The present work was submitted to the LuFG Theory of Hybrid Systems

BACHELOR OF SCIENCE THESIS

SOLVING PLANNING PROBLEMS WITH SUCCESS PROBABILITIES USING SMT

Leon Rabanus

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Dr. Gerhard Lakemeyer
Additional Advisor:
Francesco Leofante

Aachen, January 23, 2019

Abstract

Planning as satisfiability and planning with first-order logic in contrast to other planning approaches have become more popular in research in the past years. Still, most of the problems consider deterministic planning only, and those that do not tend to deal with problems other than to find a most probable solution. In this thesis, an algorithm is presented that can handle probabilistic domains with discrete probability distributions. It utilizes SMT to determine a plan with maximum success probability. Two simple algorithms are introduced for comparison. The aim of this work is to elaborate and use a suitable algorithm to identify use cases for finding most probable solutions and for applying probabilities in planning with SMT. The usage of SMT for probabilistic planning problems is examined in comparison to current planners for RDDL planning tracks of the International Planning Competition.

Eidesstattliche Versicherung

Name, Vorname

Matrikelnummer (freiwillige Angabe)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum

Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum

Unterschrift

Contents

Chapter 1

Introduction

Planning software can be used to generate instructions for autonomous robots, to navigate or to calculate a schedule for a project. As the planning domains of the International Planning Competition show (see Chapter 6 and Appendix A.4), planning problems can become very complex. There have been many approaches to formalize and solve these problems in computer science in the past decades. One of them, planning as satisfiability, relies on propositional logic and SAT solvers, and was elaborated as an alternative to deductive planning. Planning using satisfiability modulo theories (SMT) with SMT solvers like Z3 has been examined as well.

Still, probabilities have not yet been used in planning using SMT on a larger scale. Other planners, like the ones for the probabilistic tracks of the International Planning Competition, rely on heuristic tree searches, neural networks or similar ideas. As planning as satisfiability has some advantages over a deductive approach, planning using SMT might benefit from similar properties compared to other planning algorithms. Therefore, this thesis is meant to contribute to the discussion of probabilistic planning by introducing basic ideas on how to integrate probabilities in SMT solving: These include a translation from a planning language called RDDL to the language used by Z3, the integration of probabilistic statements in Z3 and a simple algorithm to generate a plan with high probability of success. The performance of the algorithm is compared to other planners that do not rely on SMT. The results can be used as a foundation for further discussion of the topic.

Background knowledge about planning as satisfiability, SMT, OMT, Z3, MDPs, the International Planning Competition (IPC) and RDDL is provided in Chapter 2. In Chapter 3, the problem examined in this thesis is formally defined. In Chapter 4, other concepts for planning in the IPC, planning using SAT and SMT and a symbolic graph search are examined. This allows for a brief comparison to other approaches and indicates the interest in the field to solve probabilistic planning problems. The general idea of the algorithm on an MDP is presented in Chapter 5. The method is also explained in form of constraints for an SMT solver. The chapter also includes a brief description of the translation of RDDL statements to Z3, especially of probabilistic statements, which need to be treated differently. Other simpler

methods and an older idea that could not be applied are mentioned as well. The resulting algorithm is then evaluated with different parameters in comparison to other planners that can handle RDDL domains in Chapter 6. Seven domains have been chosen and are analyzed separately, with focus on the performance and usefulness of the algorithm and whether it behaves correctly. The results are discussed at the end of the chapter. In Chapter 7, a conclusion is drawn to assess the use of SMT in probabilistic planning, the quality of the algorithm and the actual implementation that was tested and to give ideas for further improvements.

The publications in [BT18] and [KS92] that are cited in the following chapter could not be compared with the online versions of these papers, which were used as sources for the citations. Thus, page numbers were given for the latter instead, which are referred to in a note in the bibliography to allow for a comparison.

Chapter 2

Preliminaries

This chapter provides basic knowledge necessary for the understanding of the following discussion.

2.1 Satisfiability Modulo Theories (SMT)

Satisfiability Modulo Theories consider the satisfiability of first-order formulas given a theory T in *first-order logic* that is decidable [Seb07, p. 141], including e.g. the theory of linear arithmetic over the reals or integers [Seb07, p. 144]. The limitation to models of a theory can, as semantic restriction on the formula, make the satisfiability of such a formula *decidable* as well [BT18, see p. 2]. Thus, SMT can be seen as a restriction on first-order logic to decidable formulas. This approach allows for "specialized satisfiability procedures that exploit properties of the fragment" [BT18, p. 2] - some theories might have properties that can be used to speed up the search for a satisfying assignment. These procedures can bring a "great advantage for practical efficiency" [BT18, p. 2], so especially problems that are hard to solve in praxis could benefit from the usage of SMT.

$$\begin{aligned} \varphi &:= (a > 5) \wedge (a < 20) \\ \varphi' &:= \quad A \quad \wedge \quad B \end{aligned} \tag{2.1}$$

Problems expressed in SMT can, referring to the so-called "'lazy approach'" [BT18, p. 3], be solved with a SAT solver and specialized solvers for the used theories [BT18, p. 3]: Assume that a T-formula φ needs to be solved. The input formula's atoms are replaced by new propositional variables and the resulting formula φ' can be passed to a SAT solver [BDS02, p. 241]. An example for this procedure is shown in Equation (2.1). If a model for φ' is found, a theory solver for T checks if it is consistent to the theory [ST14, p. 5]. A model of the original problem was determined if this consistency is given. Otherwise, an additional constraint, taking a with T conflicting subset of the assignment into account, is added to the formula and the process is repeated [ST14, p. 5]. Thus, inconsistent models are excluded, until either a

solution has been found or the formula is unsatisfiable, which means that φ is T-unsatisfiable [BT18, pp. 9–10].

SMT is an approach that allows for a higher expressiveness than SAT due to the possibility to use features of first-order logic while still being decidable [BT18, p. 2]. It has already been used to solve problems in the industry [ST17, p. 1], for example in the field of automated reasoning [ST14, p. 1], and its utility in planning is still being researched (see Section 2.3, Section 2.4 and Section 4.3).

Due to the expressiveness of SMT and the advantage of specialized procedures as well as the interest of using SMT in planning, which is shown in Chapter 4, an examination of its usage in the context of probabilistic planning problems seemed reasonable.

2.2 Optimization Modulo Theories (OMT) and Z3

OMT extends SMT by allowing to "*optimize* given objectives" [ST17, p. 1], for example by using a cost variable in φ that needs to be minimized [ST12, p. 8]. Not any satisfying model, but instead an optimal model according to a cost function is desired.

Consider a problem specified by a satisfiable formula φ, given a background theory T, where the value of an integer variable *cost* should be minimized. A realization of an OMT procedure with only an upper bound could rely on $\varphi' = \varphi \wedge (cost < c_i)$ (see [ST12, p. 8]). The formula would be repeatedly passed to an SMT solver, each time with a lower value for c_i [ST12, p. 9]. In this example, the model satisfying φ with the lowest cost (if any exists) would be found [ST17, see p. 4]. Such procedures are "very naive" [ST12, p. 9], but they provide a basic understanding of the principle of OMT.

An example of its usage in νZ, "part of the SMT solver Z3" [BPF15, p. 194], which is used in this thesis, is given in Figure 2.1. νZ "allows users to solve SMT constraints and at the same time formulate optimality criteria for the solutions" [BPF15, p. 194] and thus includes an OMT procedure. In the example given in Figure 2.1, OMT is used to maximize the objective function $b - a$ under the constraints $a > 5$, $a < 20$, $b < 10$ and $a < b$. The interpretation with $a = 6$ and $b = 9$ is the optimal solution. The encoding in Z3 is shown in Figure 2.1a, the output of the solver in Figure 2.1b. This small example indicates the capabilities of OMT regarding objectives that are supposed to be optimized, which can be much more complex than demonstrated here, and, using a solver like νZ, can also be combined [BPF15, p. 195].

It is important to note that the solver can be used "for solving linear optimization problems" [BPF15, p. 194] and does not seem to be applicable for non-linear objective functions. Yet, advances in this area are anticipated by the authors of the paper [BPF15, p. 196]. As the graphic in an online chapter on optimization in Z3 [BdMNW] indicates, which describes νZ's architecture in [BPF15, p. 197], Z3 might thus currently *not support non-linear optimization*. This is important for the evaluation in Chapter 6 and the choice of the algorithm.

```
(declare−fun a () Int)
(declare−fun b () Int)
(assert (> a 5))
(assert (< a 20))
(assert (< b 10))
(assert (< a b))
(maximize (− b a))
(check−sat)
(get−model)
```

(a) OMT example in Z3

```
sat
(model
    (define−fun b () Int 9)
    (define−fun a () Int 6)
)
```

(b) Output given by Z3

Figure 2.1: Z3 example code and output

2.3 Planning as Satisfiability

Planning as Satisfiability is a method of planning using *propositional satisfiability*, which was introduced as an alternative to deduction [KS92, pp. 1–2]: Conditions and actions are formalized as axioms [KS92, p. 5], and it is "easy to specify conditions in any intermediate state" [KS92, p. 6], giving more control over desired plan properties. If specified correctly, models for a thereby resulting formula found by a SAT solver can be used as valid plans for the planning problem.

$$(precondition \wedge action) \Rightarrow effect \tag{2.2}$$

In order to formalize a plan in propositional logic, the formalization must be stronger than in deductive reasoning [KS92, p. 10]: Using deduction, initial conditions and actions together imply the goal conditions [KS92, p. 4], but a model that satisfies these deductive planning axioms alone must not represent a valid plan in the context of the planning domain [KS92, p. 4]. It can also be *"anomalous"* [KS92, p. 4], meaning that the plan found must not make sense in the world it should be applied to: For example, in Equation (2.2) the action and the effect can be interpreted as being true if the precondition is not satisfied. For the planning as satisfiability approach to work, it is therefore important to fully specify the initial state [KS92, p. 6] and all axioms necessary to encode the desired requirements as constraints on the plan [KS92, compare p. 5]. Thus, *problems must be formalized differently* than in deductive reasoning to obtain valid plans.

$$on(A,B,1) \wedge on(B,Table,1) \wedge clear(A,1) \wedge on(B,A,3) \tag{2.3}$$

To give an example, Equation (2.3) from [KS92, p. 5] describes the initial and the goal state of two blocks A and B. At time 1, A is on B, B is on the table and A is clear, meaning that no other block is on top of A at that time. At time 3, B is on A. By using this notation, implications of actions on the world's state can be formalized as well.

The satisfiability problem can be solved by any SAT solver. Using GSAT, the authors of [KS92, p. 9] noticed that adding additional axioms which could be derived from the initial

axioms can improve the performance of the computation.

2.4 Planning Using SMT

Based on the idea of planning as satisfiability, plans can also be generated using SMT or OMT in a similar manner, which can be interesting due to the expressiveness of the language. Planning problems defined in a planning language like PDDL+ can even be translated to and solved by SMT solvers [CFLM16, p. 79]. Given a set of constraints over a domain D and a set of theories T, properties of and actions in a problem definition, e.g. of navigating a robot in a small room that contains obstacles, can be formalized in first-order logic. A solver like Z3 can then be used to find a model for the formalization. Z3 is used by [CFLM16, p. 85] to find a proof, and a plan can be obtained by using the assignment of all action variables in it [CFLM16, p. 82].

2.5 Markov Decision Process

2.5.1 Transition Systems and Planning

Definition 2.5.1 (p. 20 in [BK08])
A *transition system* TS is a tuple $(S, Act, \longrightarrow, I, AP, L)$ where

- S is a set of states,
- Act is a set of actions,
- $\longrightarrow \subseteq S \times Act \times S$ is a transition relation,
- $I \subseteq S$ is a set of initial states,
- AP is a set of atomic propositions, and
- $L : S \to 2^{AP}$ is a labeling function.

A transition system is similar to a *directed graph*, where *states*, which represent the properties of the modeled system at one point in time, are represented by nodes and *nondeterministic transitions* between these states by edges [BK08, pp. 19–20]. Transitions are labeled with action names [BK08, p. 20]. The initial state is drawn nondeterministically [BK08, p. 20].

A basic knowledge of transition systems is necessary to understand Markov Decision Processes (MDPs), which are used to explain the algorithm presented in Chapter 5. In Subsection 2.5.3, a concept called *scheduler* will be introduced, and with such a scheduler a deterministic behavior can be obtained.

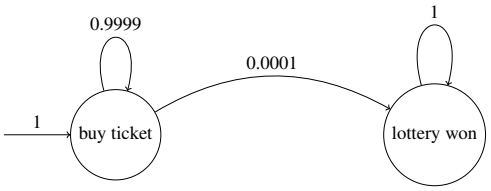

Figure 2.2: A simple Markov chain

2.5.2 Markov Chains and Rewards

Definition 2.5.2 (see pp. 747–748 in [BK08])
A *Markov chain* is defined as a tuple $(S, P, \iota_{\text{init}}, AP, L)$ where

- S is a countable, nonempty set of states, AP is a set of atomic propositions, $L : S \to 2^{AP}$ is a labeling function
- $P : S \times S \to [0,1]$ is the *transition probability function* where for all $s \in S$:

$$\sum_{s' \in S} P(s,s') = 1$$

- $\iota_{\text{init}} : S \to [0,1]$ is the initial distribution with $\sum_{s \in S} \iota_{\text{init}}(s) = 1$.

If, instead of nondeterministic transitions and actions like in transition systems, a *probability distribution* given by P determines the successor of a state $s \in S$ [BK08, p. 747], depending only on s [BK08, p. 747], that is called a Markov chain. ι_{init} is similar to I in transition systems, but the initial state is chosen probabilistically [BK08, p. 748]. Edges are labelled with the corresponding probability [BK08, p. 749] instead of action names.

An example for a Markov chain is given in Figure 2.2. It models a customer that buys new lottery tickets until he wins the lottery, which happens with a probability of 0.01%. As soon as the lottery is won, the customer stops buying new tickets. The initial distribution only includes the state "buy tickets".

A Markov chain $\mathcal{M}$ can be extended by a *reward function rew*, and the resulting model (M, rew) is called a Markov reward model, with $rew : S \to \mathbb{N}$ [BK08, p. 817]. The reward $rew(s)$ for a state $s \in S$ is earned as soon as it is left [BK08, p. 817].

Rewards will play an important role in RDDL domains, as the goal that needs to be achieved is only defined implicitly by the reward function.

2.5.3 Markov Decision Processes and Schedulers

Definition 2.5.3 (see pp. 833–834 [BK08])
A *Markov decision process* (MDP) is a tuple $(S, Act, P, \iota_{\text{init}}, AP, L)$ where

- S is a countable, nonempty set of states, Act is a set of actions, AP is a set of atomic propositions, $L : S \to 2^{AP}$ is a labeling function

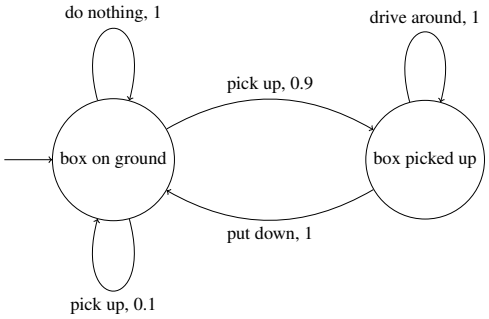

Figure 2.3: A simple MDP

- $\iota_{\text{init}} : S \to [0,1]$ is the initial distribution with $\sum_{s \in S} \iota_{\text{init}}(s) = 1$.
- $P : S \times Act \times S \to [0,1]$ is the transition probability function where for all $s \in S$ and $\alpha \in Act$:

$$\sum_{s' \in S} P(s, \alpha, s') \in \{0, 1\}$$

Act here is the set of actions for which the sum above is 1. Furthermore, *Act* is extended in each state by a "noop" action that always succeeds. It does not have any effect on the current state of the MDP. If such an action is not allowed by an RDDL instance, then the sum above must be 1 for at least one action in each state to prevent deadlocks.

MDPs can be interpreted as a combination of transition systems and Markov chains. S, Act, ι_{init}, AP and L thus have the same properties as described before. The transition function now allows for *nondeterministic and probabilistic* transitions [BK08, p. 832]. For any state $s \in S$ where more than one action can be taken, meaning $\sum_{\alpha \in Act} \sum_{s' \in S} P(s, \alpha, s') > 1$, an action $\alpha \in Act$ needs to be chosen nondeterministically [BK08, p. 834], and from all outgoing α-transitions of s one transition is chosen according to the probability distribution given by P over α [BK08, p. 834].

Edges where the probability given by P is 0 are omitted. Edges are labeled with the action name and the probability of the transition.

The MDP given in Figure 2.3 considers a robot that can perform actions like "pick up". This action succeeds with a probability of 90% and fails with a probability of 10%. If it is selected nondeterministically, the outcome of the action is therefore uncertain. In order to give a solution for planning problems formalized with an MDP, it is important to *define* what a *solution* in an MDP looks like. The choice between different actions does not need to be resolved nondeterministically. If certain states are defined as goal states and a *plan* can be generated that deterministically selects an action according to the current state and, by following the path determined by the plan, a goal state is reached from an initial state, then this is a solution for the given instance.

Definition 2.5.4 (see p. 841 [BK08])
For an MDP $\mathcal{M} = (S, Act, P, \iota_{\text{init}}, AP, L)$, the infinite sequence $s_0 \alpha_1 s_1 \alpha_2 \ldots \in (S \times Act)^\omega$ is

called an infinite *path*. It is written as

$$\pi = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \dots$$

with $P(s_i, \alpha_{i+1}, s_{i+1}) > 0$ for all $i \geq 0$. The set of all infinite paths starting in state s is given by $Paths(s)$, the set of finite paths - all distinct finite sequences π' in all paths $\pi \in Paths(s)$ - by $Paths_{fin}(s)$.

Example 2.5.1

$$\text{box on ground} \xrightarrow{\text{pick up}} \text{box picked up} \xrightarrow{\text{drive around}} \text{box picked up}$$

Definition 2.5.5 (see p. 842 [BK08])
Let $\mathcal{M} = (S, Act, P, \iota_{\text{init}}, AP, L)$. A (*deterministic*) *scheduler* for $\mathcal{M}$ is a function $\mathfrak{S} : S^+ \to Act$ such that $\mathfrak{S}(s_0 s_1 \dots s_n) \in Act(s_n)$ for all $s_0 s_1 \dots s_n \in S^+$. $\pi = s_0 \xrightarrow{\alpha_1} \dots$ is called a $\mathfrak{S}$-path if $\alpha_i = \mathfrak{S}(s_0 \dots s_{i-1})$ for all $i > 0$.

