, commonly used algorithms have been shown not to always be reliable. Value iteration, which is one of the most common algorithms used in probabilistic model checking, has been shown to not have reliable upper bounds [34].

An interesting approach is taken for tools that verify C and Java programs. During the yearly SV-COMP competition, verifiers participate in a competition where they are penalised significantly for incorrect results [27]. By doing this, mistakes can be detected in the verifiers.

Efforts have also been made to provide stronger guarantees than what we can provide with our proposed tests. There has been a start to proof the correctness of probabilistic algorithms [39]. However, this is still an active area of research.

### 1.2 Origin Of Work

Before this research, we have tried to replicate a small part of a certain multi-objective model checking paper [30]. We did this as part of the Capita Selecta course offered by the University of Twente, in which students have to select a research topic and write a paper about this topic.

During this preliminary research, we compared our implementation with the existing implementation in the existing model checker PRISM [53]. In doing this, we observed unexpected behaviour in PRISM as we will show at the start of Section 4. We were not sure at this point whether the origin of these mistakes was in the theory behind the model checker, or in the implementation. We did also not know whether such mistakes were present in new tools. This also led to other interesting observations in the types of models that cannot be evaluated, which we will explain in Section 6.

Based on these two observations, we came up with the following research questions:

- **RQ1** Which mistakes are present in existing model checkers?
- **RQ2** What is the origin of the mistakes in existing model checkers?
- **RQ3** Which mistakes are present in multi-objective model checking theory?
- **RQ4** How should invalid models be treated?

### 1.3 Contributions

Our goal with this research is to improve the state of multi-objective model checking on three facets. First, we aim to improve the existing multi-objective model checking tools. Next, we want to improve the current theory by highlighting mistakes in existing papers and developing some of our own theory. Lastly, we want to introduce a new multi-objective model checking tool such that more tools can be compared to each other to find mistakes. We will explain these steps in more detail.

First, we evaluate the most prevalent probabilistic model checkers that support multi-objective properties [31,40,53] on several types of models in Section 4. Using this approach, we show that there are mistakes and inconsistencies in the current generation of probabilistic model checking tools.

We then discuss these mistakes in greater depth and find the causes of these problems in Section 5. We made pull requests to resolve these issues, or, if fixing these problems was too time-consuming, we reported the problems to the maintainers of the relevant tools to improve the existing model checkers.

A subset of MDPs with rewards are commonly considered modelling errors in the literature and are the most common tools. We will consider these models in Section 6. We found in Section 5 that the commonly excluded subset of

models can be quite tricky to identify in practice. Therefore, we define our own restriction in Section 6.2. This approach excludes more models than the approaches used by Storm, ePMC and PRISM, but it is easier to implement and also excludes more degenerate cases.

After primarily considering existing tools, we shift our focus to new tools. In Section 7 we provide the reader with a more in-depth explanation of the two common algorithms than their papers, since we are not restricted by a page limit. Moreover, we explain geometric computations that are required in [30], but for which little guidance is given. Here, we also introduce some new theory.

We then aim to locate mistakes in the original papers [29,30] by replicating these papers. We do so by implementing the algorithms shown in the papers in the Modest Toolset [35]. We discuss the mistakes we found in these papers and highlight important details to consider when implementing these algorithms in Section 8. By learning from the mistakes found in existing tools, we aim to provide a more reliable model checker.

Lastly, we validate our changes to existing tools and our new implementation in the Modest Toolset in Section 9 to show that the new approaches are still not completely consistent, but a significant improvement has been made on the evaluated models.

## 2 Background

Before we proceed, we establish the notation that we will use. Moreover, we provide background knowledge so that the reader can familiarise themselves with the subject of probabilistic multi-objective model checking. A summary of the notation can be found in Appendix A.

### 2.1 Notation

We use several sets in this paper. First, we have natural numbers including 0, which we denote as $\mathbb{N}_0$. We also commonly use positive natural numbers, which are denoted as $\mathbb{N}_+$. If we also include $\infty$ into these two sets, we get the sets $\mathbb{N}_{0,\infty}$ and $\mathbb{N}_{+,\infty}$, respectively. We also use the set of real numbers, which we denote by $\mathbb{R}$. The empty set is denoted as $\emptyset$ and the powerset of $X$ as $\mathcal{P}(X)$.

We also use injective and partial functions. An injective function has a result for every input in its domain. In contrast, a partial function may not be defined for some inputs. We denote an injective function $f$, which has $X$ and range $Y$ as $f\colon X \to Y$. If $f$ is a partial function, we denote it as $f\colon X \rightharpoonup Y$.

To model uncertainty, we use probability distributions. A probability distribution is a function that maps all elements of a finite set $X$ to a probability. We denote this by $\mathit{Dist}\colon X \to [0,1]$. Since these are probabilities, their sum must equal one: $\sum_{x \in X} \mathit{Dist}(x) = 1$.

We often need a function that is 1 when a condition $P$ is satisfied and 0 if not. We use the Iverson bracket to denote this $[P] = \begin{cases} 1, & \text{if } P \text{ is } \mathit{true} \\ 0, & \text{otherwise} \end{cases}$.

To denote a tuple $\boldsymbol{x} \in \mathbb{R}^n$, we use $\boldsymbol{x} = \langle x_1, \dots x_n \rangle$. We can always refer to the $i$th element of a tuple $\boldsymbol{x}$ as $x_i$. A tuple of size $n$ with all $n$ elements being 0 is denoted as $\mathbf{0}_n$.

### 2.2 Markov Decision Process

The type of model that we consider for multi-objective probabilistic model checking is a Markov decision process (MDP) [43,64]. An MDP is an automaton with both nondeterministic and probabilistic choices. This means that when we are in a state, a nondeterministic choice to needs to be selected. This nondeterministic choice is referred to as an action. If we take this action, we end up in a new state via a probabilistic choice associated with the nondeterministic choice. This probabilistic choice is based on a probability distribution over the successor states.

*Example 1.* We will look at a case of a programmer who considers switching jobs. After some research, they found that a few months ago a huge group of local companies teamed up to create a transparent application process. The companies were trying to find a way to remove human bias from the application process. To do this, they came up with a pilot application process based on "hiring points".

The company hires the candidate with the highest number of hiring points. It turns out that the programmer cannot influence most of these hiring points in a few months. A lot of the hiring points are assigned to based on work experience. However, certain certifications also provide hiring points, two of those certifications the programmer does not yet possess.

The certification can be obtained by taking a physical exam. The exam has a 85% passing rate, but it can only be taken every two years, which means that the programmer can only take this exam once during their current job search. After getting this certification, they can get the second certification, which is a higher level of the first certification. The second certification can be obtained via an online exam, which can be done as often as the candidate wants. Unfortunately, this exam only has a 20% passing rate.

We can model this process using the MDP shown in Figure 1. Here, $s_I$ is the state in which the programmer starts, denoted by the arrow without input. In this state, there are two nondeterministic choices: ignore the certification process (*stop*) or take the first exam ($try_1$). If they take the exam, they have 0.85 (85%) chance of passing it, which means that they end up in $passed_1$. If they fail the exam, they end up in the *finished* state, since they cannot take the exam again during the job search and cannot take the second exam without passing the first. From $passed_1$, they again have the nondeterministic choice of taking the second exam ($try_2$) or to quit (*stop*). If they take the exam, they have a 0.2 (20%) chance of success, otherwise they fail it and get the same nondeterministic choice again.

We can now formally describe Markov decision processes in Definition 1 and show the formal notation of Example 1 in Example 2.

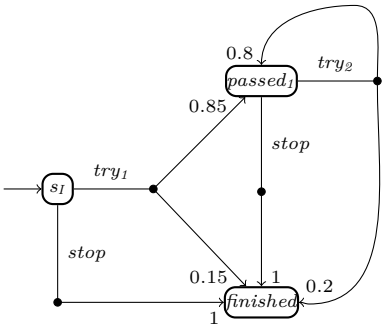



**Fig. 1.** Markov decision process of a certification process

**Definition 1.** *A **Markov decision process** (MDP) $\mathcal{M}$ is a tuple $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$, where:*

- *$\mathcal{S}$ is the state space: a finite set of states,*
- *$s_I \in \mathcal{S}$ is the initial state,*
- *$\mathcal{A}$ are all supported actions: a finite set of actions, and*
- *$\delta \colon \mathcal{S} \to \mathcal{A} \rightharpoonup \mathit{Dist}(\mathcal{S})$ is the transition function.*

*Example 2.* Figure 1 is the MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ such that:

- $\mathcal{S} = \{s_I, \mathit{passed}_1, \mathit{finished}\}$,
- $\mathcal{A} = \{\mathit{try}_1, \mathit{try}_2, \mathit{stop}\}$, and
- $$\delta = \left\{ \begin{array}{l} s_I \mapsto \left\{ \begin{array}{l} \mathit{try}_1 \mapsto \{s_I \mapsto 0, \mathit{passed}_1 \mapsto 0.85, \mathit{stop} \mapsto 0.15\}, \\ \mathit{stop} \mapsto \{s_I \mapsto 0, \mathit{passed}_1 \mapsto 0, \mathit{stop} \mapsto 1\} \end{array} \right\}, \\ \mathit{passed}_1 \mapsto \left\{ \begin{array}{l} \mathit{try}_2 \mapsto \{s_I \mapsto 0, \mathit{passed}_1 \mapsto 0.8, \mathit{stop} \mapsto 0.2\}, \\ \mathit{stop} \mapsto \{s_I \mapsto 0, \mathit{passed}_1 \mapsto 0, \mathit{stop} \mapsto 1\} \end{array} \right\}, \\ \mathit{finished} \mapsto \emptyset \end{array} \right\}.$$

In a state, not all actions might be available. For example, in Figure 1, in $s_I$ actions can be chosen $\mathit{try}_1$ and $\mathit{stop}$, but cannot be performed $\mathit{try}_2$. To obtain the set of all available actions, we use Definition 2.

**Definition 2.** *The **available actions for a state** in MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ are given by the function $\alpha_{\mathcal{M}} \colon S \to \mathcal{P}(\mathcal{A})$ such that:*

$$\alpha_{\mathcal{M}}(s) \stackrel{def}{=} \{a \in \mathcal{A} \mid \delta(s)(a) \text{ is defined}\}.$$

*Example 3.* We can see that in Figure 1, there are no actions available in the state *finished*. This can also be observed using Definition 2 and Example 2: $\alpha_{\mathcal{M}}(\mathit{finished}) = \emptyset$.

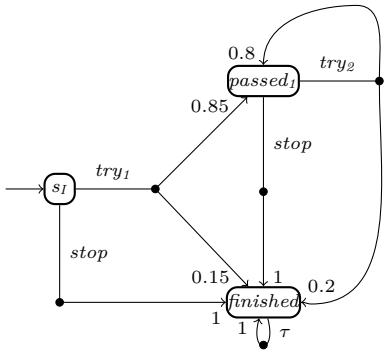



**Fig. 2.** Deadlock-free Markov decision process of a certification process

If there are no available actions in a state, this is called a deadlock since we cannot proceed from this state. Many algorithms work only for MDPs without deadlocks. Fortunately, it is possible to eliminate deadlocks with a simple transformation. For each deadlock state, a $\tau$ action is added that goes to the same deadlocked state. For the remainder of the paper, we assume that all MDPs have been made deadlock-free by applying this transformation, which is formalised in Proposition 1.

**Proposition 1.** *Let $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ be an MDP. Let $\mathcal{M}' = \langle \mathcal{S}, s_I, \mathcal{A} \cup \{\tau\}, \delta' \rangle$ be the MDP with $\delta'$ the function such that:*

$$\delta'(s) \stackrel{def}{=} \begin{cases} \delta(s), & \text{if } \alpha_{\mathcal{M}}(s) \neq \emptyset \\ \{\tau \mapsto \{s' \mapsto [s = s'] \mid s' \in \mathcal{S}\}\}, & \text{otherwise} \end{cases}$$

*Then $\mathcal{M}'$ is deadlock-free and the only difference in supported executions of $\mathcal{M}$, is that there might be an infinite sequence of $\tau$ loops appended to executions in $\mathcal{M}'$.*

*Example 4.* If we apply Proposition 1 to Figure 1, we obtain the deadlock-free MDP shown in Figure 2. The only change is the $\tau$ loop added to the *finished* state.

### 2.3 Path

We model processes using MDPs, so that we can analyse the process by analysing the MDP. Analysing an MDP in this context means computing what we would expect to observe during an execution of the MDP.

Executing an MDP means starting in a state, selecting an available action, going to the next state based on the probability distribution, and repeating this process from this next state. Such an execution is described by a path. Notice that since all deadlocks have been removed, executions can continue indefinitely.

**Definition 3.** *A **path** in MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ is an infinite sequence $\pi = s_1 a_1 s_2 a_2 \ldots$, such that:*

$$\forall i \in \mathbb{N}_{+,\infty} \colon s_i \in \mathcal{S} \wedge a_i \in \alpha_{\mathcal{M}}(s_i) \wedge \delta(s_i)(a_i)(s_{i+1}) > 0.$$

*Example 5.* If we start in $s_I$ in Figure 2 and decide to take *try*$_1$ and by chance end up in *passed*$_1$, from which we take *stop* to end up in *finished*, we get the path: $s_I$ *try*$_1$ *passed*$_1$ *stop finished* $\tau$ *finished* $\tau$ *finished* $\ldots$ Notice that we cannot leave the *finished* state, so the path will end with an infinite number of $\tau$ actions to go to the *finished* state.

We will also need to define which states are reachable from each other. Being reachable means that there exists a path from one state to another, with arbitrarily many states in between.

**Definition 4.** *The state $s_j \in \mathcal{S}$ is **reachable** from $s_i \in \mathcal{S}$ in MDP $\mathcal{M}$ if and only if there exists a path $s_i a_i \ldots s_j \ldots$ in $\mathcal{M}$. We denote this as $s_i \xrightarrow{\mathcal{M}} s_j$.*

*Example 6.* The state *passed*$_1$ is reachable from $s_I$. It can be reached by any path that starts with $s_I$ *try*$_1$ *passed*$_1$. However, the converse does not hold. $s_I$ is not reachable from *passed*$_1$, as there is no path from *passed*$_1$ that leads to $s_I$.

### 2.4 Strategy

In order to execute the MDP, we need some way of selecting the action to take in each state. The way in which we decide which action to take is called a strategy. There are several different types of strategies. First, it is possible to always select the same action in the same state. This is called a memoryless strategy, which is formalised in Definition 5. It is called memoryless, since it does not use any information about the previously visited states.

**Definition 5.** *A **memoryless strategy** for MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ is a function $\sigma_m \colon \mathcal{S} \to \mathcal{A}$, such that:*

$$\forall s \in \mathcal{S} \colon \sigma_m(s) \in \alpha_{\mathcal{M}}(s).$$

*Example 7.* If we want a strategy for Figure 2 such that we always try to stop as soon as possible, we can use the memoryless strategy:

$$\sigma_m = \{ s_I \mapsto \mathit{stop}, \mathit{passed}_1 \mapsto \mathit{stop}, \mathit{finished} \mapsto \tau \}.$$

In scenarios where we are only interested in the result after a finite number of steps (actions taken in the current path), it might be beneficial to be able to select a different action based on the length of the current path. That is why we introduce a strategy that takes into account the current length of the path to determine the action to take. This kind of strategy is called a step-positional strategy, since it considers both the position (the current state) and the step (the current length of the path).

**Definition 6.** *A **step-positional strategy** for MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ is a function $\sigma_{\mathcal{S}} \colon \mathcal{S} \times \mathbb{N}_0 \to \mathcal{A}$, such that:*

$$\forall s \in \mathcal{S}, k \in \mathbb{N}_0 \colon \sigma_{\mathcal{S}}(s,k) \in \alpha_{\mathcal{M}}(s).$$

*Example 8.* If we need a strategy for Figure 2, such that we take the first exam, but do not want to take more than three exams in total, we can use the following step-positional strategy:

$$\begin{aligned}\sigma_{\mathcal{S}} = &\{\langle s_I, i\rangle \mapsto \mathit{try}_1 \mid i \in \mathbb{N}_0\} \cup \{\langle \mathit{finished}, i\rangle \mapsto \tau \mid i \in \mathbb{N}_0\} \cup \\ &\{\langle \mathit{passed}_1, i\rangle \mapsto \mathit{try}_2 \mid i \in \{0,1,2\}\} \cup \{\langle \mathit{passed}_1, i\rangle \mapsto \mathit{stop} \mid i \in \{3,4,\dots\}\}.\end{aligned}$$

Next, we list the probabilistic versions of the memoryless and step-positional strategies in Definitions 7 and 8, respectively. The probabilistic versions introduce a probabilistic choice to take an action instead of a deterministic choice like in the previous definitions. We minimise the use of the probabilistic strategies in algorithms, but maximise their use in definitions. This allows us to use the most basic implementation while keeping the definitions as general as possible.

**Definition 7.** *A **probabilistic memoryless strategy** for MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ is a function $\sigma_{Pm} \colon \mathcal{S} \to \mathit{Dist}(\mathcal{A})$, such that:*

$$\forall s \in \mathcal{S}, a \in \mathcal{A} \colon \sigma_{Pm}(s)(a) > 0 \implies a \in \alpha_{\mathcal{M}}(s).$$

*Example 9.* We can use a probabilistic memoryless strategy for Figure 2 to flip a coin on whether or not to take an exam.

$$\sigma_{Pm} = \begin{Bmatrix} s_I \mapsto \{\mathit{try}_1 \mapsto 0.5, \mathit{try}_2 \mapsto 0, \mathit{stop} \mapsto 0.5, \tau \mapsto 0\}, \\ \mathit{passed}_1 \mapsto \{\mathit{try}_1 \mapsto 0, \mathit{try}_2 \mapsto 0.5, \mathit{stop} \mapsto 0.5, \tau \mapsto 0\}, \\ \mathit{finished} \mapsto \{\mathit{try}_1 \mapsto 0, \mathit{try}_2 \mapsto 0, \mathit{stop} \mapsto 0, \tau \mapsto 1\} \end{Bmatrix}.$$

**Definition 8.** *A **probabilistic step-positional strategy** for MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ is a function $\sigma_{Ps} \colon \mathcal{S} \times \mathbb{N}_0 \to \mathit{Dist}(\mathcal{A})$, such that:*

$$\forall s \in \mathcal{S}, k \in \mathbb{N}_0, a \in \mathcal{A} \colon \sigma_{Ps}(s,k)(a) > 0 \implies a \in \alpha_{\mathcal{M}}(s).$$

*The **set of all probabilistic step-positional strategies** in MDP $\mathcal{M}$ is denoted by $\mathit{Strat}_{\mathcal{M}} \subseteq \mathcal{S} \times \mathbb{N}_0 \to \mathit{Dist}(\mathcal{A})$.*

*Example 10.* We can construct a probabilistic step-positional strategy to Figure 2 to model the case where we flip a coin to determine whether to take an exam three times and stop if we still have not passed both exams after that.

$$\begin{aligned}\sigma_{Ps} = &\{\langle s_I, i\rangle \mapsto \{\mathit{try}_1 \mapsto 0.5, \mathit{try}_2 \mapsto 0, \mathit{stop} \mapsto 0.5, \tau \mapsto 0\} \mid i \in \mathbb{N}_0\} \cup \\ &\{\langle \mathit{passed}_1, i\rangle \mapsto \{\mathit{try}_1 \mapsto 0, \mathit{try}_2 \mapsto 0.5, \mathit{stop} \mapsto 0.5, \tau \mapsto 0\} \mid i \in \{0,1,2\}\} \cup \\ &\{\langle \mathit{passed}_1, i\rangle \mapsto \{\mathit{try}_1 \mapsto 0, \mathit{try}_2 \mapsto 0, \mathit{stop} \mapsto 1, \tau \mapsto 0\} \mid i \in \{3,4,\dots\}\} \cup \\ &\{\langle \mathit{finished}, i\rangle \mapsto \{\mathit{try}_1 \mapsto 0, \mathit{try}_2 \mapsto 0, \mathit{stop} \mapsto 0, \tau \mapsto 1\} \mid i \in \mathbb{N}_0\}.\end{aligned}$$

Note that all types of defined strategies can be described as a probabilistic step-positional strategy. A (probabilistic) memoryless strategy can be expressed by a (probabilistic) step-positional strategy, with the same action for each path length. Moreover, a non-probabilistic strategy can be expressed as a probabilistic strategy with the probability distribution that has probability 1 for the selected action. For this reason, we will express definitions using probabilistic step-positional strategies if possible. The same definitions can be used for the other strategies by applying the transformations. We will write down the formal transformations shortly, but to do so, we need a few more concepts.

In later definitions, we will need to access all paths for a strategy. These are all paths in the MDP that only use actions allowed by the strategy.

**Definition 9.** *The set of* ***all paths for a strategy*** *for MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$, for a probabilistic step-positional strategy $\sigma_{Ps} \in Strat_{\mathcal{M}}$, is the set:*

$$\Pi_{\mathcal{M}}^{\sigma_{Ps}} \stackrel{def}{=} \{s_1 a_1 s_2 a_2 \cdots \mid \forall i \in \mathbb{N}_+ \colon \sigma_{Ps}(s_i, i-1)(a_i) > 0 \wedge \delta(s_i)(a_i)(s_{i+1}) > 0\}.$$

We also need to define the chance of obtaining a certain path when using a given strategy.

**Definition 10.** *The* ***path probability*** *for a probabilistic step-positional strategy $\sigma_{Ps} \in Strat_{\mathcal{M}}$ and MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ for the path $\pi = s_1 a_1 s_2 a_2 \ldots$ is $Pr_{\mathcal{M}}^{\sigma_{Ps}} \colon Dist(\Pi_{\mathcal{M}}^{\sigma_{Ps}})$ and is defined in [47]. The main idea is that we construct cylinder sets in the form $\zeta(s_1 a_1 s_2 a_2 \ldots s_n)$ which is the set of paths starting with the prefix $s_1 a_1 s_2 a_2 \ldots s_n$ of which we take the smallest $\sigma$-algebra. The probability of such a cylinder is:*

$$Pr_{\mathcal{M}}^{\sigma_{Ps}}(\zeta(s_1 a_1 s_2 a_2 \ldots s_n)) \stackrel{def}{=} \prod_{i=1}^{n} \sigma_{Ps}(s_i, i-1)(a_i) \cdot \delta(s_i)(a_i)(s_{i+1}).$$

*Example 11.* If we consider a probabilistic step-positional strategy $\sigma_{Ps}$ for Figure 2, such that we take the first exam but stop after that, we might get the path $s_I\ try_1\ passed_1\ stop\ finished\ \tau\ finished\ \tau \ldots$. We can calculate the probability of this path as follows:

$$\begin{aligned} &Pr_{\mathcal{M}}^{\sigma_{Ps}}(s_I\ try_1\ passed_1\ stop\ finished\ \tau\ finished\ \tau \ldots) \\ &\quad = \sigma_{Ps}(s_I, 0)(try_1) \cdot \delta(s_I)(try_1)(passed_1) \cdot \\ &\qquad \sigma_{Ps}(passed_1, 1)(stop) \cdot \delta(passed_1)(stop)(finished) \cdot \\ &\qquad \sigma_{Ps}(finished, 2)(\tau) \cdot \delta(finished)(\tau)(finished) \ldots \\ &\quad = 1 \cdot 0.85 \cdot 1 \cdot 1 \cdot 1 \cdot 1 \cdots = 0.85. \end{aligned}$$

We can now define the transformations from memoryless, step-positional and probablistic memoryless strategies to probabilistic step-positional strategies. We describe these transformations in Propositions 2 to 4 respectively.

**Proposition 2.** *Let $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ be an MDP and $\sigma_m$ be a memoryless strategy. Let $\sigma_{Ps}$ be the probabilistic step-positional strategy such that:*

$$\forall s \in \mathcal{S}, k \in \mathbb{N}_0, a \in \mathcal{A} \colon \sigma_{Ps}(s, k)(a) = [\sigma_m(s) = a].$$

*Then $\Pi_{\mathcal{M}}^{\sigma_m} = \Pi_{\mathcal{M}}^{\sigma_{Ps}}$ and $Pr_{\mathcal{M}}^{\sigma_m} = Pr_{\mathcal{M}}^{\sigma_{Ps}}$.*

**Proposition 3.** *Let $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ be an MDP and $\sigma_{\mathcal{S}}$ be a step-positional strategy. Let $\sigma_{Ps}$ be the probabilistic step-positional strategy such that:*

$$\forall s \in \mathcal{S}, k \in \mathbb{N}_0, a \in \mathcal{A} \colon \sigma_{Ps}(s,k)(a) = [\sigma_{\mathcal{S}}(s,k) = a].$$

*Then $\Pi_{\mathcal{M}}^{\sigma_{\mathcal{S}}} = \Pi_{\mathcal{M}}^{\sigma_{Ps}}$ and $Pr_{\mathcal{M}}^{\sigma_{\mathcal{S}}} = Pr_{\mathcal{M}}^{\sigma_{Ps}}$.*

**Proposition 4.** *Let $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ be an MDP and $\sigma_{Pm}$ be a probabilistic memoryless strategy. Let $\sigma_{Ps}$ be the probabilistic step-positional strategy such that:*

$$\forall s \in \mathcal{S}, k \in \mathbb{N}_0 \colon \sigma_{Ps}(s,k) = \sigma_{Pm}(s).$$

*Then $\Pi_{\mathcal{M}}^{\sigma_{Pm}} = \Pi_{\mathcal{M}}^{\sigma_{Ps}}$ and $Pr_{\mathcal{M}}^{\sigma_{Pm}} = Pr_{\mathcal{M}}^{\sigma_{Ps}}$.*

### 2.5 Rewards

Now that we have defined strategies, we can execute an MDP. We are typically not interested in any strategy, but in the best strategy for a given MDP. Right now, we do not have a way to express what better means in an MDP. That is why we introduce the notion of rewards. A reward in an MDP should be given when either a positive or a negative event occurs in the modelled process. When we do this, we can calculate the sum of the rewards for a path. By calculating the reward for all possible paths that can be generated by executing a strategy, we can then calculate the expected reward for a strategy. We can then compare the strategies and determine the optimal expected reward.

**2.5.1 Reward-structures** There are several ways to give rewards. The first type of reward we will consider is the reward for taking a branch. Branches are the probabilistic choices in an MDP, so they describe which state we can end up in after taking an action. For example, the action *try₁* in Figure 2, has two branches: one that goes to *passed₁* and one that goes to *finished*. We define the rewards for a path by defining a reward-structure, which assigns a reward to each branch in the MDP.

**Definition 11.** *A **branch reward-structure** for MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ is a function $\rho \colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. The **set of all branch reward-structures** is $Struct_{\mathcal{M}} \stackrel{def}{=} \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$.*

*Example 12.* Passing the first exam in Figure 3 gives one hiring point, therefore we add a reward of +1 to the branch from the *try₁* action going to *passed₁*. In addition, passing the second exam gives three hiring points. Hence, we add an additional +3 reward on the branch going from the *try₂* action to the *finished* state. We then obtain the reward structure $\rho$, which is visualised in Figure 3 and is defined as:

$$\rho(s,a,s') = \begin{cases} +1, & \text{if } s = s_I \wedge a = try_1 \wedge s' = passed_1 \\ +3, & \text{if } s = passed_1 \wedge a = try_2 \wedge s' = finished \\ 0, & \text{otherwise} \end{cases}$$

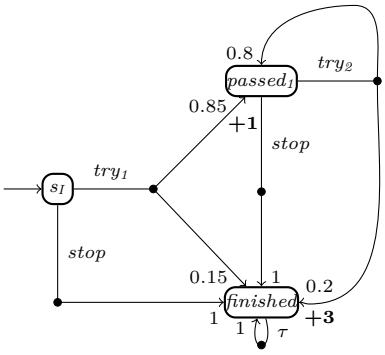



**Fig. 3.** Certification process MDP with branch rewards

Another common type of reward-structure specifies a reward for selecting an action from a state. We also call selecting an action from a state taking a transition. Therefore, we refer to this type of reward-structure as a transition reward-structure. We do not call this an action reward-structure, since we can assign a different reward for the same action if it was taken from a different state.

**Definition 12.** *A **transition reward-structure** is a function* $\rho_{\mathcal{A}} \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

Fortunately, we can easily convert a transition reward-structure to a branch reward-structure as shown in Proposition 5. This allows us to use transition reward-structures in definitions while the implementation uses branch reward-structures.

**Proposition 5.** *Let* $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ *be an MDP and* $\rho_{\mathcal{A}}$ *be an transition reward-structure. Let* $\rho$ *be a branch reward-structure, such that:*

$$\forall s \in \mathcal{S}, a \in \alpha_{\mathcal{M}}(s), s' \in \mathcal{S} \colon \rho(s, a, s') = \rho_{\mathcal{A}}(s, a).$$

*Then for each path in* $\mathcal{M}$ *the sum of transitions reward for* $\rho_{\mathcal{A}}$ *after taking* $k \in \mathbb{N}_{0,\infty}$ *transitions will be the same as the sum of branch rewards for* $\rho$ *after* $k$ *steps.*

The last type of reward-structure we consider, gives rewards for entering a state, except for the initial state. We do not give a reward for the initial state, since this would complicate the conversion to a branch reward-structure. This exclusion does not matter since each path includes the initial state, and thus all paths would just have a reward offset by this value. Therefore, it does not influence which strategy is optimal.

**Definition 13.** *A **state reward-structure** is a function* $\rho_{\mathcal{S}} \colon \mathcal{S} \to \mathbb{R}$.

Fortunately, we can also convert a state reward-structure into a branch reward-structure as shown in Proposition 6, which also allows us to express reward-structures for states while still using branch reward-structures in an implementation.

**Proposition 6.** *Let $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ be an MDP and $\rho_{\mathcal{S}}$ be a state-reward structure. Let $\rho$ be the branch-reward structure, such that:*

$$\forall s \in \mathcal{S}, a \in \alpha_{\mathcal{M}}(s), s' \in \mathcal{S} \colon \rho(s, a, s') = \rho_{\mathcal{S}}(s').$$

*Then for each path in $\mathcal{M}$ the sum of state rewards for $\rho_{\mathcal{S}}$ after taking $k \in \mathbb{N}_{0,\infty}$ transitions will be the same as the sum of branch rewards for $\rho$ after $k$ steps.*

**2.5.2 Properties** Now that we have defined reward-structures, we can compare different strategies. For this, we use expected rewards, of which we introduce a few variations. Each of these variations has their own unique use cases.

The first is the cumulative reward. This type of reward allows for a bound on the length of the paths (possibly infinity) and calculates the expected reward.

**Definition 14.** *The **cumulative reward** for MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ is the function $CumRew_{\mathcal{M}} \colon Struct_{\mathcal{M}} \to \mathbb{N}_{+,\infty} \times Strat_{\mathcal{M}} \to \mathbb{R}$, such that:*

$$CumRew_{\mathcal{M}}(\rho)(k, \sigma_{Ps}) \stackrel{def}{=} \int_{\pi = s_1 a_1 s_2 a_2 \cdots \in \Pi_{\mathcal{M}}^{\sigma_{Ps}}} \sum_{i=1}^{k} \rho(s_i, a_i, s_{i+1}) dPr_{\mathcal{M}}^{\sigma_{Ps}}.$$

*Example 13.* For the reward-structure in Figure 3, we can find the maximum cumulative reward. Obviously, this is the strategy that takes the exams when possible. Although it is impossible to list each path individually, we know that each path has a 0.85 chance of passing the first exam. If we try the second exam infinitely often, the probability of eventually passing it is 1. This means that there is a 0.85 chance of passing both exams and thus a 0.85 chance of getting 4 hiring points:

$$\begin{aligned} \max_{\sigma_{Ps} \in Strat_{\mathcal{M}}} & CumRew_{\mathcal{M}}(\rho)(\infty, \sigma_{Ps}) \\ &= (1+3) \cdot (0.85 \cdot (0.2 + 0.8 \cdot 0.2 + 0.8 \cdot 0.8 \cdot 0.2 + \dots)) \\ &= 4 \cdot 0.85 = 3.4. \end{aligned}$$

The cumulative reward however, can also become infinite quite easily. Consider the case where we would add a reward of $\rho(\mathit{finished}, \tau, \mathit{finished}) = +1$ to Figure 3. The cumulative reward for taking the $\tau$ loop infinitely often is infinite. Therefore, we also introduce the long-run average reward in Definition 15. This reward calculates the average reward we can get per transition taken. This means that the long-run average can be finite, even though the cumulative reward for the same strategy might be infinite. We also introduce a maximum number of steps for the long-run average, even though this value is typically going to be infinite. Introducing it anyways allows us for easier definitions later on.

**Definition 15.** *The **long-run average reward** for MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ is the function $LraRew_{\mathcal{M}} \colon Struct_{\mathcal{M}} \to \mathbb{N}_{+,\infty} \times Strat_{\mathcal{M}} \to \mathbb{R}$, such that:*

$$LraRew_{\mathcal{M}}(\rho)(k, \sigma_{Ps}) \stackrel{def}{=} \int_{\pi = s_1 a_1 s_2 a_2 \cdots \in \Pi_{\mathcal{M}}^{\sigma_{Ps}}} \frac{1}{k} \sum_{i=1}^{k} \rho(s_i, a_i, s_{i+1}) dPr_{\mathcal{M}}^{\sigma_{Ps}}.$$

*Example 14.* When we add the reward $\rho(\mathit{finished}, \tau, \mathit{finished}) = +1$ to the MDP in Figure 3, the long-run average reward for infinitely many steps will be $+1$.

The last type of reward-structure we consider, is a reachability reward-structure, as formalised in Definition 16. For reachability reward-structures, we calculate the reward until we reach a goal state. This can, for example, be the reward until we pass the first exam. Any rewards obtained after reaching a goal state are ignored. If we never reach a goal state, we define the reachability reward to be $\infty$. This choice is arbitrary, but we need a value to ensure that the function is well defined, which simplifies other definitions. We choose $\infty$ because we the other case in which $\infty$ can be obtained is by visiting a reward infinitely often. This is commonly considered a modelling error, as we will discuss in Section 6. Hence, when we obtain an $\infty$, we know that the model can be considered erroneous.

**Definition 16.** *The **reachability reward** for MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ is the function $RRew_{\mathcal{M}} \colon Struct_{\mathcal{M}} \times \mathcal{P}(\mathcal{S}) \to \mathbb{N}_{+,\infty} \times Strat_{\mathcal{M}} \to \mathbb{R}$, such that if we have $Pr_{\mathcal{M}}^{\sigma_{Ps}}(\{\pi = s_1 a_1 s_2 a_2 \cdots \in \Pi_{\mathcal{M}}^{\sigma_{Ps}} \mid \exists i \in \mathbb{N}_+ s_i \in G\}) = 1$, then:*

$$RRew_{\mathcal{M}}(\rho, G)(k, \sigma_{Ps}) \stackrel{def}{=} \int_{\pi = s_1 a_1 s_2 a_2 \cdots \in \Pi_{\mathcal{M}}^{\sigma_{Ps}}} \sum_{i=1}^{k} \rho(s_i, a_i, s_{i+1}) \cdot [\forall i \in \{1, \ldots, i-1\} : s_i \notin G] dPr_{\mathcal{M}}^{\sigma_{Ps}}.$$

*Otherwise:*

$$RRew_{\mathcal{M}}(\rho, G)(k, \sigma_{Ps}) \stackrel{def}{=} \infty.$$

*Example 15.* If we add the reward $\rho(\mathit{finished}, \tau, \mathit{finished}) = +1$ to Figure 3 and then add *finished* to the goal set, the reachability reward is significantly different from the cumulative reward, which would be infinite, since the loop is now only achieved after reaching a goal state. This means that we obtain the reward:

$$\begin{aligned} \sup_{\sigma_{Ps} \in Strat_{\mathcal{M}}} & RRew_{\mathcal{M}}(\{\mathit{finished}\}, \rho)(\infty, \sigma_{Ps}) \\ &= (1+3) \cdot (0.85 \cdot (0.2 + 0.8 \cdot 0.2 + 0.8 \cdot 0.8 \cdot 0.2 + \ldots)) \\ &= 4 \cdot 0.85 = 3.4 \end{aligned}$$

There are two types of properties which we can express using reward-structures. The most prevalent are properties where we search for an optimal value. In this case, we try to either minimise or maximise the expected reward for a reward-structure. Since the expected reward is a number, we call this type of property a "numerical property".