Schedulers are used for the formalization of a plan. They contain the rules that decide which action to take next, beginning in any starting point $s_0 \in S$. Example 2.5.1 is a finite path that represents one of many possible executions that could have been planned by the robot's planning software. It could also be interpreted as the beginning of an infinite $\mathfrak{S}$-path of any scheduler $\mathfrak{S}$ where $\mathfrak{S}(\text{box on ground}) = \text{pick up}$ and $\mathfrak{S}(\text{box on ground, box picked up}) = \text{drive around}$. If "box picked up" were a goal state, these schedulers could even be interpreted as solutions for the domain. But such a scheduler does not always reach a goal state, as "pick up" is an action that might infinitely often fail. Thus, schedulers do not induce paths, as, after the resolution of nondeterminism in an MDP, probabilistic statements remain.

A scheduler induces a Markov chain $\mathcal{M}$ instead (e.g. Figure 2.4). The probability that the finite path given in Example 2.5.1 is taken on $\mathcal{M}$, beginning at the initial state, is 90%, as the first action fails with a probability of 10%.

Definition 2.5.6 (see p. 843 [BK08])
A *scheduler* $\mathfrak{S}$ on an MDP $\mathcal{M} = (S, Act, P, \iota_{\text{init}}, AP, L)$ *induces the Markov chain*

$$\mathcal{M}_{\mathfrak{S}} = (S^+, P_{\mathfrak{S}}, \iota'_{\text{init}}, AP, L')$$

where for $\sigma = s_0 s_1 \dots s_n$: $P_{\mathfrak{S}}(\sigma, s_{n+1}) = P(s_n, \mathfrak{S}(\sigma), s_{n+1})$ and $L'(\sigma) = L(s_n)$. The induced initial distribution is

$$\iota'_{\text{init}} : S^+ \to [0,1], \sigma \mapsto \iota_{init}(s_n) \text{ for } \sigma = s_0 \dots s_n \in S^+$$

Let $\mathfrak{S}_{exec}$ be a scheduler where $\mathfrak{S}_{exec}(s_0 \dots s_n) = \text{pick up}$ if $s_n = \text{box on ground}$ and $\mathfrak{S}_{exec}(s_0 \dots s_n) = \text{drive around}$ otherwise. Figure 2.4 shows the beginning of the Markov chain induced by $\mathfrak{S}_{exec}$ on the MDP in Figure 2.3.

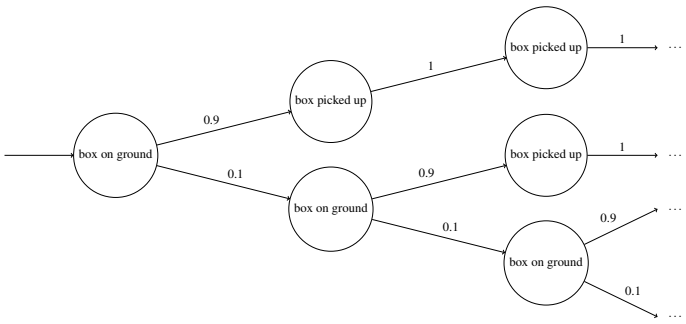

Figure 2.4: Markov chain induced by a scheduler on the MDP in Figure 2.3

2.6 International Planning Competition

In order to evaluate the solution presented in this thesis, it is compared with other planners that can operate on MDPs used in the International Planning Competition.

The International Planning Competition (IPC) has taken place multiple times since 1998. One of its benefits is to support the development of planners [CCO+12, see p. 85]. Submitted planners and the gathered data and benchmarks generated during the competition are supposed to be advantageous for the planning community [CCO+12, see p. 85]. It was thus chosen as the basis for comparison.

In 2011, the tracks of the IPC included deterministic, learning and uncertainty tracks [CCO+12, p. 83]. The latter, called probabilistic tracks in 2018 [ippc], are of interest in this thesis. First, in 2004, PPDDL, whose problems can be represented as an MDP, was used [YLWA05, see p. 851, p. 853, p. 854], but since 2011, PPDDL was replaced by RDDL [CCO+12, p. 84], which can be used to model new problems [San10, see p. 2]. The performance metric for planners of the 2011 competition was based on the received reward [CCO+12, p. 87].

The probabilistic part of the IPC will be abbreviated as *IPPC*.

2.7 RDDL

RDDL (Relational Dynamic Influence Diagram Language) is a language that is used by the IPPC to describe *probabilistic planning problems*. It supports for example concurrency, multiple variable types, different probability distributions, a reward function and logical and arithmetic expressions [San10, pp. 3–4]. As partial observability will not be considered here, all presented instances could be represented by factored MDPs as well [San10, p. 1]. Because Section 2.5 suffices for the explanation of RDDL and the algorithm in Section 5.1, the concept of factored MDPs will not be introduced.

In RDDL, domains, non-fluents and instances of domains can be declared in different blocks [San10, p. 17]. A *domain* is initially given a *name* and *requirements* [San10, p. 6], for example that the reward is deterministic. It also defines variables, conditional probabilistic

```
types {
    robot: object;
};
```

(a) Object definitions

```
objects {
    robot : {robot1, robot2, robot3};
};
```

(b) Object instantiations

Figure 2.5: User-defined objects in RDDL

functions (cpfs), state-action constraints, action-preconditions, types and a reward function [San10, pp. 18–20].

Variables, which are called *pvariables*, are parameterized [San10, p. 3] and can have one of the ranges `bool`, `int`, `real`, or are an object or enumerated [San10, p. 18]. Objects are defined by the user as shown in Figure 2.5. The only pvariable types of interest are `state-fluent` and `interm-fluent` (of level 1), which describe a property of the system that can dynamically change in each step, `non-fluent`, which describes a property of a system that does not change, and `action-fluent`, which describes actions that can be made by agents in the domain. These variables are usually declared including default values [San10, p. 18], which are, if not specified otherwise in a non-fluent or instance block, where non-fluents can be instantiated or initial states can be set [San10, p. 20], their initial values. The latter blocks also define `objects`, specify the planning horizon in `horizon`, the maximum amount of (concurrent) actions that are allowed to be taken in a time step in `max-nondef-actions` and the discount on the reward function in `discount` [San10, see p. 20].

The block *cpfs* is used to specify *transitions* for state-fluents and intermediate-fluents [San10, p. 18]. Primed pvariables represent values of the state-fluent in the next state of the system, unprimed pvariables the value in the current state [San10, see p. 18]. They can be assigned constants like `true` or `false`, integers, reals and enums and be used as part of basic logical expressions including existential and universal quantifiers, basic arithmetic expressions including `sum` and `prod` for $\sum$ and $\prod$, (in)equality comparisons and conditional expressions [San10, pp. 18–19]. To support probabilistic domains, probability distributions are part of the syntax as well [San10, p. 19]. Four of these distributions are of interest: `KronDelta(val)` and `DiracDelta(val)` state, for discrete or continuous domains, that the probability that the given argument `val` in an assignment will become the next value of a pvariable is 1.0 [San10, see p. 19]. For convenience, they are omitted and `val` will be written instead. `Bernoulli(p)`, on the other hand, represents a function that becomes true with a probability given by the statement p, where p can have values in $[0,1]$, and that is false otherwise [San10, see p. 19]. `Discrete` describes a discrete distribution between different values, for which probabilities that sum up to 1.0 are given [San10, see p. 19].

State-action constraints and *action-preconditions* provide an option to specify additional constraints in the domain that are independent of the transition functions. The *reward* expression is a function that, as the performance metric indicates [CCO+12, see p. 87], is supposed to be used for optimization. Goal states are not defined explicitly.

An example of a translation of the simple MDP in Figure 2.3 into a similar RDDL file

is shown in Figure 2.6 and Figure 2.7. Parts irrelevant for the understanding of the RDDL syntax were omitted, indicated by "...".

All planning problems considered in this thesis are written in and translated from RDDL files. Apart from finding a most probable path in an MDP represented by an RDDL problem, it was thus also relevant to elaborate a straightforward translation of the most important parts of the language to Z3. Due to the expressiveness of RDDL, anomalous models can be ruled out in the planning language itself.

2.7.1 Used Tools

RDDL files are parsed with the parser provided by Scott Sanner on GitHub, which is a part of RDDLSim [rdd]. The parser object is used in the project to translate the RDDL file to Z3. The RDDL instances can be simulated using RDDLsim [rdd], which is used in Chapter 6 to evaluate the proposed algorithms and planners of the IPPC 2018.

Z3 was chosen as SMT solver for this project because it is open source, has a good documentation and is easy to use. The solver can be downloaded at [z3G]. Its basic usage is described in Section 2.2. More documents on Z3 can be found in [z3M].

```
domain simple_mdp {
    ...
    types {
        box : object;
    };

    pvariables {
        //non−fluent, the default probability of success for picking up a box is 80%
        PROB−PICKUP : { non−fluent, real, default = 0.8 };

        //state−fluents, the default position of a box is unspecified
        on−ground(box) : { state−fluent, bool, default = false };
        picked−up(box) : { state−fluent, bool, default = false };
        pick−up−action(box) : { interm−fluent, bool, level = 1 };

        //action−fluents
        do−nothing(box) : { action−fluent, bool, default = false };
        pick−up(box) : { action−fluent, bool, default = false };
        put−down(box) : { action−fluent, bool, default = false };
        drive−around(box) : { action−fluent, bool, default = false };
    };

    cpfs {
        on−gound(?b)' = if (pick−up−action(?b))
                            then false
                        else if (put_down(?b))
                            then true
                        else on−ground(?b);

        picked−up'(?b)= if (pick−up−action(?b))
                            then true
                        else if (put_down(?b))
                            then false
                        else picked−up(?b);

        pick−up−action(?b)= if (pick−up(?b))
                                then Bernoulli(PROB−PICKUP)
                            else false;
    };

    reward = ... //Some reward function

    state−action−constraints {
        //Some actions are not allowed to be taken in certain situations
        forall_{?b : box} picked−up(?b) => ~do−nothing(?b) ^ ~pick−up(?b);
        forall_{?b : box} on−ground(?b) => ~drive−around(?b) ^ ~put−down(?b);
    };
}
```

Figure 2.6: RDDL file extract for Figure 2.3, domain. This example includes the action pick-up, which is made probabilistic and applied in the following step through the interm-fluent pick-up-action.

```
non−fluents simple_mdp_nf {
    domain = simple_mdp;
    objects {
        box: { box1 };
    };
    non−fluents {
        PROB−PICKUP = 0.9;
    }
}

instance simple_mpd_inst {
    domain = simple_mdp;
    non−fluents = simple_mdp_nf;
    init−state {
        on−ground(box1);
    }
    max−nondef−actions = 1;
    horizon = 10;
    discount = 0.8;
}
```

Figure 2.7: RDDL file extract for Figure 2.3, instance

Chapter 3

Problem Statement

The topic of this thesis is to contribute to the discussion of probabilistic planning using SMT by finding a method to combine an SMT solver with a procedure that allows for representing and solving probabilistic planning problems. While planning as satisfiability as well as planning with SMT already have been investigated in other papers, there is yet not much to be found concerning probabilistic planning with SMT (see Chapter 4). Thus, a basic algorithm to solve a certain subset of probabilistic planning problems using SMT solving is elaborated. It is compared to other planners that operate on domains which are specified in RDDL. Thus, a translation from RDDL to Z3's SMT representation is implemented as well.

3.1 The Problem

Probabilistic planning problems describe domains where certain actions, state transitions or decisions are of probabilistic nature, for example because actions might fail or because unforeseeable state transitions like a coin flip might be observed. The set of probabilistic planning domains that is of interest in this thesis can be described by MDPs (see Section 2.5).

Only plans that fulfill three criteria are of interest: A plan is only valid or *successful* if an agent that executes it reaches a goal state. The goal state does not need to be the final state, if not specified otherwise in the domain. Furthermore, all *nondeterministic behavior must be resolved* by the plan, similar to a scheduler, but with the option to remain in a state - if allowed by the domain - by choosing no action at all. Finally, the taken path from the initial to the goal state should be the *most probable path* of all paths from the initial to any goal state. Thus, a path in the MDP with maximum probability of success is desired. As nondeterminism needs to be resolved first, the most probable path of all finite paths within the search horizon of all Markov chains that are induced by any scheduler $\mathfrak{S}$ in $\mathcal{M}$ is searched for.

```
state−var−1 : { state−fluent, int, default = 0 };
...
action : { action−fluent, bool, default = false };
perform−action : { interm−fluent, bool, level = 1 };
...
perform−action = if (action)
                    then Bernoulli(0.7)
                 else false;

state−var−1' = if (perform−action)
                    then state−var−1 + 1
               else state−var−1;
state−var−2' = if (action ^ Bernoulli(0.7))
                    then state−var−1 + 1
               else state−var−1;
...
```

Figure 3.1: RDDL: simulation of probabilistic actions

3.2 Restrictions

The set of domains that are of interest in this thesis is restricted. *Only discrete probability distributions* are considered: Coin flips or dice rolls can be encoded but generating a real valued random variable between 0 and 1 is not possible.

In RDDL, the representation of actions with probabilistic outcome is not intuitive. Only state transitions can be probabilistic. All probabilistic changes of variables can be interpreted as concrete actions that are executed by the environment if the precondition of the transition is satisfied. They might not depend on decisions made by the agent but can also be related to properties of the environment only. Still, actions with probabilistic outcomes can be simulated, if desired, as shown in Figure 3.1, where an action with success probability of 0.7 is defined: The action can either influence the value of an intermediate fluent with the given probability and thus have an effect on the state variable in the next step, or a Bernoulli constraint is used in conjunction with the action-fluent in the same step - the latter only works if the action is used only once in all constraints.

The planner will always consider all probabilistic state changes that are part of a time step to calculate the success probability of that step. The probability of success of a time step in the domain is the *product of all probabilities of all state changes* that are made in this step. This means that for example probabilities specified on state variables that do not influence whether a goal state can be reached are still part of the calculation of the algorithm that searches for a most probable path. This interpretation can be problematic depending on the domain, as shown in Figure 3.2a: Here, every single state transition could depend on the outcome of the Bernoulli variable, but the transition probability would be 1.0 as soon as state-var-1 becomes true. These statements are thus not properly supported. Figure 3.2b shows an encoding of the same transition that is unproblematic regarding the proposed interpretation of probabilities. There, the transition is only considered to be probabilistic if a precondition is satisfied.

```
1 state−var−1' = state−var−1 | Bernoulli(0.5);
```

(a) Problematic Bernoulli statement

```
1 state−var−1' = if (state−var−1)
2                then true
3                else Bernoulli (0.5);
```

(b) Statement with same semantics, but different interpretation

Figure 3.2: Problematic Bernoulli statements and possible solutions

Probabilities are also interpreted incorrectly if the effect of the transition can be reverted by another action with another probability of success. If "state-var-1" needs to be false to reach a goal state, but an action can always change its value to false if it became true, then the probability given in the else-clause in Figure 3.2b would become meaningless.

The most probable path regards neither the semantics of actions nor alternatives if these actions fail. Whether the presented solution is useful is thus highly dependent on the domain. Therefore, only in domains where probabilities directly influence whether a goal state might be reached or not, and where probabilistic outcomes cannot be reverted by other actions, the presented solution makes sense semantically as well. Still, it might be interesting to see how good such a planner performs compared to planners that regard these issues, so this loose restriction does not apply to all RDDL domains tested in Chapter 6. While this approach is sufficient to introduce new ideas for treating probabilistic planning problems in SMT, it should be examined further in future work so that more domains can be supported.

For the sake of simplicity, the initial state is supposed to be unique. If more than one initial state exists, the algorithm can be applied to each one individually and a plan for each outcome can be calculated.

Other restrictions that are connected to RDDL syntax or semantics will be discussed in Chapter 5.

3.3 Formal Problem Definition

Definition 3.3.1 (Plan)
Let $\mathfrak{S}$ be a scheduler for an MDP $\mathcal{M}$ and h be the horizon for the planning problem. Any finite $\mathfrak{S}$-path π in $\mathcal{M}$ that is within h is a plan for $\mathcal{M}$. The set of all plans in $\mathcal{M}$ will be denoted as $Plans(\mathcal{M})$.

Definition 3.3.2 (Probability of Success of a Plan)
Let $\pi = s_0 \xrightarrow{\alpha_1} s_1 \ldots \xrightarrow{\alpha_{n-1}} s_n$ be a plan for an MDP $\mathcal{M} = (S, Act, P, \iota_{\text{init}}, AP, L)$. The probability of success of π is $p(\pi) = \prod_{(s,\alpha,s')\in\pi} P(s,\alpha,s')$ iff there exist $i, j \in [0,n], i \leq j$ so that s_i is an initial state and s_j is a goal state, else $p(\pi) = 0$.

Definition 3.3.3 (Maximum Plan / Plan with Maximum Probability of Success)
Let $\mathcal{M}$ be an MDP. π is a maximum plan in $\mathcal{M}$ iff $\forall_{\pi' \in Plans(\mathcal{M})} p(\pi') \leq p(\pi)$.

The problem to be solved is to think of and implement a translation of probabilistic planning problems to SMT, to develop an algorithm that finds a maximum plan in an MDP, to use it to solve RDDL domain instances under the restrictions mentioned above, and to compare its performance with other RDDL planners to elaborate its use for planning in probabilistic domains.

Chapter 4

Related Work

A lot of research has already been conducted in the field of probabilistic planning and planning using SAT or SMT solvers. This chapter considers four topics related to the problem specified in Chapter 3: Planners of the IPPC for RDDL domains are described and compared, examples are given of how probabilities are integrated in SAT solving, other approaches of planning using SMT solvers are discussed and a solution is mentioned that could have been implemented instead of the algorithm proposed in Chapter 5.

4.1 Planning Systems of the IPPC

There already exist planners for the RDDL language, made by participants of the IPPC. Most of them use heuristics and approaches based on some form of graph representation or neural networks. The variety of solutions shows that there is an interest in solving planning problems similar to those representable in RDDL. They provide a comparison to the problem considered in this thesis and to the chosen approach to solve it.

The winner of the IPPC 2018 for discrete MDP tracks is the planner *Prost-DD-1* [ippc]. It uses a *tree search* called THTS [GS, p. 1] (for more information see [KH13]) with two competing heuristics that are combined to solve more domains [GS, see p. 3]. In THTS, values of nodes in the search tree are being calculated in so-called trials until either a timeout occurred [KH13, p. 137] or an optimal value for the root node has been found [GS, p. 1]. In each trial, the tree can be expanded, using the chosen heuristic function to award values to new nodes, and a backup function is used in another phase of the trial to update the values of old nodes [KH13, p. 137]. The calculated values are used to choose actions [GS, see p. 1]. Additional information on the planning method can be found in the Prost-DD paper [GS] and in [KH13].

Prost-DD considers the planning problem as a search problem and tries to optimize the value for the initial state, ideally without the need to explore the whole problem graph. Due to the timeout, this optimization might not be completed - then again, only a part of the problem

was explored, with an estimate of the optimal value. The calculated values are used to create a plan.

The algorithm in Chapter 5 does not rely on the construction of a graph representation or on value calculations for graph nodes. Any form of optimization or path search can be handled by the SMT solver. The returned solution, if one exists, is always optimal regarding its probability. Prost-DD performed well in the IPPC 2018, meaning that it received a reward that was on average better than the rewards received by other planners. The reward is dependent on the reward function, but also on actual probabilistic outcomes in the simulation of the domain. The approach of the algorithm presented in this thesis is slightly different, as it maximizes the probability of success instead of the reward, given a threshold for the desired reward.