**Definition 17.** *A **numerical property** $\hat{\phi}$ for MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ is $\hat{\phi} = [Rew_{\mathcal{M}}]_{\widehat{\triangle}}^{\leq k}$ where:*

- *$Rew_{\mathcal{M}} \colon \mathbb{N}_{+,\infty} \times Strat_{\mathcal{M}} \to \mathbb{R}$, the type of reward to use,*

- $k \in \mathbb{N}_{+,\infty}$, *the step-bound: the maximum amount of transitions to take, and*
- $\triangle \in \{max, min\}$, *whether to maximise or minimise the reward.*

The other type of property aims to verify whether it is possible to find a strategy which satisfies a constraint. In particular, we want to verify whether there is a strategy that has an expected reward larger than or equal to $(\geq, >)$, smaller than or equal to $(\leq, <)$ or equal to $(=)$ a constant. Since we do not have any unknowns and only want to verify whether such a strategy exists, we call these types of properties "achievability properties".

**Definition 18.** *An* ***achievability property*** $\bar{\phi}$ *for MDP* $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ *is* $\bar{\phi} = [Rew_{\mathcal{M}}]^{\leq k}_{\square c}$ *where:*

- $Rew_{\mathcal{M}} \colon \mathbb{N}_{+,\infty} \times Strat_{\mathcal{M}} \to \mathbb{R}$, *the type of reward to use,*
- $k \in \mathbb{N}_{+,\infty}$, *the step-bound: the maximum amount of transitions to take,*
- $\square \in \{\leq, <, \geq, >, =\}$, *the type of comparison with the constant, and*
- $c \in \mathbb{R}$, *the constant to which the reward should be compared.*

*We say that* $\bar{\phi}$ *is satisfied in* $\mathcal{M}$ *under a probabilistic step-positional strategy* $\sigma_{Ps} \in Strat_{\mathcal{M}}$ *if:*

$$\mathcal{M}, \sigma_{Ps} \models [Rew_{\mathcal{M}}]^{\leq k}_{\square c} \overset{def}{\iff} Rew_{\mathcal{M}}(k, \sigma_{Ps}) \square c.$$

*Example 16.* If we want to express that the cumulative reward in MDP $\mathcal{M}$ over the branch reward-structure $\rho$ within 10 transitions is at least 5, we can denote this is as the achievability property $[CumRew_{\mathcal{M}}(\rho)]^{\leq 10}_{\geq 5}$.

We also need a way to evaluate properties. For this, we will use the *query* function as formalised in Definition 19. For an achievability property, this returns whether there exists a strategy such that the property is satisfied. For a numerical property, this returns the optimal value (maximum for *max* and minimum for *min*) that can be achieved by a strategy.

**Definition 19.** *A* ***query*** *for MDP* $\mathcal{M}$ *is a function* $query_{\mathcal{M}}$ *that evaluates a property* $\boldsymbol{\phi}$ *and is defined as follows:*

- *For an achievability property* $\phi$:

$$query_{\mathcal{M}}(\phi) \overset{def}{\iff} \exists \sigma_{Ps} \in Strat_{\mathcal{M}} \colon \mathcal{M}, \sigma_{Ps} \models \phi,$$

- *For a numerical property* $\phi = [Rew_{\mathcal{M}}]^{\leq k}_{max}$:

$$query_{\mathcal{M}}(\phi) \overset{def}{=} \sup_{\sigma_{Ps} \in Strat_{\mathcal{M}}} Rew_{\mathcal{M}}(k, \sigma_{Ps}), \text{ and}$$

- *For a numerical property* $\phi = [Rew_{\mathcal{M}}]^{\leq k}_{min}$:

$$query_{\mathcal{M}}(\phi) \overset{def}{=} \inf_{\sigma_{Ps} \in Strat_{\mathcal{M}}} Rew_{\mathcal{M}}(k, \sigma_{Ps}).$$

### 2.6 Probabilistic Reachability

Another way to analyse MDPs is by calculating the probability of reaching a given state of interest. We might want to ensure success or compute the probability of failure. For example, in the example MDP in Figure 1, we might want to compute the chance of achieving the first certificate. To do this, we can calculate the probability that a strategy reaches $passed_1$.

We can express probabilistic reachability properties using a state-reward structure, by transforming the MDP and giving a reward of 1 the first time we enter a goal state, and 0 in other states. To ensure that we only give a reward when entering the first goal state, we could add one "handed_out" state and force the transition function for the goal states to only transition to that state. This approach works for MDPs on which only one property is being evaluated, but it does not work for multi-objective properties, which we will consider later. That is why we take a slightly different approach in Proposition 7, which is an adapted version of [30, Proposition 2].

For each state, there are two new states, one in which the reward has been given and one in which it has not. This approach also works for the multi-objective case. We stress that this approach means that the size of the state space doubles for each probability property that needs to be evaluated on the MDP.

**Proposition 7.** *Let $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ be an MDP, and $G \subseteq \mathcal{S}$ be a set of goal states. Let $\mathcal{M}' = \langle \mathcal{S} \times \{false, true\}, \langle s_I, false \rangle, \mathcal{A}, \delta' \rangle$ be the MDP with with $\delta' \colon \mathcal{S} \times \{false, true\} \to \mathcal{A} \rightharpoonup Dist(\mathcal{S} \times \{false, true\})$ the function with each $\delta'(\langle s, b \rangle)$ being the smallest partial function, such that $\forall s, s' \in \mathcal{S} \colon \forall a \in \alpha_{\mathcal{M}}(s) \colon$*

- *$\delta'(\langle s, false \rangle)(a)(\langle s', true \rangle) = \delta(s)(a)(s')$ if $s \in G$,*
- *$\delta'(\langle s, false \rangle)(a)(\langle s', false \rangle) = \delta(s)(a)(s')$ if $s \notin G$, and*
- *$\delta'(\langle s, true \rangle)(a)(\langle s', true \rangle) = \delta(s)(a)(s')$.*

*Let $\rho_{\mathcal{S}} \colon \mathcal{S} \times \{false, true\} \to \mathbb{R}$ be the state reward-structure such that:*

$$\rho_{\mathcal{S}}(\langle s, g \rangle)) = [\neg g \wedge s \in G].$$

*Then for all strategies in MDP $\mathcal{M}$, the probability of reaching a state in $G$ is the same as the cumulative state reward of $\rho_{\mathcal{S}}$ in $\mathcal{M}'$ [30].*

*Example 17.* If we transform Figure 2 using Proposition 7, with the goal set $\{finished\}$, we get the MDP shown in Figure 4. Notice that we could ignore the states $\langle passed_1, true \rangle$ and $\langle s_I, true \rangle$, since they are unreachable from $\langle s_I, false \rangle$, but we add them for completeness.

### 2.7 Value Iteration

We formalised queries in Definition 19. However, this definition still contains integration and summations over infinite paths, making it hard to implement

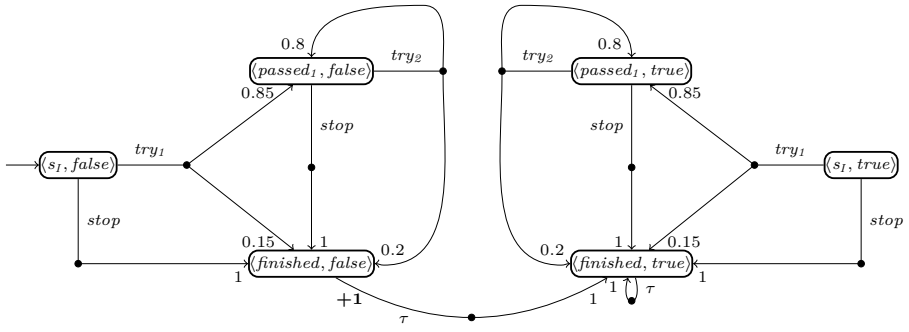



**Fig. 4.** Proposition 7 applied to Figure 2 with goal *finished*

this definition directly. Fortunately, in practice, we do not need to go over all possible paths. There are several ways to compute or approximate the result of a query. The first algorithm we consider is value iteration.

The principle of value iteration is relatively straightforward. We find the best value for all paths after 0 steps, then based on that we find the best actions to take for each state, meaning that we found the best value for all paths after 1 step, and so on, until we decide to stop. We typically stop when the changes in values from one iteration to the next are smaller than some convergence threshold, or when we reach the step-bound for the property. The early stopping criterion based on a convergence threshold makes value iteration relatively fast; however, it also makes it inaccurate. In some cases, the computed value is quite close to the actual value, but in others, the computed value can be quite far off [34].

We show the value iteration algorithm for a maximum cumulative reward in Algorithm 1. For each step, we compute the maximum reward expected reward.

*Example 18.* We take Figure 2 and give a transition reward of +1 for taking the $try_2$ action from $passed_1$. All other transitions get a reward of 0. We can then use value iteration to compute the cumulative reward of this transition reward-structure. The first few iterations are shown in Table 1.

As shown in Algorithm 1, we consider two types of convergence thresholds. The first type of convergence threshold is absolute. With an absolute convergence threshold, we consider the largest difference between the values in two consecutive iterations. A relative difference is the percentage of change between two consecutive iterations. Note that if we had an infinite reward, the algorithm might never terminate with an absolute convergence threshold. However, with a relative convergence threshold, the algorithm will terminate if it is non-zero.

*Example 19.* We will evaluate both types of convergence thresholds after iteration 5 in Table 1. Using an absolute convergence threshold requires comparing the largest difference. In this case, that is $2.5092 - 2.074 = 0.4352$. Thus, we would stop if $0.4352 < \epsilon$. If we were using a relative convergence threshold instead, we

**Alg. 1.** Maximum Cumulative Reward Value iteration

**Input:** MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$, numerical property $\phi = [CumRew_{\mathcal{M}}(\rho)]^{\leq k}_{max}$, convergence threshold $\epsilon \in [0, 1]$, convergence threshold type $t \in \{absolute, relative\}$
**Result:** $query_{\mathcal{M}}(\phi)$

$\boldsymbol{x} \leftarrow \boldsymbol{y} \leftarrow \mathbf{0}_{\|\mathcal{M}\|}$
$i \leftarrow 1$
**do**
  **foreach** $s \in \mathcal{S}$ **do**
    $y_s \leftarrow \max_{a \in \alpha_{\mathcal{M}}(s)} \sum_{s' \in \mathcal{S}} \delta(s)(a)(s') \cdot (\rho(s, a, s') + x_{s'})$
  **if** $t = absolute$ **then** $\Delta \leftarrow \max_{s \in \mathcal{S}} |y_s - x_s|$
  **else** $\Delta \leftarrow \max_{s \in \mathcal{S}} |\frac{y_s - x_s}{x_s}|$
  $\boldsymbol{x} \leftarrow \boldsymbol{y}$
  $i \leftarrow i + 1$
**while** $i \leq k \wedge \Delta > \epsilon$
**return** $x_{s_I}$

**Table 1.** Example value iteration for Figure 2

| i | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $x_{s_I}$ | 0.0000 | 0.0000 | 0.8500 | 1.5300 | 2.0740 | 2.5092 |
| $x_{passed_I}$ | 0.0000 | 1.0000 | 1.8000 | 2.4400 | 2.9520 | 3.3616 |
| $x_{finished}$ | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |

would need to calculate the largest relative change, which is $\frac{2.5092 - 2.074}{2.074} = 0.210$. Thus, we would stop if $0.210 < \epsilon$.

Since we can only increase our values in a future iteration or keep them the same if we are only using positive rewards and only decrease or keep the same for negative rewards, value iteration is monotonic if all rewards have the same sign. This means that the result from value iteration can be used as an upper or lower bound on the actual value if proper rounding is used. Since a smaller convergence threshold can only increase the iterations used or keep them the same, the value iteration results should be monotonic in terms of the convergence threshold.

### 2.8 Linear Programming

A more exact, but typically slower way of evaluating queries is by using linear programming. Linear programming is a technique that optimises a linear function, based on a set of linear functions, the requirements of which are also given as linear functions.

Several algorithms exist to solve linear programs. Solvers that implement such algorithms are called linear optimisation solvers. We will not cover these

$$
\begin{aligned}
\text{minimise } & x(s_I) + x(\mathit{passed}_1) + x(\mathit{finished}) \\
\text{subject to } & x(\mathit{finished}) = 0 \\
& x(s_I) \geq 1 \cdot (x(\mathit{finished}) + \rho(s_I)(\mathit{stop})(\mathit{finished})) \\
& x(s_I) \geq 0.15 \cdot (x(\mathit{finished}) + \rho(s_I)(\mathit{try}_1)(\mathit{finished})) \\
& \qquad + 0.85 \cdot (x(\mathit{passed}_1) + \rho(s_I)(\mathit{try}_1)(\mathit{passed}_1)) \\
& x(\mathit{passed}_1) \geq 1 \cdot (x(\mathit{finished}) + \rho(\mathit{passed}_1)(\mathit{stop})(\mathit{finished})) \\
& x(\mathit{passed}_1) \geq 0.8 \cdot (x(\mathit{passed}_1) + \rho(\mathit{passed}_1)(\mathit{try}_2)(\mathit{passed}_1)) \\
& \qquad + 0.2 \cdot (x(\mathit{finished}) + \rho(\mathit{passed}_1)(\mathit{try}_2)(\mathit{finished}))
\end{aligned}
$$

**Fig. 5.** Linear program for maximising a reachability reward for Figure 2

algorithms since we deem them out-of-scope for this research. The interested reader can look at [14]. We assume that an existing linear optimisation solver is used.

This means that we only need to express our property and MDP as a linear program, and then we hand it to a linear optimisation solver to get the result of the query. The primary downsides of this approach over value iteration are that linear programming is typically bit slower, and in most cases it requires an additional dependency to solve linear programs in implementations. However, linear programming is still polynomial in the size of the MDP [48].

It is important to mention that not all linear optimisation solver implementations are exact. Some implementations approximate the result of the query instead of calculating it exactly. Therefore their results might be incorrect [37]. Thus, if one wants to compute exact results, one should choose an exact linear optimisation solver.

*Example 20.* For the MDP shown in Figure 2, we can evaluate a maximum reachability reward property by creating a linear program. If we take the goal state to be *finished*, with a branch reward-structure $\rho$, we get the linear program shown in Figure 5.

### 2.9 End Component

When we traverse an MDP, we might at some point have the opportunity to cycle infinitely between a subset of the states using a subset of the transitions. In other words, there exist sub-MDPs for any MDP, which a strategy does not need to leave [1]. We refer to such a sub-MDP as an end component. If there are no more states or transitions we can add without the end component no longer being an end component, the end component is called a maximal end component. End components and in particular maximal end components can be useful when evaluating certain properties.

**Definition 20.** *An **end component** of MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ is any MDP $\mathcal{M}_e = \langle \mathcal{S}_e, s_{eI}, \mathcal{A}_e, \delta_e \rangle$ such that:*

- $\emptyset \neq \mathcal{S}_e \subseteq \mathcal{S}$,
- $\forall s \in \mathcal{S}_e \colon \emptyset \neq \delta_e(s) \subseteq \delta(s)$,
- $S_e = \{s' \in \mathcal{S} \mid \exists s \in \mathcal{S}_e, a \in \alpha_{\mathcal{M}_e}(s) \colon \delta_e(s)(a)(s') > 0\}$, *and*
- $\forall s, s' \in S_e \colon s \xrightarrow{\mathcal{M}} s'$.

*The **set of all end components** of MDP $\mathcal{M}$ is represented by $EC(\mathcal{M})$. An end component of MDP $\mathcal{M}$ is **maximal** if and only if there does not exist an end component $\mathcal{M}'_e = \langle \mathcal{S}'_e, s_e{}'_I, \mathcal{A}'_e, \delta'_e \rangle$ such that $\mathcal{S}_e \subseteq \mathcal{S}'_e \wedge \exists s \in \mathcal{S}_e \colon \delta_e(s) \subset \delta'_e(s)$.*

Algorithms to compute the maximum end components can be found in [1,18].

### 2.10 Linear Temporal Logic

So far, we have only used a set of goal states for probabilistic reachability and reachability rewards. However, in practice we also want to set constraints on the paths which lead to these states. For example, we might only be interested in the rate of success of a program, given that the computer does not overheat. For this, we can use linear temporal logic (LTL) [63]. LTL allows us to express constraints on paths by using operators that allow us to say that something must be true at some point in the future, must always hold, must hold in the next step or something must be true until some other condition holds.

LTL properties are part of the $\omega$-languages [22]. These $\omega$-languages can be described by several types of automata [26]. These automata are quite similar to each other, but they have different acceptance criteria. The type of automata we consider are deterministic Rabin automata [65]. These automata are deterministic automata, but instead of a set of goal states as for finite words, they have a set of pairs of sets of states which describe which infinite words, or paths as defined earlier, should be accepted.

**Definition 21.** *A **deterministic Rabin automaton** (DRA) $\mathcal{R}$ is a tuple $\mathcal{R} = \langle \mathcal{Q}, q_I, \mathcal{A}, \lambda, \mathcal{C} \rangle$, where:*

- *$\mathcal{Q}$ is the state space: a finite set of states,*
- *$q_I \in \mathcal{Q}$ is the initial state,*
- *$\mathcal{A}$ are all supported actions: a finite set of actions,*
- *$\lambda \colon \mathcal{Q} \times \mathcal{A} \to \mathcal{Q}$ is the transition function, and*
- *$\mathcal{C} \subseteq \mathcal{P}(Q) \times \mathcal{P}(Q)$ is the acceptance condition.*

*The DRA $\mathcal{R}$ **accepts** the path $\pi$ iff there exists a pair $\langle F, I \rangle \in \mathcal{C}$ such that $\pi$ visits all states in $F$ finitely often but visits at least one state in $I$ infinitely often.*

*Example 21.* If we want to check whether it is possible to stay in the $passed_1$ state for ever, so that we can evaluate Figure 2, we can construct a DRA. This DRA is visualised in Figure 6. It is the DRA $\mathcal{R} = \langle \mathcal{Q}, q_I, \mathcal{A}, \lambda, \mathcal{C} \rangle$ such that:

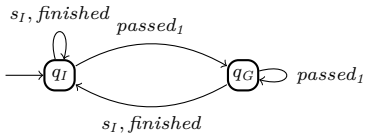



**Fig. 6.** Deterministic Rabin automaton for the certification process

- $\mathcal{Q} = \{q_I, q_G\}$,
- $\mathcal{A} = \{s_I, passed_1, finished\}$,
- $\lambda = \left\{ \begin{array}{l} \langle q_I, s_I \rangle \mapsto q_I, \langle q_I, passed_1 \rangle \mapsto q_G, \langle q_I, finished \rangle \mapsto q_I, \\ \langle q_G, s_I \rangle \mapsto q_I, \langle q_G, passed_1 \rangle \mapsto q_I, \langle q_G, finished \rangle \mapsto q_I \end{array} \right\}$, and
- $\mathcal{C} = \{\langle \{q_I\}, \{q_G\} \rangle\}$.

In order to use a DRA in combination with an MDP, we need a way to compose the two. We can compose an MDP and a DRA if the DRA has the states of the MDP as actions. The result of a composition is an MDP and a DRA is an MDP. Because the result of this composition is an MDP, we can compose an MDP with several DRAs that describe multiple properties. This is important for multi-objective properties.

**Definition 22.** *The composition of the MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ and DRA $\mathcal{R} = \langle \mathcal{Q}, q_I, \mathcal{S}, \lambda, \mathcal{C} \rangle$ is the MDP $\mathcal{M} \otimes \mathcal{R} = \langle \mathcal{S} \times \mathcal{Q}, \langle s_I, q_I \rangle, \mathcal{A}, \delta' \rangle$ with $\delta' \colon \mathcal{S} \times \mathcal{Q} \to \mathcal{A} \rightharpoonup Dist(\mathcal{S} \times \mathcal{Q})$ the function with each $\delta'(\langle s, q \rangle)$ being the smallest partial function such that:*

$$\forall s, s' \in \mathcal{S}, a \in \alpha_{\mathcal{M}}(s), q \in \mathcal{Q} \colon \delta'(\langle s, q \rangle)(a)(\langle s', \lambda(q, s') \rangle) = \delta(s)(a)(s').$$

To determine when the DRA is accepted, we need to determine the accepting end components. An end component is accepting if it contains no states that can only be visited a finite number of times and at least one state that needs to be visited an infinite number of times. We consider such an end component accepting, since we can modify any strategy to visit the state that needs to be visited infinitely often, an infinite number of times. Notice that since we need to visit the state infinitely often, this means that the modified strategy will never leave the end component. The algorithm to compute the accepting end components can be found in the appendix of the extended version of [28].

**Definition 23.** *An end component $\mathcal{M}_e = \langle \mathcal{S}_e, sq_I, \mathcal{A}, \delta' \rangle \in EC(\mathcal{M} \otimes \mathcal{R})$ of the composition $\mathcal{M} \otimes \mathcal{R}$ of the MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ and DRA $\mathcal{R} = \langle \mathcal{Q}, q_I, \mathcal{A}, \lambda, \mathcal{C} \rangle$ is **accepting** when:*

$$\mathcal{M}_e \models \mathcal{R} \overset{def}{\iff} \exists \langle F, I \rangle \in \mathcal{C} \colon (\mathcal{S} \times F) \cap \mathcal{S}_e = \emptyset \wedge (\mathcal{S} \times I) \cap \mathcal{S}_e \neq \emptyset.$$

*Example 22.* If we compose the DRA from Example 21 with the MDP for the certification process from Figure 2, we construct the MDP on which we can evaluate whether it is possible to stay in the $passed_1$ state for ever. This composition is shown in Figure 7.

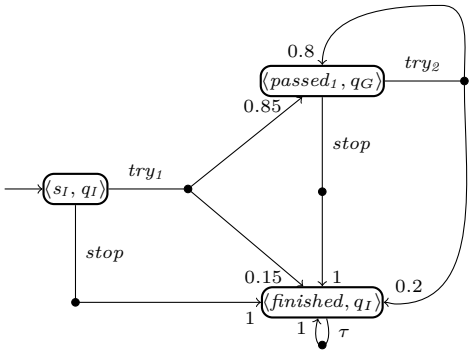



**Fig. 7.** Certification process composition with the DRA in Figure 6

We can now also formalise LTL properties as we have done for rewards in Definitions 17 and 31 and extend the definition of a query to also accept these properties.

**Definition 24.** *A **numerical LTL property** is* $\hat{\Gamma} = [\mathcal{R}]^{\leq k}_{\triangle}$ *where:*

- $\mathcal{R}$ *is a DRA,*
- $k \in \mathbb{N}_{+,\infty}$*, the maximum amount of steps, and*
- $\triangle \in \{max, min\}$*, whether to maximise or minimise the probability of accepting* $\mathcal{R}$*.*

**Definition 25.** *An **achievability LTL property** is* $\bar{\Gamma} = [\mathcal{R}]^{\leq k}_{\square c}$ *where:*

- $\mathcal{R}$ *is a DRA,*
- $k \in \mathbb{N}_{+,\infty}$*, the maximum amount of steps,*
- $\square \in \{\leq, <, \geq, >, =\}$*, the type of comparison with the constant, and*
- $c \in \mathbb{R}$*, the constant to which the reward should be compared.*

**Definition 26.** *A **query** for MDP* $\mathcal{M}$ *is a function* $query_{\mathcal{M}}$ *that evaluates an LTL property* $\Gamma$ *and is defined as follows:*

- *For a numerical LTL property* $\Gamma \stackrel{def}{=} [\mathcal{R}]^{\leq k}_{max}$*:*

$$query_{\mathcal{M}}(\Gamma) = \sup_{\sigma_{Ps} \in Strat_{\mathcal{M}}} \sum_{\pi = s_1 a_1 s_2 a_2 \cdots \in \Pi^{\sigma_{Ps}}_{\mathcal{M}}} Pr^{\sigma_{Ps}}_{\mathcal{M}}(\pi) \cdot P,$$

- *For a numerical LTL property* $\Gamma \stackrel{def}{=} [\mathcal{R}]^{\leq k}_{min}$*:*

$$query_{\mathcal{M}}(\Gamma) = \inf_{\sigma_{Ps} \in Strat_{\mathcal{M}}} \sum_{\pi = s_1 a_1 s_2 a_2 \cdots \in \Pi^{\sigma_{Ps}}_{\mathcal{M}}} Pr^{\sigma_{Ps}}_{\mathcal{M}}(\pi) \cdot P,$$

- *For an achievability LTL property* $\Gamma \stackrel{def}{=} [\mathcal{R}]^{\leq k}_{\square c}$ *with* $\square \in \{\geq, >\}$*:*

$$query_{\mathcal{M}}(\Gamma) = query_{\mathcal{M}}([\mathcal{R}]^{\leq k}_{max}) \square c, \text{ and}$$

– *For an achievability LTL property* $\Gamma \stackrel{def}{=} [\mathcal{R}]^{\leq k}_{\square c}$ *with* $\square \in \{\leq, <\}$*:*

$$query_{\mathcal{M}}(\Gamma) = query_{\mathcal{M}}([\mathcal{R}]^{\leq k}_{min})\square c.$$

*Where:*
$P = [\exists \mathcal{M}_e \in EC(\mathcal{M})\colon \mathcal{M}_e \models \mathcal{R} \wedge \exists i \in \{1, \ldots, k\} : s_i a_i s_{i+1} \cdots \in \Pi^{\sigma_{Ps}}_{\mathcal{M}_e}]$.

## 3 Multi-Objective Properties

So far, we have only considered properties with one reward-structure. However, in practice, taking a decision has multiple consequences. For example, taking an exam in Figure 2 increases our chance of getting hired, but it also costs money to take an exam. With multi-objective properties, we can express the trade-off between multiple variables by using multiple reward-structures.

We can visualise the relation between two or more reward-structures in a Pareto curve. A Pareto curve is the curve that displays all optimal solutions. A solution is on the Pareto curve if there is no other solution that has a better value for one of the reward-structures while keeping the other values the same. We also call such a vector not dominated.

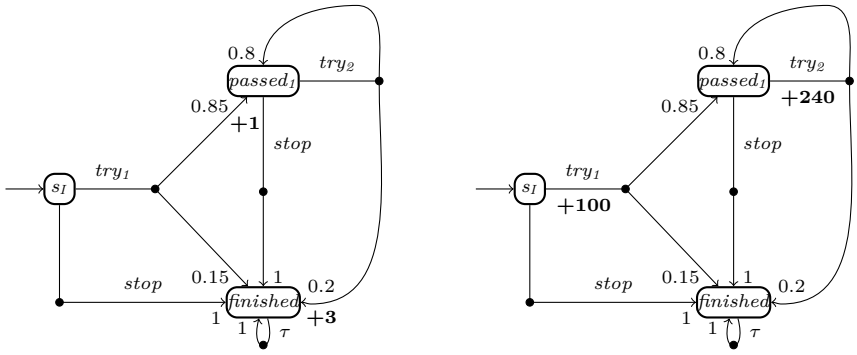
*Example 23.* If we consider the certification process again, now we will also include costs in addition to hiring points, since taking an exam is not free. Taking the first exam costs €100, while the second exam costs €240. For the hiring points, the rewards are based on the outcome of the exam. Therefore, we use the branch reward-structure $\rho_{hire}$, as shown in Figure 8a. Taking an exam however, does always cost money, no matter the outcome. Therefore, we use the transition reward-structure $\rho_{money}$ as visualised in Figure 8b.

Obviously, we want to maximise the hiring points while minimising the costs. If we do this for the reward-structures given in Figure 8, we get the Pareto curve shown in Figure 9. The achievable points consist of all points that are on the Pareto curve or are dominated by a point on the Pareto curve.

### 3.1 Definitions

**Definition 27.** *A* ***multi-objective property*** $\boldsymbol{\phi} = \langle \phi_1, \phi_2, \ldots, \phi_n \rangle$ *is a tuple of achievability and numerical subproperties* $\phi_i$*. An achievability subproperty is called a* ***constraint****. A numerical subproperty is called an* ***unknown****.*

We will now distinguish three types of multi-objective properties. The first of which has no unknowns and an arbitrary number of constraints. These are called multi-objective achievability properties (Definition 28). Next, we have multi-objective properties with one unknown and an arbitrary number of constraints, these are multi-objective numerical properties (Definition 29). Lastly, we consider properties with two or more unknowns and without constraints. These are called Pareto properties (Definition 30). Notice that there is no property with two or




**(a)** Annotated with $\rho_{hire}$ **(b)** Annotated with $\rho_{money}$

**Fig. 8.** Rewards for a certification process

more unknowns and an arbitrary number of constraints. This kind of property has not yet been implemented in any of the tools we explored and is also not common in the literature.

**Definition 28.** *A **multi-objective achievability property** is a multi-objective property consisting of zero unknowns and one or more constraints:*

$$\bar{\boldsymbol{\phi}} = \left\langle \bar{\phi}_1, \bar{\phi}_2, \ldots, \bar{\phi}_d \right\rangle .$$

*A multi-objective achievability property is satisfied in MDP $\mathcal{M}$ under a probabilistic step-positional strategy $\sigma_{Ps} \in Strat_{\mathcal{M}}$ when:*

$$\mathcal{M}, \sigma_{Ps} \models \bar{\boldsymbol{\phi}} \iff \forall i \in \{1, \ldots, d\} : \mathcal{M}, \sigma_{Ps} \models \bar{\phi}_i.$$

**Definition 29.** *A **multi-objective numerical property** is a multi-objective property consisting of one unknown in position $j \in \{1, \ldots, d\}$ and zero or more constraints:*

$$\hat{\boldsymbol{\phi}} = \left\langle \bar{\phi}_1, \ldots \hat{\phi}_j, \ldots, \bar{\phi}_d \right\rangle .$$

**Definition 30.** *A **Pareto property** is a multi-objective property consisting of two or more unknowns and zero constraints:*

$$\tilde{\boldsymbol{\phi}} = \left\langle \hat{\phi}_1, \hat{\phi}_2, \ldots, \hat{\phi}_d \right\rangle .$$

To find the optimal values such as in Figure 9, we must first determine which values are achievable. We define achievable values in Definition 31. Notice that the values on the Pareto curve itself are also achievable.

**Definition 31.** *The **achievable values** in MDP $\mathcal{M}$ for a Pareto property $\langle \phi_1, \ldots, \phi_d \rangle$ is the function $Ach_{\mathcal{M}}$ such that:*

$$Ach_{\mathcal{M}}(\left\langle \hat{\phi}_1, \ldots, \hat{\phi}_d \right\rangle) \stackrel{def}{=} \left\{ \langle x_1, \ldots x_d \rangle \in \mathbb{R}^d \mid \exists \sigma_{Ps} \colon \mathcal{M}, \sigma_{Ps} \models \langle \bar{\phi}_1, \ldots \bar{\phi}_d \rangle \right\},$$

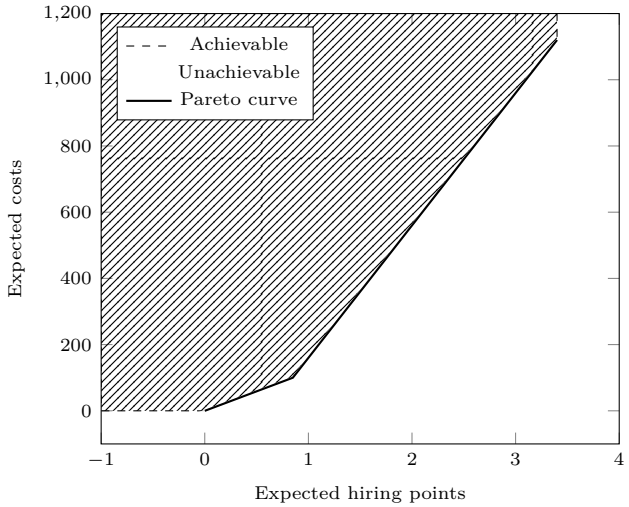



**Fig. 9.** Pareto curve of $\left\langle [CumRew_{\mathcal{M}}(\rho_{hire})]^{\leq\infty}_{max}, [CumRew_{\mathcal{M}}(\rho_{money})]^{\leq\infty}_{min} \right\rangle$

*where each* $\hat{\phi}_i$ *is an unknown in the form* $[Rew_{i\mathcal{M}}]^{\leq k}_{\triangle_i}$*. If* $\triangle_i = max$*, we get* $\bar{\phi}_i = [Rew_{i\mathcal{M}}]^{\leq k}_{\geq x_i}$*, and* $\bar{\phi}_i = [Rew_{i\mathcal{M}}]^{\leq k}_{\leq x_i}$ *if* $\triangle_i = min$*.*

We can now also define precisely what it means for a vector to dominate another vector. We do this by introducing the dominating relation under a Pareto property in Definition 32. A vector dominates another vector if it has a better value for at least one unknown and no worse value for any unknown. All vectors which are not dominated by any other vector are on the Pareto curve. This is shown, for example, in Figure 9.

**Definition 32.** *A vector* ***dominates*** *another vector under the Pareto property* $\boldsymbol{\phi} = \langle \phi_1, \ldots, \phi_n \rangle$ *iff it is in the relation* $>_{\boldsymbol{\phi}}$ *such that:*

$$>_{\boldsymbol{\phi}} = \left\{ \left\langle \boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{y} \in \mathbb{R}^d \right\rangle \mid \boldsymbol{x} \neq \boldsymbol{y} \wedge \forall i \in \{1, \ldots, d\} : x_i \bowtie_i y_i \right\},$$

*where* $\bowtie_i = \geq$ *if* $\phi_i$ *is a max unknown and* $\bowtie_i = \leq$ *otherwise.*

We also need a way to evaluate multi-objective properties. For this, we will overload the *query* function of Definition 19. For a multi-objective achievability property, this returns whether there exists a strategy such that the property is satisfied. For a multi-objective numerical property, this returns the optimal value (maximum for *max* and minimum for *min*) that can be achieved by a strategy while satisfying all constraints if it exists. If no such strategy exists, $\perp$ is returned. For a Pareto property, it returns the Pareto curve: the set of all achievable vectors that are not dominated by any other achievable vector.