The participants *A2C-Plan* and *Imitation-Net* rely on *neural networks* and learning algorithms (see [AFTc] and [AFTa]). A2C-Plan is "an offline planner that trains a policy network through Reinforcement Learning" [AFTc, p. 1]. Imitation-Net works similarly but uses supervised learning on a deep neural network [AFTa, see p. 1]. *Random-Bandit* works online and uses an ε-greedy algorithm that returns, within a time-limit, an action in each planning step, given a state [AFTb, p. 1]. The two variants of *CONFORMANT-SOGBOFA*, which are also online algorithms [CK, p. 4], are based on a *computation graph* and conformant planning and rely on estimation and optimization [CK, p. 1].

All of these solutions either depend on a graph representation or do not guarantee to calculate an optimal solution. In the context of the IPPC, the planners do not need to find a maximum plan. Still, they will be compared to the presented algorithm, because the runtime of the SMT planner in comparison with the other planners as well as the comparison of the success of the generated plan, especially with the online planners, might be of interest: It could indicate whether applying SMT to solve RDDL domains is useful, and whether a maximum plan can - on average - compete with solutions that take failure into account.

4.2 Probabilities in SAT Solving

Apart from the interest in the field of probabilistic planning in the IPPC, the current interpretation and implementation of probabilities in the context of SAT solving and SMT needs to be examined as well. There might already exist methods that could be applied to RDDL problems, or approaches on how to treat probabilities in SAT solving or SMT encodings. Other ideas might not be connected to planning but could give examples for problems where probabilities in combination with SAT solving or SMT need to be considered.

In [JÁZ+12], counterexamples for properties specified for Markov chains (discrete-time Markov chains, DTMC) are generated symbolically [JÁZ+12, p. 134] using binary decision diagrams, as states and transitions of a system might not be representable explicitly due to its potentially large size [JÁZ+12, p. 135]. Both a SAT solving and a symbolic graph search approach are introduced [JÁZ+12, p. 136]. The latter is based on an algorithm discussed in Section 4.4. The properties for which counterexamples need to be generated include a thresh-

old for the probability of reaching undesired states that should not be exceeded [JÁZ+12, p. 138]. Thus, a counterexample consists of a set of paths to undesired states whose probability is in total higher than the specification in the system allows. Finding a most probable path is not required in the SAT solving approach, but the choice of which values are set during the search of a solution is tried to be changed so that more probable paths are found first, to increase the speed of the algorithm [JÁZ+12, p. 144].

SAT solving can thus be used to falsify specifications of a system that include probabilistic statements. The example shows that there exist problems where probabilities are encoded to a SAT solving formula, or where SAT solving is used to find a path or sets of paths that exceed a certain probability. Although the most probable path might not be of particular interest here, it could be used to speed up the search for a counterexample.

Probabilistic planning problems have been encoded in SAT as well: In [ML99], two solvers based on E-MAJSAT and S-SAT are presented, which solve partially observable "probabilistic propositional planning problems" [ML99, p. 549]. Here, state variables are Boolean [ML99, p. 549], so the proposed planners, MAXPLAN and ZANDER, do not regard domains where variables can be real or integer valued. The solvers distinguish between choice and chance variables [ML99, p. 549], where choice variables are given by existential quantifiers of the formula, and chance variables by random quantifiers [ML99, p. 552]. These quantifiers are a part of S-SAT or stochastic SAT, where they alternate and are followed by a Boolean formula and a threshold for its expected value [ML99, p. 551] (based on [Pap85]). E-MAJSAT is a subset of S-SAT [ML99, p. 551]. Both languages can be used to encode probabilities directly into a modified version of SAT. They work differently from the proposed integration of probabilities in SMT, as the evaluation of a formula with an expected value is unlike probabilistic statements in RDDL, where transition probabilities and the reward function are not directly connected.

MAXPLAN and ZANDER consider the probability of a solution as well, but not the probability of a single path. MAXPLAN uses a procedure called DPLL, and a solution to a E-MAJSAT problem is the assignment to all choice variables where, over all chance variables, the sum of the probability values of all satisfiable assignments is maximal [ML99, p. 552]. It does not search for a most probable path in the domain, where only one evaluation of all chance variables is considered, but for a set of actions where the overall probability of reaching a goal state is maximal, considering all possible outcomes of the chance variables.

ZANDER solves problems based on S-SAT by interpreting them as a tree and performing a depth-first search [ML99, p. 554]. The generated plan here is a tree itself, where every possible outcome of the chance variables is covered and choice variables are assigned so that the probability is maximized [ML99, see pp. 554-555]. ZANDER thus returns more than a maximum path in the planning domain, as the generated plan is complete in a way that it offers a set of actions for any outcome of the chance variables. For the proposed algorithm, calculating a maximum plan was deemed to be sufficient because it was developed with a focus on planning problems where an undesired probabilistic outcome implies that the goal state becomes unreachable. In any other case, a new plan can be generated by invoking the

algorithm again.

Overall, [ML99] shows that there already exist tools for defining and solving probabilistic planning problems that are based on a probabilistic language extension of SAT. Still, solving planning problems with success probabilities using SMT might be interesting as well, due to the higher expressiveness of the underlying language, which also allows to translate RDDL domains more easily.

4.3 Using SMT for Planning

Deterministic (see Section 2.4) as well as probabilistic domains have already been examined in planning using SMT, but this planning method has not been used for the IPPC 2018 (see [ippc] and Section 4.1), and methods for finding a most probable path like the solution in this thesis could not be found, especially including the translation and handling of RDDL files.

Using an SMT solver is not undisputed: [HGSK07] shows how numeric variables in planning problems can be translated so that SAT solving can still be used to obtain a solution, at least if the domains of the variables are not too large, and compares the performance of the SAT approach with an SMT solver, which was mostly slower [HGSK07, pp. 1918–1919]. Still, some RDDL problems might use statements that are harder to translate, or that might not be solvable within reasonable amounts of main memory. Thus, it is still reasonable to use an SMT solver for RDDL domains.

An algorithm comparable to the solution in this thesis, based on MDPs and Z3, is presented in [LSTZ15]. It deals with probabilistic preference planning problems and uses probabilities to simulate the uncertainty of effects of actions [LSTZ15, p. 3313]. Preferences and the goal condition are expressed in an LTL-style notation [LSTZ15, p. 3313] that allows for statements like “the goal states are eventually reached with probability at least 0.95” [LSTZ15, p. 3313], which can be used to set constraints on a plan. An optimal solution for a planning problem that includes an MDP, a goal formula and the ordered preferences is a scheduler where the goal formula must be fulfilled in combination with the first satisfiable preference, starting from the initial state [LSTZ15, p. 3315]. The treatment of preferences is similar to the iteration through possible probabilities for a maximum plan presented in the next chapter. The planning problem is specified in P4 and translated to quadratic equations by PolFinder, which are then being solved by Z3 [LSTZ15, p. 3316]. The paper also mentions RDDL, but because that language neither supports temporal statements nor preferences, an encoding to it was left open [LSTZ15, p. 3318].

There is still active research in the field of SMT and probabilistic planning. Even though no approach to find a maximum plan is presented in this chapter, the work in [LSTZ15], where the goal and the underlying language differ from the strategy presented in this thesis, but an implementation of probabilistic preference planning problems in SMT is shown, indicates that such an algorithm might already exists. Nonetheless, combined with the translation of RDDL and its Bernoulli statements, the solution presented in this thesis might still introduce interesting ideas for future research, which do not necessitate an interpretation of a planning

problem as a graph and which is not related to temporal logic.

4.4 Symbolic Graph Search

A procedure that can be used to find a maximum plan in an MDP is shown in [GSS10]. It was utilized in [JÁZ+12, p. 136] and can be used to find most probable paths [GSS10, p. 13]. The algorithm relies on a symbolic approach, uses a "symbolic variant of Dijkstra's algorithm" [GSS10, p. 13] and is not based on SAT or SMT solving: Given a symbolic representation of a graph, it first calculates a set of edges similar to a spanning tree [GSS10, p. 14]. These edges are chosen iteratively using the set of formerly discovered states, which initially only contains state 0 [GSS10, see p. 14]. In each iteration, edges are chosen between the discovered set of states and all states reachable from it so that the probability calculated for each state is the maximum possible [GSS10, see p. 14]. A most probable path in the graph is then determined in the second step, by traversing backwards through the spanning tree from the goal to the initial state [GSS10, p. 14].

The procedure of finding a most probable path in an MDP, which is a key aspect of the algorithm that determines a maximum plan, could have been implemented using some form of symbolic Dijkstra algorithm. But it would have required to interpret the RDDL translation as a graph first. Also, an SMT version of Dijkstra would have to be elaborated. There also exist symbolic heuristic search procedures as in [FH02, p. 455] for MDPs, but these were not considered for an SMT approach as well. Instead, the solution translates RDDL problems to SMT directly. The algorithm uses the Bernoulli-statements in the RDDL problem description only to influence which plans are generated by the SMT solver, and thus a graph search was not implemented. The translation itself is simpler and possibly better to understand, as any statement excluding probabilistic ones can be translated without making any significant changes compared to the RDDL file. The solution takes advantage of this translation, by manipulating variables of the SMT encoding, while the underlying MDP must neither be constructed nor analyzed.

Chapter 5

The Solution

In order to solve an instance of an RDDL domain of the specified problem class using Z3, it must be translated to an SMT problem with some additional treatment of the probabilistic information, as these cannot directly be expressed in SMT syntax. The approach that is presented in this chapter includes an abstracted version of the algorithm that is introduced on an MDP, an explanation of the translation of RDDL files and the more complex actual implementation of the algorithm for RDDL which utilizes Z3.

In between, the algorithm is also compared with Dijkstra's algorithm.

5.1 The Algorithm

5.1.1 Finding a Most Probable Path

The basic concept of the method that will be presented later can be visualized using the MDP $\mathcal{M}$ in Figure 5.1 with $use\_M_1, use\_M_2, use\_M_3, \alpha, \beta \in Act$. Only the states with actions that lead to the goal state are shown. For each transition from one state to another with a success probability of p there is also a transition leading from the state to itself with probability $1-p$. These transitions are omitted for better readability. They will not be considered in the examples for the sake of simplicity but would be treated similarly. As they cannot be used to get closer to the goal state, they cannot be part of a maximum path.

The only initial state in $\mathcal{M}$ is "raw materials". The goal state is "final product". The path of interest to the goal is the finite path with the highest probability of success, the maximum plan. Only three plans lead to the goal:

$$\pi_{fin,1} = \text{raw mat.} \xrightarrow{use\_M_3} \text{final prod.}$$

$$\pi_{fin,2} = \text{raw mat.} \xrightarrow{use\_M_1} \text{A1} \xrightarrow{use\_M_1} \text{final prod.}$$

$$\pi_{fin,3} = \text{raw mat.} \xrightarrow{use\_M_2} \text{B1} \xrightarrow{use\_M_2} \text{B2} \xrightarrow{use\_M_2} \text{B3} \xrightarrow{use\_M_2} \text{B4} \xrightarrow{use\_M_2} \text{final prod.}$$

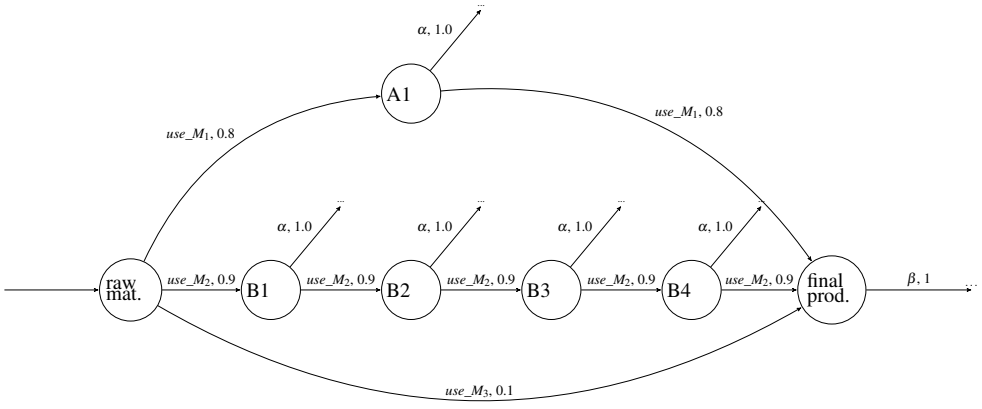

Figure 5.1: MDP: Construction of a final product from raw materials using different machines

The probabilities of success of these plans are:

$$p(\pi_{fin,1}) = 0.1 \tag{5.1}$$

$$p(\pi_{fin,2}) = 0.8 \cdot 0.8 = 0.64 \tag{5.2}$$

$$p(\pi_{fin,3}) = 0.9 \cdot 0.9 \cdot 0.9 \cdot 0.9 \cdot 0.9 = 0.59049 \tag{5.3}$$

Neither the shortest plan, nor the plan that only contains actions with the highest probability of success are a maximum plan. These criteria cannot solely be relied on. A distance metric can be used to search for a maximum plan using a search graph instead. The weights for the presented search scheme will be used for the actual algorithm as well, to restrict the set of edges that can be taken to reach a goal state.

Definition 5.1.1 (Search Graph for an MDP)
Let $\mathcal{M} = (S, Act, P, \iota_{\text{init}}, AP, L)$ be an MDP with $\iota_{init} = 1$ for some $s_0 \in S$. A *search graph for* $\mathcal{M}$ *with horizon* $h \in \mathbb{N}$ *and goal states* $G \subseteq S$ is the tuple $G_{\mathcal{M}} = (V, E, d, m, i, V_G)$ with

- $V = S \times \{0, \ldots, h\}$
- $E = \{((s,i),(s',i+1)) \in V | P(s,\alpha,s') > 0 \text{ for some } \alpha \in Act\}$
- $d : E \to \mathbb{R}, ((s,i),(s',i+1)) \mapsto max_{\alpha \in Act} \ln P(s,\alpha,s')$
- $m : E \to Act, ((s,i),(s',i+1)) \mapsto a$ for some $a \in \{\alpha \in Act | \ln P(s,\alpha,s') \geq \ln P(s,\alpha',s'), \forall \alpha' \in Act\}$
- $i = (s_0, 0)$
- $V_G = \{(s,i) \in V | s \in G\}$.

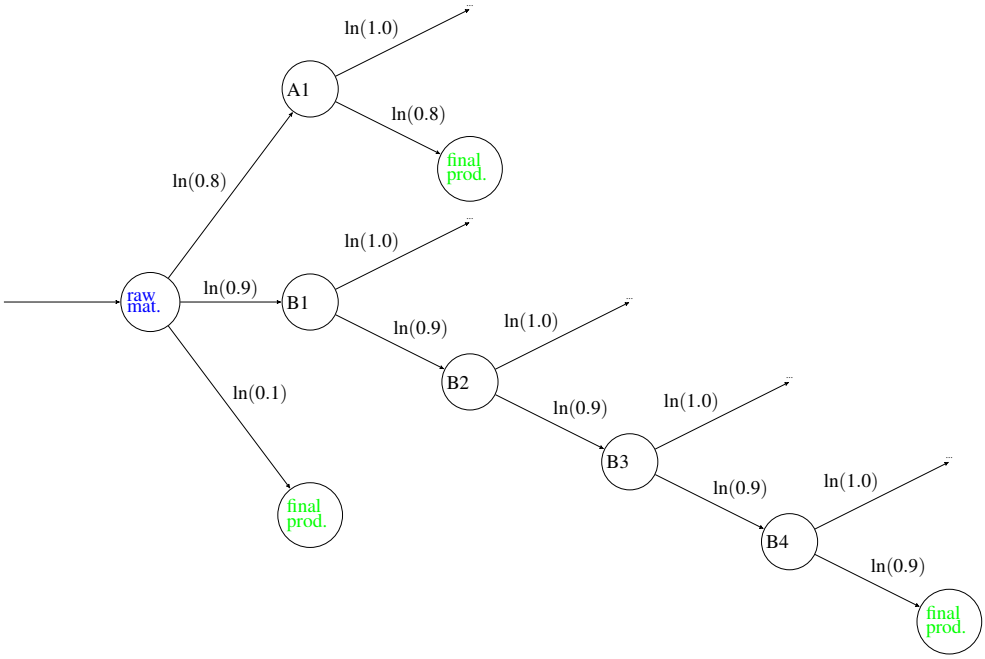

Figure 5.2: Graph for the MDP in Figure 5.1. Transitions from the states to themselves were left out in the translation for the sake of better readability. Initial states are blue, goal states are green.

Note that for d the maximum over all α is considered, as several actions with different positive probabilities may lead from s to s'. The mapping from actions in $\mathcal{M}$ to the weights of edges d in $G_{\mathcal{M}}$, m, is not unique as several transitions from a state $s \in S$ can have the same probability.

An example translation is shown in Figure 5.2: A path

$$p_{\pi,G_{\mathcal{M}}} = (s_0,0)e_0(s_1,1)e_1 \dots e_{n-1}(s_n,n), n \in \mathbb{N}, 0 \leq n \leq h$$

in $G_{\mathcal{M}}$ corresponds to a finite path (and thus to a plan) in $\mathcal{M}$,

$$\pi = s_0 \xrightarrow{m(e_0)} s_1 \xrightarrow{m(e_1)} \dots \xrightarrow{m(e_{n-1})} s_n$$

Definition 5.1.2 (Path Length in a Search Graph)
The length of a path $p = v_0 e_0 \dots e_{n-1} v_n$ in $G_{\mathcal{M}} = (V,E,d,m,i,V_G)$ is defined as:

$$D(p) \coloneqq \sum_{(v,e,v') \in p} d(v,v')$$

Corollary 5.1.1
For $a,b \in (0,1]$, iff $a > b$ then $\ln(a) > \ln(b)$ and iff $a = b$ then $\ln(a) = \ln(b)$.

Lemma 5.1.2
The longest path from the initial state to any goal state in $G_{\mathcal{M}}$ corresponds to a maximum plan in $\mathcal{M}$.