**Definition 33.** *A* ***multi-objective query*** *for MDP* $\mathcal{M}$ *is a function* $query_{\mathcal{M}}$ *that evaluates a multi-objective property* $\boldsymbol{\phi}$ *and is defined as follows:*

– *For a multi-objective achievability property* $\boldsymbol{\phi}$*:*

$$query_{\mathcal{M}}(\boldsymbol{\phi}) \overset{def}{\iff} \exists \sigma_{Ps} \in Strat_{\mathcal{M}} \colon \mathcal{M}, \sigma_{Ps} \models \boldsymbol{\phi},$$

– *For a multi-objective numerical property* $\boldsymbol{\phi} = \left\langle \bar{\phi}_1, \ldots, [Rew_{j\,\mathcal{M}}]^{\leq k_j}_{max}, \ldots, \bar{\phi}_d \right\rangle$*, with a max unknown in position* $j$*:*

$$query_{\mathcal{M}}(\boldsymbol{\phi}) \overset{def}{=} \sup \left\{ c_j \in \mathbb{R} \mid \exists \sigma_{Ps} \colon \mathcal{M}, \sigma_{Ps} \models \left\langle \bar{\phi}_1, \ldots, [Rew_{j\,\mathcal{M}}]^{\leq k_j}_{\geq c_j}, \ldots, \bar{\phi}_d \right\rangle \right\},$$

– *For a multi-objective numerical property* $\boldsymbol{\phi} = \left\langle \bar{\phi}_1, \ldots, [Rew_{j\,\mathcal{M}}]^{\leq k_j}_{min}, \ldots, \bar{\phi}_d \right\rangle$*, with a min unknown in position* $j$*:*

$$query_{\mathcal{M}}(\boldsymbol{\phi}) \overset{def}{=} \inf \left\{ c_j \in \mathbb{R} \mid \exists \sigma_{Ps} \colon \mathcal{M}, \sigma_{Ps} \models \left\langle \bar{\phi}_1, \ldots, [Rew_{j\,\mathcal{M}}]^{\leq k_j}_{\leq c_j}, \ldots, \bar{\phi}_d \right\rangle \right\}$$

– *For a Pareto property* $\boldsymbol{\phi} = \langle \phi_1, \ldots, \phi_d \rangle$*:*

$$query_{\mathcal{M}}(\boldsymbol{\phi}) \overset{def}{=} \{ \boldsymbol{x} \in Ach_{\mathcal{M}}(\boldsymbol{\phi}) \mid \neg \exists \boldsymbol{y} \in Ach_{\mathcal{M}}(\boldsymbol{\phi}) \colon \boldsymbol{y} >_{\boldsymbol{\phi}} \boldsymbol{x} \}.$$

### 3.2 Tool Support

There are a few model checkers that support multi-objective model checking. However, they do not support all types of properties and settings. In this research, we focus on PRISM [53] (version 4.8.1), Storm [40] (version 1.8.1), and ePMC [31] (commit b1ba8ab). These model checkers all support multi-objective queries. However, the reward-structures and settings they support differ. We consider two algorithms supported by most of these tools: value iteration [30] and linear programming [28].

**3.2.1 Value Iteration** For value iteration, there are a few important things to notice in the tool support as listed at the left side of Table 2. First, properties with infinite cumulative rewards cannot be evaluated by any of the tools. However, the way in which infinite cumulative rewards are handled differs from tool to tool. Storm appears to detect properties with at least one infinite cumulative reward and tells the user that it does not support multi-objective properties with infinite rewards. PRISM in some cases only detects these properties for lower convergence thresholds and then also tells the user that such properties are not supported. ePMC on the other hand appears to diverge. Another interesting part of the supported features is that Storm does not support relative convergence thresholds. In addition, PRISM and ePMC only support cumulative rewards and probabilistic reachability. They do not implement algorithms for long-run average and reachability rewards.

**Table 2.** Tool support for multi-objective properties

| | Value Iteration | | | Linear Programming | | |
|---|---|---|---|---|---|---|
| Type | PRISM | Storm | ePMC | PRISM | Storm | ePMC |
| Absolute convergence threshold | ✓ | ✓ | ✓ | n/a | n/a | n/a |
| Relative convergence threshold | ✓ | ✗ | ✓ | n/a | n/a | n/a |
| Sound | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ |
| Exact | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ |
| Probabilistic reachability | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ |
| State reward-structure | ✗ | ✓ | ✓ | ✗ | ✓ | ✗ |
| Transition reward-structure | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ |
| Branch reward-structure | ✗ | ✓ | ✓ | ✗ | ✓ | ✗ |
| Cumulative reward | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ |
| Long-run average reward | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ |
| Reachability reward | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ |
| Step-bounded reward | ✓ | ✓ | ✓ | ✗ | ✗ | ✗ |
| Infinite cumulative reward | ✗ | ✗ | ✗ | ✗ | ✗ | ✗ |
| Achievability | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ |
| Numerical | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ |
| Pareto | 2 unknowns | ✓ | ✗ | ✗ | ✗ | ✗ |

**3.2.2 Linear Programming** Unfortunately, ePMC does not support linear programming. Storm and PRISM do, although both tools support fewer properties than they do for value iteration as shown on the right side of Table 2. In particular, step-bounds and Pareto queries are not supported when using linear programming. The most important difference between the tools for linear programming is that Storm does not support numerical queries for linear programming, while PRISM does.

## 4 Mistakes In Existing Model Checkers

Before this research, we tried to replicate a small part of the value iteration approach in [30]. We evaluated the "Task-graph scheduling problem"[1], using our implementation and PRISM [53]. In our notation, the property considered is of the form:

$$\hat{\phi} = \left\langle [CumRew_{\mathcal{M}}(\rho_{time})]^{\leq\infty}_{min}, [CumRew_{\mathcal{M}}(\rho_{energy})]^{\leq\infty}_{\leq 1.45} \right\rangle .$$

We found some interesting results for the query of $\hat{\phi}$ when using a relative convergence threshold as we will discuss in Section 4.2. We now evaluate what happens if we use more models, settings, and tools. This helps us to assess what happened with the Task-graph scheduling problem and whether the same happens for different models.

[1] `https://www.prismmodelchecker.org/files/atva12mo/`

**Alg. 2.** Multi-objective numerical query approximation

**Input:** MDP $\mathcal{M} = \langle S, s_1, \mathcal{A}, \delta \rangle$, multi-objective numerical property $\hat{\boldsymbol{\phi}} = \left\langle \bar{\phi}_1, \ldots, [Rew_{j\,\mathcal{M}}]^{\leq k_j}_{max}, \ldots, \bar{\phi}_n \right\rangle$ with the unknown in position $j$, upper bound $upper \in \mathbb{R}_+$, precision $\theta \in \mathbb{R}_+$

**Result:** $query_{\mathcal{M}}(\boldsymbol{\phi})$ with an error margin of $\pm\theta$

```
1  low ← 0
2  high ← upper
3  iterations ← ⌈log_2 (upper/θ)⌉
4  foreach iteration ∈ {1, ..., iterations} do
5  |   target ← (high+low)/2
6  |   φ̄ ← ⟨φ̄_1, ..., [Rew_j M]^{≤k_j}_{≥target}, ..., φ̄_n⟩
7  |   if query_M(φ̄) then
8  |   |   low ← target
9  |   else
10 |   |   high ← target
11 return (high+low)/2
```

### 4.1 Approach

In order to evaluate the tools, we only focus on multi-objective achievability and numerical properties. We do not focus on Pareto properties since they are harder to compare, since they do not output a number, but a geometric structure. In order to assess the results of the multi-objective queries, we use numerical multi-objective properties and compare the results of the direct multi-objective numerical query with an approximation by using multi-objective achievability queries.

**4.1.1 Approximation** We approximate multi-objective numerical queries in a binary search-like fashion [21]. We start with an upper and lower bound, where the lower bound is 0 (since all models we evaluate have non-negative rewards) and the upper bound is 1 for probabilistic reachability and defined by the user for rewards (our default is 100000). The approximation only works if the actual value is between these bounds. For each iteration, we then take $target = \frac{upper - lower}{2}$ as our target value. For a *max* unknown, we change it to a constraint with the bound $\geq target$, and for a *min* property, we change it to a $\leq target$ constraint.

We can then shrink the upper or lower bound, based on whether the multi-objective achievability property is achievable. We do this until we get to a user-defined precision. The algorithm is shown in Algorithm 2. We show the algorithm for a multi-objective numerical property with a *max* unknown. For a property with a *min* unknown, the $\geq$ on line 6 should be changed to a $\leq$ and the *low* and *high* on lines 8 and 10 should be interchanged.

For the results used in this paper, we use a timeout of two minutes. If a query does not return a result in two minutes, the experiment of which it is a

part is aborted. For example, if a value iteration query with a relative convergence threshold times out for PRISM, we abort the entire value iteration using a relative convergence threshold for PRISM only. For value iteration, we ran the experiments with convergence threshold values between 0 and 0.9, with a step size of 0.01.

We use this approach for both linear programming and value iteration implementations. As shown in Section 3.2, not all models and properties are supported by these tools. For value iteration, we use both absolute and relative convergence thresholds. The implementation used to do this can be found in the complementary Github repository[2].

We also use an early stopping mechanism. If the approximation and direct query of the lowest convergence threshold are equal to the queries for a higher convergence threshold, we assume the queries for all convergence thresholds values in between to be the same. We had to include this early stopping mechanism since the program took several days to run without this early stop. Even with the early stop, it took 2 full days to run. However, this means that we might stop too early if the model checkers do not behave as we expect.

**4.1.2 Models** The models used can be found in the complementary Github repository in the `/models` directory. Some models require "constants" to generate the model. These constants give some flexibility to the user, for example, to impact the state space. These constants are also specified in the Github repository. We also list the models here and introduce them:

- *Care home*: models a robot controllers that, while trying to complete a task, also completes soft goals [56]. For this model, we use two sets of constants. The second set of variables times out for all tools. We also evaluate two properties. The first has a minimum cumulative reward unknown. The second property has a maximum reachability reward unknown.
- *Client server*: models a client-server protocol with mutual exclusion that has probabilistic errors in the clients [50]. For this model, there are no constants, and we evaluate one property which has a maximum long-run average reward unknown.
- *Dining philosophers*: models the well-known dining philosophers problem [42], but without the fairness constraint [23]. This model does not have any constants and one property, which has a maximum long-run average reward unknown.
- *Dynamic power management*: assesses the performance and power consumption of a system and its components [61]. For this model, we use one set of constants and evaluate one property which has a step-bound minimum cumulative reward unknown.
- *Hiring process*: the model described in Figure 8a. For this model, there are no constants, and we evaluate one property with a maximum cumulative reward unknown.

[2] `https://github.com/Chickenpowerrr/multiobjectiveanalysis/releases/tag/1.0`

- *Mars rover*: models the experiments a Mars rover can do during a day on Mars [36]. For this model, we use two sets of constants. We also evaluate two properties, one has a maximum probabilistic reachability unknown and the other a minimum cumulative reward unknown.
- *Network virus*: models a virus that infects a network [55]. For this model, there are no constants, and we evaluate two properties, both with a maximum long-run average reward unknown.
- *Randomised consensus*: models asynchronous processes that communicate through a shared object [3]. This model does not have any constants and we evaluate one property with a probabilistic reachability unknown.
- *Resource gathering*: models an agent that collects resources and takes them home [8]. For this model, we use one set of constants and evaluate one property with a probabilistic reachability unknown.
- *Sensor network*: models sensors that send data over a channel with a bounded buffer [50]. This model has not constants and we evaluate one property with a maximum long-run average reward unknown.
- *Task graph scheduling*: finds optimal schedulers for tasks with energy consumption and time [62]. For this model, we use one set of constants and evaluate one property with a minimum cumulative reward unknown.
- *Team formation*: models a collaboration protocol [19]. This model has no constants and we evaluate four properties, two with a maximum probabilistic reachability reward unknown and two with a maximum reachability reward unknown.
- *Zeroconf network*: models a zero-configuration dynamic network for IPv4 [54]. For this model, we use one set of constants and evaluate one property with a probabilistic reachability unknown.
- *Zeroconf time based*: the same as the zeroconf network, but also incorporates time into the model.

### 4.2 Infinite Cumulative Reward

The model on which this entire research is based is the task graph scheduling model. During our preliminary research, we found that the result of the query for PRISM by changing the relative convergence threshold was non-monotonic, as the query drops around convergence threshold = 0.006 as shown in Figure 10. This is different from single-objective properties which are always monotonic as shown in Section 2.7. Moreover, the approximation and the direct multi-objective numerical query are quite far apart for some convergence threshold values.

When running the experiment with PRISM's linear programming solver, we get a value of 6.2, while ePMC's value iteration gives us even different results than shown in Figure 10. Storm, however, does give a hint about what might be happening. Storm does not want to verify the property since it claims that there are infinite rewards. It does however still not explain the non-monotonic increase we observed in PRISM.

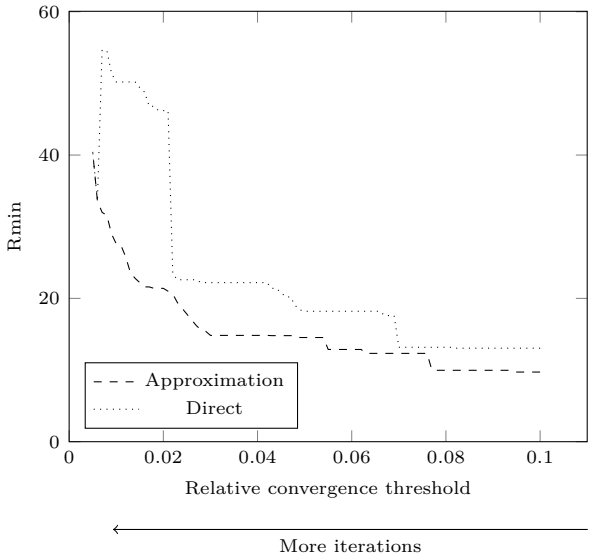



**Fig. 10.** Minimum reward for a multi-objective numerical query using PRISM

### 4.3 Results

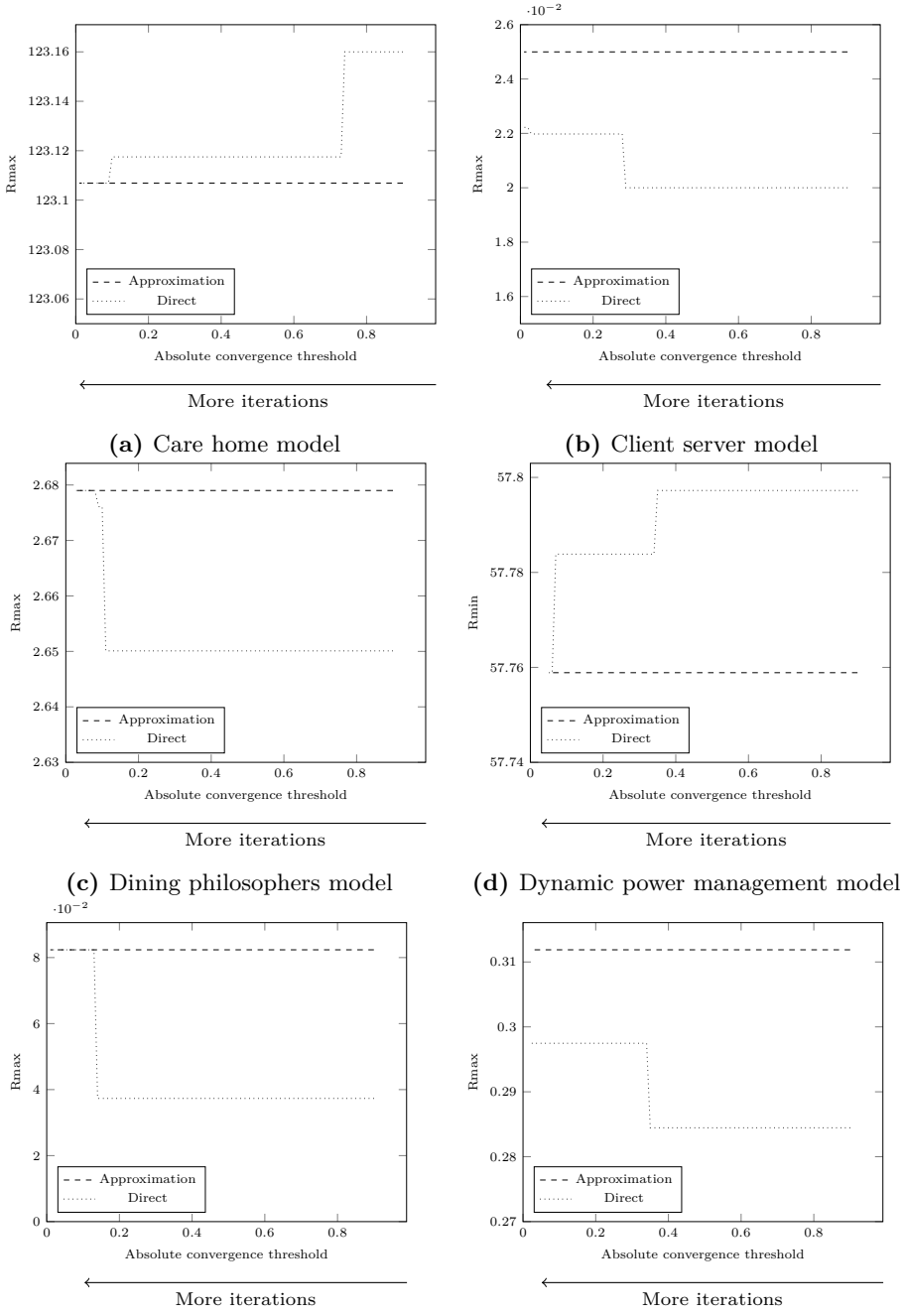
When we run our experimental setup, we get the results shown in Table 3. If either the approximation or direct query changes for different convergence thresholds, we plot both the approximation and the direct query. Otherwise, the value is shown in the table. If a value cannot be computed, for example when a timeout is triggered, the type of property or model is unsupported or because of an error in the model checker, we put -. If a direct query states that a property is unachievable, we denote it by $\perp$. We do want to note that our current implementation could not detect this for an approximation, so a 0.000 for an approximation could also mean that the property is unachievable in our setup.

The numbering of the constants corresponds to the property and constants listed in the `properties.pctl` and `constants.txt` files for each model in our repository.

**4.3.1 PRISM's Non-Monotic Behaviour** We observe that for some models, the results of the approximation are not always monotonic when using PRISM's value iteration solver. If we take the hiring process model with an absolute convergence threshold, we observe Figure 12c. The approximation with a high convergence threshold starts at 3.1, which is the actual value, only to drop and become the same as the direct query for smaller convergence thresholds. We would not expect the value to drop again for a lower convergence threshold, we would only expect the value to increase, as seen in Section 2.7. The same

**Table 3.** Experimental results

| | | | Value Iteration | | | | | | | | | | Linear Programming | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Relative convergence threshold | | | | Absolute convergence threshold | | | | | | | |
| | | | PRISM | | ePMC | | Storm | | PRISM | | ePMC | | Storm | PRISM |
| Model | Property | Constants | Approx | Direct | Approx | Direct | Approx | Direct | Approx | Direct | Approx | Direct | Approx | Approx Direct |
| Care home [56] | 1 | 1 | - | - | - | - | Figure 11a | | - | - | - | - | - | - |
| Care home [56] | 2 | 1 | - | - | 0.000 | -1000.0 | 0.000 | ⊥ | - | - | 0.000 | -1000.0 | 0.000 | - - |
| Client server [50] | 1 | n/a | - | - | - | - | Figure 11b | | - | - | - | - | - | - - |
| Dining philosophers [23] | 1 | n/a | - | - | - | - | Figure 11c | | - | - | - | - | - | - - |
| Dynamic power management [61] | 1 | 1 | 57.759 | 57.759 | - | - | Figure 11d | | 57.759 | 57.759 | - | - | - | - - |
| Hiring points (Figure 8a) | 1 | n/a | Figure 12a | | - | - | Figure 12b | | Figure 12c | | - | - | 0.000 | ∞ 3.100 |
| Mars rover [36] | 1 | 1 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 - |
| Mars rover [36] | 1 | 2 | Figure 13a | | 0.000 | 0.000 | Figure 13b | | Figure 13c | | Figure 13d | | - | - - |
| Mars rover [36] | 2 | 1 | Figure 14a | | 0.000 | 0.000 | 76.667 | 76.667 | Figure 14b | | 0.000 | 0.000 | 100.000 | 0.000 76.667 |
| Mars rover [36] | 2 | 2 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | -1.000 | - | - - |
| Network virus [55] | 1 | n/a | - | - | - | - | 3.254 | 3.254 | - | - | - | - | - | - - |
| Network virus [55] | 2 | n/a | - | - | - | - | Figure 11e | | - | - | - | - | - | - - |
| Randomised consensus [3] | 1 | n/a | 0.892 | 0.892 | 0.000 | 0.000 | 0.892 | 0.892 | 0.892 | 0.892 | 0.892 | 0.892 | 0.892 | 0.892 0.892 |
| Resource gathering [8] | 1 | 1 | Figure 15a | | 0.000 | 0.000 | Figure 15b | | Figure 15c | | Figure 15d | | 0.000 | 0.824 0.824 |
| Sensor network [50] | 1 | n/a | - | - | - | - | Figure 11f | | - | - | - | - | - | - - |
| Task graph scheduling [62] | 1 | 1 | - | - | 2.936 | 0.000 | - | - | - | - | 0.000 | 0.000 | - | 0.000 6.120 |
| Team formation [19] | 1 | n/a | - | - | - | - | 0.000 | ⊥ | - | - | - | - | 0.000 | - - |
| Team formation [19] | 2 | n/a | - | - | - | - | 0.000 | ⊥ | - | - | - | - | 0.000 | - - |
| Team formation [19] | 3 | n/a | - | - | 0.000 | -1000.0 | 0.000 | ⊥ | - | - | 1.980 | 1.980 | 0.000 | - - |
| Team formation [19] | 4 | n/a | - | - | 0.000 | -1000.0 | 0.000 | ⊥ | - | - | 1.714 | 1.714 | 0.000 | - - |
| Zeroconf network [54] | 1 | 1 | 3.1e-4 | 3.1e-4 | 0.000 | 0.000 | 3.1e-4 | 3.1e-4 | 3.1e-4 | 3.1e-4 | 3.1e-4 | 3.1e-4 | 0.000 | 3.1e-4 3.1e-4 |
| Zeroconf time based [54] | 1 | 1 | 2.0e-4 | 2.0e-4 | - | - | 2.0e-4 | 2.0e-4 | 2.0e-4 | 2.0e-4 | - | - | - | 2.0e-4 2.1e-4 |




**(a)** Care home model

**(b)** Client server model

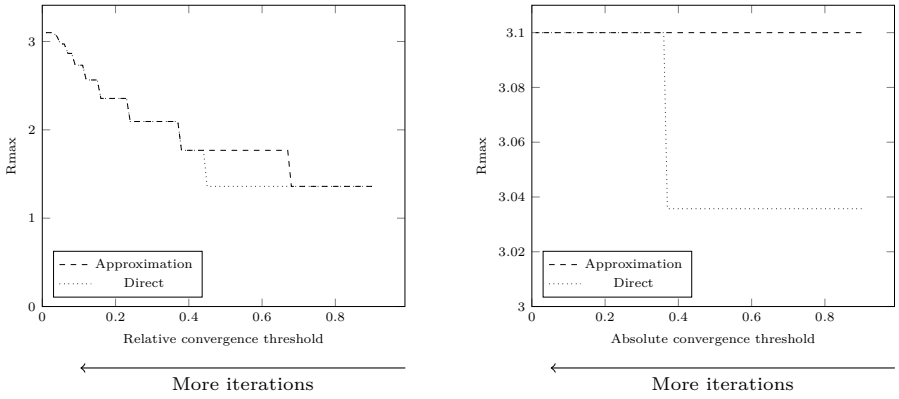
**(c)** Dining philosophers model

**(d)** Dynamic power management model

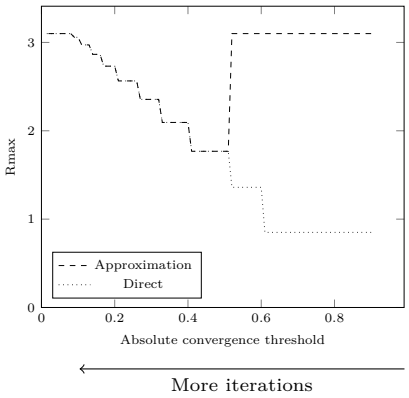
**(e)** Network virus model

**(f)** Sensor network model

**Fig. 11.** Value iteration with an absolute convergence threshold in Storm




**(a)** PRISM, relative $\epsilon$ **(b)** Storm, absolute $\epsilon$

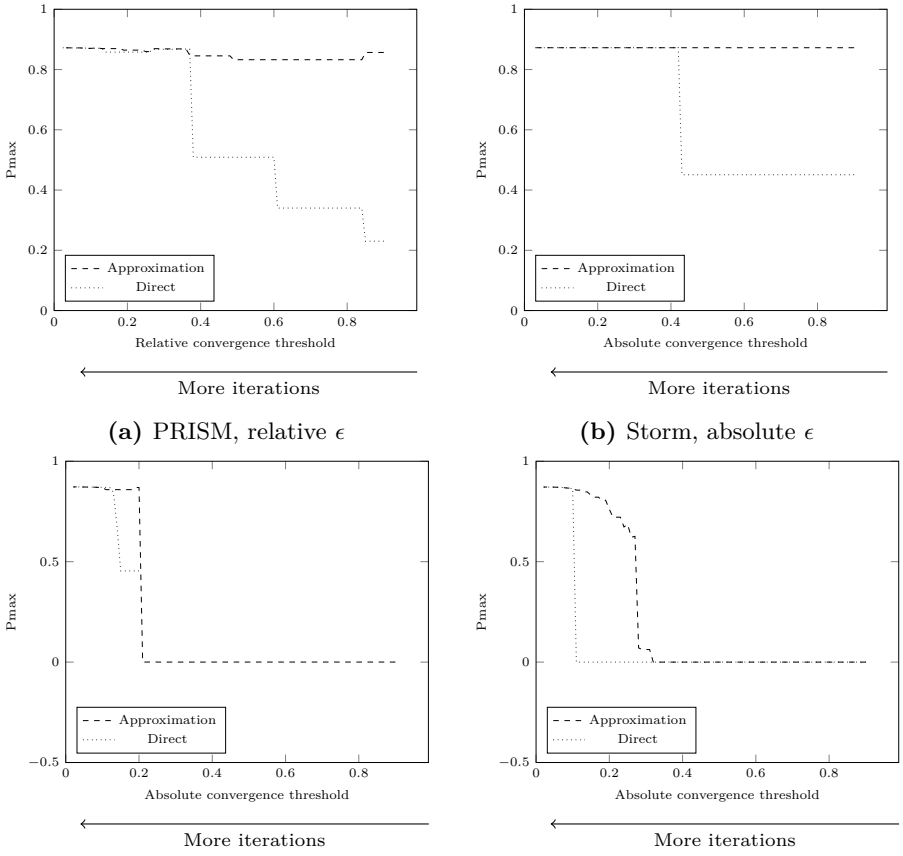



**(c)** PRISM, absolute $\epsilon$

**Fig. 12.** Value iteration for the hiring process model

non-monoticity for with an absolute convergence threshold can be observed in Figures 13c, 14b and 15c. It also occurs when using a relative convergence threshold, as shown in Figures 13a, 14a and 15a.

**4.3.2 PRISM's Problem Using Linear Programming** When we use the approximation algorithm with PRISM's linear programming solver, we get a curious result for the hiring process model. The hiring points seem to approach infinity if we add the constraint that the expected costs should be below €1000. However, from the MDP shown in Figure 8, it should be obvious that it is impossible to get more than 4 hiring points in total; in fact, the actual result should be 3.1. If we remove the expected cost constraint but check whether the multi-objective achievability property $\left\langle [CumRew_{\mathcal{M}}(\rho_{hire})]^{\leq\infty}_{\geq 10000} \right\rangle$, PRISM claims that the query is achievable. When playing around with the multi-objective achiev-




**(a)** PRISM, relative $\epsilon$

**(b)** Storm, absolute $\epsilon$

**(c)** PRISM, absolute $\epsilon$

**(d)** ePMC, absolute $\epsilon$

**Fig. 13.** Value iteration for first property of the Mars rover model

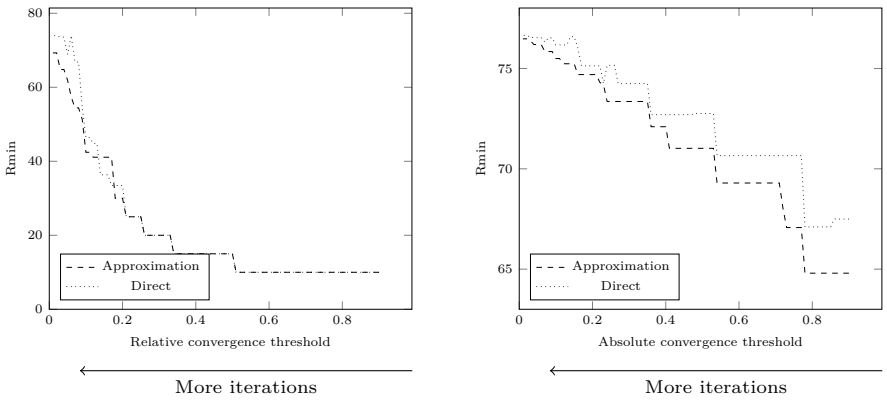
ability property containing only the expected hiring points, we found that when we update the constraint so that the expected hiring points are 3.4 or lower, the PRISM output contains the line:

```
LP problem solution found; result is 3.400000.
```

However, when we provide a higher value, which should not be achievable, the output contains the line:

```
LP problem solution not found; result is 0.000000.
```

Since the actual value is 3.4, as calculated in Example 13, PRISM seems to incidate that the query cannot be achieved. However, the result still states that the query is achievable, which is obviously incorrect. This means that something is going wrong in PRISM's linear programming engine. The output suggests that




**(a)** PRISM, relative $\epsilon$ **(b)** PRISM, absolute $\epsilon$

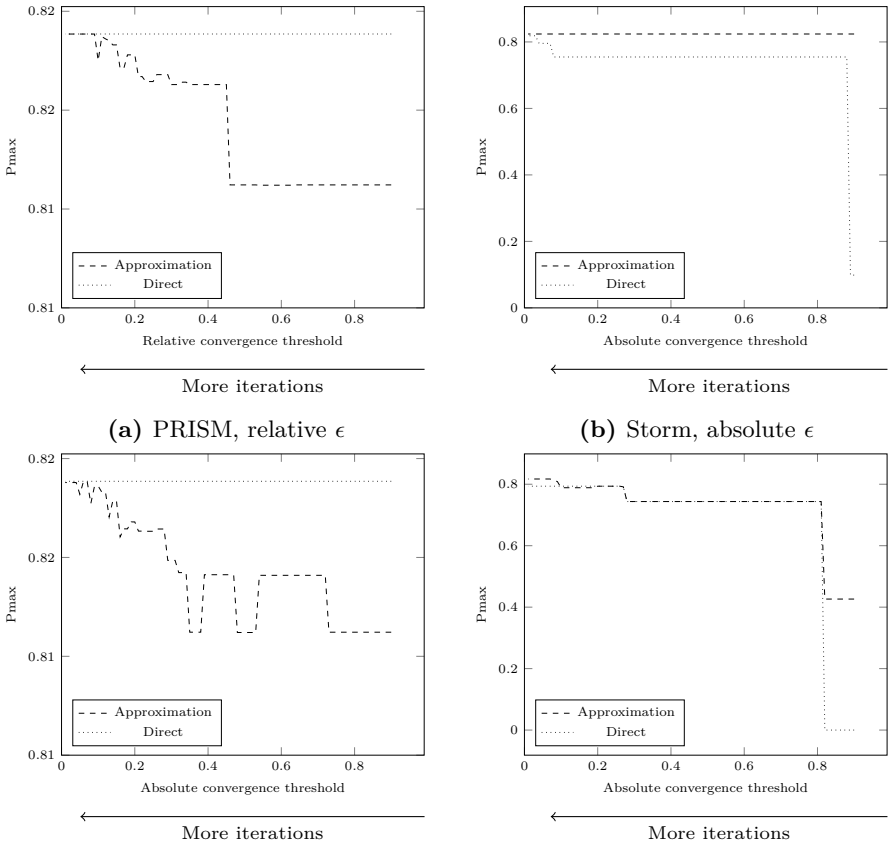
**Fig. 14.** Value iteration for second property of the Mars rover model

PRISM handles the case when no linear programming solution has been found incorrectly.

**4.3.3 Storm's Problem Using Linear Programming** For Storm we get a different problem with the hiring process model using linear programming. Whereas PRISM approached an infinite expected number of hiring points, Storm approaches 0 hiring points. This is also obviously wrong, since in Figure 8 it is obvious that if we only take the first exam, we will end up with at least 0.85 expected hiring points. If we again construct a multi-objective achievability property with only one constraint, which is the expected hiring points and check whether it is possible to get more points than a small value, such as 0.01, we find that Storm says that the query is unachievable if we are using the linear programming solver. It seems that any value above 0 is unachievable. This is also obviously incorrect, as any value below 3.4 should be achievable as shown in Example 13. The same happens for the zeroconf network model.

**4.3.4 ePMC's Problem Using Value Iteration** For ePMC, we observe something different. During the experiments, we found that ePMC produces results that are widely different from the other model checkers for a few models. Therefore, we investigate these results a bit more. In doing so, we find that ePMC produces inconsistent results. For example, for the Mars rover model using the second constants set with the default absolute convergence threshold, ePMC gives us the result:

$$query_{\mathcal{M}}(\left\langle \begin{matrix} [CumRew_{\mathcal{M}}(\rho_{time})]^{\leq\infty}_{min}, \\ [CumRew_{\mathcal{M}}(\rho_{energy})]^{\leq\infty}_{\leq 43.99999993400001} \end{matrix} \right\rangle) = -1.0000000.$$




**(a)** PRISM, relative $\epsilon$

**(b)** Storm, absolute $\epsilon$

**(c)** PRISM, absolute $\epsilon$

**(d)** ePMC, absolute $\epsilon$

**Fig. 15.** Value iteration for the resource gathering model

However, it is impossible to get a negative result, as all rewards defined in the model are non-negative. Of course, it could be the case that this is ePMC's way of saying that it is impossible to achieve the multi-objective achievability property:

$$\left\langle [CumRew_{\mathcal{M}}(\rho_{energy})]^{\leq \infty}_{\leq 43.99999993400001} \right\rangle.$$

However, if we construct multi-objective numerical properties for both these reward-structures without any constraints, we also get

$$query_{\mathcal{M}}(\left\langle [CumRew_{\mathcal{M}}(\rho_{time})]^{\leq \infty}_{min} \right\rangle) = -1.0000000, \text{ and}$$
$$query_{\mathcal{M}}(\left\langle [CumRew_{\mathcal{M}}(\rho_{energy})]^{\leq \infty}_{min} \right\rangle) = -1.0000000.$$

Both these multi-objective numerical properties do not have any constraints, so they are achievable. Since there are only non-negative rewards present in the

model, ePMC must be wrong here. ePMC also produces negative results for the team formation and care home models.

A curious result arises when running the randomised consensus model with a relative convergence threshold. In this case, we get the result:

$$query_{\mathcal{M}}(\left\langle \begin{matrix}[\mathcal{R}_{\Diamond one\_proc\_err}]^{\leq\infty}_{max}, \\ [\mathcal{R}_{\Box one\_proc\_ok}]^{\leq\infty}_{\geq 0.10833260973166493}\end{matrix} \right\rangle) = 0.0000000.$$

The other model checkers and even ePMC with an absolute convergence threshold get a value around 0.89167 (which is $1 - 0.10833260973166493$). When we investigate this again by creating two new multi-objective numerical properties containing only the reward-structures as an unknown, we get:

$$query_{\mathcal{M}}(\langle [\mathcal{R}_{\Diamond one\_proc\_err}]^{\leq\infty}_{max} \rangle) = 0.0000000, \text{ and}$$
$$query_{\mathcal{M}}(\left\langle [\mathcal{R}_{\Box one\_proc\_ok}]^{\leq\infty}_{\geq 0.10833260973166493} \right\rangle) = 0.0000000.$$

If we convert these two multi-objective numerical properties into single-objective numerical properties, we get completely different results:

$$query_{\mathcal{M}}([\mathcal{R}_{\Diamond one\_proc\_err}]^{\leq\infty}_{max}) = 1.0000000, \text{ and}$$
$$query_{\mathcal{M}}([\mathcal{R}_{\Box one\_proc\_ok}]^{\leq\infty}_{\geq 0.10833260973166493}) = 1.0000000.$$

This means that even if the relative convergence threshold stays the same, the two engines compute completely different results. This seems to indicate that there might be an issue in either of these computations; however, we cannot be 100% certain yet since the computations are not identical, just similar.

**4.3.5 Storm's Segmentation Fault** In Storm, we can get a segmentation fault. This occurs when using the first property for the Mars rover models using the linear programming solver and the second set of constants. This does not produce any wrong results, but it is inconvenient that the query cannot be computed. Moreover, the underlying problem that causes this segmentation fault might have an influence on other computations that do not crash.

**4.3.6 ePMC's NullPointerException** Lastly, we found with ePMC that for the team formation model, we get a NullPointerException if we try to evaluate the properties. If we run the model checker with assertions enabled, we get an AssertionError instead. However, this appears to be the way ePMC handles long-run average and reachability rewards, since they are not supported. This could be more user-friendly to avoid people trying to evaluate unsupported properties.

## 5 Solutions To Mistakes In Existing Model Checkers

We start by analysing PRISM, Storm and ePMC and try to find causes of the issues found in Section 4.3. To do this, we dive into the code and in some cases manage to resolve the problems and create bug reports for the ones that are too complex to quickly solve.

### 5.1 Storm

As shown in Section 4, we have seen that there are a few mistakes in the Storm model checker. In particular, the segmentation fault and the problem using linear programming.

**5.1.1 Segmentation Fault** The segmentation fault in Storm seems to arise from a StackOverflowError. This happens in the Z3ExpressionAdapter's visit function, which invokes the BinaryNumericalFunctionExpressions's accept function, which then invokes the Z3ExpressionAdapter's visit function again.[3] Debugging the Mars rover model itself was too challenging for us, since it has 161410 states and 302642 transitions. However, we reported the problem and the developers found that the problem arises since the model is so large that a part of the code could not handle the computation.[4] The problem has now been addressed, so no segmentation fault occurs in future versions of Storm.[5]

**5.1.2 Problem Using Linear Programming** We found that the linear programming problem in Storm arises from inaccuracy in double precision numbers. If we use Figure 16 in Storm, with probabilities $p_1$ and $p_2$ to end up in respectively $s_I$ and *goal* and query the multi-objective property consisting of only the maximum probabilistic reachability property which asks whether it is possible to reach *goal*, the query should obviously be true. However, we only get this when we use decimal values for $p_1$ and $p_2$ which can be expressed exactly using the IEEE 754 format. These are values that can be expressed as $\frac{n}{2^m}, n, m \in \mathbb{Z}$ [44]. For example, for $p_1 = 0.25, p_2 = 0.75$ or $p_1 = 0.125, p_2 = 0.875$, we get that the query is true. When we use a value that cannot be represented exactly like $p_1 = 0.3, p_2 = 0.7$ or $p_1 = 0.2, p_2 = 0.8$, no solution to the linear program can be found, and thus the result is false.

This problem probably arises from the fact that there are strict equality signs for each state in the linear program [28]. It is hard to satisfy several equality constraints if they all need to round to an exact result if the values in between are getting rounded. A way to solve this would be to add a small value $\epsilon > 0$, and for each equality $x = y$ try to achieve $x + \epsilon \geq y \wedge x - \epsilon \leq y$ instead.

Another way to solve the issue is by using the `--exact` flag, this ensures that Storm represents decimal values as fractions instead of floating-point numbers, which bypasses the inaccuracy. This means that in the current implementation in Storm, the linear programming solver is only useful when using the `--exact` flag since otherwise the chances of getting meaningful results are very slim. This is why it would be useful to add a warning when using the linear programming solver without the `--exact` flag. This ensures that Figure 16 also works with values for $p_1$ and $p_2$ that would otherwise have been rounded due to double precision.

[3] `https://github.com/moves-rwth/storm/blob/9fba97a4a14f346427b7c8954cb5750ce311afd6/src/storm/adapters/Z3ExpressionAdapter.cpp#L226`

[4] `https://github.com/moves-rwth/storm/issues/625`

[5] https://github.com/moves-rwth/storm/pull/626

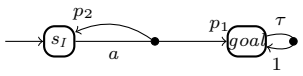



**Fig. 16.** Double-precision problem in Storm

### 5.2 PRISM

In Section 4 we have also detected a problem using linear programming in PRISM. We will discuss the origin of this issue and resolve it.

**5.2.1 Problem Using Linear Programming** In Section 4.3.2 we found that PRISM has a linear programming problem. Fortunately, it turned out that this problem was quite easy to solve. In PRISM, the unknowns should always come before the constraints. Since the linear programming solver cannot solve Pareto queries, their native C++ code only needed to check whether the first property in the multi-objective property is an unknown or a constraint to determine whether the multi-objective property was a multi-objective numerical or achievability property. However, since reward properties and probabilistic reachability properties are stored in two different arrays, the first element of both of these arrays was checked. Unfortunately, no bounds check on the length of these arrays was performed before doing this. This means that if only reward properties or only probabilistic reachability properties were used, it was undefined behaviour whether the multi-objective property would be recognised as a multi-objective reachability or numerical property. However, since the value of the out-of-bounds field was typically 0 on the machine the experiments were performed on, multi-objective properties with only reward properties, the multi-objective achievability properties consisting of only reward properties were often classified as numerical properties. For this reason, an unachievable result was reported as NaN which is the way for PRISM to say that a multi-objective numerical query is unachievable, instead of 0.0, which is PRISM's way of communicating an unachievable multi-objective achievability query. Therefore, NaN was incorrectly classified as an achievable result, since it was not 0.0. After checking the bounds of the arrays when classifying the multi-objective property, this issue has been resolved.[6]

### 5.3 ePMC

Most problems found in Section 4 originate in the ePMC model checker. We will discuss the root of these problems. Unfortunately, we are not able to resolve all of these problems, since they are quite fundamental to the model checker.

[6] `https://github.com/prismmodelchecker/prism/pull/244`

**5.3.1 NullPointerException** In Section 4.3.6, we determined that ePMC throws a NullPointerException for certain properties. Upon further inspection of the code, this turned out to be not caused by the long-run average and reachability rewards being unsupported. Instead, it was a fairly simple oversight in a constructor. A variable was passed on as a parameter, however, it was never assigned to its corresponding field. The field to which it should have been assigned was then used later on without being assigned. After updating this field in the constructor, the issue was resolved.[7]

**5.3.2 Unachievable Numerical Queries** When going over the code, we found that unachievable numerical queries were not reported to the user. There was a boolean value which reported whether a given property was achievable, however it was never used. The code that reported it to the user was commented out. Upon contacting the authors of the tool, it turned out that they did not know why they commented it out. When we uncommented this code, it seemed that unachievable numerical queries were reported correctly.[8]

**5.3.3 Problem Using Value Iteration** As shown in Section 4.3.4, ePMC has a value iteration problem. Upon investigating the issue, we found that ePMC attempts to take a more efficient approach for giving rewards for reaching a goal state in a probabilistic reachability property than that is described in Proposition 7. Instead, they give a "stop reward", when a goal state is reached, if there is no opportunity to reach a state that satisfies more reachability properties. While this approach works for single-objective properties and seems to work for multi-objective properties which consist of only probabilistic reachability properties, it breaks down when rewards get involved. The stop rewards are in some cases also handed out for reward properties. This means that we can get a reward of $+1$ for reaching a state for a reward property, even though no reward has been specified. The cause of this part of the problem is in the calculation of accepting states, which is done based on Rabin automata as shown in Definition 23. The goal set used for probabilistic reachability is then determined using [26].

The $-1.000$ result shown in Section 4.3 also arises from this problem. Since the algorithm used can only maximise the rewards, the rewards are negated before the start of the algorithm if the reward property should be minimised. For probabilistic reachability properties, they do not invert the reward. In this case, they take the complement of the goal set if the property should be minimised, meaning that a reward for reaching a goal state is always non-negative. However, the stop reward is used instead of the value from the reward-structure, if the stop reward is higher than the reward in the reward-structure. Since the rewards are negative for minimising reward properties, the stop reward is always going to be

[7] `https://github.com/Chickenpowerrr/ePMC/commit/dfd56134d14d282481b9b9519ef0cca5372c242a`

[8] `https://github.com/iscas-tis/ePMC/pull/11/commits/0dc1e136153b8a6ba1d0c1fb77d1d34c24c663e8`

**Table 4.** Status of problems

| Problem | Status |
|---|---|
| Storm's segmentation fault | Problem reported and resolved |
| Storm's problem using linear programming | The `--exact` flag should be used |
| PRISM's problem using linear programming | Problem found and resolved |
| ePMC's NullPointerException | Problem found and resolved |
| ePMC's unachievable numerical queries | Problem found and resolved |
| ePMC's problem using value iteration | Problem found, reported and partially resolved |

higher than any reward. This means that it is impossible to correctly evaluate any minimising reward property since its value can only be between $-1$ and 0 in the current implementation because it will always decide to take the stop reward instead of a reward from the reward structure. We have created a pull request to resolve this problem.[9]

The $-1000.0$ result arises since the model checker has not been fully implemented. The $-1000.0$ is set as a hack until the code is properly implemented.[10] However, upon contacting the authors of the tool, it appears that they do not have the resources required to update the tool.

### 5.4 Status

As a summary of this section, we have listed all the problems discussed and show what we have done with them in Table 4. This shows which problems are currently still open.

## 6 Infinite Cumulative Rewards

As shown in Table 2, the current tools do not support models with at least one infinite cumulative reward. One of the reasons for this is that the algorithm used by all tools to convert LTL probabilistic reachability properties into cumulative rewards assumes that there are no infinite maximising cumulative rewards [28]. Both Storm and PRISM perform this check, ePMC on the other hand, does never detect them.

The concept of the algorithm including the check is shown in Algorithm 3. An infinite cumulative reward can only occur if we can reach an end component with a nonzero reward. If this is the case, we can decide to stay in the end component for ever and obtain the reward an infinite number of times, thus obtaining an infinite reward. Notice that in Algorithm 3 a property is only considered invalid when an infinite cumulative reward can be obtained while satisfying all (other)

[9] `https://github.com/iscas-tis/ePMC/pull/11`

[10] `https://github.com/iscas-tis/ePMC/blob/b1ba8abf9f52c05f23421efb1abd1be7618f426d/plugins/propertysolver-multiobjective/src/main/java/epmc/multiobjective/MultiObjectiveUtils.java#L147`

**Alg. 3.** Algorithm to convert LTL properties into cumulative rewards [28]

**Input:** An MDP, DRAs for the probabilistic reachability properties $\{\phi_1, \ldots, \phi_n\}$

**Result:** The modified MDP and reward-structures representing the probabilistic reachability properties

1 Construct the composition of the MDP and the DRAs for all probabilistic reachability properties. For each maximising cumulative reward property and the unknowns, determine whether it is possible to reach an end component which contains a reward higher than 0 while satisfying all constraints of the multi-objective property. If this is the case, stop because there is an infinite cumulative reward.

2 Remove all actions that will not be used because the assumption holds. These are all the actions with a reward higher than 0 for a maximising cumulative reward property that are contained in an end-component. Then repeatedly remove all states with no outgoing actions and all actions that contain a branch that leads to a non-existing state until no changes occur.

3 We turn all probabilistic reachability properties into cumulative rewards. To do this, we add the state $s_{dead}$ to the state space. If we have probabilistic reachability properties $\{\phi_1, \ldots, \phi_n\}$, for each $R \in \mathcal{P}(\{\phi_1, \ldots, \phi_n\})$, we find the end-components that accept all properties in $R$ and do not contain any reward. For each of the states in these end components, we add a transition to the $s_{dead}$ state with a reward of $+1$ for all properties in $R$.

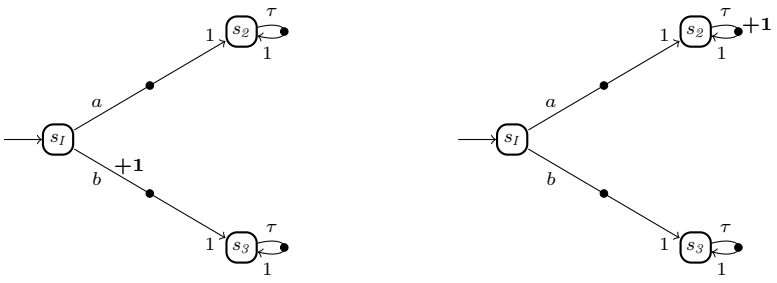

**Fig. 17.** Infinite cumulative reward with reward-structure $\rho_\infty$

constraints. This means that we do not only need to check whether we can reach such an infinite end component from the initial state, but can do so while satisfying all constraints. This requires an additional multi-objective query, just to validate the validity of a model.

### 6.1 Existing Approaches

Right now, PRISM does not perform the check shown in Algorithm 3 on the unknown in case of a multi-objective property with a minimising cumulative reward unknown. For example, if we evaluate the multi-objective property

$$\left\langle [CumRew_{\mathcal{M}}(\rho_\infty)]^{\leq \infty}_{min} \right\rangle$$




**(a)** Reward-structure $\rho_1$ **(b)** Reward-structure $\rho_{\infty'}$

**Fig. 18.** Infinite cumulative reward edge case

on the MDP shown in Figure 17, we have an infinite cumulative reward of $\infty$. However, since PRISM does not perform the check in this case, PRISM will continue to verify the property. Although value iteration does not converge on this model, the linear program solver will report a value of 0.0, which is obviously incorrect. We reported this problem to the maintainers.[11]

Storm handles infinite cumulative rewards slightly differently. It discards any strategies that contain an infinite cumulative reward for any of the subproperties. This means that if we for example evaluate the multi-objective property

$$\left\langle [CumRew_{\mathcal{M}}(\rho_{\infty'})]^{\leq\infty}_{min}, [CumRew_{\mathcal{M}}(\rho_1)]^{\leq\infty}_{\leq 0} \right\rangle$$

on Figure 18, Storm does not report the infinite cumulative reward, even though the result of the query would expected to be infinity. Instead, it reports that the query is unachievable, since it does not consider any strategy that includes the action $a$, since it would lead to an infinite cumulative reward.[12] This can be confusing to users, especially since

$$\left\langle [CumRew_{\mathcal{M}}(\rho_1)]^{\leq\infty}_{\leq 0} \right\rangle$$

is achievable according to Storm.

Also, if we consider the model Figure 18 and evaluate the query

$$\left\langle [CumRew_{\mathcal{M}}(\rho_{\infty'})]^{\leq\infty}_{max}, [CumRew_{\mathcal{M}}(\rho_1)]^{\leq\infty}_{\geq 2} \right\rangle$$

using PRISM and Storm, both output that the model has an infinite cumulative reward. However, if we look at the model, we see that it is impossible to satisfy the constraint that $\rho_1$ becomes 2. This means that both tools do not only exclude models for which all constraints are satisfied as shown in Algorithm 3. This saves a separate multi-objective query, since detecting an infinite cumulative reward can be done using a graph algorithm as we will see in Section 6.2.

[11] `https://github.com/prismmodelchecker/prism/issues/253`
[12] `https://github.com/moves-rwth/storm/issues/624`

Moreover, Storm and PRISM do not report which cumulative rewards are infinite, making it harder to debug.

### 6.2 Our Approach

As seen in Section 6.1, if an infinite cumulative reward is detected in the current tools, it is not verified whether it can actually be obtained by a strategy satisfying the constraints as shown in Algorithm 3. We assume that this approach has been chosen since performing this requires a multi-objective query on the same MDP using another reward-structure. This means that the check would take a similar amount of time as the main check. This significantly slows down the model checkers for only a small set of missing models.

Since we also would not like to slow down our implementation so drastically, we also choose a different approach than the one used in Algorithm 3. However, since infinite cumulative rewards typically denote modelling errors, we choose to exclude *all* reward-structures for which there exists a strategy that leads to an infinite cumulative reward, even when minimising rewards. Otherwise, the user might still provide negative rewards and minimise to end up with a negative infinite reward. This is essentially equivalent to maximising for a positive reward to end up with and infinite reward, just the signs are flipped. This means that we can evaluate a slightly smaller set of models than PRISM and Storm. However, we know for sure that we can evaluate all the models that pass the check. We opted not to choose the Storm approach since, as shown in Section 6.1 the outputs can be confusing.

In order to do this, we could use an algorithm that works for the non-multi-objective case and apply it separately for each reward-structure. However, if we have several reward structures, we can also perform this computation only once. For this, we came up with the approach shown in Algorithm 4. It is inspired by the approach taken by Storm. We prevent each model with an end component with a non-negative reward that is reachable from the initial state. We do this since the reward being in an end component means that we can construct a strategy such that we stay in the end component for ever while obtaining the reward an infinite number of times.

## 7 Multi-Objective Model Checking Algorithms

So far, we have found problems in multi-objective algorithms without diving into the algorithms themselves. To better understand what is going on behind the scenes, we present the multi-objective model checking algorithms used so far in this section. We do this in more depth than the respective papers, since we have no page limit and can thus provide more insight into the algorithms.

We will use this knowledge to replicate the papers [29,30] in Section 8. This allows us to find errors in the underlying papers, which can help future implementations avoid falling into these traps. Moreover, this allows us to attempt to implement a model checker that learns from the mistakes found in Section 4.

**Alg. 4.** Algorithm to detect infinite cumulative rewards for MDP $\mathcal{M}$

**Input:** The MDP $\mathcal{M}$ and reward-structures $\boldsymbol{\rho} \in Struct^d_{\mathcal{M}}$
**Result:** Whether the MDP contains infinite cumulative rewards

1 For the MDP $\mathcal{M}$ calculate the set of maximal end components $MEC_{\mathcal{M}} = \{EC_1, \ldots, EC_k\}$, where each $EC_i = \langle \mathcal{S}_{EC_i}, s_{EC_i}, \mathcal{A}_{EC_i}, \delta_{EC_i} \rangle$.
2 Compute the function $l$ which labels each maximal end component with the reward-structures which have a non-zero reward in the maximal end component. Formally: $l(EC_i) = \{\rho_j \mid \exists s, s' \in \mathcal{S}_{EC_i} \colon \exists a \in \alpha_{EC_i}(s) \colon \rho_j(s)(a)(s') \neq 0\}$.
3 Calculate the set of reward-structures with an infinite reward by computing the union of the labels of maximal end components reachable from initial state $s_I$. $l_\infty = \bigcup_{EC \in \left\{ EC_i \in MEC_{\mathcal{M}} \mid \exists s' \in \mathcal{S}_{EC_i} \colon s_I \xrightarrow{\mathcal{M}} s' \right\}} l(EC)$.
4 If $l_\infty = \emptyset$ the model is valid so perform the actual model checking algorithms. Otherwise, report to the user that the reward-structures in $l_\infty$ have infinite cumulative rewards and do not perform the actual model checking algorithms.

### 7.1 Value Iteration Approach

The first way in which we will compute multi-objective queries is by using value iteration [30]. This approach is an extension of the approach shown for single-objectives in Section 2.7.

**7.1.1 Intuition** We will start by looking at the intuition behind the algorithm. This will help to understand the principle behind the algorithm and show the necessity of some mathematical definitions. We consider the case of a Pareto query, since the multi-objective achievability and numerical queries work in the same way, but with some optimisations.

The example we will consider is the certification process model shown in Figure 8. Since we are looking for the Pareto curve, we know that the result should be equivalent to Figure 9 at the end of the algorithm.

To simplify the algorithm, we only consider multi-objective properties with cumulative rewards for which all constraints have a $\geq$ bound, and all unknown are maximising. To convert properties into this form, we use Proposition 8.

**Proposition 8.** *Let $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ be an MDP, then:*

$$query_{\mathcal{M}}([Rew_{\mathcal{M}}]^{\leq k}_{\leq c}) = query_{\mathcal{M}}([-Rew_{\mathcal{M}}]^{\leq k}_{\geq -c}), \text{ and}$$
$$query_{\mathcal{M}}([Rew_{\mathcal{M}}]^{\leq k}_{min}) = -query_{\mathcal{M}}([-Rew_{\mathcal{M}}]^{\leq k}_{max}).$$

*Proof. For the first part of the proposition, we only need Definitions 18 and 19 and the simple relation:* $a \leq b \iff -a \geq -b$.

$$\begin{aligned}
&query_{\mathcal{M}}([Rew_{\mathcal{M}}]^{\leq k}_{\leq c}) \\
&= \exists \sigma_{Ps} \in Strat_{\mathcal{M}} \colon \mathcal{M}, \sigma_{Ps} \models [Rew_{\mathcal{M}}]^{\leq k}_{\leq c} \\
&= \exists \sigma_{Ps} \in Strat_{\mathcal{M}} \colon Rew_{\mathcal{M}}(k, \sigma_{Ps}) \leq c \\
&= \exists \sigma_{Ps} \in Strat_{\mathcal{M}} \colon -Rew_{\mathcal{M}}(k, \sigma_{Ps}) \geq -c \\
&= \exists \sigma_{Ps} \in Strat_{\mathcal{M}} \colon \mathcal{M}, \sigma_{Ps} \models [-Rew_{\mathcal{M}}]^{\leq k}_{\geq -c} \\
&= query_{\mathcal{M}}([-Rew_{\mathcal{M}}]^{\leq k}_{\geq -c}).
\end{aligned}$$

*For the second part of the proposition, we need Definitions 17 and 19 and the duality of the infimum and supremum operators on a set* $X$ *of real numbers:* $\inf X = -\sup -X$.

$$\begin{aligned}
&query_{\mathcal{M}}([Rew_{\mathcal{M}}]^{\leq k}_{min}) \\
&= \inf_{\sigma_{Ps} \in Strat_{\mathcal{M}}} Rew_{\mathcal{M}}(k, \sigma_{Ps}) \\
&= - \sup_{\sigma_{Ps} \in Strat_{\mathcal{M}}} -Rew_{\mathcal{M}}(k, \sigma_{Ps}) \\
&= -query_{\mathcal{M}}([-Rew_{\mathcal{M}}]^{\leq k}_{max}).
\end{aligned}$$

□

Since the certification process model uses the property

$$\left\langle [CumRew_{\mathcal{M}}(\rho_{hire})]^{\leq \infty}_{max}, [CumRew_{\mathcal{M}}(\rho_{money})]^{\leq \infty}_{min} \right\rangle,$$

we should evaluate the property

$$\left\langle [CumRew_{\mathcal{M}}(\rho_{hire})]^{\leq \infty}_{max}, [CumRew_{\mathcal{M}}(-\rho_{money})]^{\leq \infty}_{max} \right\rangle$$

and change the sign of *money* after the algorithm has finished.

In each step of the algorithm, we will compute a "weight vector". This weight vector assigns various weights to all reward-structures in the multi-objective property. These weights are all between 0 and 1, and they should sum up to one. Notice that this is equivalent to constructing a probability distribution over the points. A linear combination where the constants are a weight vector is called a convex combination, which we will cover in greater depth in Section 7.1.2.

With a weight vector $\boldsymbol{w} \in [0,1]^d$, we can then find the step-positional strategy $\sigma_{\mathcal{S}}$ for multi-objective property

$$\left\langle [CumRew_{\mathcal{M}}(\rho_1)]^{\leq k_1}_{\circ_1}, \ldots, [CumRew_{\mathcal{M}}(\rho_d)]^{\leq k_d}_{\circ_d} \right\rangle,$$

that achieves the highest value for the sum $\sum_{i=1}^{d} w_i \cdot CumRew_{\mathcal{M}}(\rho_i)(k_i, \sigma_{\mathcal{S}})$ and the rewards achieved by using this strategy $\boldsymbol{q} = \langle CumRew_{\mathcal{M}}(\rho_i)(k_i, \sigma_{\mathcal{S}}) \rangle_{i \in \{1,\ldots,d\}}$.

The algorithm for computing the step-positional strategy and point $\boldsymbol{q}$ will be covered in Section 7.1.8. Notice that we do not compute the best probabilistic step-positional strategy, but the non-probabilistic step-positional strategy. We do this since if we know all points that can be achieved using non-probabilistic step-positional strategies, the points from all probabilistic step-positional strategies can be computed using convexity which will be covered in Section 7.1.2.

*Example 24.* In the certification process it turns out that the candidate points when using any weight vector are: $\langle 0, 0\rangle$, $\langle 0.85, -100\rangle$, and $\langle 3.4, -1120\rangle$. These are the points corresponding to memoryless strategies, since a step-positional strategy cannot be better than a memoryless strategy if only infinite step-bounds are used [29]. These points are not coincidentally the boundary points shown in Figure 9. Any point in a line segment between two of those points would correspond to a probabilistic step-positional strategy, which has a $p$ chance of taking the strategy corresponding to one point and a $1 - p$ chance of taking the strategy corresponding to the other. If we take the weight vector $\langle 0.99, 0.01\rangle$, of these boundary points, $\langle 0, 0\rangle$ has the highest sum: $0.99 \cdot 0 + 0.01 \cdot 0 = 0$, $0.99 \cdot 0.85 + 0.01 \cdot -100 = -0.1585$, and $0.99 \cdot 3.4 + 0.01 \cdot -1120 = -7.834$. Hence, from the computation we would get $\boldsymbol{q} = \langle 0, 0\rangle$ with this weight vector.

Since we maximise the sum, we know that no achievable value can have a higher sum. This means that if we compute $\boldsymbol{q}$, we have an upper bound on the Pareto curve. In two dimensions, this upper bound is a line. The line has the normal vector $\boldsymbol{w}$ and goes through $\boldsymbol{q}$. This line is described by the equation $\boldsymbol{w} \cdot \boldsymbol{x} = \boldsymbol{w} \cdot \boldsymbol{q}$. This equation holds in all dimensions. In higher dimensions, it is known as a hyperplane, which will be covered in more detail in Sections 7.1.3 and 7.1.10. Please note that if one tries to draw the vector $\langle 0.99, 0.01\rangle$ from Example 24 in Figure 9 and draws the line that is 90° to that, it seems to go through both the achievable and unachievable regions. This happens since the y-axis is scaled by a factor of 200 for visualisation purposes. However, if both axis are not scaled, the vector is also normal to the line in the visualisation.

Now that we have an upper bound, we also need a lower bound. The lower bound can be found by constructing the curve that connects all the points found $\boldsymbol{q}$, or, in higher dimensions, the convex hull of these points as we cover in Section 7.1.5. All points below the curve are also achievable, as they are dominated by the points on this curve, as shown in Definition 32. It turns out that connecting these points works since connecting the points is equivalent to taking a convex combination of these points. This convex combination represents the probabilistic step-positional strategy with a probability distribution over the step-positional strategies that lead to the connected points. This set of points is called the downward closure, which we will discuss in Section 7.1.6.

*Example 25.* Recall that we are using the certification process model and needed to flip the sign for the expected costs. We start the algorithm by finding the maximum value for each of the subproperties and finding which values for the other subproperties are found. In our example, we do this using the weight vectors $\langle 1, 0\rangle$ and $\langle 0, 1\rangle$. Using the weight vector $\langle 1, 0\rangle$, we find the point $\boldsymbol{q} =$

$\langle 3.4, -1120\rangle$. This means that all points dominated by this point and the point itself form the lower bound. For the upper bound, we now know that there is no achievable point with more hiring points than 3.4, as it would otherwise have had a higher sum and should therefore have been selected instead of $\langle 3.4, -1120\rangle$. This is shown in Figure 19a.

We then use the weight vector $\langle 0, 1\rangle$. This gives us the point $\boldsymbol{q} = \langle 0, 0\rangle$. This means that we can update the upper bound again, since we now also know that it is impossible to achieve a cost higher than 0. Moreover, we can update the lower bound, since all points dominated by a convex combination of $\langle 0, 0\rangle$ and $\langle 3.4, -1120\rangle$ are now confirmed to be achievable. We show this in Figure 19b.

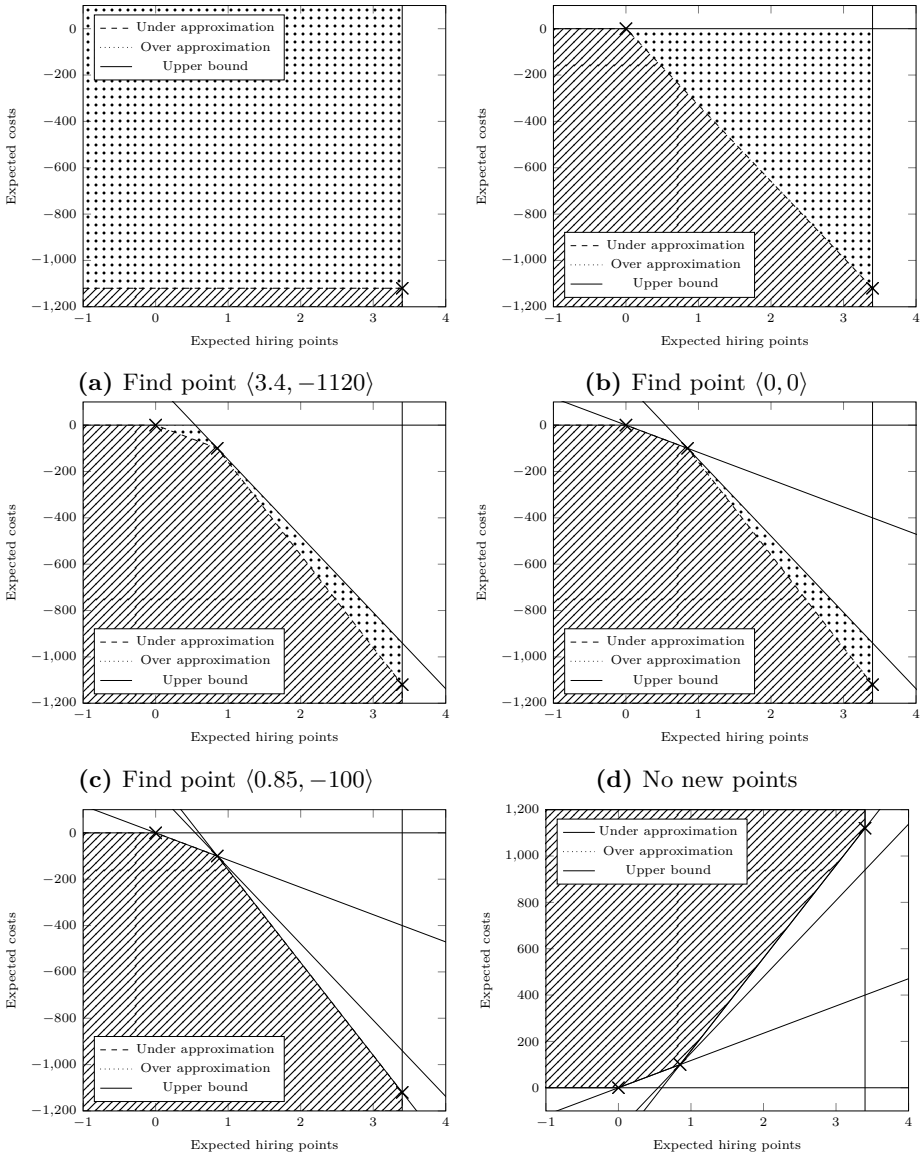
Now that the most obvious weight vectors have been used, we need to find a new weight vector such that the over and under approximation get closer to each other. We also know that the weight vector represents a line. Therefore, it makes the most sense to select the weight vector, such that if there is no new point, the upper and under approximation are equal. This can be achieved by using the weight vector that is normal to the line from $\langle 0, 0\rangle$ to $\langle 3.4, -1120\rangle$. If there is no point above this line, the newly found upper bound will go through both $\langle 0, 0\rangle$ and $\langle 3.4, -1120\rangle$ and we can terminate the algorithm. In this case however, we find a new point. This point is $\boldsymbol{q} = \langle 0.85, -100\rangle$, as shown in Figure 19c.

We now have two new lines at the boundary of the under approximation. We again use the weight vectors normal to these lines. We start with taking the weight vector for the line through $\langle 0, 0\rangle$ and $\langle 0.85, -100\rangle$ and then the line through $\langle 0.85, -100\rangle$ and $\langle 3.4, -1120\rangle$. This time, there are no points above these lines. Because of the choices of the weight vectors, the upper and under approximation will be equivalent after these to steps as demonstrated in Figures 19d and 19e.

Since the under and over approximation are equivalent, we must now have constructed the entire curve. Now, the last step to take is to switch the sign of the expected costs as shown in Proposition 8, since it was initially a minimising unknown. Note that after doing this, Figure 19f is equivalent to Figure 9, as required.

**7.1.2 Convexity** As we have seen, the algorithm is heavily dependent on the concept of convexity [15]. In geometric terms, convexity means that if we have a shape and we pick any two points inside of a shape and draw a line segment between those two points, no point on the line segment is outside of the shape. If a shape is not convex, we call it concave. We show this in Figure 20, for the convex shape, there is no way in which we can draw a line segment between two points inside of the shape such that the line segment leaves the shape. In the concave shape, on the other hand, we can easily find two points inside the shape, but the line segment leaves the shape.

Another important geometric concept we will use are polytopes [57]. A polytope is the higher-dimensional equivalent of a polygon. This means that for a $d$-dimensional space, the shape consists only of $(d-1)$-dimensional flats, also called facets. These facets are in turn constructed from $(d-2)$-dimensional flats,




**(a)** Find point $\langle 3.4, -1120 \rangle$

**(b)** Find point $\langle 0, 0 \rangle$

**(c)** Find point $\langle 0.85, -100 \rangle$

**(d)** No new points

**(e)** No new points

**(f)** Invert expected costs

**Fig. 19.** Multi-objective value iteration example

**(a)** Convex Shape **(b)** Concave Shape

**Fig. 20.** Convexity Example

**(a)** Polytope **(b)** Non-Polytope

**Fig. 21.** Polytope Example

which are called ridges. In 2 dimensions, this means that a facet is a line segment and a ridge a point. In Figure 21 we can see that our concave shape from Figure 20 is a polytope, since it consists only of straight line segments. A circle is not a polytope since it contains curved edges.

Most shapes we will consider after this are both convex and a polytope. We call these shapes convex polytopes. For example, a square is both convex and a polytope.

We will also express convexity using sets and equations. If we have a set of Pareto optimal points, such as $\{\langle 0,0\rangle, \langle 0.85,100\rangle, \langle 3.4,1120\rangle\}$ as in Figure 9, we can find other achievable points using a probability distribution between the strategies used to generate these achievable points. Because each point corresponding to a strategy gets a value between 0 and 1, with the sum of these values summing up to 1, we can also express this using a concept called a convex combination [15].

**Definition 34.** *A **convex combination** of a set of vectors* $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d\} \subseteq \mathbb{R}^n$ *is the tuple* $\sum_{i=1}^{d} w_i \cdot \boldsymbol{x}_i$*, with* $\boldsymbol{w} = \langle w_1, \ldots, w_d\rangle \in \mathbb{R}^d$*,* $\forall i \in \{1, \ldots, d\} : w_i \geq 0$*,* $\sum_{i=1}^{d} w_i = 1$*.*

*Example 26.* If we take the strategy obtaining the achievable point $\langle 0,0\rangle$ with a probability of 0.05, the strategy resulting in $\langle 0.85,100\rangle$ with probability 0.1 and in the remaining cases take the strategy resulting in $\langle 3.4,1120\rangle$, we obtain the convex combination $0.05 \cdot \langle 0,0\rangle + 0.1 \cdot \langle 0.85,100\rangle + 0.85 \cdot \langle 3.4,1120\rangle = \langle 2.97,962\rangle$. Hence, the point for this probabilistic strategy would be $\langle 2.97,962\rangle$.