Proof. A path with the highest distance in $G_{\mathcal{M}}$ should correspond to a maximum plan in $\mathcal{M}$. For the translation from probabilities to a distance metric, the relation $\geq$ and $=$ thus need to stay the same on all paths regarding the maximum path and corresponding paths in $G_{\mathcal{M}}$. As transitions with probabilities of success of 0 are treated like they would not exist, all probabilistic values are in $(0,1]$ and thus Corollary 5.1.1 holds:

Let $\pi = v_0 \xrightarrow{\alpha_0} \ldots \xrightarrow{\alpha_{n-1}} v_n$ be a maximum plan in $\mathcal{M}$, then, for all plans in the MDP with $\pi' = v'_0 \xrightarrow{\alpha'_0} \ldots \xrightarrow{\alpha'_{m-1}} v'_m$:

$$\begin{aligned}
&P(v_0,\alpha_0,v_1)\cdot\ldots\cdot P(v_{n-1},\alpha_{n-1},v_n) \geq P(v'_0,\alpha'_0,v'_1)\cdot\ldots\cdot P(v'_{m-1},\alpha'_{m-1},v'_m)\\
\Leftrightarrow\ & ln(P(v_0,\alpha_0,v_1)\cdot\ldots\cdot P(v_{n-1},\alpha_{n-1},v_n)) \geq ln(P(v'_0,\alpha'_0,v'_1)\cdot\ldots\cdot P(v'_{m-1},\alpha'_{m-1},v'_m))\\
\Leftrightarrow\ & ln(P(v_0,\alpha_0,v_1))+\ldots+ln(P(v_{n-1},\alpha_{n-1},v_n)) \geq ln(P(v'_0,\alpha'_0,v'_1))+\ldots\\
&\qquad + ln(P(v'_{m-1},\alpha'_{m-1},v'_m))
\end{aligned}$$

The latter corresponds to the distance metric in $G_{\mathcal{M}}$. Paths in $G_{\mathcal{M}}$ can be mapped to paths within the planning horizon in $\mathcal{M}$ and vice versa: States can be mapped using the state definition in Definition 5.1.1. Edges can be mapped to actions using m. Transitions with actions in $\mathcal{M}$ can be mapped to edges in $G_{\mathcal{M}}$ using the definiton of edges in Definition 5.1.1. The transitions of a maximum plan must be maximal regarding all transitions that could be made between the states of the plan, or else it would not be a maximum plan. The weights of the edges of the corresponding path g in $G_{\mathcal{M}}$ must thus use the values of the transition probabilites of π. Any other path in $G_{\mathcal{M}}$ cannot have a higher distance, or else the path in $\mathcal{M}$ it could be mapped to would have a higher probability of success than π. So, for all plans π, π' in $\mathcal{M}$ and all finite paths g, g' in $G_{\mathcal{M}}$, where g corresponds to π and g' to π', if π is a maximum plan:

$$\begin{aligned}
p(\pi) \geq p(\pi') \Leftrightarrow D(g) \geq D(g')\\
p(\pi) = p(\pi') \Leftrightarrow D(g) = D(g')
\end{aligned}$$

Only exactly those finite paths within the horizon that lead from the initial to a goal state in the MDP also lead from the initial to a goal state in their representation in the search graph due to its definition. All finite paths in $\mathcal{M}$ up to the horizon are represented in $G_{\mathcal{M}}$, and there is no path in $G_{\mathcal{M}}$ that cannot be mapped to a finite path in the MDP.

Thus, for any path corresponding to π' in $G_{\mathcal{M}}$ and therefore for all paths in the search graph from the initial to a goal state, a path with maximum distance in the graph corresponds to π, and if such a path exists, π must exist as well. □

A *longest path* in a search graph can be found easily using Dijkstra's algorithm: If an

inversed distance metric is used as weights for all edges, the shortest path of the set of shortest paths from i to all goal states $v_g \in V_G$ found by Dijkstra is the longest path from the initial state to any goal state in $G_{\mathcal{M}}$. As the mapping of edges in $G_{\mathcal{M}}$ to actions in $\mathcal{M}$ is saved, the path found by that algorithm could be translated to a set of actions that need to be taken from each of the visited states to obtain the most probable path to a goal state. That information suffices to generate a plan. Nonetheless, Dijkstra's algorithm will not be used to solve instances of RDDL domains. In Subsection 5.1.3, the advantages of the algorithm presented in this chapter compared to Dijkstra's algorithm are being discussed.

5.1.2 Formalization of the Algorithm

Instead of using a search graph to find a most probable path from an initial state to a goal state in an MDP, the *search proposed in this thesis is symbolic* and utilizes the possibility of defining further constraints in addition to the translation of an RDDL domain instance in SMT.

For now, specific properties of RDDL and SMT are being ignored, and the algorithm is explained without describing the details of its implementation. The discussion takes place on the level of *abstraction* of graphs and MDPs, like in Subsection 5.1.1. It is assumed that an algorithm is given that decides whether a specific search problem is *solvable* given a set of constraints on the amount of transitions that can be taken, where solvable means that a path from an initial to a goal state can be found using these transitions. Edges with a probability of success of 0 are being omitted. That algorithm is called IS_SATISFIABLE. Later, Z3 will fulfill that purpose.

Any transition with a probability of success of 1.0 can be taken anytime and will not be restricted by a constraint. The set of transitions that can be chosen from with a probability lower than 1.0 is empty in the beginning of the search. It is expanded step by step, until a solution can be found. The set of allowed transitions at that point can then be used to find a maximum plan for the given instance.

Definition 5.1.3 (Probabilities, Occurrences, Weights)
Given an MDP $\mathcal{M} = (S, Act, P, \iota_{\text{init}}, AP, L)$ and

$$P' = \{P(s,\alpha,s') | s,s' \in S, \alpha \in Act, P(s,\alpha,s') \in (0,1)\}$$

all distinct values $p_1, \ldots, p_n \in P', n \in \mathbb{N}$, sorted from highest to lowest, are the *probabilities of success* in $\mathcal{M}$.

The *frequencies of occurrence* $x_1, \ldots, x_n \in \mathbb{R}$, given a plan π in $\mathcal{M}$, are defined as:

$$x_i = |\{(s,\alpha,s') | (s,\alpha,s') \in \pi, P(s,\alpha,s') = p_i\}|$$

Weights $g_1, \ldots, g_n \in \mathbb{R}$ correspond to the probabilities of success and are always positive:

$$g_i = \frac{\ln p_i}{\ln p_1}, p_i \in (0,1), i \in \{1, \cdots, n\}$$

These variables p_i, x_i, g_i for $i \in \{1, \ldots, n\}$ will be used to determine a set of frequencies of occurrence that corresponds to a maximum plan. Constraints on transitions with a probability of 1.0 are not required - these transitions always succeed. Transitions with a probability of 0.0 are omitted.

Figure 5.1 contains three different probabilities of success, $p_1 = 0.9, p_2 = 0.8, p_3 = 0.1$. Any plan with $x_1 = 5, x_2 = 0, x_3 = 0$ must contain the longest path in the middle. The probability of success of this path is $0.9^5 \cdot 0.8^0 \cdot 0.1^0 = p_1^{x_1} p_2^{x_2} p_3^{x_3}$. Constraints using $x_i, i \in \{1, \cdots, n\}$ can be defined to set how often transitions with probability p_i must be taken in a valid plan. Due to Definition 5.1.3, it follows that:

Corollary 5.1.3 (Probability of a plan using frequencies of occurrence)
If a plan can be found with $x_1, \ldots, x_n$ for $p_1, \ldots, p_n$, the probability of success is $\prod_{i=1}^{n} p_i^{x_i}$.

The search for a set of values for x_i that corresponds to a maximum plan is performed in steps. In each step, the value of a *search variable s* is incremented by 1. Constraints on the SMT formula are used to set bounds for the values of $x_1, \cdots, x_n$ so that no plan can be found with a probability below a certain threshold.

The search is terminated as soon as the set of constraints in a step is satisfiable, or else after $x_i = horizon \cdot$ amount-of-state-variables, $i \in \mathbb{N}$ has been tried out, where `horizon` is the planning horizon of the problem. This guarantees that the algorithm terminates, given that Z3 terminates if the problem is undecidable. Also, no solution is left out because no more than $horizon \cdot$ amount-of-state-variables $=: max\_length$ potentially probabilistic state transitions can be performed (if each transition function of a state variable is considered to be represented by a single transition in the MDP). It is important to note that all used parameter interpretations need to be considered for the upper bound: Transitions are defined for functions that represent state- or interm-fluents, and probabilistic transitions in each step of the plan could be made for all parameter interpretations of these functions.

The value of max_length is h in Definition 5.1.1.

The frequencies of occurrence for a step $j \in \mathbb{N}$ will be denoted as $x_{j,1}, \ldots, x_{j,n}$, where the second index corresponds to the probability value with the same index. In the encoding a variable s is used which is incremented as follows:

$$s_0 = 0 \tag{5.4}$$

$$s_j = s_{j-1} + 1, \quad j \in \mathbb{N}_{>0} \tag{5.5}$$

The linear equations that limit the set of plans that can be found in step $j \in \mathbb{N}_{>0}$ are:

$$x_{0,i} = 0, \quad i \in \{1, \cdots, n\} \tag{5.6}$$

$$\sum_{i=1}^{n} x_{j,i} g_i \leq s_j \tag{5.7}$$

$$\sum_{i=1}^{n} x_{j,i} g_i > s_{j-1} \tag{5.8}$$

$$0 \leq x_{j,i} g_i \leq s_j, \quad i \in \{1, \cdots, n\} \tag{5.9}$$

$$\sum_{i=1}^{n} x_{j,i} \leq \text{max\_length}, \quad i \in \{1, \cdots, n\} \tag{5.10}$$

The right part of Equation (5.9) is redundant, because it is implicitly contained in Equation (5.7). However, as stated in Chapter 2, it can be beneficial to explicitly use additional deducible constraints in SMT because they can reduce the runtime. Note that in Equation (5.8) $s_{j-1} = s_j - 1$.

Lemma 5.1.4
In step $j, j \in \mathbb{N}_{>0}$, the least probable plan that can be constructed using transitions that satisfy the constraints specified above has a probability of success of $p_1^{s_j}$.

Proof. For $n \in \mathbb{N}, i \in \{1, \cdots, n\}, j \in \mathbb{N}_{>0}$, given any set of frequencies of occurrence $x_{j,i}$ that satisfy the constraints above in step j for the probabilities p_i, the probability of any plan with these frequencies $x_{j,i}$ is $\prod_{i=1}^{n} p_i^{x_{j,i}}$ due to Corollary 5.1.3.

As $p_i \in (0,1), ln(p_i) < 0$ holds. Because the constraints must be fulfilled, it can be followed that:

$$\begin{aligned}
&\sum_{i=1}^{n} x_{j,i} g_i \leq s_j \\
\Leftrightarrow &\sum_{i=1}^{n} x_{j,i} \frac{\ln p_i}{\ln p_1} \leq s_j \\
\Leftrightarrow &\sum_{i=1}^{n} x_{j,i} \ln p_i \geq s_j \ln p_1 \\
\Leftrightarrow &\prod_{i=1}^{n} p_i^{x_{j,i}} \geq p_1^{s_j}
\end{aligned}$$

Thus, $p_1^{s_j}$ is the lowest probability of success for a path that satisfies the constraints. □

Theorem 5.1.5
If a solution is found in step $j, j \in \mathbb{N}_{>0}$, then that solution has a higher probability of success than any solution found in any following step $j+m, m \in \mathbb{N}_{>0}$.

Proof. Lemma 5.1.4 states that the lowest probability of success of a solution that was found

in step j is $p_1^{s_j}$. If all plans found in step $j+1$ have a lower probability than any possible plan in step j, then the theorem is proven.

Assume that a plan exists for some $x_{j+1,i}, i \in \{1,\cdots,n\}, n \in \mathbb{N}$ where its probability is higher than the probability of a plan in step j. Then, according to Lemma 5.1.4:

$$\begin{aligned}
&\prod_{i=1}^{n} p_i^{x_{j+1,i}} \geq p_1^{s_j} \\
\Leftrightarrow &\sum_{i=1}^{n} x_{j+1,i} \ln p_i \geq s_j \ln p_1 \\
\Leftrightarrow &\sum_{i=1}^{n} x_{j+1,i} \frac{\ln p_i}{\ln p_1} \leq s_j \\
\Leftrightarrow &\sum_{i=1}^{n} x_{j+1,i} g_i \leq s_j
\end{aligned}$$

This would contradict Equation (5.8). Also, solutions using this bound could have already been found in the former iteration. □

Algorithm 1 The Search Algorithm, Step 1

```
1: procedure ALGORITHM1(problem, p_1, ..., p_n, x_1, ..., x_n)   ▷ The translation of the problem "problem" (which includes the planning horizon), the extraction of probability values and the definition of x_i will be given by the RDDL translation
2:     s ← 0
3:     for i ← 1, n do
4:         g_i ← ln(p_i / p_1)
5:     end for
6:     solvable ← false
7:     repeat
8:         problem' ← problem
9:         ADD_ADDITIONAL_CONSTRAINT(problem', Σ_{i=1}^{n} x_i g_i ≤ s)
10:        ADD_ADDITIONAL_CONSTRAINT(problem', Σ_{i=1}^{n} x_i g_i > s − 1)
11:        ADD_ADDITIONAL_CONSTRAINT(problem', ∀_{i∈1,...,n} 0 ≤ x_i g_i ≤ s)
12:        ADD_ADDITIONAL_CONSTRAINT(problem', Σ_{i=1}^{n} x_i ≤ max_length)   ▷ No plan can be longer than the planning horizon allows - this is already implicitly encoded in the RDDL translation (see Subsection 5.2.1)
13:        solvable ← IS_SATISFIABLE(problem')
14:        s ← s + 1
15:    until solvable or s > g_n · max_length   ▷ Until x_n > max_length is possible
16:    if not solvable then
17:        additional_constraints ← null
18:    end if
19:    return additional_constraints, s
20: end procedure
```

Algorithm 1 uses the search constraints mentioned above to search for the range of prob-

abilities that includes the probability of a maximum plan. The parameters of the algorithm were either explained before or are mentioned in the comments. The problem translation is copied, and the additional constraints are added to it. The search is performed until either the program terminates because all paths within the planning horizon do not lead from the initial to a goal state or because a satisfying set of search constraints has been found. The satisfiability is checked with IS_SATISFIABLE, which here represents a procedure of an SMT solver like Z3 that checks whether the problem together with additional constraints is satisfiable.

Theorem 5.1.6
There exists a finite path in an MDP $\mathcal{M}$ of a given RDDL domain instance that leads from the initial to a goal state with no more transitions than allowed by the planning horizon if and only if Algorithm 1 returns a set of constraints that is satisfied by a maximum plan in $\mathcal{M}$.

Proof. Assume such a plan π exists. As it would be a possible solution to the problem, IS_SATISFIABLE must return true if it fulfills all additional constraints (Equation (5.6), Equation (5.7), Equation (5.8), Equation (5.9), Equation (5.10)). Equation (5.6) defines the initial values for the search procedure and is not used explicitly in Algorithm 1. Equation (5.10) is fulfilled by definition. Due to the planning horizon and if all state changes are encoded as separate transitions, no more than *max_length* transitions can be made, therefore:

$$\sum_{i\in\{1,\dots,n\}} x_i \leq \text{max\_length} \tag{5.11}$$

$$\xRightarrow{g_n>0} \sum_{i\in\{1,\dots,n\}} x_i g_n \leq \text{max\_length} \cdot g_n \tag{5.12}$$

$$\xRightarrow{s\leq \text{max\_length}\cdot g_n} \sum_{i\in\{1,\dots,n\}} x_i g_n \leq s \text{ (for s in some iteration)} \tag{5.13}$$

$$\xRightarrow{g_i\leq g_n} \sum_{i\in\{1,\dots,n\}} x_i g_i \leq s \tag{5.14}$$

$$\tag{5.15}$$

Thus, Equation (5.7) and the right part of Equation (5.9) must be satisfied by π. It is also not possible for $x_1,\dots,x_n$ to be smaller than 0 due to their definition. As $0 \leq s \leq \text{max\_length} \cdot g_n$ and $0 \leq \sum_{i\in\{1,\dots,n\}} x_i g_i \leq s$, in the first iteration where Equation (5.7) is satisfied, Equation (5.8) is satisfied as well, and such an iteration must exist. Thus, π can be found, as the according set of search constraints is satisfiable. Because of Theorem 5.1.5, the first satisfiable set of constraints must therefore include a maximum plan.

Assume the algorithm returns a set of constraints for a maximum plan. This means that a solution for the problem encoded by the MDP was found by the procedure IS_SATISFIABLE where all constraints are satisfied. Even if this solution would contain additional transitions with a probability of success of 1, the problem translation prevents the construction of any plan that is longer than max_length (see Subsection 5.2.1). Assuming that IS_SATISFIABLE

works correctly - which should be the case for the used SMT solver Z3 - there must therefore exist a finite path within the planning horizon from the initial to a goal state in the MDP. □

The main purpose of the first algorithm is to give a range in which to search for the maximum plan. As stated in Lemma 5.1.4, the least probable plan in step s has a probability of success of p_1^s. Any plan π that satisfies the conditions returned by Algorithm 1 for some $s \in \mathbb{N}_{\geq 0}$, if one exists, must satisfy the condition $p_1^s \leq p(\pi) < p_1^{s-1}$, because Equation (5.7) and Equation (5.8) must hold. As could be seen in Corollary 5.1.3, the frequencies of occurrence can be used to calculate the probability of a plan. They can be used as constraints on the actual problem translation to search for a plan with that probability. This idea is used by Algorithm 2.

Algorithm 2 The Search Algorithm, Step 2

```
1:  procedure ALGORITHM2(problem, p_1, ..., p_n, x_1, ..., x_n, s)
2:      [x_1, ..., x_n] ← x_1, ..., x_n
3:      all_combinations ← CREATE_ALL_COMBINATIONS([x_1, ..., x_n], s)
4:      all_combinations ← SORT_BY_PROBABILITY(all_combinations, p_1, ..., p_n)
5:      for [x_1, ..., x_n] ← all_combinations do
6:          if p_1^s ≤ p([x_1, ..., x_n]) < p_1^(s-1) then
7:              if IS_SATISFIABLE(problem, [x_1, ..., x_n]) then
8:                  return [x_1, ..., x_n]
9:              end if
10:         end if
11:     end for
12:     return null
13: end procedure
```

The algorithm takes the value for s found by Algorithm 1, if a value could be found, and else is not invoked. It uses CREATE_ALL_COMBINATIONS to determine the finite distinct set of all arrays of size n where each entry can have a value in $0, \ldots, s$. The resulting values for $x_i, i \in 1, \ldots, n$ can be used to calculate a success probability for each array according to Corollary 5.1.3, and can be sorted in descending order using these probabilities, which is performed by SORT_BY_PROBABILITY. The first satisfiable set is returned. Else, the algorithm terminates when all combinations have been tried out. In the implementation, due to the size of *all_combinations*, the combinations are not stored but calculated again in each iteration to find the subset with the highest probability of success to save main memory.

Theorem 5.1.7
If Algorithm 2 uses the values given by Algorithm 1 as input values and the latter found a solution, then Algorithm 2 returns frequencies of occurrences of a maximum plan.

Proof. Only combinations within the bound determined by Algorithm 1 are considered in the for-loop. The first satisfiable array, which corresponds to a maximum plan within *all_combinations*, is returned.

The algorithm must return frequencies of occurrence of a valid plan, because it iterates through all values of possible paths allowed by the constraints set in Algorithm 1, of which at least one must exist if Algorithm 1 found that there exists a solution, and else no plan can be found. These constraints must include a maximum plan due to Theorem 5.1.6. That path is found first because of the sorting operation. □

Given concrete values $[x_1, \ldots, x_n]$, the probability of the maximum plan is already known after Algorithm 2 returns. Using these values as a constraint, an SMT solver can find a model from which the maximum plan can be extracted. An abbreviated example run of the algorithm can be found in Appendix A.2.