**7.1.3 Affine Hyperplane** An affine hyperplane is the generalisation of a plane in a 3-dimensional space to arbitrary dimensions. In general, it is a $(d-1)$-

flat in $d$-dimensional space [57]. This means that in 2-dimensional space, a hyperplane is a line and a plane in 3-dimensional space. We will need affine hyperplanes to describe convex hulls in Section 7.1.5 and to understand separating hyperplanes in Section 7.1.10, which are fundamental building blocks of Algorithms 8, 10 and 11. We can express an affine hyperplane using its normal vector and an offset from the origin.

Notice the word "affine" in "affine hyperplane". The affine prefix indicates that the hyperplane is part of an affine space. In the words of the mathematician Berger "An affine space is nothing more than a vector space whose origin we try to forget about, by adding translations to the linear maps" [13]. This simply means that we do not care whether the origin is included in the hyperplane. However, when we perform computations on geometric structures in affine spaces such as affine hyperplanes, we will typically require to shift the geometric structure, to include the origin.

**Definition 35.** *An **affine hyperplane** is the tuple $H = \langle \langle a_1, \dots a_d \rangle, b \rangle$, and contains all points:*

$$\left\{ \langle x_1, \dots, x_d \rangle \in \mathbb{R}^d \mid \sum_{i=1}^{d} a_i \cdot x_i = b \right\}.$$

*The set of all affine hyperplanes in $d$-dimensional space, is denoted as $\mathcal{H}_d$.*

In order to find the affine hyperplane through $d$ affine independent points $\{\boldsymbol{x_1}, \dots, \boldsymbol{x_d}\} \in \mathbb{R}^d$, we can create a system of linear equations such that for each point $\sum_{j=1}^{d} a_j \cdot x_{ij} - b = 0$. Recall that a null space of the matrix $M$ is:

$$Null(M) = \left\{ \boldsymbol{x} \in \mathbb{R}^{d+1} \mid M\boldsymbol{x} = \boldsymbol{0}_d \right\}.$$

This means that we should find the null space of the matrix: $M_{hyp} = \begin{bmatrix} \boldsymbol{x_1}^T & -1 \\ \vdots & \\ \boldsymbol{x_d}^T & -1 \end{bmatrix}$. For any vector $\langle y_1, \dots, y_d, y_{d+1} \rangle$ in the null space of $M_{hyp}$, the affine hyperplane $\langle \langle y_1, \dots y_d \rangle, y_{d+1} \rangle$ contains all points $\boldsymbol{x_i}$.

An important feature of hyperplanes is that they divide the space into two half-spaces [4]. Using hyperplanes, we can determine in which of these two half-spaces any given point lies. The overlap between several half-spaces can be considered as a shape. We will use this fact to express a convex polygon as a set of hyperplanes in Section 7.1.5.

To compute in which half-space from an affine hyperplane $\langle \langle a_1, \dots a_d \rangle, b \rangle$ a vector $\langle x_1, \dots, x_d \rangle \in \mathbb{R}^d$ lies, we can evaluate the following sum $\sum_{i=1}^{d} a_i \cdot x_i - b$. All points in the same half-space will have the same sign. So for all points in one half-space of the hyperplane, the summation will be positive, and for all points in the other half-space of the hyperplane the outcome will be negative.

Another important property of hyperplanes, which we will use, is that we can calculate the distance from a hyperplane to a point by scaling the sum that we used to determine the half-space.

**Proposition 9.** *The distance from the hyperplane* $\langle \boldsymbol{a}, b\rangle \in \mathcal{H}_d$ *and the point* $\boldsymbol{x} \in \mathbb{R}^d$*, is:*

$$\frac{|\boldsymbol{a}\cdot\boldsymbol{x} - b|}{\|\boldsymbol{a}\|}.$$

*Proof. Since* $\boldsymbol{a}$ *is the normal vector of the hyperplane, the distance between the point* $\boldsymbol{x}$ *and the hyperplane, is the distance between* $\boldsymbol{x}$ *and the intersection of the line* $\boldsymbol{x} + \lambda\boldsymbol{a}$ *and the hyperplane. At this point, the distance is minimised. For* $\boldsymbol{x} + \lambda\boldsymbol{a}$ *to be on the hyperplane, by Definition 35 we must have:*

$$\begin{aligned}
\boldsymbol{a}\cdot(\boldsymbol{x} + \lambda\boldsymbol{a}) &= b\\
\boldsymbol{a}\cdot\boldsymbol{x} + \lambda\boldsymbol{a}^2 &= b\\
\boldsymbol{a}\cdot\boldsymbol{x} + \lambda\|a\|^2 &= b\\
\lambda &= \frac{-\boldsymbol{a}\cdot\boldsymbol{x} + b}{\|a\|^2}
\end{aligned}$$

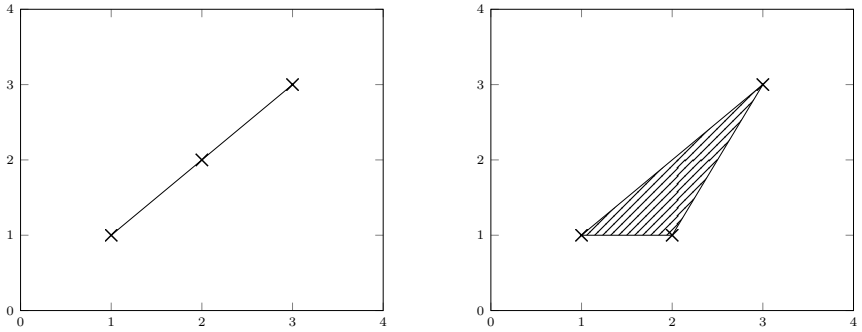
*So we now need to calculate the distance between* $\boldsymbol{x}$ *and* $\boldsymbol{x} + \frac{-\boldsymbol{a}\cdot\boldsymbol{x}+b}{\|a\|^2}\cdot\boldsymbol{a}$*. We see that both vectors are offset by* $\boldsymbol{x}$*, so the distance between these two points is the length of the vector* $\frac{-\boldsymbol{a}\cdot\boldsymbol{x}+b}{\|a\|^2}\cdot\boldsymbol{a}$*. Well:*

$$\begin{aligned}
&\|\frac{-\boldsymbol{a}\cdot\boldsymbol{x} + b}{\|\boldsymbol{a}\|^2}\cdot\boldsymbol{a}\|\\
=&|\frac{-\boldsymbol{a}\cdot\boldsymbol{x} + b}{\|\boldsymbol{a}\|^2}|\cdot\|\boldsymbol{a}\|\\
=&\frac{|-\boldsymbol{a}\cdot\boldsymbol{x} + b|}{\|\boldsymbol{a}\|^2}\cdot\|\boldsymbol{a}\|\\
=&\frac{|-\boldsymbol{a}\cdot\boldsymbol{x} + b|}{\|\boldsymbol{a}\|}\\
=&\frac{|\boldsymbol{a}\cdot\boldsymbol{x} - b|}{\|\boldsymbol{a}\|}.
\end{aligned}$$

□

**7.1.4 Affine Subspace** To compute the convex hull as covered in Section 7.1.5 in an arbitrary number of dimensions, the points should not all lie in a lower-dimensional shape. Otherwise, the set of points is said to be degenerate [32].

*Example 27.* If we have the points $\left\{\begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}2\\2\end{bmatrix}, \begin{bmatrix}3\\3\end{bmatrix}\right\}$, if we take each point to be represented as $\begin{bmatrix}v_1\\v_2\end{bmatrix}$ all these points are on the line $v_1 = v_2$ as shown in Figure 22a. This means that all linear combinations of these points will end up on this line. Since their linear combinations only span a line and not the entire affine space, we consider the points to span an affine subspace.



**(a)** Points span 1 dimension **(b)** Points span 2 dimensions

**Fig. 22.** Examples of points in an affine space

If on the other hand, we have the points $\left\{\begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}2\\1\end{bmatrix}, \begin{bmatrix}3\\3\end{bmatrix}\right\}$, we cannot express $v_1$ or $v_2$ in terms of the other. As shown in Figure 22b observe that in 2 dimensions this results in the shape having an area. If we would now take the set of all linear combinations of these points, we would obtain the entire affine space.

If points only span an affine subspace, some dimensions are linearly dependent on the other dimensions, and thus we can eliminate these linearly dependent dimensions without losing information. For example, if $v_1 = v_2$, it means that the first dimension is linearly dependent on the second. Therefore, we can find a linear transformation to remove these linearly dependent dimensions from each point to transform each point to a lower dimension. In this case, we go from 2 dimensions to 1, since we remove the first dimension. The advantage of using a linear transformation is that if a point is inside of the convex hull in the subspace, it is also inside of the convex hull if we transform all points back to the original space. To use a linear transformation, we must offset the points such that the origin is included, since a linear transformation uses a vector space instead of an affine space. This does not change the relative positioning of the points and thus does not change the structure of the convex hull. The offset can be achieved by subtracting any of the points from all points.

*Example 28.* If we take the set $\left\{\begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}2\\2\end{bmatrix}, \begin{bmatrix}3\\3\end{bmatrix}\right\}$ again, we can express each point as $\begin{bmatrix}v_1\\v_2\end{bmatrix} = \begin{bmatrix}v_2\\v_2\end{bmatrix}$. Hence, we can remove $v_1$ without loss of information. We can achieve this by using the linear transformation $\begin{bmatrix}0 & 1\end{bmatrix}$. For a linear transformation to work, we need to shift the points so that the origin is included. We do this by subtracting $\begin{bmatrix}1\\1\end{bmatrix}$ from each point before the transformation. We then obtain $\begin{bmatrix}0 & 1\end{bmatrix}(\begin{bmatrix}v_1\\v_2\end{bmatrix} - \begin{bmatrix}1\\1\end{bmatrix}) = \begin{bmatrix}v_2 - 1\end{bmatrix}$. The point $\begin{bmatrix}v_2 - 1\end{bmatrix}$ has only one dimension, but

**Alg. 5.** Subspace Transformation - *ComputeSubspace*

**Input:** Points $\{\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}\} \in \mathcal{P}(\mathbb{R}^d)$
**Result:** Transformation $T$, inverse transformation $T^{-1}$, offset $\boldsymbol{o} \in \mathbb{R}^d$

$\langle \boldsymbol{y_1}, \ldots, \boldsymbol{y_n} \rangle \leftarrow \langle \boldsymbol{x_1} - \boldsymbol{x_1}, \ldots, \boldsymbol{x_n} - \boldsymbol{x_1} \rangle$
$M \leftarrow \mathit{RowReducedEchelonForm}(\begin{bmatrix} \boldsymbol{y_1} & \boldsymbol{y_2} & \cdots & \boldsymbol{y_n} & I_d \end{bmatrix})$
$f \leftarrow \left\{ \langle di, ri \rangle \mid di, ri \in \{1, \ldots, d\}, M[ri, n + di] \neq 0, \; \forall ci \in \{1, \ldots, n + di - 1\} : M[ri, ci] = 0 \right\}$
$r \leftarrow \{\langle di \in \{1, \ldots, d\}, \|\{di' \in \{1, \ldots, d\} \mid di' \notin f, di' \leq di\}\| \rangle \mid \forall ri \colon \langle di, ri \rangle \notin f\}$

$T \leftarrow (d - \|r\|) \times d$ matrix filled with zeros
$T^{-1} \leftarrow d \times (d - \|R\|)$ matrix filled with zeros
**foreach** $di \in \{1, \ldots, d\}$ **do**
  **if** $\forall ri \colon \langle di, ri \rangle \notin f$ **then**
    **foreach** $\langle di', di'' \rangle \in r$ **do**
      $T^{-1}[di, r(di')] \leftarrow -M[f(di), di']$
  **else**
    $T[r(di), di] \leftarrow 1$
    $T^{-1}[di, r(di)] \leftarrow 1$
**return** $T, T^{-1}, \boldsymbol{x_1}$

it contains the same information as $\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$, since $v_1 = v_2$. Therefore, the point can be converted back using a linear transformation as well. To do this, we first apply the inverse transformation $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and then add the offset $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$ that we used to convert the set points to a vector space: $(\begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} v_2 - 1 \end{bmatrix}) + \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} v_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$.

The procedure we use to convert the points into the corresponding subspace is shown in Algorithm 5. On the first line, we transform the affine space into a vector space by ensuring that the origin is in the set of points without changing the relative positioning of the points. We then construct the matrix $\begin{bmatrix} \boldsymbol{y_1} & \boldsymbol{y_2} & \cdots & \boldsymbol{y_n} & I_d \end{bmatrix}$. Since the vector space spanned by the vectors $\{\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}\}$ is expressed as $\{\sum_{i=1}^{n} \lambda_i \boldsymbol{y_i} \mid \boldsymbol{\lambda} \in \mathbb{R}^n\}$, each row in the matrix $M$ corresponds to the equation for a dimension. The first $n$ columns on row $ri$, represents the sum $\sum_{i=1}^{n} \lambda_i y_{i_{ri}}$ and the equation is equal to the sum of dimensions in the last $d$ columns. We then find the row reduced echelon form [58] of this system of linear equations to find which dimensions depend on each other. This is the case where there is no $\lambda_i$ left in an equation but there are still dimensions present.

*Example 29.* We take the set of points $\left\{ \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \begin{bmatrix} 3 \\ 3 \end{bmatrix} \right\}$ again. This gives us the matrix $\begin{bmatrix} 0 & 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 0 & 1 \end{bmatrix}$, whose row reduced echelon form is $\begin{bmatrix} 0 & 1 & 2 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$. This matrix represents the equations $\lambda_2 + 2\lambda_3 = v_2$ and $0 = v_1 - v_2$, which means that $v_1 = v_2$. Hence, we know that $v_1$ is linearly dependent on $v_2$.

On line 3, we search for the equations where no $\lambda_i$ is present, since equations without any $\lambda_i$ can be written in the form of $v_i = \sum_{j=i+1}^{d} a_j v_j, \boldsymbol{a} \in \mathbb{R}^d$. This means that the $i$th dimension is fixed based on the other dimensions. We store the dimension $i$ and the line on which the equation is written, so that we can use the equation later in the algorithm.

*Example 30.* We start with $M = \begin{bmatrix} 0 & 1 & 2 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 \end{bmatrix}$. We need to store the dependent dimension, which as we determined is $v_1$, which is represented by a 1, the index of its corresponding column. The linear dependence on the other variables is described in the second row. Hence, we obtain $f = \{\langle 1, 2\rangle\}$.

Now that we know which dimensions can be expressed in terms of the other dimensions, we can compute the transformation to a subspace by removing all fixed dimensions. We do this on line 4 by storing for each free dimension *di* how many free dimensions are present in the dimensions up to and including *di*. This means that each free dimension has its own unique number between 1 and the number of free dimensions, which corresponds to the dimension it will end up in after the transformation.

*Example 31.* The free dimensions are the dimensions that are not fixed and therefore are not part of $f$. In our case, this means that 2 is the only free dimension. Since there is no smaller free dimension, we notice that the second dimension will end up as the first dimension after the transformation. This results in $r = \{\langle 2, 1\rangle\}$.

After that, we create the forward transformation from the higher to the lower dimension on line 5, and the inverse transformation on line 6. For any linear transformation, observe that

$$\begin{bmatrix} x_{1,1} & \cdots & x_{1,d} \\ \vdots & \ddots & \vdots \\ x_{d',1} & \cdots & x_{d',d} \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_d \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{d} x_{1,i} \\ \vdots \\ \sum_{i=1}^{d} x_{d',i} \end{bmatrix}.$$

For the forward transformation, we need to ensure that for each free variable in dimension *di*, that ends up in dimension $r(di)$. To do this, we need to set $T_{r(di),di} = 1$, since this will ensure that $v_{di}$ ends up in position $r(di)$ in the new vector.

For the inverse transformation, we need to ensure that the rows in the resulting vector are equal to the weights for the linear combination of the free variables. Observe that the row *di* in the inverse transformation $T^{-1}$ should contain the weights for the linear combination of the *di*th dimension after the inverse transformation has been applied. Hence, for a free variable, in position *di*, we only need to set $T^{-1}_{di,r(di)}$. We do this on line 13. For a fixed variable, we need to set the value equal to the linear combination found on line 2. This is done on line 10.

*Example 32.* Since only the second dimension is free, we obtain the forward transformation $T = \begin{bmatrix} 0 & 1 \end{bmatrix}$. Furthermore, the linear combination $v_1$ is $v_2$ and $v_2$ is stored in the first dimension of the transformed vector, the same holds for $v_2$. Therefore, we obtain the inverse transformation $T^{-1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

Notice that $T^{-1}T \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = T^{-1} \begin{bmatrix} v_2 \end{bmatrix} = \begin{bmatrix} v_2 \\ v_2 \end{bmatrix}$, which is indeed the form of all the vectors in the subspace spanned by the points $\left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 2 \\ 2 \end{bmatrix} \right\}$.

In fact, an important property of Algorithm 5 is that, when applied to the set $X$, for each point $\boldsymbol{y}$ in the same subspace as the points in $X$, we have $T^{-1}T(\boldsymbol{y} - \boldsymbol{o}) = \boldsymbol{y} - \boldsymbol{o}$. The inverse also holds. Moreover, if a point is not in this subspace, this equation does not hold. Therefore, we can check using Algorithm 5 whether a point is in the same subspace.

**Proposition 10.** *When $T$, $T^{-1}$, and $\boldsymbol{o}$ are obtained by applying Algorithm 5 to the set $X$ we have $T^{-1}T(\boldsymbol{y} - \boldsymbol{o}) = \boldsymbol{y} - \boldsymbol{o}$ if and only if $(\boldsymbol{y} - \boldsymbol{x'}) \in Span(\{\boldsymbol{x} - \boldsymbol{x'} \mid \boldsymbol{x} \in X\})$ with $\boldsymbol{x'} \in X$.*

*Proof sketch. We have chosen $\boldsymbol{x'}$ to be an arbitrary vector in $X$, just like $\boldsymbol{o}$. For the remainder of this sketch, we will only consider the case $\boldsymbol{x'} = \boldsymbol{o}$. In this case, we can take the set $Y = \{\boldsymbol{x} - \boldsymbol{o} \mid \boldsymbol{x} \in X\}$ and show that when $T$, $T^{-1}$, and $\boldsymbol{o}$ are obtained by applying Algorithm 5 to the set $Y$, then $T^{-1}T\boldsymbol{z} = \boldsymbol{z}$ if and only if $\boldsymbol{z} \in Span(Y)$.*

- *$Span(Y)$ is the subspace spanned by the vectors in $Y$. When we have $\boldsymbol{z} \in Span(Y)$, $T\boldsymbol{z}$ will remove only the dimensions that are fixed for each vector in the subspace that contains all the points in $Y$, and thus all the points in $Span(Y)$. These fixed dimensions are reconstructed from its linear combination when $T^{-1}(T\boldsymbol{z})$ is applied as explained above, so $T^{-1}T\boldsymbol{z} = \boldsymbol{z}$.*
- *When we have $T^{-1}T\boldsymbol{z} = \boldsymbol{z}$, we know that by its construction $T\boldsymbol{z}$ is the same as $\boldsymbol{z}$, but potentially with fewer dimensions. In the case where the number of dimensions is equivalent, we know that there were no fixed dimensions. Therefore, each dimension is free and $Span(Y)$ must be the entire vector space and thus $\boldsymbol{z} \in Span(Y)$. If the number of dimensions is smaller, at least one dimension must have been fixed and removed in $T\boldsymbol{z}$ compared to $\boldsymbol{z}$. Since $T^{-1}T\boldsymbol{z} = \boldsymbol{z}$, we know that the linear combination for these removed dimensions must have been the same as for each vector in $Y$. Since all removed dimensions can be expressed using the same linear combinations required for each point in $Y$, we can come up with a linear combination of the vectors in $Y$ to construct $\boldsymbol{z}$, thus $\boldsymbol{z} \in Span(Y)$.*

*Since $\boldsymbol{z} \in Span(Y)$ implies that $T^{-1}T\boldsymbol{z} = \boldsymbol{z}$ and the other way around, we know that $T^{-1}T\boldsymbol{z} = \boldsymbol{z}$ if and only if $\boldsymbol{z} \in Span(Y)$.*

*Example 33.* We take $T = \begin{bmatrix}0\ 1\end{bmatrix}$, $T^{-1} = \begin{bmatrix}1\\1\end{bmatrix}$, and $\boldsymbol{o} = \begin{bmatrix}1\\1\end{bmatrix}$ as in the previous example. In this case, we observe that a point outside of the subspace such as $\begin{bmatrix}0\ 1\end{bmatrix}$ would result in:

$$\begin{aligned}
&\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}0\ 1\end{bmatrix}\left(\begin{bmatrix}0\\1\end{bmatrix}-\begin{bmatrix}1\\1\end{bmatrix}\right)\\
=&\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}0\ 1\end{bmatrix}\begin{bmatrix}-1\\0\end{bmatrix}\\
=&\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}0\end{bmatrix}\\
=&\begin{bmatrix}0\\0\end{bmatrix}\\
\neq&\begin{bmatrix}0\\1\end{bmatrix}.
\end{aligned}$$

We observe that a point $\boldsymbol{y}$ outside of the subspace indeed does not satisfy the condition $T^{-1}T(\boldsymbol{y}-\boldsymbol{o}) = \boldsymbol{y}-\boldsymbol{o}$.

**7.1.5 Convex Hull** If we take all the convex combinations of a set of vectors $X$, we get the convex hull of that set of vectors [12]. If $X$ is a finite set, then the convex hull will be a convex polytope. In this paper, we will only consider the convex hull of finite sets, so their convex hull will always be a convex polytope.

**Definition 36.** *The **convex hull** of a set of vectors $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d\} \subseteq \mathbb{R}^n$ is the set of all convex combinations:*

$$conv(X) = \left\{ \sum_{i=1}^{d} w_i \cdot \boldsymbol{x}_i \mid w_i \in \mathbb{R}, \forall i \in \{1,\ldots,d\} : w_i \geq 0, \sum_{i=1}^{d} w_i = 1 \right\}.$$

*Example 34.* The convex hull of the points $X = \{\langle 0,0\rangle, \langle 0.85,100\rangle, \langle 3.4,1120\rangle\}$ is shown in Figure 23. The points in $X$ correspond to the memoryless strategies in Figure 8.

A convex hull can also be described in a geometric way. In essence, we try to find the smallest convex polytope that contains all the given points. We describe these convex polytopes using hyperplanes at the boundary of the convex polytope, for which the entire convex polytope is in one half-space of each hyperplane [59,72]. This will allow us to determine whether a point is inside of the convex hull by checking whether the point is in the half-space of the convex hull for all of the hyperplanes.

**Definition 37.** *A **convex hull** in d-dimensional space can be described a tuple $\langle F, \boldsymbol{c}, \langle T, T^{-1}, \boldsymbol{o}, SC\rangle\rangle$, where:*

- *$F$ is the set of facets, where each facet is a tuple $\langle R, H\rangle$, with $R$ a set of $d$ ridges, where each ridge is a set of $d-1$ points in $\mathbb{R}^d$, and $H \in \mathcal{H}_d$ a hyperplane containing the ridges in $R$,*

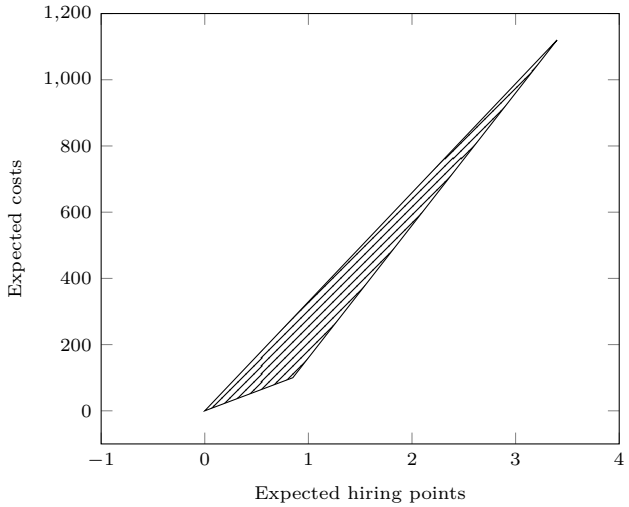



**Fig. 23.** Convex hull of $\{\langle 0, 0\rangle, \langle 0.85, 100\rangle, \langle 3.4, 1120\rangle\}$

**Alg. 6.** *ConvexHullContains*

**Input:** The convex hull description $C = \left\langle F, \boldsymbol{c}, \left\langle T, T^{-1}, \boldsymbol{o}, SC\right\rangle\right\rangle$, the point to check $\boldsymbol{x} \in \mathbb{R}^d$
**Result:** Whether $\boldsymbol{x}$ is inside the convex hull described by $C$

**if** $\left\langle T, T^{-1}, \boldsymbol{o}, SC\right\rangle \neq \bot$ **then**
  **return** $T^{-1}T(\boldsymbol{x} - \boldsymbol{o}) = \boldsymbol{x} - \boldsymbol{o} \wedge \mathit{ConvexHullContains}(SC, T(\boldsymbol{x} - \boldsymbol{o}))$
**return** $\forall \langle \boldsymbol{R}, \langle \boldsymbol{a}, \boldsymbol{b}\rangle\rangle \in F \colon \boldsymbol{a} \cdot \boldsymbol{x} \leq b$

- $\boldsymbol{c} \in \mathbb{R}^d$ *is a point inside of the convex hull,*
- $T$ *is the transformation to the subspace,*
- $T^{-1}$ *is the inverse transformation from the subspace,*
- $\boldsymbol{o}$ *the offset for the subspace transformation, and*
- $SC$ *is the convex hull in the subspace.*

This tuple description of a convex hull is more convenient in steps later on. Notice that to check whether a point is in the same half-space as the convex hull, we only need one point inside the convex hull, which is the point $\boldsymbol{c}$ in the description. Hence, if for all hyperplane describing a convex hull the point $\boldsymbol{c}$ is in the same half-space as some target point, the target point is inside the convex hull. We show the computation to check whether a point is inside a convex hull in Algorithm 6.

*Example 35.* In Figure 23 we have drawn a convex hull. Notice that if we use the tuple description for the same convex hull and draw the hyperplanes and point $\boldsymbol{c}$ inside of the convex hull, we obtain Figure 24. Notice that for all points inside of the convex hull, they are in the same half-space as the point $\boldsymbol{c}$ for each

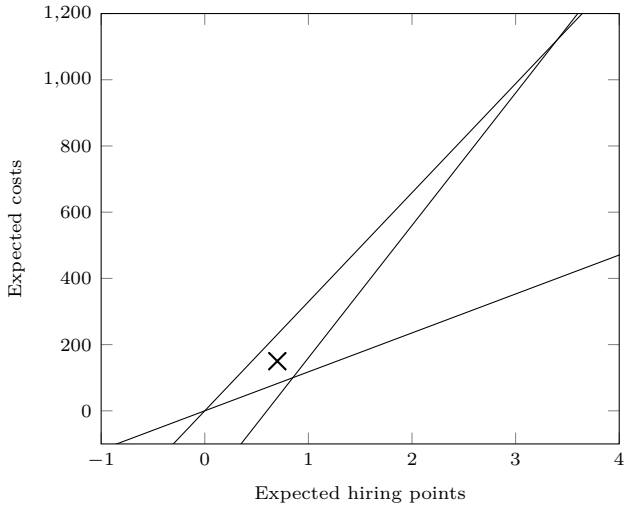



**Fig. 24.** Tuple description of Figure 23

hyperplane. For any point outside of the convex hull however, there is at least one hyperplane for which $\boldsymbol{c}$ is in the other half-space.

To find the tuple description of a convex hull, we will use the Quickhull algorithm [7]. We chose to choose this algorithm over alternatives [17,33,45,49] since Quickhull works in arbitrary many dimensions, is straightforward to implement, and it keeps track of the facets during the construction.

We adapt Quickhull to handle the degenerate case in which all the points are in an affine subspace. This degenerate case is often noted in literature, but usually no solution is given to handle this case. We use the approach of Section 7.1.4 to transform the points into a lower dimension. Moreover, since in our approach we iteratively add points to our convex hull, instead of knowing the points beforehand, we provide the way to add a point to a convex hull in Algorithm 7.

On the first line, we check whether the point is already inside of the convex hull. If this is the case, the convex hull does not grow and thus does not need to be modified. On the second line, we get all the points based on which the convex hull is constructed and store this in the set $X$. We then compute the subspace on line 3.

We use this to check whether all points lie in a lower-dimensional structure on line 4. If all the points in $X$ all lie in a lower dimensional structure, we create a convex hull in this dimension by adding all the points in $X$ on line 8. On line 9, we then check whether adding one more affine independent point will create a facet. If this is the case, we construct the ridges such that we can use them in the if block on line 12 when this point is added.

**Alg. 7.** Adapted Quickhull Algorithm - *ConvexHullAdd*

**Input:** The current convex hull description $C = \langle F, \boldsymbol{c}, \langle T, T^{-1}, \boldsymbol{o}, SC \rangle \rangle$, new point $\boldsymbol{y} \in \mathbb{R}^d$

**Result:** The convex hull description containing $\boldsymbol{y}$ and all points in $C$

**if** *ConvexHullContains*$(C, \boldsymbol{y})$ **then return** $C$
$X \leftarrow \bigcup \{ \bigcup R \mid \exists H\colon \langle R, H \rangle \in F \} \cup \{\boldsymbol{y}\}$
$T, T^{-1}, \boldsymbol{o} \leftarrow$ *ComputeSubspace*$(X)$
**if** $T.rows \neq T.columns$ **then**
  $F' \leftarrow \emptyset$
  $\boldsymbol{c'} \leftarrow \mathbf{0}_d$
  **foreach** $\boldsymbol{x} \in X$ **do**
    $\langle F', \boldsymbol{c'}, SS' \rangle \leftarrow$ *ConvexHullAdd*$(\langle F', \boldsymbol{c'}, \bot \rangle, T(\boldsymbol{x} - \boldsymbol{o}))$
  **if** $T.rows < d - 1$ **then return** $\langle \bot, \bot, \langle T, T^{-1}, \boldsymbol{o}, \langle F', \boldsymbol{c'}, \bot \rangle \rangle \rangle$
  $F'' \leftarrow \left\{ \left\langle \left\{ T^{-1}\boldsymbol{x'} + \boldsymbol{o} \mid \boldsymbol{x'} \in \bigcup \{ R \mid \exists H\colon \langle R, H \rangle \in F' \} \right\}, \bot \right\rangle \right\}$
  **return** $\langle F'', \boldsymbol{c'}, \langle T, T^{-1}, \boldsymbol{o}, \langle F', \boldsymbol{c'}, \bot \rangle \rangle \rangle$
**if** $\|F\| = 1$ **then**
  $\boldsymbol{c'} \leftarrow \sum_{\boldsymbol{x} \in X} \frac{\boldsymbol{x}}{\|X\|}$
  $F' \leftarrow \emptyset$
  **foreach** $\langle R, H \rangle \in F$ **do**
    **foreach** $r \in R$ **do**
      $R' \leftarrow \{\{\boldsymbol{y}\} \cup r \setminus \boldsymbol{z} \mid \boldsymbol{z} \in r \cup \{\boldsymbol{y}\}\}$
      $\langle \boldsymbol{a}, b \rangle \leftarrow$ *AffineHyperplane*$(r \cup \{\boldsymbol{y}\})$
      **if** $\boldsymbol{a} \cdot \boldsymbol{c'} > b$ **then** $\langle \boldsymbol{a}, b \rangle \leftarrow \langle -\boldsymbol{a}, -b \rangle$
      $F' \leftarrow F' \cup \{\langle R', \langle \boldsymbol{a}, b \rangle \rangle\}$
  **return** $\langle F', \boldsymbol{c'}, \bot \rangle$
$VF \leftarrow \{\langle R, \langle \boldsymbol{a}, b \rangle \rangle \in F \mid \boldsymbol{a} \cdot \boldsymbol{y} \geq b\}$
$F' \leftarrow F \setminus VF$
$HR \leftarrow \{R \mid \exists H\colon \langle R, H \rangle \in F'\} \cap \{R \mid \exists H\colon \langle R, H \rangle \in VF\}$
**foreach** $R \in HR$ **do**
  **foreach** $r \in R$ **do**
    $R' \leftarrow \{\{\boldsymbol{y}\} \cup r \setminus \boldsymbol{z} \mid \boldsymbol{z} \in r \cup \{\boldsymbol{y}\}\}$
    $\langle \boldsymbol{a}, b \rangle \leftarrow$ *AffineHyperplane*$(r \cup \{\boldsymbol{y}\})$
    **if** $\boldsymbol{a} \cdot \boldsymbol{c} > b$ **then** $\langle \boldsymbol{a}, b \rangle \leftarrow \langle -\boldsymbol{a}, -b \rangle$
    $F' \leftarrow F' \cup \{\langle R', \langle \boldsymbol{a}, b \rangle \rangle\}$
**return** $\langle F', \boldsymbol{c}, \bot \rangle$

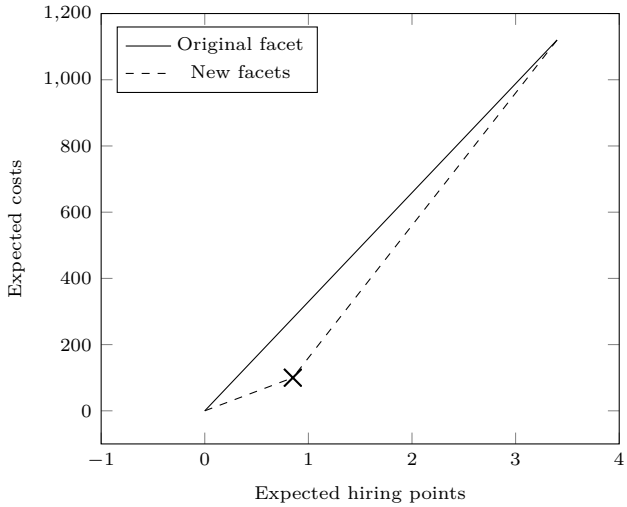



**Fig. 25.** From facet to convex polytope with a new point

When the condition on line 12 is satisfied, we know that the points do not lie in a subspace, since line 4 is not satisfied. Moreover, only one facet is constructed. This can only happen if line 12 has not been executed before, since this can only happen if line 10 has been performed and line 20 has not ran. From this facet, we will then construct a convex polygon by connecting the new point to the constructed facet as we visualise in Example 36. Since each ridge is a (d-2)-dimensional structure while the facet is a (d-1)-dimensional structure, each ridge excludes one of the points of the facet. This is done on line 17. On line 18 we then compute the hyperplane that includes the entire facet, using the procedure shown in Section 7.1.3. On line 19, we ensure that for each hyperplane, the points inside of the convex hull are in the negative half-space. Notice that this update is valid, since $\boldsymbol{a} \cdot \boldsymbol{x} = b$ accepts the same points $\boldsymbol{x}$ as the equation $-\boldsymbol{a} \cdot \boldsymbol{x} = -b$. This update is not strictly necessary, but simplifies the implementation, since we can ignore $\boldsymbol{c}$ to check whether a point is in the same half-space as $\boldsymbol{c}$.

*Example 36.* In Figure 25 we have drawn the facet between $\langle 0, 0\rangle$ and $\langle 3.4, 1120\rangle$. We then add the point $\langle 0.85, 100\rangle$ to construct two new facets, which together form a convex polytope.

If we already have a convex polygon, we reach line 22. In this case, we find all "visible facets", which are the facets that can "see" the new point $\boldsymbol{y}$. Seeing the point is equivalent to the point being in the other half-space of the hyperplane than the interior of the convex polygon. To verify this, we use the technique shown in Section 7.1.3. We store the new facets in the set $F'$, each facet that cannot see the new point will not be changed and is thus already added to $F'$ on line 23. We then search for the "horizon ridges", which are the ridges that lie on the horizon of being seen and not being seen. This means that the ridge

is part of a facet that can be seen but also of a facet that cannot be seen. From these ridges, we construct the facets that connect the new point $\boldsymbol{y}$ on lines 26 to 30. Notice that this construction is equivalent to lines 16 to 20.

*Example 37.* The convex hull of the points $\{\langle 0,0\rangle, \langle 0.85,100\rangle, \langle 3.4,1120\rangle\}$, is shown in Figure 26a. We now want to add the point $\langle 2,400\rangle$ which is not yet part of the convex hull as is clear from Figure 26b. Consequently we will need to grow it, the first step of this process is to determine the visible facets. Notice that only the facet that includes the points $\langle 0,0\rangle$ and $\langle 3.4,1120\rangle$ cannot see the point $\langle 2,0\rangle$ as shown in Figure 26c. We then determine the horizon ridges, which are points in 2 dimensions. We observe that the points $\langle 0,0\rangle$ and $\langle 3.4,1120\rangle$ are both part of visible and invisible facets. Therefore, they are the horizon ridges which we show in Figure 26d. Finally we construct the new ridges from the horizon ridges to the new point as shown in Figure 26e. This last construction is equivalent to the approach taken in Example 36.

**7.1.6 Downward Closure** As we have seen in Definition 31, all values dominated by an achievable value are achievable. To describe all achievable values, we will use the downward closure. For simplicity, we will only consider the case where we want to maximise each property. We have seen in Proposition 8 that we can transform any multi-objective property into a form where this is sufficient.

**Definition 38.** *The **downward closure** of a set of vectors $X = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d\} \subseteq \mathbb{R}^n$ is the set of vectors that are below or equal to an element of the convex hull of $X$:*

$$down(X) = \{\boldsymbol{y} \in \mathbb{R}^n \mid \exists \boldsymbol{x} \in conv(X) \colon \forall i \in \{1,\ldots,n\} : y_i \leq x_i\}.$$

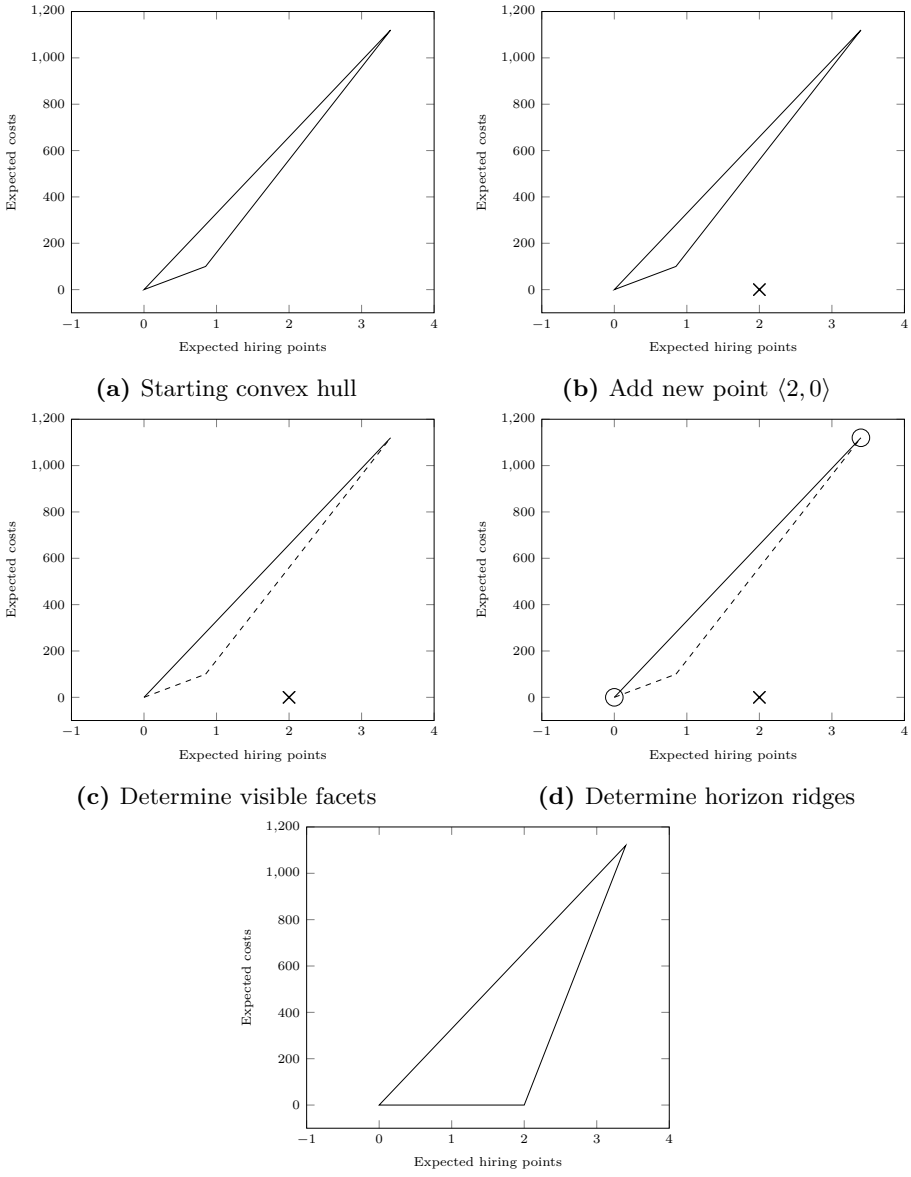
*Example 38.* In Figure 19e the under approximation is the downward closure of the points $\{\langle 0,0\rangle, \langle 0.85,-100\rangle, \langle 3.4,-1120\rangle\}$.

Convex hulls are quite well known, therefore it is typically easier to come by a convex hull algorithm than a downward closure algorithm. Fortunately, we can express a downward closure using a convex hull using Proposition 11. All we need to do is add $d$ points in $d$-dimensional space, so that each dimension is extended to negative infinity.

**Proposition 11.** *The downward closure of a set $X \in \mathcal{P}(\mathbb{R}^d)$ is the set of convex combinations of:*

$$\hat{X} = X \cup \left\{ \left\langle -\infty \cdot [i=j] + (\min_{\boldsymbol{x} \in X} x_i) \cdot [i \neq j] \right\rangle_{i \in \{1,\ldots,d\}} \mid j \in \{1,\ldots,d\} \right\}.$$

*Proof. In order to prove that the downward closure of $X$ is equivalent to the set of convex combinations of $\hat{X}$, we will first prove that $down(X) \subseteq conv(\hat{X})$ and then $conv(\hat{X}) \subseteq down(X)$.*




**(a)** Starting convex hull

**(b)** Add new point $\langle 2, 0 \rangle$

**(c)** Determine visible facets

**(d)** Determine horizon ridges

**(e)** Construct new facets

**Fig. 26.** Convex hull construction example

- *By the definition of a downward closure, we have that for any* $\boldsymbol{y} \in \mathit{down}(X)$, $\exists \boldsymbol{x} \in \mathit{conv}(X) \colon \forall j \in \{1, \ldots, n\} : y_j \leq x_j$. *This means that for an arbitrary number of dimensions, the value of* $\boldsymbol{y}$ *is smaller than* $\boldsymbol{x}$. *For each such dimension* $j$, *we can take the convex combination of* $\boldsymbol{x}$ *and an infinitely small fraction of the vectors* $\boldsymbol{e_j} = \langle -\infty \cdot [i = j] + (\min_{\boldsymbol{x} \in X} x_i) \cdot [i \neq j] \rangle_{i \in \{1,\ldots,d\}}$ *to obtain* $\boldsymbol{y}$. *Since* $\boldsymbol{x}$ *and all* $\boldsymbol{e_j}$ *are in* $\hat{X}$, $\boldsymbol{y}$ *is also present in* $\mathit{conv}(\hat{X})$. *Since the choice of* $\boldsymbol{y}$ *was arbitrary, we obtain* $\mathit{down}(X) \subseteq \mathit{conv}(\hat{X})$.
- *Each element* $\boldsymbol{x} \in \mathit{conv}(\hat{X})$ *is a convex combination. We modify the weights of this combination so that the weights for all vectors in* $\hat{X} \setminus X$ *are set to* 0 *and the sum of their original weights is added to the weight of one of the vectors in* $X$ *to obtain the convex combination* $\boldsymbol{y}$. *Since the values in* $\hat{X} \setminus X$ *are lower bounds on the values in* $X$, *we conclude that* $\forall i \in \{1, \ldots, n\} : x_i \leq y_i$. *Moreover, since only the vectors in* $X$ *are used in the convex combination of* $\boldsymbol{y}$, *we get* $\boldsymbol{y} \in \mathit{conv}(X)$. *Combining these two facts, we get* $\boldsymbol{x} \in \mathit{down}(X)$ *by the definition of the downward closure. Since the choice of* $\boldsymbol{x}$ *was arbitrary, we obtain* $\mathit{conv}(\hat{X}) \subseteq \mathit{down}(X)$.

*Since* $\mathit{down}(X) \subseteq \mathit{conv}(\hat{X})$ *and* $\mathit{conv}(\hat{X}) \subseteq \mathit{down}(X)$, *we infer that* $\mathit{down}(X) = \mathit{conv}(\hat{X})$. □

**7.1.7 Pareto Query** The first type of query that we consider is the Pareto query. This is a strongly altered version of the one shown in [30]. In the original paper, only the 2-dimensional case is considered, however, we want to support more than two unknowns. We do only consider maximising cumulative rewards. Recall that we can turn probabilistic reachability properties into cumulative rewards using Proposition 7. Moreover, we can turn *min* into *max* targets as shown in Proposition 8.

The intuition behind this approach is shown in Section 7.1.1. We aim to generate the entire Pareto curve with this algorithm. The adapted algorithm is shown in Algorithm 8. The set $X$ contains boundary points of the achievable values. The set $Y$ contains a queue of facets for which we still need to check whether there is an achievable point on the outside of the facet.

As shown in Section 7.1.1, we first construct the initial facet by finding the maximum value for each dimension. We then try to expand the convex polygon of achievable solutions by finding the point that is the furthest away from that facet. If there exists a point outside the facet, we can grow the convex polygon to contain the point. We can then construct $d$ new facets by including this new point. We then check for all of these facets whether they can still be grown. Notice that this construction is extremely similar to the Quickhull algorithm [7], however, we do not know the points in advance.

On line 3, we construct the weight vector that will maximise for the $i$th dimension. We then compute the achievable point $\boldsymbol{q}$ obtains the maximum distance between the hyperplane $\langle \boldsymbol{w}, 0 \rangle$ and the rewards on lines 4 and 5. This computation is discussed in Section 7.1.8. Unfortunately, we might then still end up with a degenerate point cloud. For example, we might have a Pareto property

**Alg. 8.** Pareto queries

**Input:** MDP $\mathcal{M}$, Pareto property $\tilde{\boldsymbol{\phi}} = \langle [\mathit{CumRew}_{\mathcal{M}}(\rho_1)]^{\leq k_1}_{max}, \ldots, [\mathit{CumRew}_{\mathcal{M}}(\rho_d)]^{\leq k_d}_{max} \rangle$
**Result:** Finite set $X \subset \mathbb{R}^d$, such that $\mathit{down}(X) = \mathit{Ach}_{\mathcal{M}}(\tilde{\boldsymbol{\phi}})$

```
1  X ← ∅
2  foreach i ∈ {1, ..., d} do
3  |  w ← ⟨[i = j]⟩_{j∈{1,...,d}}
4  |  Find strategy σ_Ps maximising Σ_{j=1}^{d} w_j · CumRew_M(ρ_j)(k_j, σ_Ps)
5  |  q ← ⟨CumRew_M(ρ_j)(k_j, σ_Ps)⟩_{j∈{1,...,d}} − 1
6  |  if q ∈ X then
7  |  |  q_i ← −|q_i| − Σ_{x∈X} |x_i|;
8  |  X ← X ∪ {q}
9  Y ← {X}
10 while ∃F ∈ Y do
11 |  Y ← Y \ F
12 |  Find w such that ∀x_1, x_2 ∈ F: w · x_1 = w · x_2, Σ_{i∈{1,...,d}} w_i = 1
13 |  if w ≠ ⊥ then
14 |  |  Find strategy σ_Ps maximising Σ_{i=1}^{d} w_j · CumRew_M(ρ_i)(k_i, σ_Ps)
15 |  |  q ← ⟨CumRew_M(ρ_i)(k_i, σ_Ps)⟩_{i∈{1,...,d}}
16 |  |  if q ∉ down(X) then
17 |  |  |  X ← X ∪ {q}
18 |  |  |  foreach x ∈ F do
19 |  |  |  |  Y ← Y ∪ ({q} ∪ F \ {x})
20 return X
```

with two exactly the same subproperties. In this case, the points that maximise for these two points might be equivalent. To prevent this from happening, we include lines 6 and 7, which compute an achievable point, which is not a convex combination of any of the other vectors. This point is achievable since the updated point is dominated by the point before the update.

Since we do not know the points in advance, we need a different way to handle degenerate convex hulls. It is important for the algorithm to work that a facet is actually a facet and not a lower-dimensional structure. To ensure that the initial facet is not a lower-dimensional structure, we use line 6 and 7. If we want to maximise two different dimensions and end up with the same point, we obviously lose one dimension, since we need at least $d$ points to describe a $(d-1)$-dimensional structure. For example, we need 2 points to describe a line. Fortunately, if all points are unique, because we are optimising different dimensions, the points are independent. If two points are the same, we alter the dimension that it was maximising, such that the value is still achievable, yet not Pareto optimal. Since it is equivalent to another point, this newly constructed point is still maximal for the other dimension for which this point was generated.

We then loop over the facets to check whether there are points on their outside. On line 12 we determine the weight vector that is the normal vector of the facet $F$, which can be computed using the techniques of Section 7.1.3. The summation does require that all its dimensions sum up to one. This can be easily com-

puted from the hyperplane $\langle \boldsymbol{a}, b \rangle$ in $d$ dimensions as $\boldsymbol{w} = \left\langle \frac{a_i}{\sum_{j=1}^{d} a_j} \right\rangle_{i \in \{1,\ldots,d\}}$. On line 13, we continue if the points in $F$ are not actually a facet, but all lie in an affine subspace, which might happen due to line 19.

We then compute the point with the maximum distance from the hyperplane through all points in $\boldsymbol{q}$, which is $\boldsymbol{q}$ using the techniques of Section 7.1.8 on lines 14 and 15. On line 16, we check whether this point was not already achievable. This computation is discussed in Section 7.1.6. If $\boldsymbol{q}$ was outside of the previous under approximation, we update the under approximation on line 17. We then construct the new candidate facets on lines 18 and 19 in the same way that we construct new facets from a new point used in the Quickhull algorithm discussed in Section 7.1.5.

**7.1.8 Value Iteration Algorithm** One of the main contributions of [30] is its value iteration implementation. The value iteration part of the paper computes a strategy that maximises the dot product between a given weight vector $\boldsymbol{w}$ and the rewards with the corresponding step-bounds for a given property. This is equivalent to finding the achievable point with the maximum distance from the hyperplane $\langle \boldsymbol{w}, 0 \rangle$. We show our adapted version in Algorithm 9. We have only included an absolute convergence threshold, but, just as in Section 2.7 a relative convergence threshold might be used instead.

In the first loop, we compute the strategy that maximises the dot product between the weight vector and the rewards, for subproperties with an infinite step-bound. This corresponds to the reward for taking infinitely many steps. Notice that this computation is extremely similar to its simple single-objective counterpart shown in Algorithm 1. In each iteration, we take the action for which the expected reward is the highest. We do this by computing for each branch the sum of the expected sum in the resulting state and the reward for taking the branch on line 6. On line 7, we compute the action that was taken on line 6, notice that since the summations are equal, these two values can be computed alongside at the same time in an implementation. We exit the loop at line 11 if the error is small enough. The type of value iteration is in this loop is called Jacobi value iteration [66]. For users interested in optimising the run time, the Gauss-Seidel method [67] might be considered. However, covering this difference is beyond the scope of this paper.

In the second loop, we then compute the value for each individual reward-structure with an infinite step-bound if the computed strategy is being used. This is done on line 13, where again the equation is very similar to line 6, with the exception that it does check all actions but only uses the previously computed action. If all step-bounds are infinite, we are now done, otherwise we will still need to enter the third loop in order to compute the values for the subproperties with a finite step-bound.

In the third loop, we essentially perform the single-objective approach of Section 2.7 again on lines 19 and 20. Notice that they are almost equivalent to lines 6 and 7. The only difference is that we now also include the properties with

**Alg. 9.** Value iteration algorithm

**Input:** MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$, weight vector $\boldsymbol{w} = \langle w_1, w_2, \cdots, w_d \rangle$, multi-objective achievability property $\left\langle [\mathit{CumRew}_{\mathcal{M}}(\rho_1)]^{\leq k_1}_{\geq c_1}, \ldots, [\mathit{CumRew}_{\mathcal{M}}(\rho_d)]^{\leq k_d}_{\geq c_d} \right\rangle$, convergence threshold $\epsilon \in [0,1]$

**Result:** Step-positional strategy $\sigma_{\mathcal{S}}$ maximising $\sum_{i=1}^{d} w_i \cdot \mathit{CumRew}_{\mathcal{M}}(\rho_i)(k_i, \sigma_{\mathcal{S}})$ and $\boldsymbol{q} = \langle \mathit{CumRew}_{\mathcal{M}}(\rho_i)(k_i, \sigma_{\mathcal{S}}) \rangle_{i \in \{1,\ldots,d\}}$

$\boldsymbol{x} \leftarrow \boldsymbol{y} \leftarrow \boldsymbol{0}_{\|\mathcal{S}\|}$
**foreach** $i \in \{1, \ldots, d\}$ **do** $\boldsymbol{x}^i \leftarrow \boldsymbol{y}^i \leftarrow \boldsymbol{0}_{\|\mathcal{S}\|}$
**foreach** $s \in \mathcal{S}$ **do** $\sigma_{\mathcal{S}}(s, \infty) \leftarrow \bot$
**do**
  **foreach** $s \in \mathcal{S}$ **do**
    $y_s \leftarrow \max_{a \in \alpha_{\mathcal{M}}(s)} \left( \sum_{s' \in \mathcal{S}} \delta(s)(a)(s') \cdot (x_{s'} + \sum_{\{i \mid k_i = \infty\}} w_i \rho_i(s, a, s')) \right)$
    $\sigma_{\mathcal{S}}(s, \infty) \leftarrow \arg\max_{a \in \alpha_{\mathcal{M}}(s)} \left( \sum_{s' \in \mathcal{S}} \delta(s)(a)(s') \cdot (x_{s'} + \sum_{\{i \mid k_i = \infty\}} w_i \rho_i(s, a, s')) \right)$
  $\Delta \leftarrow \max_{s \in \mathcal{S}} |y_s - x_s|$ $\boldsymbol{x} \leftarrow \boldsymbol{y}$
**while** $\Delta > \epsilon$
**do**
  **foreach** $s \in \mathcal{S}$ **do**
    **foreach** $\{i \mid k_i = \infty\}$ **do**
      $y^i_s \leftarrow \left( \sum_{s' \in \mathcal{S}} \delta(s)(\sigma_{\mathcal{S}}(s', \infty))(s') \cdot (x^i_{s'} + \sum_{\{i \mid k_i = \infty\}} w_i \rho_i(s, \sigma_{\mathcal{S}}(s, \infty), s')) \right)$
  $\Delta \leftarrow \max_{i \in \{1,\ldots,d\}} \max_{s \in \mathcal{S}} |y^i_s - x^i_s|$
  **foreach** $i \in \{1, \ldots, d\}$ **do** $\boldsymbol{x}^i \leftarrow \boldsymbol{y}^i$
**while** $\Delta > \epsilon$
**for** $l = \max \{k_i \mid i \in \{1, \ldots, d\}, k_i \neq \infty\}$ *down to* 1 **do**
  **foreach** $s \in S$ **do**
    $y_s \leftarrow \max_{a \in \alpha_{\mathcal{M}}(s)} \left( \sum_{s' \in S} \delta(s)(a)(s') \cdot (x_{s'} + \sum_{\{i \mid k_i \geq l\}} w_i \rho_i(s, a, s')) \right)$
    $\sigma_{\mathcal{S}}(s, l) \leftarrow \arg\max_{a \in \alpha_{\mathcal{M}}(s)} \left( \sum_{s' \in S} \delta(s)(a)(s') \cdot (x_{s'} + \sum_{\{i \mid k_i \geq l\}} w_i \rho_i(s, a, s')) \right)$
    **foreach** $\{i \in \{1, \ldots, d\} \mid k_i \geq j\}$ **do**
      $y^i_s \leftarrow \sum_{s' \in S} \delta(s)(\sigma_{\mathcal{S}}(s, l))(s') \cdot (x_{s'} + \rho_i(s, \sigma_{\mathcal{S}}(s, l), s'))$
  **foreach** $i \in \{1, \ldots, d\}$ **do** $\boldsymbol{x^i} \leftarrow \boldsymbol{y^i}$
**foreach** $i \in \{1, \ldots, d\}$ **do** $q_i \leftarrow y^i_{s_I}$
**return** $\sigma_{\mathcal{S}}$, $\boldsymbol{q}$

**Alg. 10.** Multi-objective achievability query

**Input:** MDP $\mathcal{M}$, multi-objective achievability property
$\bar{\phi} = \left\langle [CumRew_{\mathcal{M}}(\rho_1)]^{\leq k_1}_{\geq c_1}, \ldots, [CumRew_{\mathcal{M}}(\rho_d)]^{\leq k_d}_{\geq c_d} \right\rangle$
**Result:** $query_{\mathcal{M}}(\bar{\phi})$

1 $X \leftarrow \emptyset, \boldsymbol{c} = \langle c_1, \ldots, c_d \rangle$
2 **do**
3 Find $\boldsymbol{w}$ separating $\boldsymbol{c}$ from $down(X)$
4 Find strategy $\sigma_{Ps}$ maximising $\sum_{i=1}^{d} w_i \cdot CumRew_{\mathcal{M}}(\rho_i)(k_i, \sigma_{Ps})$
5 $\boldsymbol{q} \leftarrow \langle CumRew_{\mathcal{M}}(\rho_i)(k_i, \sigma_{Ps}) \rangle_{i \in \{1,\ldots,d\}}$
6 **if** $\boldsymbol{w} \cdot \boldsymbol{q} < \boldsymbol{w} \cdot \boldsymbol{c}$ **then return** false
7 $X \leftarrow X \cup \{\boldsymbol{q}\}$
8 **while** $\boldsymbol{c} \notin down(X)$
9 **return** true

a finite step-bound of at least $l$. Since this will be true exactly $k_i$ times for a subproperty at index $i$ with a finite step-bounds, we include the $i$th subproperty only $k_i$ times, as intended. On line 22, we perform the equivalent of line 13, but now we also include the subproperties with a finite step-bound. This is the computation of the value for the individual subproperty.

**7.1.9 Multi-Objective Achievability Query** To compute an achievability query we could compute the entire Pareto curve and determine whether the target point is contained in the downward closure of the Pareto curve. However, it turns out that we can take some shortcuts that significantly speed up the computation [30]. We show the algorithm in Algorithm 10.

On the first line, we set $\boldsymbol{c}$ as the point for which we want to verify whether it is achievable. On line 3 we find a new operation. Here, we search for any hyperplane for which the downward closure of $X$, which is under approximation discussed in Section 7.1.1, is in one half-space from the hyperplane, while the point $\boldsymbol{c}$ is in the other. This differs from the approach of Section 7.1.7 since we can discard any facet for which the point are in the same half-space as the under approximation, since this can never grow the under approximation closer to $\boldsymbol{c}$. This significantly speeds up the algorithm. We discuss this computation in Section 7.1.10.

On lines 4 and 5 we use the approach from Section 7.1.8 again to compute the point $\boldsymbol{q}$ that has the largest distance to the hyperplane $\langle \boldsymbol{w}, 0 \rangle$. On line 6 we find another way to exit early. We check whether the distance from $\boldsymbol{c}$ to the hyperplane $\langle \boldsymbol{w}, 0 \rangle$ is larger than the maximum distance for any achievable point, which was obtained by $\boldsymbol{q}$. If $\boldsymbol{c}$ is farther away than the furthest achievable point, it must be impossible to achieve. This is equivalent to checking whether the $\boldsymbol{c}$ point is outside the upper approximation. The distance between a hyperplane and a point is discussed in Section 7.1.3.

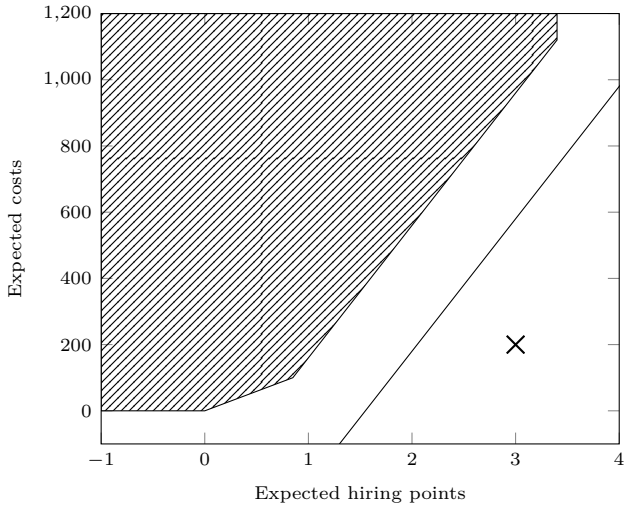



**Fig. 27.** Separating hyperplane for the point $\langle 3, 200 \rangle$

On line 8 we perform another optimisation. If we know that the point $\boldsymbol{c}$ is in the downward closure of $X$, we know that $\boldsymbol{c}$ is in the under approximation of the Pareto curve. If it is, then it must mean that the point is achievable, so we can exit. This computation is discussed in Section 7.1.6.

**7.1.10 Separating Hyperplane** If we have a point that is not inside a convex hull, we can always find a hyperplane that separates the point from the convex hull according to the separating hyperplane theorem [68]. This is equivalent to finding a hyperplane for which the entire convex hull is in one half-space, while the point is in the other.

*Example 39.* The point $\langle 3, 200 \rangle$ can be separated from the Pareto curve for the hiring points, using the hyperplane $\langle \langle 400, -1 \rangle, 620 \rangle$, as shown in Figure 27.

In some cases, a separating hyperplane is used as the weight vector as demonstrated in Section 7.1.1. While we can start with the standard basis vectors $\{[i = j]_{\{1,\ldots,d\}} \mid i \in \{1, \ldots, d\}\}$, we want one that represents a facet of the downward closure after that. Therefore, we choose the hyperplane with the maximum distance to the point. This seems to have the best results in practice [30]. We can find this hyperplane, since in Section 7.1.6 we have shown how to compute a downward closure using a convex hull description, which contains the hyperplanes of the facets.

Furthermore, we even know that this hyperplane has a normal vector which consists only of non-negative values [30]. We can scale these values such that the sum of the values is equal to 1. This ensures that the scaled normal vector can be used as the weight vector for a convex combination as done in Section 7.1.1.

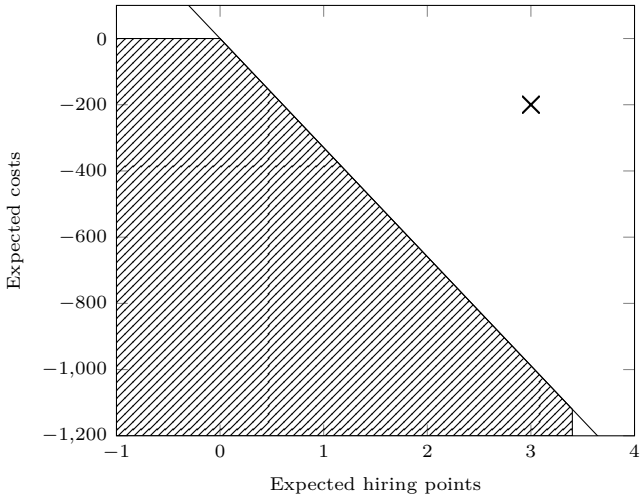



**Fig. 28.** Separating hyperplane for the point $\langle 3, -200 \rangle$

In the context of the downward closure, we will consider such a scaled normal vector, a separating vector.

*Example 40.* The point $\langle 3, -200 \rangle$ can be separated from the Pareto curve for the hiring points, using the hyperplane $\langle \langle 1120, 3.4 \rangle, 0 \rangle$, which is the hyperplane through the boundary points $\langle 0, 0 \rangle$ and $\langle 3.4, -1120 \rangle$, as shown in Figure 28. We can scale this normal vector to $\left\langle \frac{1120}{1120+3.4}, \frac{3.4}{1120+3.4} \right\rangle$, so that its values sum up to one. We use this as the separating vector of the point $\langle 3, -200 \rangle$ from the downward closure.

**7.1.11 Multi-Objective Numerical Query** The approach for multi-objective numerical queries is very similar to the approach for multi-objective achievability queries shown in Section 7.1.9. We can take fairly similar shortcuts to avoid having to compute the entire Pareto curve. The procedure is shown in Algorithm 11.

We start with the most pessimistic target point $\boldsymbol{c}$ on line 1 and iteratively increase it throughout the algorithm. Therefore, we take the lowest possible value for the reward in the unknown position. This value can be computed using any technique for model checking one objective [5].

On line 3, we again find a separating hyperplane. We do this using the approach of Section 7.1.10. It is now important that $w_j > 0$, since we always need to select a point $\boldsymbol{q}$ such that there does not exist another point $\hat{\boldsymbol{q}}$, such that $\forall i \in \{1, \ldots, d\}\, \hat{q}_i \geq q_i$ and $\hat{q}_j > q_j$ [30]. For example, if we have the multi-objective numerical property

$$\left\langle [CumRew_{\mathcal{M}}(\rho_1)]^{\leq \infty}_{\leq max}, [CumRew_{\mathcal{M}}(\rho_2)]^{\leq \infty}_{\geq 1} \right\rangle,$$

**Alg. 11.** Multi-objective numerical query

**Input:** MDP $\mathcal{M}$, multi-objective numerical property $\hat{\boldsymbol{\phi}} = \left\langle [CumRew_{\mathcal{M}}(\rho_1)]^{\leq k_1}_{\geq c_1}, \ldots, [CumRew_{\mathcal{M}}(\rho_j)]^{\leq k_j}_{max}, \ldots, [CumRew_{\mathcal{M}}(\rho_d)]^{\leq k_d}_{\geq c_d} \right\rangle$ with the unknown in position $j \in \{1, \ldots, d\}$

**Result:** $query_{\mathcal{M}}(\hat{\boldsymbol{\phi}})$

1 $X \leftarrow \emptyset, \boldsymbol{c} = \left\langle c_1, \ldots, query_{\mathcal{M}}([CumRew_{\mathcal{M}}(\rho_j)]^{\leq k_j}_{min}), \ldots, c_d \right\rangle$
2 **do**
3 Find $\boldsymbol{w}$ separating $\boldsymbol{c}$ from $down(X), w_j > 0$
4 Find strategy $\sigma_{Ps}$ maximising $\sum_{i=1}^{d} w_i \cdot CumRew_{\mathcal{M}}(\rho_i)(k_i, \sigma_{Ps})$
5 $\boldsymbol{q} \leftarrow \langle CumRew_{\mathcal{M}}(\rho_i)(k_i, \sigma_{Ps}) \rangle_{i \in \{1,\ldots,d\}}$
6 **if** $\boldsymbol{w} \cdot \boldsymbol{q} < \boldsymbol{w} \cdot \boldsymbol{c}$ **then return** $\perp$
7 $X \leftarrow X \cup \{\boldsymbol{q}\}$
8 $c_j \leftarrow \max\{c_j, \max\{\bar{c_j} \in \mathbb{R} \mid \langle c_1, \ldots, \bar{c_j}, \cdots, c_d \rangle \in down(X)\}\}$
9 **while** $\boldsymbol{c} \notin down(X) \vee \boldsymbol{w} \cdot \boldsymbol{q} > \boldsymbol{w} \cdot \boldsymbol{c}$
10 **return** $c_j$

we would never want to have $\boldsymbol{q} = \langle 0, 2 \rangle$, if there exists a strategy such that $\hat{\boldsymbol{q}} = \langle 1, 2 \rangle$. We can achieve this by adding a small value to $w_i$ if it is 0. Another way to implement this without the $w_j > 0$ constraint is covered in Section 8.3.

We then continue with lines 4 and 5 by computing the point $\boldsymbol{q}$ with the maximum distance to the hyperplane $\langle \boldsymbol{w}, 0 \rangle$ with the calculation shown in Section 7.1.8. On line 6, we check whether the point $\boldsymbol{c}$ is farther away from this hyperplane than the furthest point $\boldsymbol{q}$. Since $c_j$ is always an under approximation, if $\boldsymbol{c}$ is farther away than the furthest achievable point $\boldsymbol{q}$, must mean that the property is unachievable. This is equivalent to checking whether the point $\boldsymbol{c}$ is outside of the upper approximation. On line 7 we update the under approximation of the Pareto curve. On line 8, we find the optimal value for $c_j$ in the downward closure of our known points, given that all constraints need to be satisfied. This is the maximum point on the current Pareto curve under approximation, where all constraints are satisfied. Notice that this point might not (yet) exist. We do this using the approach of Section 7.1.12. This is now the best point we can now achieve.

We stop the algorithm on line 9 only if $\boldsymbol{c}$ is part of the under approximation and if it is on the edge of the upper approximation. Checking whether the point is part of the under approximation is important, since before we exit, we need to ensure that the point $\boldsymbol{c}$ is actually achievable. If it is not part of the under approximation, we are not yet sure that it can be achieved. Checking whether the point is part of the under-approximation is done by checking whether the point is part of the downward closure of $X$, using the techniques of Section 7.1.6. If the point is part of the downward closure and is on the edge of the upper approximation, we are finished. We cannot do any better than achieving a point on the upper approximation. Checking whether the point is part of the upper

**Alg. 12.** Maximal Point of a Convex Hull - *MaxConv*

**Input:** The convex hull $C = \langle F, \boldsymbol{c}, \langle T, T^{-1}, \boldsymbol{o}, SC\rangle\rangle$, the constraints $\boldsymbol{c} = \langle c_1, \ldots, c_d\rangle \in \mathbb{R}^d$ and the dimension to maximise $j \in \{1, \ldots, d\}$
**Result:** $\max\{\bar{c_j} \in \mathbb{R} \mid \langle c_1, \ldots, \bar{c_j}, \cdots, c_d\rangle \in C\}$

1 **if** $\langle T, T^{-1}, \boldsymbol{o}, SC\rangle = \bot$ **then**
2 $\quad r \leftarrow \bot$
3 $\quad$ **foreach** $\langle R, \langle \boldsymbol{a}, b\rangle\rangle \in C$ **do**
4 $\quad\quad c_j \leftarrow \frac{b - \sum_{i \in \{1,\ldots,d\}\setminus\{j\}} a_i \cdot c_i}{a_j}$
5 $\quad\quad$ **if** *ConvexHullContains*$(C, \boldsymbol{c})$ **then** $r \leftarrow c_j$
6 $\quad$ **return** $r$
7 **else if** $\exists j' \colon T_{j',j} = 1$ **then**
8 $\quad$ **if** $T^{-1}T(\boldsymbol{c} - \boldsymbol{o}) \neq \boldsymbol{c} - \boldsymbol{o}$ **then return** $\bot$
9 $\quad$ **return** *MaxConv*$(SC, T(\boldsymbol{c} - \boldsymbol{o}), j')$
10 **else**
11 $\quad \boldsymbol{c}' \leftarrow T^{-1}T(\boldsymbol{c} - \boldsymbol{o}) + \boldsymbol{o}$
12 $\quad c_j \leftarrow c'_j$
13 $\quad$ **if** $\boldsymbol{c} = \boldsymbol{c}' \wedge$ *ConvexHullContains*$(SC, \boldsymbol{c})$ **then return** $c_j$
14 $\quad$ **else return** $\bot$

approximation is done by checking the distance to the hyperplane $\langle \boldsymbol{w}, 0\rangle$ as discussed in Section 7.1.3.