5.1.3 Comparison to Dijkstra

The algorithm is different than the k-shortest path Dijkstra algorithm that was briefly described in Section 4.4. Instead of performing a search in the graph and calculating weights for nodes using the probability of a path from the initial state to them, it iterates through all possible combinations of transitions that lower the probability of success within the planning horizon in descending order. Edges in the search graph are not being traversed, but in each iteration the SMT solver is used to determine whether there exists a path from the initial to the goal state that uses exactly x_i transitions with probability p_i. To speed up the search, the set of combinations which need to be enumerated is restricted using a minimum and maximum bound for their probability of success. A graph representation is not required.

The model for a non-symbolic Dijkstra algorithm *would grow exponentially* in the length of a path. There exist symbolic concepts which partially avoid this problem, but the presented approach was implemented instead.

5.2 RDDL Plan Translation

A correct translation of RDDL domain instances as well as the actual implementation of the algorithm that depends on the translation of the `Bernoulli` statements need to be elaborated. After some other strategies were tried out, where the transitions that are decoded as state changes in RDDL were decoded as action-based transitions in Z3 like in Section 2.3, it was decided that, due to the expressiveness of RDDL, anomalous models need to be ruled out by the RDDL domain description instead, so the translation is now state-based. Apart from Bernoulli statements, quantifiers, sums and products, all constraints are translated similar to their representation in the RDDL files.

5.2.1 Variable Translation

All variable definitions are part of the pvariables section in an RDDL domain description [San10, see p. 18]. They are interpreted as *n-ary functions*. Their range and their domain are specified together with default values and the information of what they represent - a

```
domain ... {
    types { car : object; ... };
    pvariables {
        PROB_FLATTIRE : { non−fluent, real, default = 0.1 };
        onHighway(car) : { state−fluent, bool, default = true };
        relativeSpeed(car, car) : { state−fluent, int, default = 0 };
    };
}
non−fluents ... {
    objects { car : {c1, c2, c3}; ... };
}
```

(a) RDDL variables and types

```
(declare−datatypes () ((car c1 c2 c3)))
...
(declare−fun PROB_FLATTIRE () Real)
(declare−fun onHighway (car) Bool)
(declare−fun relativeSpeed (car, car) Int)
```

(b) Translation

Figure 5.3: RDDL variables / types example and translation to Z3

state property, an action or a property of the modeled system that does not change over time [San10, see pp. 15, 18]. The parameter types of the variables supported by this translation are objects or enums.

Variables, objects and enumerated types can directly be translated into their Z3 equivalents. An example translation is shown in Figure 5.3. The function declarations are saved dependent on the information they represent - for example, action-fluents need to be accessed after the translation of the RDDL file was finished, because additional concurrency constraints are only given implicitly by the keyword `max-nondef-actions` but must be included explicitly.

In order to get an expression for which initial values and constraints can be defined, *parameters* need to be applied to function declarations. These parameters can either be uninterpreted or represent a concrete value in the domain, for example c1 in Figure 5.3a. If the range of all parameters in the domain of a function declaration is finite, constraints can be translated for every set of parameter interpretation (parameters can be enumerated), as shown in Figure 5.4a. Alternatively, parameters remain *uninterpreted*, as shown in Figure 5.4b. If possible, the performance of both approaches will be compared in Chapter 6. In all following examples, parameters are uninterpreted, and the definition of the parameter variables will be omitted. This allows for shorter example code.

The *planning horizon* of an RDDL domain instance is always finite [San10, p. 4]. The value of the keyword `horizon` is used to determine how many time steps can be made in a domain. This property is translated to all variables that do not represent non-fluents, as only non-fluents do not change over time [San10, see p. 15]. The variables are defined independently for each step, as shown in Figure 5.5. With each time step another state in the system

```
(assert (= (relativeSpeed c1 c1) 0))
(assert (= (relativeSpeed c1 c2) 0))
(assert (= (relativeSpeed c1 c3) 0))
(assert (= (relativeSpeed c2 c1) 0))
...
```

(a) Using concrete parameter values

```
(declare-const _x_1 car)
(declare-const _x_2 car)
(assert (= (relativeSpeed _x_1 _x_2) 0))
```

(b) Using uninterpreted parameters

Figure 5.4: Usage of parameters in a constraint

```
(declare-fun onHighway_0 (car) Bool)
(declare-fun onHighway_1 (car) Bool)
(declare-fun onHighway_2 (car) Bool)
```

Figure 5.5: All variable definitions for onHighway with horizon = 2

can be reached. Due to the initial values, there are $horizon + 1$ definitions of each variable that is not a non-fluent. The maximum length of a path is thus already encoded in the RDDL translation.

State-fluents and non-fluents are given *default values* [San10, p. 18] - values for action-fluents are chosen by the SMT solver, and the values of intermediate fluents are determined by their state transition function. The *intial values* might differ from the default values in the domain definition if they are specified in a `non-fluents` or `init-state` block [San10, see p. 20], see Figure 5.6 (using pvariables as defined in Figure 5.3a).

5.2.2 Constraint Translation

Most of the RDDL operators are supported by the RDDL translation. The parsed object of the RDDL domain file provides a nested representation of `state-action-constraints`, `action-preconditions` and `cpfs`. They can be translated *recursively*. If a state variable is *primed*, then all other variables referred to in the constraint are variables of a former time step, else the same time step is referred to [San10, see p. 6]. Every single constraint is added to the SMT solver, and thus the resulting formula is a conjunction of all constraints translated from the RDDL file. An example translation is shown in Figure 5.7.

There are RDDL statements which require a function interpretation for all possible function values, like negated existential quantifiers, universal quantifiers or sum or product expressions. To support these statements, the functions involved in these statements are defined over the range of all parameter values that are being quantified. Thus, even if uninterpreted parameters are chosen in general, constraint translations need to be made for all parameter interpretations for *quantified parameters*, see Figure 5.8.

As quantified functions are defined with more than one set of parameters, an additional set of constraints must be introduced: If the function parameters are the same, then the interpretation of the function must be the same as well.

Another constraint that needs to be passed to the solver considers concurrency. No more

```
non−fluents {
    PROB_FLATTIRE = 0.05;
};
init−values {
    ~onHighway(c1);
};
```

(a) Initial values in RDDL

```
(assert (= (PROB_FLATTIRE 0.05))
(assert (=> (= _x_1 c1) (= (onHighway_0 _x_1) false)))
(assert (=> (not (= _x_1 c1)) (= (onHighway_0 _x_1) true)))
```

(b) Translation

Figure 5.6: Translation of initial values

```
//Usually, only interm−fluents (intermediate) are unprimed, while state−fluents are primed
relativeSpeed(?c1, ?c2) = if (onHighway(?c1) ^ ~onHighway(?c2)) then 1
                          else 0;
//start−drive here is an action−fluent
onHighway'(?c) = start−drive(?c) | onHighway(?c);
```

(a) Some constraints in a cpfs block

```
//ite means if−then−else; the same constraints need to be translated for step 2, 3, ...
(assert (ite (and (onHighway_1 _x_1) (not (onHighway_1 _x_2)))
        (= (relativeSpeed_1 _x_1 _x_2) 1)
        (= (relativeSpeed_1 _x_1 _x_2) 0)))
(assert (= (onHighway_1 _x_3) (or (start−drive_0 _x_3) (onHighway_0 _x_3))))
```

(b) Translation

Figure 5.7: Translation of cpfs, primed and unprimed

than `max-nondef-actions` actions can have non default values in each time step [San10, see p. 7].

5.2.3 Determining Goal States

Instead of defining specific objectives or goal states, RDDL domain files just define a *reward function* that may use any metric that can be expressed using the RDDL language [San10, see p. 19]. The equation specifying the reward value is an unprimed expression [San10, see p. 19] and is translated like `cpfs`, using the variable "reward_i" in time step *i*.

The algorithm expects the goal states to be explicitly defined. As the reward statement is problem-specific, desired minimal values for the overall reward need to be passed to the program manually. An optimizer is used to determine the highest reward value "max_reward", and a value relative to that maximum needs to be passed, e.g. 0.8, meaning that the overall reward must at least be $0.8 \cdot$ max_reward, and that all states with this value are goal states.

Using the planner without such a *reward approximation* would not suffice. As no goal states are defined, any solution where any arbitrary number of actions is taken would be a valid plan. In some of the domains in Chapter 6, every maximum plan would have a probability of success of 1, as not taking any action could be considered a solution. Thus, a set of desirable final states must be known before the algorithm is run. A solution will most

```
fastestCar'(?c) = ~exists_{?c2 : car}[relativeSpeed(?c2, ?c) > 0];
```

(a) Some cpf with quantifier

```
(assert (= (fastestCar_1 _x_1) (not (or (
    (relativeSpeed_0 c1 _x_1)
    (relativeSpeed_0 c2 _x_1)
    (relativeSpeed_0 c3 _x_1)
)))))
```

(b) Translation

Figure 5.8: Translation of cpfs with quantifier

likely be obtained for a domain that the user is familiar with, and thus a lower threshold for the reward can coincide with the desired minimal quality of the plan in the context of the domain. Due to the highly domain-dependent reward definition, the trade-off between computation time, reward, probability and manual procedures was deemed to be an acceptable first step of integrating the reward function.

5.2.4 Algorithm Implementation

Only the probabilistic statements `Bernoulli`, `Discrete` (partially), `DiracDelta` and `KronDelta` are being supported. The restricted set of statements is sufficient to translate most IPPC RDDL domains. Furthermore, the algorithm only works with discrete probability distributions.

`KronDelta` and `DiracDelta` can be translated with a success probability of 1.

`Discrete` is only partially supported, for simple probabilistic statements that are not dynamic, i.e. that do not rely on other variables than nonfluents, and not for uninterpreted parameters. `Discrete` could be fully supported with an implementation similar to `Bernoulli` statements.

The expression "prob-statement" in a Bernoulli statement "Bernoulli(prob-statement)" can either refer to constants and non-fluents only, or to other fluents as well. The former case will be called *static probability*, the latter *dynamic probability*. Both are supported by the algorithm, but its implementation does not consider all dynamic probability expressions: Only `Bernoulli` statements and state-fluents that have a finite range are supported. Further implementation was not required in the context of the evaluation.

All possible interpretations of all variables that are part of a dynamic statement are evaluated by checking whether they are satisfiable, and then combined to a *list of all probabilities* that can be expressed by the statement. Placeholder variables for all possible outcomes in $[0, 1]$ are created, and they are used similarly to static probability variables, which are explained in the following paragraph. Figure 5.9 shows examples for both variants.

When an RDDL statement is translated that includes a `Bernoulli` expression, it is replaced by a *placeholder* variable. `Discrete` expressions are treated similarly. The value

```
1 //Statement with static probability, every car has a different probability of having a flat
2 has−flat−tire'(?c) =if (moved(?c))
3                             then Bernoulli(PROB_FLATTIRE(?c));
4                       else false;
5 //Statement with dynamic probability
6 road−is−intact' = Bernoulli(1 / (1 + sum_{?c : car}[onHighway(?c)]));
```

Figure 5.9: Bernoulli statements (here without actions involved)

```
1 (assert (ite (= (moved_0 _x_1) true)
2     (= (has−flat−tire_1 _x_1) (or _Bernoulli_1 _Bernoulli_2))
3     (= (has−flat−tire_1 _x_1) false)))
4 x1 = [(and _Bernoulli_2 (= (PROB_FLATTIRE _x_1) 0.98) (= (moved_0 _x_1) true)), (and (not _Bernoulli_1)
      (= (PROB_FLATTIRE _x_1) 0.02) (= (moved_0 _x_1) true))]
5 x2 = [(and _Bernoulli_1 (= (PROB_FLATTIRE _x_1) 0.02) (= (moved_0 _x_1) true)), (and (not _Bernoulli_2)
      (= (PROB_FLATTIRE _x_1) 0.98) (= (moved_0 _x_1) true))]
```

Figure 5.10: Bernoulli statement translation for Figure 5.9

of the probabilistic statement is evaluated for each possible interpretation of the parameters. Both the Boolean variables created for the expression and the probabilities they are connected with are used by the algorithm.

In Figure 5.10, two new placeholder variables are introduced. The state fluent can only become true if one of the Bernoulli variables and the condition "moved" is true. Each possible outcome is covered by the statements in x1 and x2. They are *used to count the frequencies of occurrence*: If the condition is true, one of the four statements must be satisfied. The first Bernoulli variable becomes true with a probability of 0.02 and false with a probability of 0.98 and the second Bernoulli variable becomes true with a probability of 0.98 and false with a probability of 0.02. Which of the statements is satisfied depends on the interpretation of "_x_1".

Each Bernoulli variable is unique only to the RDDL expression, the current step and the actual interpretation of the parameters used. If "has-flat-tire" was quantified and constraints for e.g. "has-flat-tire_1(c1)" would need to be defined, the above approach might not be sufficient. This is because, in an interpretation where "_x_1 = c1", both constraints *represent the same function* in the same step, but their frequency of occurrence would be counted twice. Thus, additional constraints are used that refer to the value of already translated functions in the same step to prevent this.

It is important to note that the algorithm itself determines whether a Bernoulli statement should become true or false, even if that decision is made by the simulator of the environment. By setting upper and lower bounds for x_i, only some state changes with the according probability can be made. The Bernoulli variables can be used to determine the probabilities of success as well as the frequencies of occurrence, which need to be passed to the algorithm. In Figure 5.10, the frequencies of occurrence can be counted using x1 and x2 as in Figure 5.11. In general, a plan cannot recover from any deviation from the anticipated probabilistic out-

```
1 p1 = 0.98, p2 = 0.02
2 x1 = (and (moved_0 _x_1) (= (PROB_FLATTIRE _x_1) 0.02) (not _Bernoulli_1))
3     + (and (moved_0 _x_1) (= (PROB_FLATTIRE _x_1) 0.98) _Bernoulli_2 true)
4 x2 = (and (moved_0 _x_1) (= (PROB_FLATTIRE _x_1) 0.02) _Bernoulli_1 true)
5     + (and (moved_0 _x_1) (= (PROB_FLATTIRE _x_1) 0.98) (not _Bernoulli_2))
```

Figure 5.11: Algorithm interpretations of Figure 5.10

comes, as it might depend on the state of the variables affected by this change.

Definition 5.2.1 ($x_1, \ldots, x_n$ in the RDDL Translation)
Let `Bernoulli(prob-statement)` be a statement in an RDDL domain that is used under a condition "condition", which is either always true or given by an if expression. In the translation of the probabilistic statement, each probability value $p \in (0,1)$ is connected to a set S_p of all statements "condition $\wedge$ prob-statement" for state transitions where the probability of the Bernoulli variable taking the value of "prob-statement" is p under the condition "condition".

If $p_1, \ldots, p_n \in (0,1)$ are all probability values occurring in the RDDL file that are not 0 or 1, sorted from highest to lowest, then $x_i := \sum_{s \in S_{p_i}} s, i \in \{1, \ldots, n\}$, where satisfied statements are identified as 1, and unsatisfied statements as 0.

For more detailed information about the implementation of the algorithm, commented code snippets can be found in Appendix A.7.

5.2.5 Extracting a Plan

Like the planners in Section 4.1, the algorithm returns actions that need to be performed upon request. If no alternative is supposed to be calculated if the plan fails, a plan is only calculated once, and the states received by the RDDLsim simulator are being ignored. In this case, the precalculated set of actions for each step is being returned, and if they cannot be applied in the current state, then the plan might fail. Else, the received states are translated, and a new plan is calculated if the new problem is still satisfiable.

Actions are extracted using the *interpretation of all action fluents* in each step in the RDDL domain given by the model returned from Z3. All action fluents of the current step that are true for any interpretation of the parameters are being returned. The details of this implementation will not be mentioned, as they only depend on the syntax specified by the RDDL simulator that is used for the evaluation and by Z3.

If no simulator is used, the plan can also alternatively just be returned as a set of action and state variable interpretations for each step. The expected received reward if the plan succeeds and its probability can be returned as well.

5.2.6 Further Restrictions

In addition to Section 3.2 and Subsection 5.2.4, the implementation includes further rules and restrictions that need to be mentioned.

Some of these are of syntactical nature: Underscores, besides those used for RDDL keywords, are not allowed to be used in the RDDL files. The names of parameters, if quantified, need to be unique. Actions are only allowed to be Boolean if `max-nondef-actions` is used, but support for other types could be added easily. The names of the definition blocks are ignored - it is assumed that the instance block passed to the program is supposed to be applied to the received domain block. Requirements are ignored as well - *full observability is assumed*. The reward must be deterministic. Maximum and minimum statements, TVAR_EXPR and FUN_EXPR are not supported, as well as switch statements - if statements can be used instead. All functions are ranged over Boolean, real or integer values or enumerated or object types. Parameters can only be ranged over enumerated or object types.

Reward and discount need to be defined. Intermediate fluents are only supported on level 1 for the sake of simplicity - most of the domains did not require deeper levels.

Dynamic probability and Discrete statements and the RDDL2 syntax are only partially supported. Discrete statements have been used in some IPPC 2018 domains (see [ippc]), but none of these domains were solvable by all proposed algorithms within the timeout, so their translation remains untested. Probabilistic statements in if expressions can currently not be used due to the fact that the condition is a part of the probabilistic constraint, but this could be resolved in a future implementation.

Furthermore, the *number precision* is potentially problematic. The usage of the natural logarithm and divisions is assumed to work fine in most of the problems. For file translations, probability values are translated with a precision of up to 40 decimal places, using Javas BigDecimal class, but they are casted to doubles before they are being used by the algorithm. Values extracted from a Z3 model are read with a number precision of 20 decimal places. Due to the potentially inexact translation, probability value constraints for probabilistic statements like (= (- 1.0 PROB_FLATTIRE) p_i) might not work because Z3 might calculate a slightly different value. Thus, a small ε value is added and subtracted to/from p_i and it is instead checked if the probabilistic statement is within that range. To support at least ten decimal places, $\varepsilon = 10^{-11}$ was chosen. The distance between probabilistic values of an expression should thus be at least ε. While these precautions should be sufficient for simple domains, a more sophisticated approach should be taken if the algorithm needs to be used in praxis.

Overall, most probabilistic planning problems of the IPPC are supported and can be evaluated. The translation can thus be considered *essential* to discuss the problem statement. Exact calculations might not be possible due to rounding errors, but the solution could be verified exactly if desired, using the values of the RDDL domain.

5.3 Other Algorithms and Algorithm Names

Two simple algorithms are used to test alternative approaches to solving RDDL domains. The algorithm presented before will be called "*search algorithm*", the other two "*reward optimizer*" and "*probability optimizer*". All three can easily be analyzed regarding their probabilities, whereas for the other RDDL planners the probabilities would have to be calculated

manually. They operate on the same RDDL translation.