**7.1.12 Maximal Downward Closure** As we have seen in Section 7.1.11, we need to find the maximum value of an unknown on the under approximation given a set of constraints. This means that we need to find the maximum value in a downward closure, with one dimension being unknown, while the others are fixed. Formally, for a convex hull description $C = \langle F, \boldsymbol{c}, \langle T, T^{-1}, \boldsymbol{o}, SC\rangle\rangle$, a point $\langle c_1, \ldots, c_d\rangle \in \mathbb{R}^d$ and the unknown dimension $j \in \{1, \ldots, d\}$, we want to find $\max\{\bar{c_j} \in \mathbb{R} \mid \langle c_1, \ldots, \bar{c_j}, \cdots, c_d\rangle \in C\}$.

To do this, we observe that any point with one unknown and a set of constraints must lay on a line, where the unknown is the only variable. Moreover, since we are looking for a point on the edge of the convex hull, the maximal point, if it exists, must be at the intersection of one of the facets of the downward closure and this line. For each such intersection, we must verify whether the intersection is part of the convex hull described by $C$. The result is the valid intersection with the highest value for $\bar{c_j}$. We show these steps in Algorithm 12.

On the first line of this algorithm, we check whether the convex hull is not in a subspace. If not, we continue to line 2, where $r$ will store the result. We loop over all facets and compute the intersection of the hyperplane and on line 4. This computation works, because each point $\boldsymbol{x}$ on the hyperplane must satisfy the equation $\boldsymbol{a} \cdot \boldsymbol{x} = b$ as shown in Section 7.1.3. Since we have only one unknown

in $\boldsymbol{c}$, we can rewrite this formula to obtain $c_j$:

$$\begin{aligned} \boldsymbol{a} \cdot \boldsymbol{c} &= b \\ \sum_{i=1}^{d} a_i \cdot c_i &= b \\ a_j \cdot c_j &= b - \sum_{i \in \{1,\ldots,d\} \setminus \{j\}} a_i \cdot c_i \\ c_j &= \frac{b - \sum_{i \in \{1,\ldots,d\} \setminus \{j\}} a_i \cdot c_i}{a_j} \end{aligned}$$

Notice that we ensure that $a_j \neq 0$ when this computation is reached from Algorithm 11, because of the $w_j > 0$ constraint on line 3 of that algorithm. On line 5 we then verify whether the computed point $\boldsymbol{c}$ is in the convex hull described by $C$, this computation is shown in Section 7.1.5. We then return the best value for $c_j$ on line 6.

If the entire convex hull described by $C$ lies in a subspace, we do not have a polytope in $d$ dimensions. Therefore, we use a different approach. In this case, the line describing all potential values for $\boldsymbol{c}$ is a line in $d$-dimensional space, but in the subspace it might not be. It could be that the entire line is only a point in the subspace, or it could be that it does not even pass the subspace. We start on line 7 by checking whether the unknown dimension is a free dimension. If it is, we know that the line must either be a line in the subspace as well, or it is not in the subspace at all. Because of this, we can pick any point on the line and check whether it is in the subspace, which is done on line 8 using the theory of Section 7.1.4. If the point is in the subspace, the entire line and we can return the maximum point for the convex hull in the subspace. Otherwise, the line does not intersect the subspace, so there is no solution.

If the unknown is a fixed dimension, there is no way in the subspace for the variable on the line to change, hence the line must be a point in the subspace or not be in the subspace at all. On lines 11 to 13 we compute whether the point representing the line is in fact part of the subspace using the technique shown in Section 7.1.4. If the point is in the subspace, the point is the maximal point for this convex hull if and only if it is contained in the convex hull.

## 7.2 Linear Programming Approach

The other prevalent way to implement multi-objective queries is by using the linear programming approach from [29]. This approach should be easier to understand for most readers, since the complexity of the analysis is hidden behind a linear optimisation solver. Since we assume the linear optimisation solver to be a library, we will not cover its internals. For this approach, we only need to be able to express the MDP and the constraints of the multi-objective property as constraints in the linear program.

In case of a multi-objective numerical property, we can let the linear program optimise for its unknown. In case of a multi-objective achievability property, we

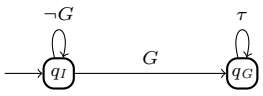



**Fig. 29.** Deterministic Rabin automaton for $\Diamond G$

do not need to optimise for anything, we merely need to verify whether the linear program is feasible.

Notice that we can optimise for only one linear function in a linear program. Therefore, the approach from [29] does not cover Pareto properties. However, it is straightforward to extend the approach taken using a multi-objective linear optimisation solver by optimising for all unknowns [11,25]. These are for example included in MATLAB[13] and ALGLIB[14]. Another possibility would be to use linear programming instead of the value iteration approach shown in Algorithm 9 as part of Algorithms 8, 10 and 11.

**7.2.1 Construct MDP** So far, we have used Proposition 7 to transform the MDP to query probabilistic reachability properties. The algorithm for linear programming however, uses a more general approach that supports all LTL properties instead of just the reachability of a goal set as we have used so far. Unfortunately, the Modest Toolset, in which we implement this algorithm, does not support arbitrary LTL specifications right now. Only a subset is supported. Since allowing the Modest Toolset to handle arbitrary LTL specifications would require a lot of work in an area that is not the focus of this paper, we stick with the reachability of a set of states. We will do so using its LTL equivalent to demonstrate how LTL properties can be handled.

The reachability of a set of states, corresponds to the LTL property $\Diamond$. If we want to eventually reach any state in $G$, this can be described in LTL as $\Diamond G$. The DRA corresponding to this LTL property is $\mathcal{R} = \langle \{q_I, q_G\}, q_I, \mathcal{A}, \lambda, \langle \emptyset, \{q_G\} \rangle \rangle$ shown in Figure 29. If we now take the approach from Section 2.10, we can compose the DRA with the MDP $\mathcal{M}$ on which we want to evaluate the property. The MDP which embeds the LTL property is then the composition $\mathcal{M} \otimes \mathcal{R}$. Notice that this MDP is isomorphic to the MDP resulting from Proposition 7. The only difference is the name of the states, each *false* is replaced by $q_I$ and each *true* by $q_G$.

*Example 41.* When we compose the MDP for the certification process model from Figure 2 with the DRA for $\Diamond \{\textit{finished}\}$ as shown in Figure 29, we obtain the MDP shown in Figure 30.

Now that the LTL property is embedded in the composition of the MDP and DRA, we would like to evaluate the property. To do this, it is convenient to

[13] `https://mathworks.com/discovery/multiobjective-optimization.html`
[14] `https://www.alglib.net/multi-objective-optimization/`

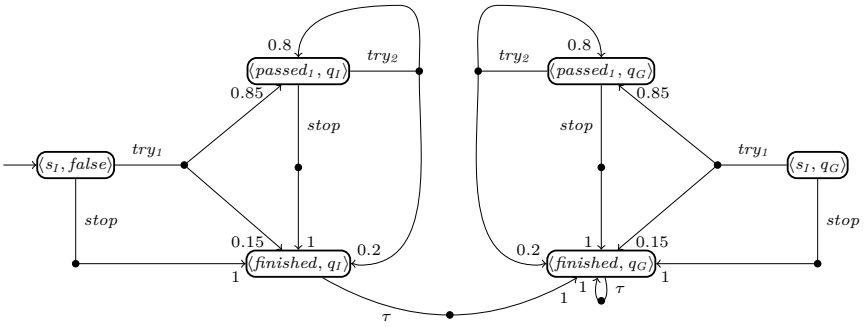



**Fig. 30.** Figure 2 composed with the DRA for $\Diamond \{finished\}$

construct a reward-structure again such that we can constrain the algorithms to reward-structures.

An LTL property is satisfied for a path if the path ends by staying within an accepting end component forever. Therefore, we will add a transition from each end component to a new state $s_{dead}$. Taking a transition to $s_{dead}$ corresponds to staying in the end component from which the transition was taken for ever. For these $s_{dead}$ transitions, we set a reward of $+1$ for each LTL property for which the end component is accepting and 0 otherwise. Of course, this approach only guarantees a correct outcome if there are no rewards in the end component itself. However, we consider such end component modelling errors, and hence we do not evaluate properties on MDPs containing such rewards as discussed in Section 6.2. We show the transformation to add $s_{dead}$ in Proposition 12.

**Proposition 12.** *Let $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ be an MDP. Let*

$$\mathcal{M}' = \langle \mathcal{S} \cup \{s_{dead}\}, s_I, \mathcal{A} \cup \{loop\}, \{\delta(s) \cup \delta'(s) \mid s \in \mathcal{S}\} \rangle$$

*be the MDP, with $\delta'$ the smallest partial function such that:*

- $\forall \langle \mathcal{S}_e, s_e, \mathcal{A}_e, \delta_e \rangle \in EC(\mathcal{M}), s \in \mathcal{S}_e \colon \delta'(s) = \{loop \mapsto \{s' \mapsto [s' = s_{dead}]\}\}$,
- $\delta'(s_{dead}) = \{loop \mapsto \{s \mapsto [s = s_{dead}]\}\}$.

*Then for every branch reward-structure $\rho$, we have*

$$query_{\mathcal{M}}([CumRew_{\mathcal{M}}(\rho)]_{\circ}^{\leq \infty}) = query_{\mathcal{M}'}([CumRew_{\mathcal{M}'}(\rho)]_{\circ}^{\leq \infty})$$

*when $\mathcal{M}$ does not contain infinite cumulative rewards [29].*

*Proof. Shown in [29, Proof of Proposition 3].* □

*Example 42.* When we apply Proposition 12 to the certification process model in Figure 30, we obtain Figure 31. The only end component in the MDP is the $\langle finished, q_G \rangle$ state with its $\tau$ transition. Therefore, we only need a transition from $\langle finished, q_G \rangle$ to $s_{dead}$.

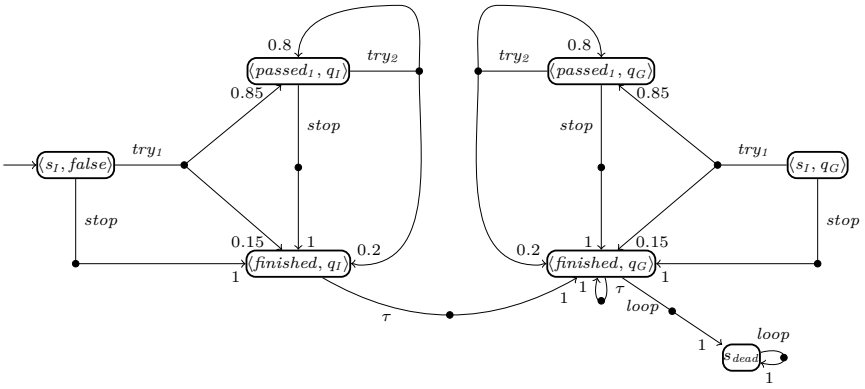



**Fig. 31.** Proposition 12 applied to Figure 30

To transform the DRA to a reward-structure, we now use Proposition 13. This allows us to only verify cumulative reward properties.

**Proposition 13.** *Let $\mathcal{M}' = \langle \mathcal{S}', s_I, \mathcal{A}', \delta' \rangle$ be an MDP transformed by Proposition 12 and $\mathcal{R} = \langle \mathcal{Q}, q_I, \mathcal{A}, \lambda, \mathcal{C} \rangle$ be a DRA. Let $\rho_{\mathcal{S}} \in Struct_{\mathcal{M}'}$ be the branch reward-structure such that:*

- $\forall s \in \mathcal{S}' \colon \forall a \in \mathcal{A} \colon \forall s' \in \mathcal{S}' \colon \rho(s, a, s') = 0$, *and*
- $\rho(s, loop, s_{dead}) = [\exists \mathcal{M}_e = \langle \mathcal{S}_e, s_e, \mathcal{A}_e, \delta_e \rangle \in EC(\mathcal{M}') \colon s \in \mathcal{S}_e \wedge \mathcal{M}_e \models \mathcal{R}]$.

*Then $query_{\mathcal{M}'}([\mathcal{R}]_{\circ}^{\leq k}) = query_{\mathcal{M}'}([CumRew_{\mathcal{M}}(\rho)]_{\circ}^{\leq k})$ [29].*

*Proof. Shown in [29, Proof of Proposition 3].* □

*Example 43.* When we apply Proposition 13 to the certification process model in Figure 31, we obtain Figure 32. We only need to add a reward on the *loop* transition from $\langle finished, q_G \rangle$ to $s_{dead}$. Notice the similarity to Figure 4.

**7.2.2 Algorithm** Now that we know how to construct the MDP using Proposition 12 and create reward-structures for LTL properties that are suitable to be checked by a linear program using Proposition 13, we need to construct the linear program itself. We show this linear program in Algorithm 13.

The paper requires step-bounds to be infinite. This makes sense since linear programming is not an approach that goes over each step such as value iteration, but is instead able to compute the value in the limit. When all step-bounds are infinite, it turns out that a probabilistic memoryless strategy suffices [29]. A strategy can only benefit from being step-aware if the step-bounds are finite. Otherwise, the choice in the $k$th step should be the same as in the $k+1$th step. Hence, we do need to ensure that the linear program computes the result based on a memoryless step-positional strategy.

We do so by computing the expected number of times each transition is taken. The expected number of times the transition with the action $a$ is taken from the

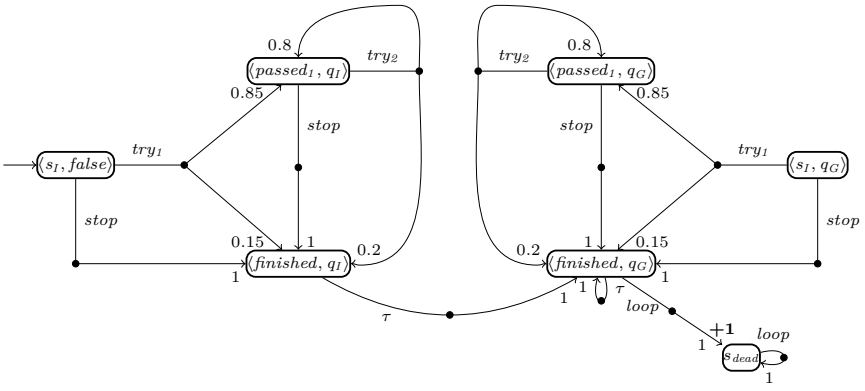



**Fig. 32.** Proposition 13 applied to Figure 31

state $s$ is modelled as the variable $y_{\langle s,a\rangle}$. To model compute the expected number of times each transition is taken, we use the fact that the number of times a state is entered must be equal to the number of times it is left, except for the initial state, which is left one more time than it is entered. This is the second constraint of the linear program. Moreover, we know that the expected number of times a transition is taken cannot be negative, which is the fourth constraint in the linear program.

The constraints of the multi-objective property are encoded as the first constraint in the linear program. The expected reward for a branch-reward structure is the sum of each branch-reward multiplied by the expected number of times the corresponding branch is visited. The expected number of visits to a branch is equal to the expected number of times the transition containing the branch is taken, multiplied by the probability of taking that branch when the transition is taken.

When we optimise for a numerical query, we optimise for the sum of the expected rewards of the unknown.

To ensure that the rewards are used, we also need to ensure that the sum of the expected number of visits to the transitions leading to $s_{dead}$ is 1. This is the third constraint of the linear program.

If we compute a multi-objective numerical query using this linear program, the final result of $query_{\mathcal{M}}(\boldsymbol{\phi})$ should just like for the constraints be the sum of the branch-rewards multiplied by the expected times the branches are visited:

$$\sum_{s\in\mathcal{S}\setminus\{s_{dead}\}}\sum_{a\in\alpha_{\mathcal{M}}(s)}\sum_{s'\in\mathcal{S}}\rho_j(s,a,s')\cdot\delta(s)(a)(s')\cdot y_{\langle s,a\rangle}.$$

If we want to compute a multi-objective achievability query instead, we can slightly alter the property such that it becomes a multi-objective numerical property. We transform one of the constraints to an unknown and check whether the value computed for this unknown satisfies the required bounds. We show this in Algorithm 14.

**Alg. 13.** Multi-objective numerical query - *LPNumQuery*

**Input:** MDP $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ obtained through Proposition 12, multi-objective numerical property $\hat{\boldsymbol{\phi}} = \left\langle [CumRew_{\mathcal{M}}(\rho_1)]^{\leq\infty}_{\geq c_1}, \ldots, [CumRew_{\mathcal{M}}(\rho_j)]^{\leq\infty}_{max}, \ldots, [CumRew_{\mathcal{M}}(\rho_d)]^{\leq\infty}_{\geq c_d} \right\rangle$ with the unknown in position $j \in \{1, \ldots, d\}$

**Result:** $query_{\mathcal{M}}(\hat{\boldsymbol{\phi}})$

1 Solve the linear program

$$\text{maximise} \sum_{s \in \mathcal{S} \setminus \{s_{dead}\}} \sum_{a \in \alpha_{\mathcal{M}}(s)} \sum_{s' \in \mathcal{S}} \rho_j(s,a,s') \cdot \delta(s)(a)(s') \cdot y_{\langle s,a \rangle}$$

$$\text{subject to} \sum_{s \in \mathcal{S} \setminus \{s_{dead}\}} \sum_{a \in \alpha_{\mathcal{M}}(s)} \sum_{s' \in \mathcal{S}} \rho_i(s,a,s') \cdot \delta(s)(a)(s') \cdot y_{\langle s,a \rangle} \geq c_i \qquad \forall \boldsymbol{i} \in \{1, \ldots, d\} \setminus \{j\}$$

$$\sum_{a \in \alpha_{\mathcal{M}}(s)} y_{\langle s,a \rangle} - \sum_{s' \in \mathcal{S}} \sum_{a' \in \alpha_{\mathcal{M}}(s')} \delta(s')(a')(s) \cdot y_{\langle s',a' \rangle} = [s = s_I] \qquad \forall s \in \mathcal{S} \setminus \{s_{dead}\}$$

$$\sum_{s \in \mathcal{S}} y_{\langle s,loop \rangle} = 1$$

$$y_{\langle s,a \rangle} \geq 0 \qquad \forall s \in \mathcal{S}, a \in \alpha_{\mathcal{M}}(s)$$

2 **return** $\sum_{s \in \mathcal{S} \setminus \{s_{dead}\}} \sum_{a \in \alpha_{\mathcal{M}}(s)} \sum_{s' \in \mathcal{S}} \rho_j(s,a,s') \cdot \delta(s)(a)(s') \cdot y_{\langle s,a \rangle}$

**Alg. 14.** Multi-objective achievability queries

**Input:** MDP $\mathcal{M}$, multi-objective achievability property $\bar{\boldsymbol{\phi}} = \left\langle [CumRew_{\mathcal{M}}(\rho_1)]^{\leq\infty}_{\geq c_1}, \ldots, [CumRew_{\mathcal{M}}(\rho_d)]^{\leq\infty}_{\geq c_d} \right\rangle$

**Result:** $query_{\mathcal{M}}(\bar{\boldsymbol{\phi}})$

1 $\boldsymbol{\phi'} \leftarrow \left\langle \begin{cases} [CumRew_{\mathcal{M}}(\rho_i)]^{\leq\infty}_{\geq c_i}, & i \neq 1 \\ [CumRew_{\mathcal{M}}(\rho)]^{\leq\infty}_{max}, & \text{otherwise} \end{cases} \right\rangle_{i \in \{1,\ldots,d\}}$

2 **return** $LPNumQuery(\mathcal{M}, \boldsymbol{\phi'}) \geq c_1$

## 8 Replication Of Most Prevalent Papers

So far, we have evaluated existing implementations of model checking algorithms. We will now also evaluate the papers that describe these algorithms. To do this, we implement the approach laid out in these papers to find mistakes and curiosities in these papers. We implement these approaches in the Modest Toolset [35], since it already has a solid framework for probabilistic model checking but does not yet support multi-objective model checking.

We provide practical information on important parts of the implementation we noticed during our implementation, which are not always present in these mostly theoretical papers. The results of our algorithms implemented are discussed in Section 9.

### 8.1 Value Iteration

During the replication of [30], we found a few peculiarities in the paper. Firstly, all algorithms only support cumulative rewards, although the paper briefly mentions in [30, Extensions] that it is possible to combine the algorithm with Rabin automata to support $\omega$-regular properties, which allow for checking of LTL properties. However, all models provided, except for two, use LTL properties.[15] This makes it difficult for programmers to test whether their basic implementation works as intended. It would be convenient to have models that only use properties with cumulative rewards in the format required, for example, by Algorithm 11. This would allow for testing only the specified algorithms without the need for additional transformations.

Next, we also noticed that the linear program given in [30, Heuristics and optimisations], appears to be incorrect. This linear program solves line 3 of Algorithms 10 and 11 using linear programming instead of finding the separating hyperplane using geometry.

The linear program uses $\boldsymbol{q}$ in its computation, which is not available in the context on line 3, although we could interpret this as the $\boldsymbol{q}$ computed in the previous iteration. However, if we change $\boldsymbol{q}$ to $\boldsymbol{c}$, the linear program makes more sense. This would find the vector that has the maximum distance to the reward, instead of the last used vector, which thus gives us the maximum separating hyperplane. This would also address the question of why the paper mentions that point $\boldsymbol{c}$ and set $X$ are being used for the linear program, while $\boldsymbol{c}$ is not used in the linear program.

In the same linear program, we also notice that "$w_i \cdot (q_i - x_i) \geq d$ for all $\boldsymbol{x} \in X$" is used. However, there is no way to access $i$ in the equation. However, given the description two lines above, the equation should implement a dot product. This can be achieved by adding a $\sum_{i=1}^{n}$ in front, to get: "$\sum_{i=1}^{n} w_i \cdot (q_i - x_i) \geq d$ for all $\boldsymbol{x} \in X$".

To use models that are not described in the "basic form", [30, Proposition 2] is used. This conversion aims to do the same as our transformation shown in Proposition 8. However, instead of the equality

$$query_{\mathcal{M}}([Rew_{\mathcal{M}}]^{\leq k}_{\leq c}) = query_{\mathcal{M}}([-Rew_{\mathcal{M}}]^{\leq k}_{\geq -c}),$$

the paper claims that

$$query_{\mathcal{M}}([Rew_{\mathcal{M}}]^{\leq k}_{\leq c}) = query_{\mathcal{M}}([-Rew_{\mathcal{M}}]^{\leq k}_{\geq c}).$$

The difference between these two equations is that in their approach the $c$ is not negated. However, as can be seen from the proof of Proposition 8, this would correspond to $a \leq b \iff -a \geq b$, which is obviously incorrect. Hence, $c$ should also be negated.

Another problem can be found in [30, Alg. 2], which is their version of the value iteration part of the algorithm shown in Algorithm 9. Due to Proposition 8,

[15] `https://www.prismmodelchecker.org/files/atva12mo`

we know that the rewards might be positive and negative. The paper computes $\Delta$ by computing the largest difference between two consecutive iterations, without taking an absolute value. However, this means that $\Delta$ can also be negative when no absolute value is taken (since value iteration can only decrease rather than increase values for negative rewards). Moreover, taking a negative convergence threshold would not solve the problem since this would also require a sign switch from $>$ to $<$. Moreover, even with the sign switch, if positive and negative rewards are mixed, the approach breaks down again. Therefore, the absolute value of $\Delta$ should be used so that $\epsilon$ can always be a small positive number, as shown in Algorithm 9.

Another small notational problem lies in their value iteration algorithm [30, Alg. 2]. In our adaptation of the algorithm shown in Algorithm 9, we used $\Delta$ for the largest difference between two iterations. However, the original paper uses $\delta$ for this difference, while it also uses $\delta$ as the transition function of the MDP.

Moreover, while the approach compares the speed of the approach in the experiments shown in [30, Experimental results] to the linear programming approach, it does not compare the results of the queries. The results of these queries are extremely interesting, since value iteration can only approximate the result of a query. It is fairly easy to quickly output any number, but the number is meaningless if it is not clear how accurate the number is.

### 8.2 Linear Programming

The linear programming paper [29] appears to contain significantly fewer mistakes than the value iteration paper [30]. Also in [29, Experimental Results], in addition to the speed, they compare the results of the queries themselves to the previously known method.

Only a small issue appears to be present in [29, Fig. 2.]. This is the linear program shown in Algorithm 13. In their notation, they use $\mu'(s)$ while $\mu'$ is not defined in the context. However, since the summation in which it occurs sums the sum of the expected number of times the state $s$ is entered, $\mu'(s)$ should be changed to $\hat{\mu}(s)$. This corresponds to our use of $\delta(s')(a')(s)$ in the second constraint of Algorithm 13.

### 8.3 Floating-Point Values

Most implementations, including ours, do not use exact representations for numbers, but instead use double-precision numbers as defined in IEEE 754 [44]. This means that theoretical equalities, such as in the linear program described in Algorithm 13, should not be programmed as equalities. Otherwise, the same issue from Section 5.1.2 would arise. This would mean that due to floating point inaccuracies, not all constraints might be satisfied, thus failing the entire program. To achieve this, we can change such equalities $x = y$, to two inequalities $x \geq y - \epsilon$ and $x \leq y + \epsilon$, for some small $\epsilon$. In our program, we use $\epsilon = 10^{-6}$.

The same rounding errors must be taken into account when implementing inequalities. For example, on line 6 of the value iteration approach shown in

Algorithm 11. In this case, to achieve a strictly greater than, we can change $\boldsymbol{w} \cdot \boldsymbol{q} < \boldsymbol{w} \cdot \boldsymbol{c}$ to $\boldsymbol{w} \cdot \boldsymbol{q} < \boldsymbol{w} \cdot \boldsymbol{c} + \epsilon$. It is important to note that such changes *do* slightly alter the precision of the outcome of the algorithm. The bigger the choice of $\epsilon$, the more inaccurate the result might get. However, if $\epsilon$ is too small, the program might fail since an (in)equality might not be satisfied using double-precision values, while with exact values the (in)equality would be satisfied.

Another important inequality is the $w_j > 0$ constraint for the separating vector on line 3 of Algorithm 11. This ensures that when there are two strategies that produce the same values for all dimensions, except for the unknown dimension, the strategy with the highest value in the unknown dimension is selected. This means that this inequality can also be checked inside Algorithm 9 by storing the value for the unknown as well. In fact, this is also how the authors of the original paper implemented this inequality.[16]

### 8.4 Infinity

In practice, it can be hard to model the downward closure using infinities as described in Proposition 11. The reason for this is that for example $\infty - \infty$ is undefined. However, for steps such as Gaussian elemination, we would require $\infty - \infty = 0$. If we are using double-precision numbers, we can approximate infinity using any large number. We choose the maximum signed integer that can be described using 32 bits. However, another large number might also be used as well. It is important to not choose a number that is too small, since this will, for example, influence the Gaussian elimination, making the results even more inaccurate.

Moreover, the value should not be chosen too large. If this is the case, the Gaussian elimination will fail again, but this time it will fail due to an overflow or underflow. Otherwise, we might, for example, end up with $-\infty' - \infty' > 0$, where $\infty'$ is the value that we use to represent infinity in the program.

### 8.5 LTL Properties

We have not implemented LTL properties, but we understand that future implementers might want to do this. For this, most model checking tools use the Spot library [24]. This library can generate an $\omega$-automaton that is equivalent to a given LTL property. A user can then provide an LTL property, since this is user-friendlier than asking the user for a DRA.

## 9 Experimental Validation

To check the effectiveness of the changes shown in Section 5 and the algorithms implemented in Section 8, we use an approach very similar to that taken in Section 4.

[16] `https://github.com/prismmodelchecker/prism/blob/1f86cc2241fa4353fd16c58d1c69256ba8be7a5b/prism/src/sparse/PS_NondetMultiObjGS.cc#L342`

### 9.1 Approach

We use the same approximation approach as shown in Algorithm 2 and the following models from Section 4: care home (CH), client server (CS), dining philosophers (DP), dynamic power management (DPM), hiring points (HP), Mars rover (MR), network virus (NV), randomised consensus (RC), resource gathering (RG), sensor network (SN), task graph scheduling (TGS), zeroconf network (ZN), zeroconf time based (ZTB). These are all models from Section 4, with the exception of the "team formation" model. This model is excluded since the approach to convert the model to an input formalism accepted by Modest cannot convert this model.

To convert these models, we use the *storm-conv* executable shipped with Storm [40] to convert the model to JANI [16] and then use the Modest [35] *convert* tool to convert the JANI model to a Modest model.

### 9.2 Results

Using this approach, we get the data shown in Table 5. While we do not perform a benchmark, the speed for the approaches differs significantly. Storm seems to be the fastest tool on this set of models. It is followed by PRISM and our implementation which have comparable speeds. ePMC takes the longest time on these models.

In Table 5, we also show the reason why a value is not present if it was not computed. We list "$\cdots$" if the computation did not finish in 2 minutes for a value iteration query and 10 minutes for a linear programming problem. For infinite cumulative rewards, we use "$I_\infty$", for unsupported properties "-", "↑" when we run out of memory. The code for these improved experiments is also available.[17]

Just like in Section 4, we denote a non-achievable multi-objective numerical property by $\perp$ and show a value if it was constant for all convergence thresholds, otherwise we show a plot of how the result changes based on the convergence threshold.

We observe in Table 5 that most entries are similar to Table 3. However, for all tools, there have been a few changes.

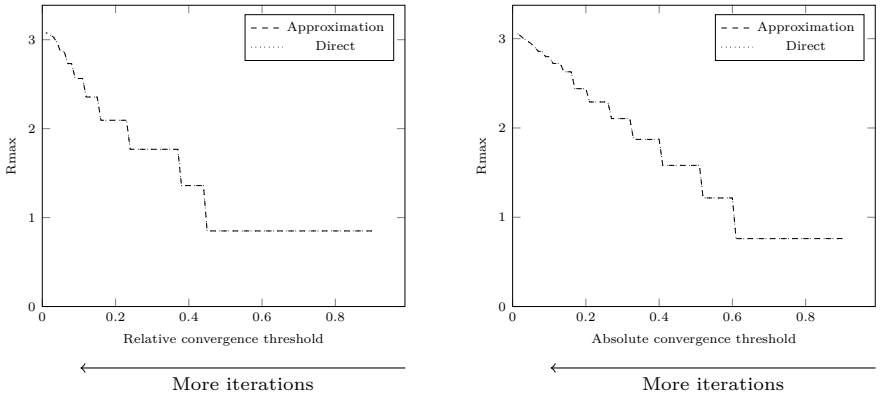
**9.2.1 Storm** For Storm, we observe that its linear programming results now align with the other model checkers. This suggests that it is important to use the `--exact` flag for Storm when using linear programming as suggested in Section 5.1.2. The segmentation fault has been resolved and instead, the computation is timed out, suggesting that the fix from Section 5.1.1 has worked.

**9.2.2 PRISM** The PRISM results also seem more promising when using linear programming. The approximation results align much more closely with the results for the other tools and PRISM's own direct query. This suggests that the change made in Section 5.2.1 has a positive effect on the results.

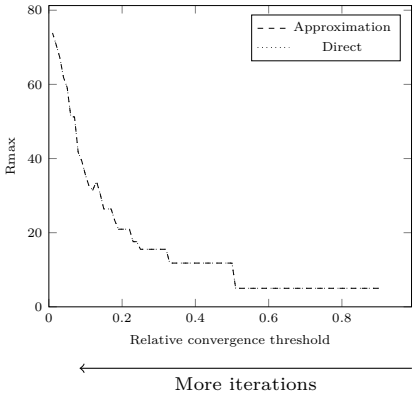
[17] `https://github.com/Chickenpowerrr/multiobjectiveanalysis/releases/tag/v1.1`

**Table 5.** Validating results

| | | | Value Iteration | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Relative convergence threshold | | | | | | Absolute convergence threshold | | | | | | | | | | Linear Programming | | | | | |
| | | | PRISM | | ePMC | | Modest | | Storm | | PRISM | | ePMC | | Modest | | Storm | | PRISM | | Modest | |
| Model | Property | Constants | Approx | Direct | Approx | Direct | Approx | Direct | Approx | Direct | Approx | Direct | Approx | Direct | Approx | Direct | Approx | Direct | Approx | Direct | Approx | Direct |
| CH | 1 | 1 | … | … | ⊥ | 1000.0 | $I_\infty$ | $I_\infty$ | Figure 11a | | … | … | … | … | $I_\infty$ | $I_\infty$ | … | 68.0000 | 68.0000 | $I_\infty$ | $I_\infty$ | $I_\infty$ |
| CH | 1 | 2 | … | … | … | … | $I_\infty$ | $I_\infty$ | … | … | … | … | … | … | $I_\infty$ | $I_\infty$ | ↑ | … | … | $I_\infty$ | $I_\infty$ | $I_\infty$ |
| CH | 2 | 1 | - | - | ⊥ | 0.0000 | - | - | ⊥ | ⊥ | - | - | ⊥ | 0.0000 | - | - | ⊥ | - | - | - | - | - |
| CH | 2 | 2 | - | - | … | … | - | - | … | … | - | - | … | … | - | - | ↑ | - | - | - | - | - |
| CS | 1 | n/a | - | - | ⊥ | −1000.0 | - | - | Figure 11b | | - | - | … | … | - | - | - | - | - | - | - | - |
| DP | 1 | n/a | - | - | ⊥ | −1000.0 | - | - | Figure 11c | | - | - | … | … | - | - | - | - | - | - | - | - |
| DPM | 1 | 1 | 57.7589 | 57.7589 | 0.0000 | 0.0000 | - | - | Figure 11d | | 57.7589 | 57.7589 | 0.0000 | 0.0000 | - | - | - | - | - | - | - | - |
| HP | 1 | n/a | Figure 12a | | … | … | Figure 33a | | Figure 12b | | Figure 12c | | … | … | Figure 33b | | 3.1000 | 3.1000 | 3.1000 | 3.1000 | 3.1000 | 3.1000 |
| MR | 1 | 1 | 0.0000 | 0.0000 | ⊥ | ⊥ | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | ⊥ | ⊥ | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| MR | 1 | 2 | Figure 13a | | ⊥ | ⊥ | ⊥ | ⊥ | Figure 13b | | Figure 13c | | Figure 13d | | ⊥ | ⊥ | … | … | … | 0.8724 | 0.8724 | |
| MR | 2 | 1 | Figure 14a | | 0.0000 | 0.0000 | Figure 33c | | 76.6667 | 76.6667 | Figure 14b | | 0.0000 | 0.0000 | … | … | 76.6667 | 76.6667 | 76.6667 | 76.6667 | 76.6667 | 76.6667 |
| MR | 2 | 2 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| NV | 1 | n/a | - | - | 4.0000 | 4.0000 | - | - | 3.2537 | 3.2537 | - | - | … | … | - | - | - | - | - | - | - | - |
| NV | 2 | n/a | - | - | 1.0000 | 4.0000 | - | - | Figure 11e | | - | - | … | … | - | - | - | - | - | - | - | - |
| RC | 1 | n/a | 0.8917 | 0.8917 | ⊥ | ⊥ | 1.0000 | 1.0000 | 0.8917 | 0.8917 | 0.8917 | 0.8917 | 0.8917 | 0.8917 | 1.0000 | 1.0000 | 0.8917 | 0.8917 | 0.8917 | 0.8917 | 1.0000 | 1.0000 |
| RG | 1 | 1 | Figure 15a | | ⊥ | ⊥ | … | … | Figure 15b | | Figure 15c | | Figure 15d | | … | … | 0.8238 | 0.8238 | 0.8239 | 0.8238 | 0.8239 | 0.8239 |
| SN | 1 | n/a | - | - | 0.0000 | −1000.0 | - | - | Figure 11f | | - | - | … | … | - | - | - | - | - | - | - | - |
| TGS | 1 | 1 | … | … | 0.0000 | 0.0000 | $I_\infty$ | $I_\infty$ | $I_\infty$ | $I_\infty$ | … | … | 0.0000 | 0.0000 | $I_\infty$ | $I_\infty$ | $I_\infty$ | 6.2000 | 6.2000 | $I_\infty$ | $I_\infty$ | |
| ZN | 1 | 1 | 0.0003 | 0.0003 | ⊥ | ⊥ | ⊥ | ⊥ | 0.0003 | 0.0003 | 0.0003 | 0.0003 | 0.0003 | 0.0003 | ⊥ | ⊥ | 0.0003 | 0.0003 | 0.0003 | ⊥ | ⊥ | |
| ZTB | 1 | 1 | 0.0002 | 0.0002 | ⊥ | ⊥ | ⊥ | ⊥ | 0.0002 | 0.0002 | 0.0002 | 0.0002 | … | … | ⊥ | ⊥ | 0.0002 | 0.0002 | 0.0002 | ⊥ | ⊥ | |




**(a)** Hiring process, relative $\epsilon$ **(b)** Hiring process, absolute $\epsilon$




**(c)** Mars rover, relative $\epsilon$

**Fig. 33.** Value iteration using our Modest implementation

**9.2.3 ePMC** We observe that ePMC can now communicate unachievable multi-objective numerical properties. In addition, the $-1.000$ result has been resolved. Unfortunately, the $-1000.0$ entries are still present. It does not seem likely that this issue will be resolved soon due to the lack of developers' resources. Therefore, ePMC does not seem like a solid choice as a model checker for most purposes.