5.3.1 Reward Optimizer

As stated in Subsection 5.2.3, an optimization procedure in Z3 is used to determine the maximum reward. Its solution can be used as a plan, although its probability can be arbitrarily bad. It can be configured using all parameters specified in Subsection 6.2.2 except the reward approximation.

5.3.2 Probability Optimizer

Instead of searching for a probability range and then for the frequencies of occurrence with maximum success probability, the product

$$\prod_{i=1}^{n} p_i^{x_{j,i}}$$

as in Corollary 5.1.3 could be maximized using Z3's optimization procedure. As Z3 presumably only supports linear equations for its optimization (see Section 2.2), the equivalent sum

$$\sum_{i=1}^{n} x_{j,i} g_i$$

(as shown in Subsection 5.1.2) is minimized, although it suffers from the same problem regarding the number precision as the search algorithm.

5.3.3 Failed Algorithm Idea

An earlier approach did not rely on log-statements, but could not be used instead of Algorithm 1, because it would not always return correct upper and lower bounds for the frequencies of occurrence. The original idea for the algorithm is mentioned in Appendix A.3. It was based on a *search table*: Whenever the step count of the search was incremented, the next row of the search table was computed. The columns referred to the probabilities in the domain, in descending order, and the numbers in one column were the maximum allowed frequencies of occurrence of all probabilities up to the probability of that column. In each step, the SAT solver would have been used to determine whether the given problem is solvable under the constraints of a cell in the current row, which were gone through from right to left. As soon as a cell $n_{i,j}$ delivered a solvable constraint, it was used to determine the set of transitions that make up the maximum path in $\mathcal{M}$. The table alone could not be used to cover all possible combinations within a probability range, so it was decided to perform that calculation in SMT, by using the weighted sum as in Equation (5.7).

Chapter 6

Final Implementation and Evaluation

The presented algorithms are evaluated by applying them in practice using RDDLsim and comparing their performance to other RDDL planners. Especially the *usefulness* and *performance* of the computed solution, concluded from the received reward, the probability and the runtime, is analyzed in different RDDL domains from the IPPC 2011, 2014 and 2018.

6.1 Method

The IPPC 2018 uses RDDLsim to evaluate the solutions computed by the different planners [ippc]. Domains are simulated with RDDLsim and plans are requested multiple times, 75 times in [ippc]. This makes it possible to e.g. evaluate the average reward obtained by a planner. A similar approach is also used in this chapter.

Following the PROTOCOL.txt file and the client and server implementation in [rdd], a client was implemented to communicate with the RDDL simulator. It translates the model returned by Z3 so that actions taken in the current step of the simulation can be transmitted to the server and receives the current state by the server so that, if desired, these states can be used as constraints and another plan considering the new set of states can be calculated as well. The decision between domains and domain instances is made by changing configuration variables in the main file, but these settings could also be taken as command line parameters.

Each planning domain instance is evaluated in 30 (instead of 75) *rounds* for each planner, with a timeout of 60 minutes. This is due to the time that the computation of a solution can take. If at least one solution can be found in time, but if not all rounds can be evaluated, the results are not discarded, but instead used to estimate the expected performance for a higher timeout.

In comparison to other planners, the received rewards and the time left are being observed. Furthermore, different configurations of the presented solutions and similar approaches are

being compared regarding the expected reward, the highest possible reward according to the reward optimization, the probabilities and the time taken for the RDDL translation as well.

The expected reward r_{exp} - the reward the program expects to receive in case of success - and the mean probability p can be used to assess whether the implementation treats the probabilistic statements correctly: If either the expected reward is received in case of success, or else the worst reward in the domain r_{worst} is received in case of failure, then the expected value calculated as shown below should be close to the mean reward.

$$r_{exp}p + r_{worst}(1-p) = r_{expectation}$$

This will be called *reward expectation*. The probabilistic nature of the problem and the few runs do not allow for any form of verification, but they can indicate whether the implementation works as desired. In domains where more than two rewards are received, the mean expected reward is used for r_{exp} and the worst received reward for r_{worst}. The deviation from the mean received reward can be greater because this estimation is rather inaccurate, but it can indicate wrong results if the deviation is very high compared to the received reward. The value is not calculated if a new plan is generated in each step, but for Academic Advising, because then different plans might be calculated.

The evaluation was performed on a computer with 16GB of main memory and an Intel Core i5-8600K CPU (3,60GHz x 6) using Ubuntu 18.10 installed on a SSD.

6.2 Planner Options

The planner created for this thesis can use three different algorithms and takes four configuration parameters.

6.2.1 Planners

The *search algorithm*, the *reward optimizer* and the *probability optimizer* are compared.

The output of the reward optimizer allows for a valuation of the reward received by the other planners. Its probability can also be used to measure the minimum probability of success that the plan calculated by the search algorithm should have.

The correctness of the probability of the maximum plan returned by the search algorithm can be estimated using the output of the probability optimizer, although not verified: The benchmarks show that the weighted sum is not always optimized correctly. Furthermore, the comparison allows to assess the speed of the solution. As Z3 probably uses highly optimized procedures, the results of the probability optimizer will show that the translation can be used to obtain a goal much faster than possible with the presented solution.

6.2.2 Parameters

All planners can operate differently depending on their configuration. The translation can either use only *uninterpreted* parameters, if that is possible in the domain, partly uninterpreted parameters, where only quantifiers require interpretations, or *interpreted* parameters only.

The most important parameter is the *reward approximation*. It needs to be set carefully: If the approximation is set too high, the probability of success of the plans found by the planner might be too low in comparison to the maximum probability that could be obtained with lower values, resulting in a worse reward expectation. If the approximation is too low, and the domain definition allows for invalid or "noop" actions, then a plan with high likelihood but low reward might be found, even if a plan with higher reward and the same probability might exist. Therefore, to obtain proper results, often different values for the reward approximation need to be tried out first before the actual computation can begin. This process could be automated, but it would take up additional computation time. Whenever it was reasonable, the author decided that an approximation of at least 25% would be sufficient.

In some domains, like the "triangle-tireworld" domain, undesired outcomes of actions, which might lead to not reaching the goal state, can be repaired, e.g. by fixing a flat tire. It might be possible in these domains that, if a maximum plan fails after some steps because it does not match the RDDL simulation after a probabilistic outcome different than expected, another plan from this state on might still lead to the goal. The "*planEveryStep*" parameter can be set to true if the planner is supposed to be invoked each time the former plan failed, to try to find a new solution according to the current and former states. This might make the planner more competitive in comparison to RDDL online planners. As better values than the reward approximation could be obtained, planning will not stop when a goal is reached.

6.2.3 Code Changes

Two small errors have been detected in the code shortly before the submission of the thesis. The first error prevented the search algorithm from being able to plan again properly. It was resolved, including a check for the satisfiability of a problem before the search procedure is started to prevent it from going through all constraints up to the planning horizon. The search algorithm has only been used twice with this parameter in the Triangle Tireworld domain. The change should not have affected other runs, so only there two additional benchmarks for the fixed version were made.

The second error was caused by the class that is responsible for printing the results of a benchmark. It failed whenever no probabilities of a plan could be calculated. This never happened in the tested domains. It was fixed as well in the submitted code.

6.3 Other Planners

None of the other planners consider the probability of a solution only (see Section 4.1). Still, a comparison of the received rewards allows for an assessment of the usefulness of plans with

maximum probability in probabilistic planning.

The overall performance is only partially comparable. The RDDL planners use Singularity and thus a container and might run a little bit slower than the presented algorithms. It is also likely that the planners have been optimized performance-wise, whereas the presented solution is not. The received rewards are thus of higher interest than the time left.

Of all IPPC 2018 planners, only PROST and Imitation-Net were able to correctly handle `max-nondef-actions`, which is used in all domains but in those of the 2018 competition. Because of the bad performance of Imitation-Net in the IPPC (see [ippc]), also when tested in the first navigation instance where its runtime was high as well, only PROST was used for those instances. For Academic Advising and Cooperative Recon, domains of the IPPC 2018, the planner Random Bandit was used as well. It was chosen because it performed mediocre, about 30% worse than PROST in the competition regarding its overall score (see [ippc]).

6.4 Results

Seven domains of the IPPC were used to analyze the performance of the planner. All domains are described briefly, and the expected outcome and most important results are being discussed. The analyzed data that is referred to in each subsection can be found in Appendix A.5. All data refers to the value of the initially calculated plan only, so if "planEveryStep" is true the values for the expected reward and the probability might differ for plans generated in later steps. Domain and instance descriptions are in Appendix A.4.

Not all parameter combinations were used for all planners. Depending on the domain, some settings are pointless, for example planning in every step in the navigation domain. The choice of parameters is either justified by a short domain analysis or by the result of applying the parameters for one domain instance.

Because the domain instances grow in complexity, they were tested in ascending order. Whenever a procedure failed on an instance, it was usually not used on the following instances again.

The reward optimizer was mostly used with one setting only: In most of the domains, it was faster with interpreted parameters. Furthermore, due to the low amount of time for finding an optimal solution, planning again was also mostly turned on.

6.4.1 Navigation Domain (IPPC 2011)

In this domain, as stated in Appendix A.4.5, a robot must navigate in a 2D world. Each coordinate next to the robot's current position can be reached with a possibly different probability. The robot disappears in case the transition to another coordinate fails. A plan is successful if the robot reaches the goal state, which is the case if the reward is higher than $-\text{horizon}$. In all tested instances, the planning horizon is 40.

Expectation

Only the state transition of the robot can fail. Any failed action makes it impossible to get to the goal state. Thus, the most successful plan to reach the goal state must be the maximum plan, because all probabilistic statements in the domain affect the overall probability of success and negative outcome cannot be repaired.

Given a proper reward approximation, the planner should thus be competitive. The expected and actual reward and the outcome of the probability optimizer should allow to estimate the correctness of the calculated probability.

Results

In all tested instances, the probability of the probability optimizer and the search algorithm was, if a solution could be computed in time, at least 10 times higher than the probability of the plans given by the reward optimizer. The values of the probability optimizer did not change per round in all instances except in instance 2 and 10, with only little deviation in the former instance. Up to instance 7, the probability was higher than 85%, but then decreased from 60% in instance 8 down to 36% in instance 10. The mean probabilities of the search algorithm and the probability optimizer were always similar, with the same maximum value.

The reward expectation in this domain was mostly within a range of 10% of the received reward.

The reward optimization failed to receive a better reward than the worst reward in instances 3, 4, 7 and 10. Its best mean received reward was -36. In instance 9, the reward optimizer was successful once even though the probability of success was only 0.2%. The expected reward of the other algorithms mostly was equal to the worst reward that still satisfied the reward constraint, and in some steps no actions or actions without any effect on the state transitions were taken. The received reward of the probability optimizer was better than the one received by the search algorithm in 2 and 6, but the expected reward of the latter was better than the one of the former in 3, 4 and 6. In instance 6, with the same probability of success, only the search algorithm received the minimum reward of -40. In instance 9 and 10, the received reward of the probability optimizer was close to the worst reward.

The approximation in all instances was chosen so that the worst possible reward of a maximum path would be up to -24, except for instances 9 and 10 where it was up to -32.

Only in instance 1 and 2, the time taken by the algorithms was comparable, depending on the program parameters. From instance 3 on, the reward optimizer was faster than the probability optimizer, which in turn was faster than the search algorithm. The search algorithm got slower by over 20 minutes in instance 4 and 6, timed out after 4 runs in instance 7 and failed to obtain a solution in instance 8. It was not used afterwards.

In the first instance, planning again and using uninterpreted parameters took longer than the other approaches. The former seemed to result in a better reward, the latter did not have any effect on the reward or the probability. They were not used in the other instances, except for the reward optimizer in the second instance, which used planning again.

PROST only received significantly better rewards than the algorithms that calculated a maximum path in instance 3 and 4. It was always faster than the other algorithms - in instance 10, it took about 20 minutes, whereas the probability optimizer took twice as long. The planner usually got faster in each round, and seemed to learn from previous mistakes, so that sometimes the reward improved in each round as well.

Analysis

The *most successful plan* in the navigation domain is, as expected, the *maximum plan*. Its reward is much higher than the reward received by the reward optimizer due to the high probability of success. The probability algorithms perform very well, although the deviations of the probability optimizer show that the probabilities are not always calculated correctly when Z3's optimization procedure is used. This might be due to rounding problems. In domains where the search algorithm fails because of a timeout, the results of the probability optimizer can still be used to estimate how the search algorithm could perform if it had more time or were optimized. The comparable results regarding the probability and the reward expectations also indicate that, in this domain, the probability is considered correctly and that the calculated plans are maximum plans for the chosen reward approximation. Small deviations regarding the reward expectation can be explained by the small sample of 30 rounds and the stochastic nature of the problem which can cause similar plans to have different results in praxis. Thus, all results of the evaluation need to be treated with care.

Due to the lower probabilities, the reward that the reward optimizer received is lower. Especially in instances with probabilities of success lower than 1%, the plans calculated by the latter are mostly unsuccessful in all rounds. The rewards received by the other algorithms indicate that the *reward approximation is very important*, because the plans tend to have the lowest possible reward as no action or actions without an effect can be taken in any round. With the same probability, plans with a higher reward might have been found with a higher reward approximation. This is especially relevant in the last two domains, where the expected reward is already close to the worst reward. Lower approximations were chosen there so that the probability of a maximum plan would be at least 33%.

The differences in time could be expected, as the probability optimizer and search algorithm depend on the reward optimizer. They indicate that the *Z3 optimization procedure is much faster* than the search algorithm, but it is, with the chosen translation, also less exact, as the deviations show.

Uninterpreted parameters were not beneficial because all parameters were quantified in the domain. Planning again also should not be advantageous, as the robot vanishes as soon as an action fails. Thus, the slightly better results should be negligible.

Finally, *PROST outperformed* the Z3 optimization procedure regarding the runtime and performed comparably well or even better in some instances regarding the reward. As the IPPC planner improves over time, it has a clear advantage over the chosen method for the other algorithms, where each run is treated separate.

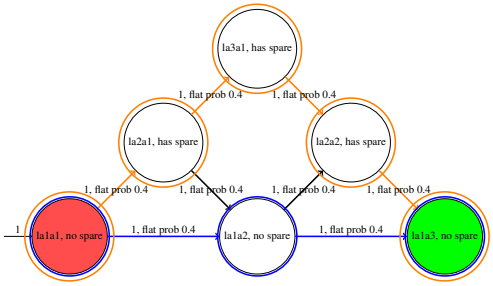

Figure 6.1: Instance 1 of Triangle Tireworld as a(n) (incomplete) graph (only actions that allow the car to move are shown)

The navigation domain proved to be a good testing domain for the algorithm. The programs calculated acceptable solutions but were less robust than the winner of the IPPC 2018: The search algorithm could not handle more complex instances, and the probability optimizer's runtime in the last instance indicates a similar disadvantage compared to PROST.

6.4.2 Triangle Tireworld Domain (IPPC 2014)

In the triangle tireworld (see Appendix A.4.7), a car is supposed to move from an initial state to a goal state by traversing differently connected locations. It can or cannot have a spare tire in the initial state and can collect other spare tires in certain locations. After each movement, the car can have a flat, and it can only continue if it has a spare and repairs the flat tire.

Due to the reward definition and the planning horizon, the reward is either -40 in case of failure or else at least 61.

Expectation

Like in the navigation domain, only the movement is probabilistic, but this time undesired outcomes of actions can be repaired, and a goal state can be reached even if the car got a flat. The maximum plan can be less successful than a plan that takes advantage of the possibility to change tires. It can therefore be expected that the algorithm will underperform in comparison to PROST.

The domain itself uses quantifiers or sums over all variables, so all parameters need to be interpreted - thus, the runtime of the uninterpreted approach is probably worse, as more constraints are generated.

Example Run for Instance 1

In the first instance, the car must navigate from position "la1a1" to "la1a3", and the probability to get a flat tire is 40% whenever it moves. A flat tire can be repaired if a spare has been picked up before at a state where a spare was available. Thus, when traversing the orange path, the car will always reach the goal state.

But the blue path takes less steps, so the product of all transition probabilities is lower there. The maximum plan therefore is the blue path. It is found by the search algorithm with the same settings as for the search algorithm in Appendix A.5 for this domain instance. The probability for not getting a flat is, despite the name, `FLAT-PROB`. The car can get a flat in the second transition and still reach the goal, so the maximum probability of success is $0.6 \cdot 0.4 = 0.24$. Because the resulting plan is also successful if the car does not get a flat after the second movement, the actual probability of success is $0.24 + 0.4 \cdot 0.4 = 0.4$.

After a test run, the actions of a maximum plan were stored. They are depicted in Appendix A.6. Most of the chosen actions do not have any effect - they could have been ruled out using `state-action-constraints`. The plan complies with the blue path.

Results

Only 5 out of 10 domain instances could be checked, and the search algorithm could only be used in the first two instances and timed out in the third one - it only got to s=5 using the search scheme in the first round, because Z3 took too long for the tested reward approximations. In these first instances, the reward optimizer found very likely paths as well.

The probability optimizer failed in the fifth instance, so no further instances were tested. From the third to the fifth instance, it could not finish all 30 rounds in time.

The probability of the paths found by the search algorithm for one instance did not change, but the expected reward was not always the same. The probabilities did deviate sometimes when Z3's optimization was used, but no probabilities were higher than the value of the paths found by the search algorithm.

The probabilities depended on the instance: The first two had maximum probabilities higher than 24%, the second two not much more than 5%. The mean reward was much higher in the first two instances as well.

When the translation time grew by the factor 8, the search algorithm stopped delivering results. When it grew again by 10, the same happened to the probability optimizer.

Using uninterpreted parameters mostly had a negative effect on the time left. Adjusting the plan had a similar effect, but always had a positive effect on the received reward. It was not used anymore when the algorithms started to time out even with the feature turned off.

The mean reward of the probability optimizer and the search algorithm were always better than the worst reward. The reward varied widely.

Only the reward optimizer was faster than PROST in instances 3 to 5. PROST always received better rewards.

The fixed version of the search algorithm did not deviate significantly from the previous version but was faster than before.

Analysis

The possible combination of parameters in the domain grows quadratically per instance. This property probably started to become relevant in the third instance. The *instance translation*

grew too large to allow for fast solving, so the time left declined until the search algorithm and the probability optimizer started to time out.

The optimization procedure did not always return a path with the same probability because it *could not always minimize the weighted sum* that it was given. The optimization problem might not be a linear optimization problem because of the usage of Boolean formulas to determine the frequencies of occurrence and thus might be unsupported by Z3 (see Section 2.2). Still, its results for the maximum path were close to and never higher than the results obtained by the search algorithm. It is thus likely that, given that the weighted sum was formulated correctly, the search for a maximum path succeeded in the first two instances.

The sample standard deviation of the expected reward for the search algorithm shows that there was more than one set of actions that could be used to reach a goal state.

As expected, uninterpreted parameters increased the time to search for a solution, as additionally all parameters also had to be quantified. The received reward using uninterpreted parameters was mostly higher, but that might be due to the higher expected reward. In the more complex translation, a shorter path to the goal with less "noops" might be found earlier than in the interpreted translation. Furthermore, 30 runs might not be enough to reflect possible changes in the translation when using uninterpreted parameters due to the behavior of Z3 in optimization and the stochastic nature of the domain simulation.

The collected data is not sufficient to make a statement about the correctness of the implementation, but the results in the first two instances seem to be promising, as paths with a rather high probability of success have been found. The example run for the first instance shows that the *search algorithm works correctly there*, but also calculates useless actions due to the low reward approximation. The overall reward would have been better with additional constraints that prevent this behavior. It also shows that the calculated probability does not take deviations from the plan into account: The actual success probability of the plan is 0.4, as the plan succeeds regardless of the outcome of the final transition. Thus, values for the reward expectation with 0.24 are lower than the actual mean reward. This shows that the reward expectation can only be used as an estimate in domains where the plan does not succeed if any transition fails.

Due to the higher complexity of the later instances, the runtime of the SMT-based approach was *faster than PROST* for the task of maximizing the reward. But as the algorithms did not consider the change-tire actions properly, *PROST was more successful* overall.

6.4.3 Academic Advising Domain (IPPC 2018)

As stated in the domain description (see Appendix A.4.1), a student is required to pass a given set of courses and gets a penalty for every step in which they did not pass all of them. Courses can be taken in any step and are passed with a certain probability, which is higher if courses marked as prerequisites have been completed in a former time step - the domain uses dynamic probabilities. In the examined domain instance, the reward only takes the penalty into account, which is -5. The worst reward in the chosen domain instance is -100 due to the

penalty.

In this domain, uninterpreted parameters were not used, as the constraints quantify all parameters.

Expectation

A plan with maximum probability should consist of the most likely order and combination of mandatory courses and their prerequisites, where courses that are prerequisites are only taken if the probability of passing them and the mandatory courses that rely on them is higher than passing the mandatory courses alone. The plan should take each course into account only once, as otherwise the probability of success could be higher if the course is not taken again. Thus, the plan should not be able to deal with failure.

With low probabilities, planning in every step should give a much better result. The domain fits the problem description and therefore the algorithm should perform well compared to IPPC planners.

Results

The probability of a plan calculated by the probability optimizer was higher with a lower approximation. The sample standard deviation for the probability was 0, meaning that, in this domain, the optimization procedure does not show unexpected changes in its outcome. The expected reward of the reward optimizer as well as the actual reward were higher than the rewards of the probability optimizer, the reward expectation for the former procedure was lower than the received reward. Planning again did not improve the reward of the latter with an approximation of $1/2$, but it improved with $1/3$. The search algorithm failed to calculate any solution in time - it got stuck at $s = 35$.

PROST was slower than the other algorithms that could generate a plan, which operated similarly, but it got faster in every round and calculated better plans. It received the best mean reward, only the reward optimizer performed comparable. Random Bandit was slower than PROST and received a mean reward that lies between the reward optimizer and PROST.

Analysis

If more plans can be generated, because of a lower reward constraint, then more likely plans can be obtained. The better performance of the reward optimizer is caused by using planning again, which improved the result, as the deviation of the reward expectation for the first plan from the received reward shows. The low probabilities of success and the performance of PROST indicate this as well. The results of the probability optimizer show that only the reward optimizer could plan again properly. This was due to the reward constraint for the higher approximation, as the mean reward received by the reward optimizer was lower than the reward constraint of -50 for the probability optimizer, and because of the low approximation value for the second run, where a plan with a higher reward value could have been

computed in the first step. In a future approach, it might be advisable to compute a maximum plan with maximum reward regarding all maximum plans.

The search algorithm failed to obtain a plan because the satisfiability check made by Z3 took too long for each constraint on s. This behavior is domain dependent and shows that the *presented idea is not ideal for all planning problems*.

PROST and Random Bandit outperformed the other approaches. But they also operated more slowly, meaning that in this domain the *algorithms using SMT had an advantage regarding their computation time*.

6.4.4 Cooperative Recon Domain (IPPC 2018)

According to the file description in Appendix A.4.2, this domain models planetary rovers. They can be moved, use tools, repair these tools and support other rovers, to increase the probability of successfully using the tools or because all rovers are equipped with different sets of tools. Thus, probabilities can be dynamic in this domain. They are rewarded for taking pictures of life in different cells of the 2D grid they navigate in, and the tools can be used to take pictures and to detect water or life if water was detected. Bad pictures with a broken camera give a lower reward and broken tools contaminate the examined object. Tools can get broken based on the cell where they are used.

The worst reward in this domain is 0, if no picture of life was taken.

Expectation

The planner should only consider repairing a tool if in some part of the plan it is more likely that a tool breaks than not. So, without planning in every step, life might get contaminated quickly and thus less pictures of life are expected to be taken.

If it is likely that a tool fails to detect life even with the support of another rover, or that a tool gets damaged, then the results should be much better if a new plan is generated whenever the old plan failed. The online RDDL planners should therefore perform better in the chosen instance.

Plans could possibly fail often in this domain, as constraints that depend on the outcome of probabilistic actions are part of the `action-preconditions`.

Results

As expected, plans often fail due to the action preconditions.

The probability optimizer only worked with an approximation lower than 0.25, although the received reward was sometimes higher than that with the 0.1-approximation. It calculated plans where sometimes a picture of the first object was taken with a reward of 11.63, and sometimes of the second object with 1.5. The search algorithm did not compute similar plans and was able to calculate the same probability in each of the 5 runs that could be performed

before a timeout occurred, whereas the probability optimizer at least once calculated a plan with very low probability of success.

When planning again, the results were better for the reward as well as the probability optimizer. The plan generated by the reward optimizer without planning again was much more successful than the associated probability would have implied. Also, the reward expectation deviated from the received reward.

PROST got a maximum reward equal to the reward calculated by the reward optimizer. It took much longer than any other method that terminated in time, but the average reward was more than twice as high compared to the highest received reward by the other algorithms. Random Bandit was not able to obtain a higher reward than 0 in two trials (only one run is shown in the table).

Analysis

The reward function seems to be too complex to be easily predictable by using the expected reward and the probability of a plan. One good picture of the first object already receives a reward of 11.63, and even though a less likely plan contains many probabilistic actions where grid cells are investigated and pictures are taken, the probability to take a good picture once is high - tools can only get broken after their first use, and *the probabilities that occur in total make the success of the solution seem less probable than it actually is*. A maximum plan in this domain does thus not correlate to a useful plan, because the probabilities are independent for each cell and a plan with a high probability of success should consider all cells with the highest possible probability of success individually. Therefore, the *reward optimizer was much more successful* than the other approaches, also because it always preferred the first object, which resulted in a higher reward.

The problematic behavior of the probability optimizer could again be explained by a *wrong optimization in Z3* or a translation error. The reward optimizer showed a similar behavior. As the search algorithm did not calculate different paths regarding the probability and the expected reward, the problem could be related to Z3 more than to the implementation. This is also indicated by the 0.1-approximation: A 0.25-approximation was evaluated as being true for the model of the 0.1-approximation. But if the former was added as constraint and the satisfiability of the same problem was checked again, no model could be found and the problem was considered to be unsatisfiable. The reason for this behavior of Z3 could not be explained.

The maximum reward PROST received indicates that the value calculated by the reward optimizer could be maximal. Overall, the domain is *not suitable* to examine the correctness of the results obtained by the algorithms and shows that there are domains where the *maximum path is not desirable*. Still, considering the performance of Random Bandit, the algorithms performed acceptable.

6.4.5 Crossing Traffic Domain (IPPC 2014)

Following the domain description in Appendix A.4.3, a robot navigates a 2D grid with a goal and an initial state. The domain is deterministic but for the obstacles that can randomly spawn at the rightmost side of the grid, except in the top or bottom row, and that move to the left. The robot needs to reach the goal and avoid the obstacles. It disappears after a collision with an obstacle.

A reward of -1 is assigned in each step if the robot did not reach the goal, else the reward is 0.

Expectation

None of the algorithms presented in this thesis can take the obstacles into account properly. The reward optimizer should calculate the shortest route to the goal state, and the other two planners should either assume that an object always spawns if that is more likely than the probability that it does not spawn, making it, depending on the domain and the initial position, impossible to reach the goal, or they will never consider that an object might appear.

If the goal or the initial position is in the rightmost state, all planners should fail whenever an object spawns. Otherwise, only planning in every step should work properly. Even then, the reward optimizer should calculate a more successful solution within the planning horizon, as the reward is higher for shorter paths. The other planners do not optimize the reward and might even fail because of the reward constraint. They should perform worse than the reward optimizer and the RDDL planners.

Results

The probabilities calculated by all algorithms were very low, but higher when uninterpreted parameters were chosen. The values did not change within 30 rounds. The rewards received by all algorithms except for the reward optimizer were low as well. The reward optimizer received its expected maximum reward at least once. The probability optimizer was run with the same settings twice but received different results although the expected reward was similar and the probabilities of success were equal.

With respect to the range of possible reward values from -1 to -40, the sample standard deviation of the reward was high with over 11. The reward expectation values deviated from the received reward.

The translation and calculation time increased when using the uninterpreted planning approach. The search algorithm took too long to obtain any path.

PROST was slower than most but performed better than all of the other algorithms.

Analysis

Because of the spawning property, the probabilities of all plans were close to zero. While the reward optimizer did not seem to produce errors, the different behavior for interpreted

and uninterpreted parameters could indicate an *error in the translation* or treatment of probabilistic statements. On the other hand, the Z3 optimization procedure did not produce stable results in most of the domains. The different values could be related to the different set of variables in the uninterpreted problem translation. Due to the higher number of variables, the uninterpreted approach took longer.

As expected, planning again and using the reward optimizer was much more useful than considering the probabilities. As the *probabilities did not regard the success of an action*, PROST could handle the domain better than the proposed algorithms.

The high deviation shows that the cars were either able to reach their goal if nothing happened, or else were unable to do so in case of failure, as described in the problem description.

6.4.6 Prop DBN Domain

This domain is taken from [rdd] and modified regarding its second instance. It was not part of the IPPC 2011/2014/2018. According to the domain description (see Appendix A.4.4), it represents a simple Dynamic Bayes Network with three state fluents and one action fluent.

Expectation

This is the only domain where no quantifiers are used. Therefore, the algorithms should be faster when uninterpreted parameters are used. The domain should work well with probabilistic planning due to the simple Bernoulli statements.

Results

In the first instance, down to a reward approximation of 12.5%, the expected reward and the probabilities of all calculated plans were equal, and the reward was similar as well, except for one deviation due to a timeout. No actions were taken in any step. For the probability optimizer and search algorithm, not choosing uninterpreted variables caused a timeout, with 21 successful rounds for the former and 3 successful rounds for the latter approach. The search algorithm also timed out when uninterpreted variables were used, with 8 completed rounds. The reward expectation value was calculated for the first entry only because deviations from the original plan are likely but their effect on the reward depends on when the plan failed. Thus, no conclusions regarding the correctness of the algorithms can be drawn from this value.

Since no actions needed to be taken, a second instance with different initial values for the variables was created. There, differences between the uninterpreted and the interpreted approach can be observed: The reward optimizer was run multiple times and delivered the same expected reward but different probabilities depending on these parameters. The probabilities were also different for the probability optimizer, with a higher maximum path found when using uninterpreted parameters, but a lower mean probability. The expected reward was different as well.

This time, actions were taken, and the time left was similar for all algorithms. PROST took a similar amount of time but got a higher minimum reward. Only the search algorithm was terminated manually at $s = 101$ after over an hour.

Analysis

The first instance is not suitable for evaluating the planners, because its optimal plan seems to include no actions. But it indicates that the translation and handling of the domain is similar for all planners, and the problems that arise in other domains do not appear. Furthermore, this is the only domain where no quantifiers, sums or products are used. Thus, as expected, the *uninterpreted approach is faster* than enumerating all parameters. This property could be *advantageous in large domains* without quantifiers - or with existential quantifiers that are not negated - in comparison to other IPPC planners, because the whole state in a time step, given by all interpretations of all functions with all possible combinations of their parameters, does not need to be considered in SMT solving then, but might be required in the other solving methods.

The *search procedure was too slow* because two probabilistic actions need to be taken in all steps and the time taken to solve each step was high. Still, the probabilistic values for all algorithms show that the *search algorithm worked as expected*. Due to the initial values, a transition with a probability of success of 0.8 needs to be taken at least once, else 0.9-transitions can be chosen. The highest probability of success thus is $0.9^{39} \cdot 0.8$, which is equal to the probability of all plans calculated by the planners.

In the second instance, an optimal plan regarding the received reward was harder to determine. Thus, PROST performed better than the other, simpler approaches, but they were not far behind. The decision between interpreted and uninterpreted parameters again has an impact on the solution found by Z3. The differences regarding the probabilities of the paths when using the probability optimizer are similar to the other problems shown before in other domains.

Overall, the proposed algorithms are less suitable for this domain, where, after a transition failed, a high reward can still be earned, and where thus a high probability alone is not enough to calculate a competitive plan.

6.4.7 Skill Teaching Domain (IPPC 2014)

The skill teaching domain is similar to the academic advising domain (see Appendix A.4.6): As explained in the RDDL file, a student is taught skills. Each skill can be improved by the agent by giving hints, which can improve a skill up to a medium proficiency, or by asking multiple choice questions. Skills can have “prerequisites” in form of other skills, which increase the probability of answering a question about it correctly. This property is encoded using if expressions, so no dynamic probabilities are used. Prerequisites must be known for hints to work. If questions are answered correctly, the proficiency of a skill increases, else it decreases, which might also happen randomly if the proficiency is high.

The reward is positive for each high proficiency level and negative for low levels, and dependent on the weight of each skill.

Expectation

As a high proficiency level can be lost anytime with some probability, it might be beneficial for the planner not to obtain a high level due to the probability optimization, depending on the combination of probabilities for losing the high proficiency level, answering questions correctly with the different levels and obtaining a higher level. But due to the reward constraint, high proficiency levels should still be preferred over lower levels.

A high reward should correlate with higher probabilities because of the changed likelihood of answering questions correctly for a high proficiency. Thus, both the reward optimizer and the algorithms that consider the probability should perform acceptable.

The approach of planning again should also have an advantage, because it can be used for correction if a higher proficiency level was not reached.

In comparison to the other planners, it is unclear how the solution performs.

Results

The results of the search algorithm, the probability optimizer and the reward optimizer (without uninterpreted parameters) did not deviate in the first instance.

For different approximation values - $0.15, 0.25, 0.35, 0.45, 0.55$ - the probability of a plan found by the probability optimizer did not decrease except for the last approximation value, where a timeout occurred after 10 rounds. In the third instance, only 3 rounds could be completed by the probability optimizer in time.

Plan rewards were higher when planning again was enabled. The reward optimizer obtained the highest rewards in all instances except for PROST.

The search algorithm terminated in time with an approximation of 0.25 but timed out after 25 steps with 0.45. The latter obtained a mean reward that was twice as good compared to the first. The probability value of the plans was equal to the highest probability in case a question was asked on a skill with medium proficiency. The reward expectation was mostly within a range of 10% of the received reward. The highest deviation occurred in the second instance, where the reward of the search algorithm should have been 5 times as high. The algorithm was less than half as fast compared to the other planners.

PROST could calculate a solution quickly in all instances and received the highest rewards.

Analysis

It seems that, with rising complexity, the optimization values for the probabilities begin to scatter. Either the translation or the optimization procedure of Z3 then begin to cause inexact results.

The different values for the approximation show that, when choosing an approximation value, probability and desired reward need to be weighed to obtain a maximum path with high probability of success when using the proposed algorithm. Also, restricting the amount of possible interpretations using the reward constraint seems to *increase the time* required to calculate a satisfying plan.

As expected, planning again was advantageous in this domain. Also, the reward optimizer obtained better results, and the difference regarding the probabilities indicates why: A plan is more likely to succeed if no questions are asked and a high proficiency level is not reached. Although a path with high probability of success could be found, and although the algorithm seems to work as required, that path is not desirable in the context of the domain, for teaching a student, and the approach that determines a plan with maximum reward is more successful when planning again.

The bad search result in the second instance was similar in a second trial. No explanation could be found for the high deviation from the reward expectation, especially because the problem did not arise with similar solutions of the other algorithms.

Again, PROST performed better due to its focus on the reward as well.

6.5 Summary

All algorithms introduced in this thesis proved to be able to handle RDDL domains for planning. In some domains, calculating a maximum plan or a plan with maximum reward was beneficial and lead to results that were comparable to the winner of the IPPC 2018, although the solving time was mostly higher.

Nonetheless, the current solution for the search algorithm is *not performant enough* to deal with complex planning problems and cannot benefit from the speed of the SMT solver, which was comparable to or faster than PROST when the reward optimization was chosen. The optimization procedures could handle more domains, but *did not always calculate reliable results*, either due to an implementation error, a translation or rounding problem or some problem with Z3's optimization procedure. For both approaches, the results regarding the probability of the plans were not unexpected, which indicates that the *calculation of a maximum path could be implemented successfully*, although not necessarily efficiently. Only the results of the search algorithm in Skill Teaching could not be explained.

From all maximum plans, a plan with maximum reward should be obtained to improve the results. Both the reward and the probability optimization alone only lead to a good reward in domains where a most probable plan or high rewards were encouraged. Else, as the different results for the reward approximation show, it is overall more advantageous in the context of the IPPC RDDL domains to maximize the reward expectation.

The Prop DBN domain indicates that the usage of uninterpreted parameters, if possible, can be beneficial regarding the runtime. Another positive effect on the runtime could be noted when the reward approximation was lower.

The evaluation also allows for conclusions regarding the initial problem statement. These will be discussed in the final chapter.

Chapter 7

Conclusion

7.1 Summary

In this thesis, a novel approach to solving planning problems with success probabilities using SMT was presented. Based on the domain definition language RDDL, 3 simple algorithms were elaborated that exploit different properties of the underlying planning problem: One algorithm optimizes the reward function only, the other two calculate a plan with maximum probability of success. They consider the product of all probabilities of the transitions made in all planning steps of a plan to be equal to its success probability. This product can be reformulated as a sum using the natural logarithm and can either be optimized directly or be translated in form of constraints. These constraints can be used to iteratively search for a maximum plan by calling the SMT solver in each iteration to evaluate their satisfiability. A former approach based on a similar idea that did not rely on the natural logarithm, but it did not work correctly. It would have been more numerically stable.