**9.2.4 Our implementation** Our own implementation appears quite promising. It is the only tool that does not seem to have a difference between its direct query and the approximation. Moreover, the value iteration results seem to be approaching the linear programming results.

Unfortunately, the results still often differ slightly from the results of other model checkers, in particular PRISM and Storm. This might be due to the loose rounding we have used in our implementation. We, for example, model infinity

as the largest integer that can be stored in 32 bits and as a result threat all values smaller than $10^{-6}$ as being equivalent to 0. The other tools do not model infinity since they compute the downward closure in a different way and typically have a smaller number than $10^{-6}$ to handle rounding errors. This means that our results might be less accurate.

Moreover, we do not use the same model file as the other model checkers since Modest does not accept the same formalism. It is possible that the conversion to another input formalism has introduced a small mistake. Either by the tool or by the parts converted by hand. Unfortunately, the *storm-conv* utility we used to convert PRISM files to JANI, only works one way. We can therefore not check whether the results stay stable for PRISM and Storm after converting to JANI and back.

Another important observation is that our implementation considers the first property in the care home model invalid. This might explain why the results using PRISM's linear programming and Storm's value iteration differ by so much.

## 10 Conclusion

We have explored and improved the state of multi-objective model checking. We have done so by building upon our previous research, based on which we laid out our research questions which we will revisit in this section.

In general, we found that Storm offers the most reliable results, moreover it seems to be the fastest tool. However, in particular, when using linear programming it is extremely important to use the `--exact` flag. Based on a conversation with the authors of the tool, we have to conclude that ePMC will not receive the required updates. Hence, ePMC should not be used in most cases for multi-objective model checking, as a significant amount of results appear to be incorrect without a prospect of future change.

### 10.1 RQ1 Which mistakes are present in existing model checkers?

In Sections 4 and 6.1 we have identified concrete mistakes in existing model checkers. This list might not be extensive, but it shows that many such mistakes are present in the current generation of model checkers. With this approach, we have shown that current implementations are still error-prone. Many mistakes are made when implementing multi-objective model checking algorithms, which find their way into the tools. Hence, a better approach to testing these tools should be found. In fact, the entire Modest Toolset only tests by running a few models to verify whether their results are correct. It would be beneficial to at least include unit tests for critical transformations. This would already improve confidence in the system.

Also, for Storm, PRISM and ePMC, more extensive tests should be present for multi-objective model checking. The test suite we used in this research is also by no means extensive in itself either. Most evaluated multi-objective numerical

properties seem to be quite close to the Pareto curve. This might disproportionately highlight the effect of rounding errors. Moreover, there are relatively few models that are small enough to be computed by hand so that the correct result is known. This would simplify the finding of errors in model checkers and could serve as a great test case. Right now we primarily need to rely on comparing tools to themselves.

### 10.2 RQ2 What is the origin of the mistakes in existing model checkers?

For most of the mistakes found in the existing model checkers, we were able to find the origin as shown in Sections 5 and 6.1. Some of these have been resolved by us, and we have made pull requests to resolve these issues. In the cases where we were able to determine the problem, but solving the problem would have been too time-consuming, we came up with minimal examples and reported the problems to the maintainers with the minimal examples. This helps the maintainers of the tool understand the problem as well and resolve the issue. The big exception here is ePMC. We have had a meeting with their maintainers, who explained that they do not have sufficient resources to update the tool. We have still made some pull requests to fix some issues, but it is unlikely that more time-consuming fixes will be made in the near future.

### 10.3 RQ3 Which mistakes are present in multi-objective model checking theory?

To find mistakes in the current theory, we have replicated the results of [29,30]. We have found several mistakes in these papers, which we have shown in Section 8. This helps future implementers avoid falling into these pitfalls, since we also provide an explanation on how these mistakes should be corrected. To further support future implementers, we explained the replicated papers in more detail in Section 7. We also provided intuition for the algorithms, which, due to the page limit in other papers, might have been hard to provide in the original papers. This also helps to prevent mistakes, since understanding what you are implementing can make it easier to spot mistakes. This is harder if one only implements exactly what is stated in a paper and does not understand why the algorithm is supposed to work.

### 10.4 RQ4 How should invalid models be treated?

In Section 6 we have explored the current ways in which models might be considered invalid. For some algorithms such as the linear programming approach [29] such constraints on the model are required for the algorithm to work. However, we have also seen that in current model checkers, these constraints are not always guaranteed. The procedure shown in [29] only disregards the smallest set of models that it cannot model check. However, as we have seen, this requires

another multi-objective query, which is computationally expensive. Moreover, the approach taken by Storm can lead to confusing results. Therefore, we came up with our own approach in Section 6.2 that declares all models that need to be invalid according to [29] invalid. However, it also declares any model with an infinite minimising reward as invalid. This makes the computation efficient, and also avoids the loophole in which users would change an infinite maximising reward to a minimising reward while changing the sign of the rewards.

### 10.5 Future Research

A new question that arose from our research is why the value iteration approach [30] seems to be non-monotonic in some cases, in particular for PRISM as seen in Section 4. This would still require future research to verify whether this is an inherent property of the algorithm or whether it is due to the implementation.

To ensure that model checkers can be compared more easily, it would be beneficial if all of them could use the same models. Unfortunately, the JANI language [16], which has been introduced to achieve interoperability between several model checkers, does not yet support multi-objective properties. If this were to be added, multi-objective models could be added to the Quantitative Verification Benchmark Set [38], which can be used to benchmark model checkers.

More confidence in the current generation of model checkers can be achieved by formally proving the underlying algorithms. There are still a lot of unproven algorithms that could benefit from proofs. We hope that in the future a model checker is developed that is formally proven. This would provide the ultimate guarantee.

## References


1. de Alfaro, L.: Formal verification of probabilistic systems. Ph.D. thesis, Stanford University, USA (1997), https://searchworks.stanford.edu/view/3910936
2. Andriushchenko, R., Bork, A., Budde, C.E., Češka, M., Grover, K., Hahn, E.M., Hartmanns, A., Israelsen, B., Jansen, N., Jeppson, J., Junges, S., Köhl, M.A., Könighofer, B., Křetínský, J., Meggendorfer, T., Parker, D., Pranger, S., Quatmann, T., Ruijters, E., Taylor, L., Volk, M., Weininger, M., Zhang, Z.: Tools at the Frontiers of Quantitative Verification. In: Beyer, D., Hartmanns, A., Kordon, F. (eds.) TOOLympics Challenge 2023. Lecture Notes in Computer Science, vol. 14550, pp. 90–146. Springer (2024). https://doi.org/10.1007/978-3-031-67695-6_4
3. Aspnes, J., Herlihy, M.: Fast Randomized Consensus Using Shared Memory. Journal of Algorithms **11**(3), 441–461 (1990). https://doi.org/10.1016/0196-6774(90)90021-6
4. Bagdasaryan, A.: On the Partition of Space by Hyperplanes. European Journal of Pure and Applied Mathematics **16**(2), 893–898 (2023). https://doi.org/10.29020/nybg.ejpam.v16i2.4713
5. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model Checking Probabilistic Systems. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 963–999. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_28
6. Bals, S., Evangelidis, A., Kretínský, J., Waibel, J.: MULTIGAIN 2.0: MDP controller synthesis for multiple mean-payoff, LTL and steady-state constraints. In: Ábrahám, E., Jr., M.M. (eds.) Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2024, Hong Kong SAR, China, May 14-16, 2024. pp. 24:1–24:7. ACM (2024). https://doi.org/10.1145/3641513.3650135
7. Barber, C.B., Dobkin, D.P., Huhdanpaa, H.: The Quickhull Algorithm for Convex Hulls. ACM Transactions on Mathematical Software **22**(4), 469–483 (1996). https://doi.org/10.1145/235815.235821
8. Barrett, L., Narayanan, S.: Learning all optimal policies with multiple criteria. In: Cohen, W.W., McCallum, A., Roweis, S.T. (eds.) Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008. ACM International Conference Proceeding Series, vol. 307, pp. 41–47. ACM (2008). https://doi.org/10.1145/1390156.1390162
9. Bashar, A., Muhammad, S., Mohammad, N., Khan, M.: Modeling and Analysis of MDP-based Security Risk Assessment System for Smart Grids. In: 2020 Fourth International Conference on Inventive Systems and Control (ICISC). pp. 25–30. IEEE (2020). https://doi.org/10.1109/ICISC47916.2020.9171072
10. Basset, N., Kwiatkowska, M.Z., Topcu, U., Wiltsche, C.: Strategy Synthesis for Stochastic Games with Multiple Long-Run Objectives. In: Baier, C., Tinelli, C. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9035, pp. 256–271. Springer (2015). https://doi.org/10.1007/978-3-662-46681-0_22
11. Benayoun, R., de Montgolfier, J., Tergny, J., Laritchev, O.: Linear programming with multiple objective functions: Step method (stem). Mathematical Programming **1**(1), 366–375 (1971). https://doi.org/10.1007/BF01584098

12. de Berg, M., Cheong, O., van Kreveld, M.J., Overmars, M.H.: Computational geometry: algorithms and applications, 3rd Edition. Springer (2008). https://doi.org/10.1007/978-3-540-77974-2
13. Berger, M.: Geometry I. Springer (1987)
14. Bertsimas, D., Tsitsiklis, J.N.: Introduction to linear optimization, Athena scientific optimization and computation series, vol. 6. Athena Scientific (1997)
15. Boyd, S.P., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2014). https://doi.org/10.1017/CBO9780511804441
16. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: Quantitative Model and Tool Interaction. In: Legay, A., Margaria, T. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10206, pp. 151–168. Springer (2017). https://doi.org/10.1007/978-3-662-54580-5_9
17. Chan, T.M.: Optimal Output-Sensitive Convex Hull Algorithms in Two and Three Dimensions. Discrete & Computational Geometry **16**(4), 361–368 (1996). https://doi.org/10.1007/BF02712873
18. Chatterjee, K., Henzinger, M.: Faster and Dynamic Algorithms for Maximal End-Component Decomposition and Related Graph Problems in Probabilistic Verification. In: Randall, D. (ed.) Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011. pp. 1318–1336. SIAM (2011). https://doi.org/10.1137/1.9781611973082.101
19. Chen, T., Kwiatkowska, M.Z., Parker, D., Simaitis, A.: Verifying Team Formation Protocols with Probabilistic Model Checking. In: Leite, J., Torroni, P., Ågotnes, T., Boella, G., van der Torre, L. (eds.) Computational Logic in Multi-Agent Systems - 12th International Workshop, CLIMA XII, Barcelona, Spain, July 17-18, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6814, pp. 190–207. Springer (2011). https://doi.org/10.1007/978-3-642-22359-4_14
20. Cicotti, G., Coronato, A.: Towards a Probabilistic Model Checking-based approach for Medical Device Risk Assessment. In: 2015 IEEE International Symposium on Medical Measurements and Applications, MeMeA 2015, Torino, Italy, May 7-9, 2015. pp. 180–185. IEEE (2015). https://doi.org/10.1109/MEMEA.2015.7145195
21. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Fourth Edition. MIT Press (2022)
22. Courcoubetis, C., Yannakakis, M.: Verifying Temporal Properties of Finite-State Probabilistic Programs. In: 29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988. pp. 338–345. IEEE Computer Society (1988). https://doi.org/10.1109/SFCS.1988.21950
23. Duflot, M., Fribourg, L., Picaronny, C.: Randomized dining philosophers without fairness assumption. Distributed Computing **17**(1), 65–76 (2004). https://doi.org/10.1007/S00446-003-0102-Z
24. Duret-Lutz, A., Lewkowicz, A., Fauchille, A., Michaud, T., Renault, E., Xu, L.: Spot 2.0 - A Framework for LTL and $\omega$-Automata Manipulation. In: Artho, C., Legay, A., Peled, D. (eds.) Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9938, pp. 122–129. Springer (2016). https://doi.org/10.1007/978-3-319-46520-3_8

25. Ecker, J.G., Kouada, I.: Finding all efficient extreme points for multiple objective linear programs. Math. Program. **14**(1), 249–261 (1978). https://doi.org/10.1007/BF01588968
26. Farwer, B.: $\omega$-automata. In: Grädel, E., Thomas, W., Wilke, T. (eds.) Automata, Logics, and Infinite Games: A Guide to Current Research. Lecture Notes in Computer Science, vol. 2500, pp. 3–20. Springer (2001). https://doi.org/10.1007/3-540-36387-4_1
27. Finkbeiner, B., Kovács, L. (eds.): State of the Art in Software Verifcation and Witness Validation: SV-COMP 2024, Lecture Notes in Computer Science, vol. 14572. Springer (2024). https://doi.org/10.1007/978-3-031-57256-2_15
28. Forejt, V., Kwiatkowska, M.Z., Norman, G., Parker, D., Qu, H.: Quantitative Multi-objective Verification for Probabilistic Systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6605, pp. 112–127. Springer (2011). https://doi.org/10.1007/978-3-642-19835-9_11
29. Forejt, V., Kwiatkowska, M.Z., Norman, G., Parker, D., Qu, H.: Quantitative Multi-objective Verification for Probabilistic Systems. In: Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6605, pp. 112–127. Springer (2011). https://doi.org/10.1007/978-3-642-19835-9_11
30. Forejt, V., Kwiatkowska, M.Z., Parker, D.: Pareto Curves for Probabilistic Model Checking. In: Chakraborty, S., Mukund, M. (eds.) Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7561, pp. 317–332. Springer (2012). https://doi.org/10.1007/978-3-642-33386-6_25
31. Fu, C., Hahn, E.M., Li, Y., Schewe, S., Sun, M., Turrini, A., Zhang, L.: EPMC gets knowledge in multi-agent systems. In: Finkbeiner, B., Wies, T. (eds.) Verification, Model Checking, and Abstract Interpretation - 23rd International Conference, VMCAI 2022, Philadelphia, PA, USA, January 16-18, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13182, pp. 93–107. Springer (2022). https://doi.org/10.1007/978-3-030-94583-1_5
32. Gallier, J.: Basics of Affine Geometry, pp. 7–63. Springer (2011). https://doi.org/10.1007/978-1-4419-9961-0_2
33. Graham, R.L.: An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set. Information Processing Letters **1**(4), 132–133 (1972). https://doi.org/10.1016/0020-0190(72)90045-2
34. Haddad, S., Monmege, B.: Reachability in MDPs: Refining Convergence of Value Iteration. In: Ouaknine, J., Potapov, I., Worrell, J. (eds.) Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8762, pp. 125–137. Springer (2014). https://doi.org/10.1007/978-3-319-11439-2_10
35. Hartmanns, A., Hermanns, H.: The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification. In: Ábrahám, E., Havelund, K. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 20th Interna-

tional Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8413, pp. 593–598. Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_51
36. Hartmanns, A., Junges, S., Katoen, J., Quatmann, T.: Multi-cost Bounded Tradeoff Analysis in MDP. Journal of Automated Reasoning **64**(7), 1483–1522 (2020). https://doi.org/10.1007/S10817-020-09574-9
37. Hartmanns, A., Junges, S., Quatmann, T., Weininger, M.: A Practitioner's Guide to MDP Model Checking Algorithms. In: Sankaranarayanan, S., Sharygina, N. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13993, pp. 469–488. Springer (2023). https://doi.org/10.1007/978-3-031-30823-9_24
38. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The Quantitative Verification Benchmark Set. In: Vojnar, T., Zhang, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11427, pp. 344–350. Springer (2019). https://doi.org/10.1007/978-3-030-17462-0_20
39. Hartmanns, A., Kohlen, B., Lammich, P.: Fast Verified SCCs for Probabilistic Model Checking. In: André, É., Sun, J. (eds.) Automated Technology for Verification and Analysis - 21st International Symposium, ATVA 2023, Singapore, October 24-27, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14215, pp. 181–202. Springer (2023). https://doi.org/10.1007/978-3-031-45329-8_9
40. Hensel, C., Junges, S., Katoen, J., Quatmann, T., Volk, M.: The probabilistic model checker Storm. International Journal on Software Tools for Technology Transfer **24**(4), 589–610 (2022). https://doi.org/10.1007/S10009-021-00633-Z
41. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate Probabilistic Model Checking. In: Steffen, B., Levi, G. (eds.) Verification, Model Checking, and Abstract Interpretation, 5th International Conference, VMCAI 2004, Venice, Italy, January 11-13, 2004, Proceedings. Lecture Notes in Computer Science, vol. 2937, pp. 73–84. Springer (2004). https://doi.org/10.1007/978-3-540-24622-0_8
42. Hoare, C.A.R.: Communicating Sequential Processes. Communications of the ACM **21**(8), 666–677 (1978). https://doi.org/10.1145/359576.359585
43. Howard, R.A.: Dynamic Programming and Markov Processes. MIT Press (1960)
44. IEEE: IEEE Standard for Floating-Point Arithmetic. IEEE Std 754-2019 (Revision of IEEE 754-2008) pp. 1–84 (2019). https://doi.org/10.1109/IEEESTD.2019.8766229
45. Jarvis, R.A.: On the Identification of the Convex Hull of a Finite Set of Points in the Plane. Information Processing Letters **2**(1), 18–21 (1973). https://doi.org/10.1016/0020-0190(73)90020-3
46. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research **4**, 237–285 (1996). https://doi.org/10.1613/JAIR.301
47. Kemeny, J.G., Snell, J.L., Knapp, A.W.: Denumerable Markov Chains. Springer (1976). https://doi.org/10.1007/978-1-4684-9455-6
48. Khachiyan, L.: Polynomial algorithms in linear programming. USSR Computational Mathematics and Mathematical Physics **20**(1), 53–72 (1980). https://doi.org/10.1016/0041-5553(80)90061-0

49. Kirkpatrick, D.G., Seidel, R.: The Ultimate Planar Convex Hull Algorithm? SIAM J. Comput. **15**(1), 287–299 (1986). https://doi.org/10.1137/0215021
50. Komuravelli, A., Pasareanu, C.S., Clarke, E.M.: Assume-Guarantee Abstraction Refinement for Probabilistic Systems. In: Madhusudan, P., Seshia, S.A. (eds.) Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings. Lecture Notes in Computer Science, vol. 7358, pp. 310–326. Springer (2012). https://doi.org/10.1007/978-3-642-31424-7_25
51. Kretínský, J., Meggendorfer, T.: Of Cores: A Partial-Exploration Framework for Markov Decision Processes. In: Fokkink, W.J., van Glabbeek, R. (eds.) 30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands. LIPIcs, vol. 140, pp. 5:1–5:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPICS.CONCUR.2019.5
52. Kwiatkowska, M., Norman, G., Parker, D., Santos, G.: PRISM-games 3.0: Stochastic Game Verification with Concurrency, Equilibria and Time. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12225, pp. 475–487. Springer (2020). https://doi.org/10.1007/978-3-030-53291-8_25
53. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6806, pp. 585–591. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_47
54. Kwiatkowska, M.Z., Norman, G., Parker, D., Sproston, J.: Performance analysis of probabilistic timed automata using digital clocks. Formal Methods in System Design **29**(1), 33–78 (2006). https://doi.org/10.1007/S10703-006-0005-2
55. Kwiatkowska, M.Z., Norman, G., Parker, D., Vigliotti, M.G.: Probabilistic Mobile Ambients. Theoretical Computer Science **410**(12-13), 1272–1303 (2009). https://doi.org/10.1016/J.TCS.2008.12.058
56. Lacerda, B., Parker, D., Hawes, N.: Multi-Objective Policy Generation for Mobile Robots under Probabilistic Time-Bounded Guarantees. In: Barbulescu, L., Frank, J., Mausam, Smith, S.F. (eds.) Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017. pp. 504–512. AAAI Press (2017). https://doi.org/10.1609/icaps.v27i1.13865
57. Lay, D.C., Lay, S.R., McDonald, J.J.: Linear Algebra and Its Applications. Pearson (2014)
58. Meyer, C.D.: Matrix Analysis and Applied Linear Algebra. SIAM (2000). https://doi.org/10.1137/1.9780898719512
59. Minkowski, H.: Allgemeine Lehrsätze über die konvexen Polyeder, pp. 121–139. Springer Vienna, Vienna (1989). https://doi.org/10.1007/978-3-7091-9536-9_5
60. Naumowicz, A., Thiemann, R. (eds.): 14th International Conference on Interactive Theorem Proving, ITP 2023, July 31 to August 4, 2023, Białystok, Poland, LIPIcs, vol. 268. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPIcs.ITP.2023
61. Norman, G., Parker, D., Kwiatkowska, M.Z., Shukla, S.K., Gupta, R.: Using probabilistic model checking for dynamic power management. Formal Aspects of Computing **17**(2), 160–176 (2005). https://doi.org/10.1007/S00165-005-0062-0

62. Norman, G., Parker, D., Sproston, J.: Model checking for probabilistic timed automata. Formal Methods in System Design **43**(2), 164–190 (2013). https://doi.org/10.1007/S10703-012-0177-X
63. Pnueli, A.: The Temporal Logic of Programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977. pp. 46–57. IEEE Computer Society (1977). https://doi.org/10.1109/SFCS.1977.32
64. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley Series in Probability and Statistics, Wiley (1994). https://doi.org/10.1002/9780470316887
65. Rabin, M.O.: Decidability of Second-Order Theories and Automata on Infinite Trees. Transactions of the American Mathematical Society **141**, 1–35 (1969). https://doi.org/https://doi.org/10.2307/1995086
66. Saad, Y.: Iterative methods for sparse linear systems. SIAM (2003). https://doi.org/10.1137/1.9780898718003
67. Seidel, L.: Über ein Verfahren, die Gleichungen, auf welche die Methode der kleinsten Quadrate führt, sowie lineäre Gleichungen überhaupt, durch successive Annäherung aufzulösen. Abhandlungen der Mathematisch-Physikalischen Classe der Königlich Bayerischen Akademie der Wissenschaften **11**(3), 81–108 (1874), `https://www.biodiversitylibrary.org/item/110049`
68. Soltan, V.: Support and separation properties of convex sets in finite dimension. Extracta Mathematicae **36**(2), 241–278 (12 2021). https://doi.org/10.17398/2605-5686.36.2.241
69. Tomy, M., Lacerda, B., Hawes, N., Wyatt, J.L.: Battery charge scheduling in long-life autonomous mobile robots via multi-objective decision making under uncertainty. Robotics and Autonomous Systems **133**, 103629 (2020). https://doi.org/10.1016/J.ROBOT.2020.103629
70. Visser, W., Mehlitz, P.C.: Model Checking Programs with Java PathFinder. In: Godefroid, P. (ed.) Model Checking Software, 12th International SPIN Workshop, San Francisco, CA, USA, August 22-24, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3639, p. 27. Springer (2005). https://doi.org/10.1007/11537328_5
71. Webster, M., Western, D.G., Araiza-Illan, D., Dixon, C., Eder, K., Fisher, M., Pipe, A.G.: A corroborative approach to verification and validation of human-robot teams. The International Journal of Robotics Research **39**(1) (2020). https://doi.org/10.1177/0278364919883338
72. Weyl, H.: Elementare Theorie der konvexen Polyeder. Commentarii Mathematici Helvetici **7**, 290–306 (1934). https://doi.org/10.1007/BF01292722
73. Xu, R., Li, L., Zhan, B.: Verified Interactive Computation of Definite Integrals. In: Platzer, A., Sutcliffe, G. (eds.) Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12699, pp. 485–503. Springer (2021). https://doi.org/10.1007/978-3-030-79876-5_28
74. Younes, H.L.S., Simmons, R.G.: Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling. In: Brinksma, E., Larsen, K.G. (eds.) Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings. Lecture Notes in Computer Science, vol. 2404, pp. 223–235. Springer (2002). https://doi.org/10.1007/3-540-45657-0_17

# A Notation

| Notation | Meaning |
|---|---|
| $\mathbb{N}_0$ | Natural numbers including 0 |
| $\mathbb{N}_+$ | Natural number excluding 0: $\mathbb{N}_0 \setminus \{0\}$ |
| $\mathbb{N}_{0,\infty}$ | Natural numbers, including 0 and infinity: $\mathbb{N}_0 \cup \{\infty\}$ |
| $\mathbb{N}_{+,\infty}$ | Natural numbers, excluding 0, including infinity: $\mathbb{N}_+ \cup \{\infty\}$ |
| $\mathbb{Z}$ | Integers |
| $\mathbb{R}$ | Real numbers |
| $[0,1]$ | Closed domain from 0 to 1: $\{x \in \mathbb{R} \mid 0 \leq x \leq 1\}$ |
| $\emptyset$ | Empty set |
| $\mathcal{P}(X)$ | Powerset of $X$ |
| $f\colon X \to Y$ | Function definition for $f$ with domain $X$ and range $Y$ |
| $f\colon X \rightharpoonup Y$ | Partial function definition for $f$ with domain $X$ and range $Y$ |
| $\mathit{Dist}\colon X \to [0,1]$ | Probability distribution over $X$ |
| $[P]$ | Iverson bracket: 1 if P is true, 0 otherwise |
| $\boldsymbol{x} = \langle x_1, x_2 \rangle$ | Tuple with elements $x_1$ and $x_2$ |
| $\mathcal{M} = \langle \mathcal{S}, s_I, \mathcal{A}, \delta \rangle$ | Markov decision process (Definition 1) |
| $\mathcal{S}$ | State space, finite set of states (Definition 1) |
| $s_I \in \mathcal{S}$ | Initial state (Definition 1) |
| $\mathcal{A}$ | Supported actions, finite set of actions (Definition 1) |
| $\delta\colon \mathcal{S} \to \mathcal{A} \rightharpoonup \mathit{Dist}(\mathcal{S})$ | Transition function (Definition 1) |
| $\alpha_{\mathcal{M}}\colon S \to \mathcal{P}(\mathcal{A})$ | The available actions for a state in MDP $\mathcal{M}$ (Definition 2) |
| $\pi = s_1 a_1 s_2 a_2 \dots$ | Path (Definition 3) |
| $s_i \xrightarrow{\mathcal{M}} s_j$ | $s_j$ is reachable from $s_i$ (Definition 4) in MDP $\mathcal{M}$ |
| $\sigma_m\colon \mathcal{S} \to \mathcal{A}$ | Memoryless strategy (Definition 5) |
| $\sigma_{\mathcal{S}}\colon \mathcal{S} \times \mathbb{N}_0 \to \mathcal{A}$ | Step-positional strategy (Definition 6) |
| $\sigma_{Pm}\colon \mathcal{S} \to \mathit{Dist}(\mathcal{A})$ | Probabilistic memoryless strategy (Definition 7) |
| $\sigma_{Ps}\colon \mathcal{S} \times \mathbb{N}_0 \to \mathit{Dist}(\mathcal{A})$ | Probabilistic step-positional strategy (Definition 8) |
| $\mathit{Strat}_{\mathcal{M}} \subseteq \mathcal{S} \times \mathbb{N}_0 \to \mathit{Dist}(\mathcal{A})$ | All probabilistic step-positional strategies for the MDP $\mathcal{M}$ (Definition 8) |
| $\Pi^{\sigma}_{\mathcal{M}}$ | All paths for a strategy $\sigma$ in MDP $\mathcal{M}$ (Definition 9) |
| $Pr^{\sigma}_{\mathcal{M}}\colon \Pi^{\sigma}_{\mathcal{M}} \to [0,1]$ | Path probability of $\pi$ for strategy $\sigma$ in MDP $\mathcal{M}$ (Definition 10) |
| $\mathit{Struct}_{\mathcal{M}} = \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ | Set of all branch reward-structures (Definition 11) |
| $\rho\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ | Branch reward-structure (Definition 11) |
| $\rho_{\mathcal{A}}\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ | Transition reward-structure (Definition 12) |
| $\rho_{\mathcal{S}}\colon \mathcal{S} \to \mathbb{R}$ | State reward-structure (Definition 13) |
| $\mathit{CumRew}_{\mathcal{M}}\colon \mathit{Struct}_{\mathcal{M}} \to \mathbb{N}_{+,\infty} \times \mathit{Strat}_{\mathcal{M}} \to \mathbb{R}$ | Cumulative reward in MDP $\mathcal{M}$ (Definition 14) |
| $\mathit{LraRew}_{\mathcal{M}}\colon \mathit{Struct}_{\mathcal{M}} \to \mathbb{N}_{+,\infty} \times \mathit{Strat}_{\mathcal{M}} \to \mathbb{R}$ | Long-run average reward in MDP $\mathcal{M}$ (Definition 15) |
| $\mathit{RRew}_{\mathcal{M}}\colon \mathit{Struct}_{\mathcal{M}} \times \mathcal{P}(\mathcal{S}) \to \mathbb{N}_{+,\infty} \times \mathit{Strat}_{\mathcal{M}} \to \mathbb{R}$ | Reachability reward in MDP $\mathcal{M}$ (Definition 16) |
| $\bar{\phi} = [\mathit{Rew}_{\mathcal{M}}]^{\leq k}_{\bar{\square}_c}$ | Achievability property (Definition 18) |
| $\hat{\phi} = [\mathit{Rew}_{\mathcal{M}}]^{\leq k}_{\hat{\triangle}}$ | Numerical property (Definition 17) |
| $\phi = [\mathit{Rew}_{\mathcal{M}}]^{\leq k}_{\bar{\circ}}$ | Achievability or numerical property |

| Notation | Meaning |
|---|---|
| $\mathcal{M}, \sigma \models \bar{\phi}$ | In MDP $\mathcal{M}$, $\bar{\phi}$ is satisfied under strategy $\sigma$ (Definitions 18 and 28) |
| $query_{\mathcal{M}}(\phi)$ | Query in MDP $\mathcal{M}$ (Definitions 19 and 33) |
| $\epsilon$ | Convergence threshold |
| $\langle \mathcal{S}_e, s_{eI}, \mathcal{A}_e, \delta_e \rangle$ | End component (Definition 20) |
| $\mathcal{R} = \langle \mathcal{Q}, q_I, \mathcal{A}, \lambda, \mathcal{C} \rangle$ | Deterministic Rabin automaton (Definition 21) |
| $\mathcal{Q}$ | State space: a finite set of states (Definition 21) |
| $q_I \in \mathcal{Q}$ | Initial state (Definition 21) |
| $\mathcal{A}$ | Supported actions: a finite set of actions (Definition 21) |
| $\lambda \colon \mathcal{Q} \times \mathcal{A} \to \mathcal{Q}$ | Transition function (Definition 21) |
| $\mathcal{C} \subseteq \mathcal{P}(Q) \times \mathcal{P}(Q)$ | Acceptance condition (Definition 21) |
| $\mathcal{M} \otimes \mathcal{R}$ | Composition of MDP $\mathcal{M}$ and DRA $\mathcal{R}$ (Definition 22) |
| $\mathcal{M}_e \models \mathcal{R}$ | $\mathcal{M}_e$ is an accepting end component for DRA $\mathcal{R}$ (Definition 23) |
| $\bar{\Gamma} = [\mathcal{R}]^{\leqq k}_{\square_c}$ | Achievability LTL property (Definition 18) |
| $\hat{\Gamma} = [\mathcal{R}]^{\leq k}_{\hat{\triangle}}$ | Numerical LTL property (Definition 24) |
| $query_{\mathcal{M}}(\Gamma)$ | LTL query in MDP $\mathcal{M}$ (Definition 26) |
| $\boldsymbol{\phi} = \langle \phi_1, \phi_2, \ldots, \phi_n \rangle$ | Multi-objective property (Definition 27) |
| $\bar{\boldsymbol{\phi}} = \langle \bar{\phi}_1, \bar{\phi}_2, \ldots, \bar{\phi}_n \rangle$ | Multi-objective achievability property (Definition 28) |
| $\hat{\boldsymbol{\phi}} = \left\langle \bar{\phi}_1, \ldots \hat{\phi}_j, \ldots, \bar{\phi}_d \right\rangle$ | Multi-objective numerical property, unknown at $j$ (Definition 29) |
| $\tilde{\boldsymbol{\phi}} = \left\langle \hat{\phi}_1, \hat{\phi}_2, \ldots, \hat{\phi}_d \right\rangle$ | Pareto property (Definition 30) |
| $Ach_{\mathcal{M}}(\boldsymbol{\phi}) \in \mathcal{P}(\mathbb{R}^d)$ | The achievable values for Pareto property $\tilde{\boldsymbol{\phi}}$ with $d$ unknowns (Definition 31) |
| $>_{\boldsymbol{\phi}} \in \mathcal{P}(\mathbb{R}^d \times \mathbb{R}^d)$ | Dominating relation under multi-objective property $\boldsymbol{\phi}$ with $d$ subproperties (Definition 32) |
| $\perp$ | No value, null |
| $\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$ | Vector form of $\langle x_1, \ldots x_n \rangle$ |
| $M = \begin{bmatrix} M_{1,1} & \cdots & M_{1,n} \\ \vdots & \ddots & \vdots \\ M_{m,1} & \cdots & M_{m,n} \end{bmatrix} \in \mathbb{R}^{mn}$ | $m \times n$ matrix |
| $M^T \in \mathbb{R}^{nm}$ | Transpose matrix of $M \in \mathbb{R}^{mn}$ |
| $H = \langle \boldsymbol{a}, b \rangle$ | Hyperplane satisfying $\boldsymbol{ax} = b$ (Definition 35) |
| $\mathcal{H}_d$ | The set of all $d$-dimensional hyperplanes (Definition 35) |
| $\boldsymbol{w} \in [0,1]^d$ | Weight vector for convex combination with $d$ variables (Definition 34) |
| $conv(X) \in \mathcal{P}(\mathbb{R}^d)$ | Set of all points in the convex hull of $X \in \mathcal{P}(\mathbb{R}^d)$ (Definition 36) |
| $\langle F, \boldsymbol{c}, \langle T, T^{-1}, SC \rangle \rangle$ | Convex hull (Definition 37) |
| $down(X) \in \mathcal{P}(\mathbb{R}^d)$ | The downward closure of $X \in \mathcal{P}(\mathbb{R}^d)$ (Definition 38) |