Subsequently, the different algorithms were evaluated on different IPPC planning domains, in comparison to participants of the IPPC 2018. On average, they took more time than PROST, the only planner they could be compared with in all domains, but were able to calculate solutions that complied with the expectations regarding their probability. The probability optimizer proved to be able to deal with more complex domains and operated faster than the search algorithm, but it was also less reliable. In domains where a high probability is important to receive a good average reward, the algorithms that determine a plan with maximum probability perfom well. The search for a maximum path was most successful when probabilities were directly connected to the possibility to reach a goal state, and a failed transition made any goal state unreachable. In other domains, especially when the probabilities were low or planning again with focus on a maximum reward was more successful because failure did not prevent to reach a goal state, the reward optimization performed better. Using uninterpreted function parameters was only useful in domains where most parameters were not quantified. In some domains, the rewards were close to the results of PROST, but overall

PROST obtained higher rewards. Random Bandit also obtained higher rewards but operated slower in Academic Advising, whereas in Cooperative Recon it could not calculate a useful solution.

Some of the results, for example in the Skill Teaching domain, and the deviations regarding the results of the probability optimizer show that the current solution might need to be revised. Explanations for this behavior could be related to numeric stability problems and the optimization behavior of Z3.

7.2 Discussion

An essential question for solving probabilistic planning problems with an SMT solver was how to encode probabilities in a language that does not explicitly support probabilistic expressions. The authors of [LSTZ15] decided to perform an intermediate step by expressing the problem in P4 and then translating this problem to quadratic equations that were solved by Z3. In this thesis, the problems are first expressed in another language as well, in RDDL. Probabilities are translated in form of constraints, as a weighted sum, but their semantic is omitted. They are treated similarly to a reward function that is either optimized using a search procedure that is based on the satisfiability of iteratively changing constraints or using Z3's optimization procedure. The actual reward function is ignored but for an approximation constraint.

This translation allows to calculate the probability of any plan that is found by the SMT solver. The treatment of these probabilities affects the runtime and the stability of the program. The implementation was not numerically stable, although floating point precision could be guaranteed to some degree except for the natural logarithm, so that most problems with simpler probabilistic statements should be solved correctly. The deviations of the reward and probability values indicate that the numeric stability of the problem or the formulation of the weighted sum caused a nondeterministic behavior in Z3's optimization procedure. Thus, the implementation is still improvable. The RDDL translation itself was likely correct, as generated plans always complied to all RDDL constraints that were not related to probabilistic outcomes. The proposed solution worked as desired but could not handle large domains and proved to be too slow or to provide plans that were not competitive to other approaches in some instances as well. Most probable solutions do not seem to be sufficient in domains that include a reward function. The search algorithm might be more useful in other areas than planning, for example for counterexample generation as in Section 4.2.

Although the reward function was not ignored by any algorithm, the reward approximation constraint was not always enough to define goal states. In a goal-based language, without any reward function, the algorithms should perform better, as the length of the path or similar properties that could be rewarded would not be relevant. The trade-off between runtime, high probabilities and a high reward as with the reward approximation could then be avoided.

As the translation was mostly much faster than the solving time, the constraints used to translate probabilistic RDDL statements might need to be improved further, or the statements

should be translated differently. Another solver than Z3 might also perform better for this problem class. As shown in Section 4.2, a direct translation to a SAT solver could also be beneficial. The solution leaves room for improvement, and some ideas will be discussed in the next section.

The results show that SMT solvers can be used to solve planning problems with success probabilities. Although the presented approach was rather naive, the performance of the implementation was not far off from the winner of the IPPC 2018 in some domains. Yet, the runtime in other domain instances indicates that it might not be suitable for all planning problems. Nonetheless, especially the performance in the navigation domain and the speed of the probability optimizer in most domains motivates further examinations of SMT solving in probabilistic planning for discrete probability distributions.

The translation of continuous distributions, on the other hand, is not covered by this thesis, cannot be performed with the introduced approach and might prove to be difficult, as the set of possible outcomes of a planning step might be infinite. Thus, for those planning problems, other methods than SMT solving might be preferable.

All in all, the presented work proposes a working translation of probabilistic expression to an SMT language, identifies use cases for most probable solutions and indicates that, at least for a subset of probabilistic planning problems, probabilistic planning using SMT could be an alternative to present planners.

7.3 Future Work

Based on the proposed translation of probabilistic expressions to SMT, more complex algorithms can be developed that do not rely on probabilistic values only. Optimizing both the reward and probabilities would lead to better results when the RDDL language is used as a basis. Another language with explicit goal states could be considered for the algorithm as well, as it was designed with goal states in mind. When using RDDL, it could be more suitable to define probability constraints instead of reward constraints, to generate a plan with maximum reward that at least succeeds with the given probability, if one exists. Thresholds for both the probability and the reward could be formulated as well.

The solutions found by PROST could improve in each round of the execution. Therefore, additionally to a more suitable algorithm, former plans and their properties, like the expected reward, received reward and success probability, could be stored after each run and the program could learn from past mistakes or use the found solution in the next run. Parameters like planning again, using interpreted or uninterpreted variables or the reward approximation could also be changed dynamically and be adjusted depending on the effect on the received reward.

Otherwise, another translation could be elaborated and might further improve the runtime. Larger domains could be handled more efficiently if a symbolic approach would be used. The presented search algorithm could also benefit from another search scheme, for example a binary search within a given range. Alternatively, if an exact solution is not required, the

approximate solution within the search range of Algorithm 1 could be sufficient and would work faster. Runtime could also be saved if the reward optimization procedure would not be used for the reward approximation, but if instead a fixed value for the minimum reward would be required - this should be possible if the user is informed about the planning problem.

The problem of numeric stability could be addressed if an alternative to the weighted sum could be elaborated. The former failed algorithm might indicate a solution.

All things considered, future algorithms that integrate these changes could be able to compete with probabilistic planners of the IPPC.

Bibliography

[AFTa] Murugeswari Issakkimuthu Alan Fern and Prasad Tadepalli. Imitation-Net: A supervised learning planner. URL: https://ipc2018-probabilistic.bitbucket.io/planner-abstracts/imitation-net.pdf [cited 22.11.2018].

[AFTb] Murugeswari Issakkimuthu Alan Fern and Prasad Tadepalli. Random-Bandit: An online planner. URL: https://ipc2018-probabilistic.bitbucket.io/planner-abstracts/random-bandit.pdf [cited 22.11.2018].

[AFTc] Murugeswari Issakkimuthu Alan Fern, Anurag Koul and Prasad Tadepalli. A2C-Plan: A reinforcement learning planner. URL: https://ipc2018-probabilistic.bitbucket.io/planner-abstracts/a2c-plan.pdf [cited 22.11.2018].

[BdMNW] Nikolaj Bjørner, Leonardo de Moura, Lev Nachmanson, and Christoph Wintersteiger. Programming Z3 - 8. optimization. URL: http://theory.stanford.edu/~nikolaj/programmingz3.html#sec-optimization [cited 06.01.2019].

[BDS02] Clark W. Barrett, David L. Dill, and Aaron Stump. Checking satisfiability of first-order formulas by incremental translation to SAT. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *Lecture Notes in Computer Science*, pages 236–249. Springer, 2002. ISBN: 3-540-43997-8. URL: https://doi.org/10.1007/3-540-45657-0_18, doi:10.1007/3-540-45657-0_18.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008. ISBN: 978-0-262-02649-9.

[BPF15] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. νz - an optimizing SMT solver. In Christel Baier and Cesare Tinelli, editors, *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, volume 9035 of *Lecture Notes in Computer Science*, pages 194–199. Springer, 2015. ISBN: 978-3-662-46680-3. URL: https://doi.org/10.1007/978-3-662-46681-0_14, doi:10.1007/978-3-662-46681-0_14.

[BT18] Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking.*, pages 305–343. Springer, 2018. The page numbers might differ from the original publication - as it could not be examined, the source that was cited was taken from http://theory.stanford.edu/ barrett/pubs/BT18-abstract.html. It refers to the Springer book, which was still unpublished when the comments on the website regarding the publication were written. URL: https://doi.org/10.1007/978-3-319-10575-8_11, doi:10.1007/978-3-319-10575-8_11.

[CCO+12] Amanda Jane Coles, Andrew Coles, Angel García Olaya, Sergio Jiménez Celorrio, Carlos Linares López, Scott Sanner, and Sungwook Yoon. A survey of the seventh international planning competition. *AI Magazine*, 33(1):83–88, 2012. URL: http://www.aaai.org/ojs/index.php/aimagazine/article/view/2392.

[CFLM16] Michael Cashmore, Maria Fox, Derek Long, and Daniele Magazzeni. A compilation of the full PDDL+ language into SMT. In *Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'16, pages 79–87. Association for the Advancement of Artificial Intelligence (AAAI), 2016. ISBN: 1-57735-757-4, 978-1-57735-757-5. URL: http://dl.acm.org/citation.cfm?id=3038594.3038605.

[CK] Hao Cui and Roni Khardon. The SOGBOFA system in IPC 2018: Lifted BP for conformant approximation of stochastic planning. URL: https://ipc2018-probabilistic.bitbucket.io/planner-abstracts/conformant-sogbofa-ipc18.pdf [cited 22.11.2018].

[FH02] Zhengzhu Feng and Eric A. Hansen. Symbolic heuristic search for factored Markov decision processes. In Rina Dechter, Michael J. Kearns, and Richard S. Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial*

Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada., pages 455–460. AAAI Press / The MIT Press, 2002. URL: http://www.aaai.org/Library/AAAI/2002/aaai02-069.php.

[GS] Florian Geißer and David Speck. PROST-DD-utilizing symbolic classical planning in THTS. URL: http://gki.informatik.uni-freiburg.de/papers/geisser-speck-ippc2018.pdf [cited 13.01.2019].

[GSS10] Michael Günther, Johann Schuster, and Markus Siegle. Symbolic calculation of k-shortest paths and related measures with the stochastic process algebra tool CASPA. In *Proceedings of the First Workshop on DYnamic Aspects in DEpendability Models for Fault-Tolerant Systems*, DYADEM-FTS '10, pages 13–18, New York, NY, USA, 2010. ACM. ISBN: 978-1-60558-916-9. URL: http://doi.acm.org/10.1145/1772630.1772635, doi:10.1145/1772630.1772635.

[HGSK07] Jörg Hoffmann, Carla P. Gomes, Bart Selman, and Henry A. Kautz. SAT encodings of state-space reachability problems in numeric domains. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 1918–1923, 2007. URL: http://ijcai.org/Proceedings/07/Papers/309.pdf.

[ippa] Icaps 2011 international probabilistic planning competition (IPPC). URL: http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/ [cited 07.01.2019].

[ippb] Icaps 2014 international probabilistic planning competition (IPPC) discrete track. URL: https://ssanner.github.io/IPPC_2014/ [cited 07.01.2019].

[ippc] International planning competition 2018 probabilistic tracks. URL: https://ipc2018-probabilistic.bitbucket.io/# [cited 22.11.2018].

[JÁZ+12] Nils Jansen, Erika Ábrahám, Barna Zajzon, Ralf Wimmer, Johann Schuster, Joost-Pieter Katoen, and Bernd Becker. Symbolic counterexample generation for discrete-time Markov chains. In Corina S. Pasareanu and Gwen Salaün, editors, *Formal Aspects of Component Software, 9th International Symposium, FACS 2012, Mountain View, CA, USA, September 12-14, 2012. Revised Selected Papers*, volume 7684 of *Lecture Notes in Computer Science*, pages 134–151. Springer, 2012. ISBN: 978-3-642-35860-9. URL: https://doi.org/10.1007/978-3-642-35861-6_9, doi:10.1007/978-3-642-35861-6_9.

[KH13] Thomas Keller and Malte Helmert. Trial-based heuristic tree search for finite horizon MDPs. In Daniel Borrajo, Subbarao Kambhampati, Angelo Oddi, and Simone Fratini, editors, *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, June 10-14, 2013*, pages 135–143. AAAI, 2013. ISBN: 978-1-57735-609-7. URL: http://www.aaai.org/ocs/index.php/ICAPS/ICAPS13/paper/view/6026.

[KS92] Henry Kautz and Bart Selman. Planning as satisfiability. In *ECAI 92: 10th European Conference on Artificial Intelligence*, volume 92, pages 359–363. Wiley, 1992. The page numbers differ from the original publication - as it could not be examined, the source that was cited was taken from CiteSeerX in http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.134.3573&rank=1. The site refers to the original document.

[LSTZ15] Meilun Li, Zhikun She, Andrea Turrini, and Lijun Zhang. Preference planning for Markov decision processes. In Blai Bonet and Sven Koenig, editors, *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 3313–3319. AAAI Press, 2015. ISBN: 978-1-57735-698-1. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9730.

[ML99] Stephen M. Majercik and Michael L. Littman. Contingent planning under uncertainty via stochastic satisfiability. In Jim Hendler and Devika Subramanian, editors, *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA.*, pages 549–556. AAAI Press / The MIT Press, 1999. ISBN: 0-262-51106-1. URL: http://www.aaai.org/Library/AAAI/1999/aaai99-078.php.

[Pap85] Christos H. Papadimitriou. Games against nature. *J. Comput. Syst. Sci.*, 31(2):288–301, 1985. URL: https://doi.org/10.1016/0022-0000(85)90045-5, doi:10.1016/0022-0000(85)90045-5.

[rdd] ssanner/rddlsim - code. URL: https://github.com/ssanner/rddlsim [cited 07.01.2019].

[San10] Scott Sanner. Relational dynamic influence diagram language (RDDL): Language description. 2010. URL: http://users.cecs.anu.edu.au/~ssanner/IPPC_2011/RDDL.pdf [cited 04.10.2018].

[Seb07] Roberto Sebastiani. Lazy satisfiability modulo theories. *Journal on Satisfiability, Boolean Modeling and Computation*, 3:141–224, 2007.

[ST12] Roberto Sebastiani and Silvia Tomasi. Optimization in SMT with LA(Q) cost functions. *CoRR*, abs/1202.1409, 2012. URL: http://arxiv.org/abs/1202.1409 [cited 13.01.2019], arXiv:1202.1409.

[ST14] Roberto Sebastiani and Patrick Trentin. Pushing the envelope of optimization modulo theories with linear-arithmetic cost functions. *CoRR*, abs/1410.5568, 2014. URL: http://arxiv.org/abs/1410.5568 [cited 13.01.2019], arXiv:1410.5568.

[ST17] Roberto Sebastiani and Patrick Trentin. On optimization modulo theories, MaxSMT and sorting networks. *CoRR*, abs/1702.02385, 2017. URL: http://arxiv.org/abs/1702.02385 [cited 13.01.2019], arXiv:1702.02385.

[YLWA05] Håkan L. S. Younes, Michael L. Littman, David Weissman, and John Asmuth. The first probabilistic track of the international planning competition. *J. Artif. Intell. Res.*, 24:851–887, 2005. URL: https://doi.org/10.1613/jair.1880, doi:10.1613/jair.1880.

[z3G] Z3Prover/z3 - code. URL: https://github.com/Z3Prover/z3 [cited 07.01.2019].

[z3M] Z3Prover/z3 - wiki - home. URL: https://github.com/Z3Prover/z3/wiki#background [cited 07.01.2019].

Appendix A

A.1 Abbreviations

IPC International Planning Competition

IPPC International Probabilistic Planning Competition

MC Markov Chain

MDP Markov Decision Process

OMT Optimization Modulo Theories

RDDL Relational Dynamic Influence Diagram Language

SMT Satisfiability Modulo Theories

A.2 Algorithm Example

The algorithm is performed on the MDP in Figure A.1 which is similar to Figure 5.1. Some transitions were left out - additionally to transitions leading from a state to another state with probability p, transitions with the same action lead from the state to itself with probability $1 - p$. These transitions will not be considered in the algorithm example for the sake of simplicity but would be treated similarly. They are considered by the actual algorithm. As they cannot be used to get closer to the goal state, they are not part of a maximum path. The initial state in Figure A.1 is “raw mat.”, the goal state is “final prod.”.

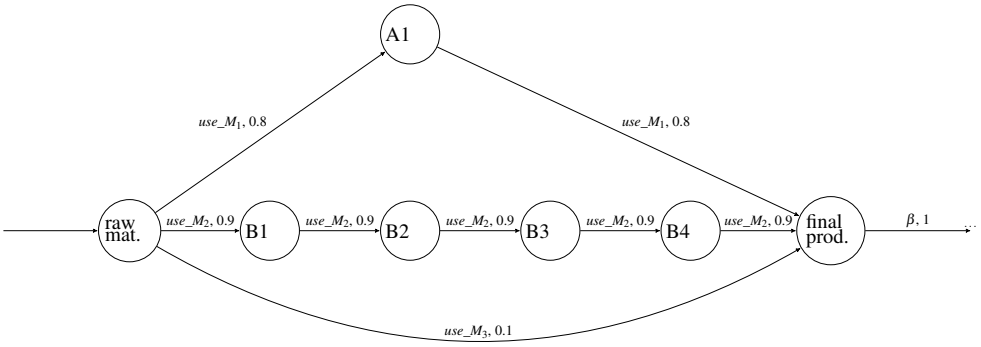

Figure A.1: MDP: Construction of a final product from raw materials using different machines

A.2.1 Algorithm 1

Remember: The algorithm for RDDL planning problems is not based on an actual graph search, but rather restricts the amount of transitions with a probability lower than 1.0 that can be taken in each step until a solution is found. The highest lower probability of success is 0.9.

The search constraints alone would normally be satisfied by subsets of the MDP that do not include the initial or the goal state as well. All parts of the MDP that could be considered before the consistency with the problem definition is checked could be shown here. To save space and for reasons of simplicity, the set of reachable states in each step from the initial state is shown instead. The latter must be part of any solution. The last figure is the first one that is consistent with the planning problem, meaning that here a path from the initial to a goal state was found. At this point algorithm 1 terminates, and the s value is passed to the second algorithm.

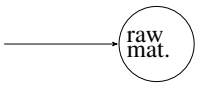

Figure A.2: s = 0

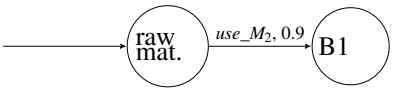

Figure A.3: s = 1

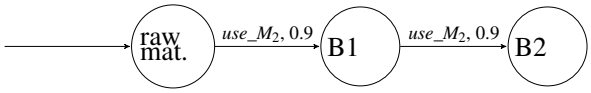

Figure A.4: s = 2

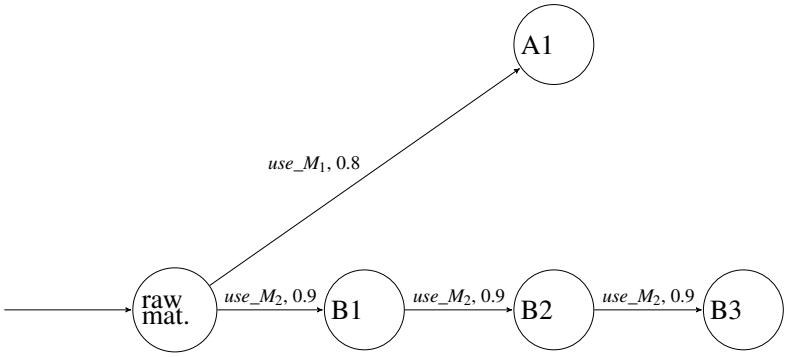

Figure A.5: s = 3

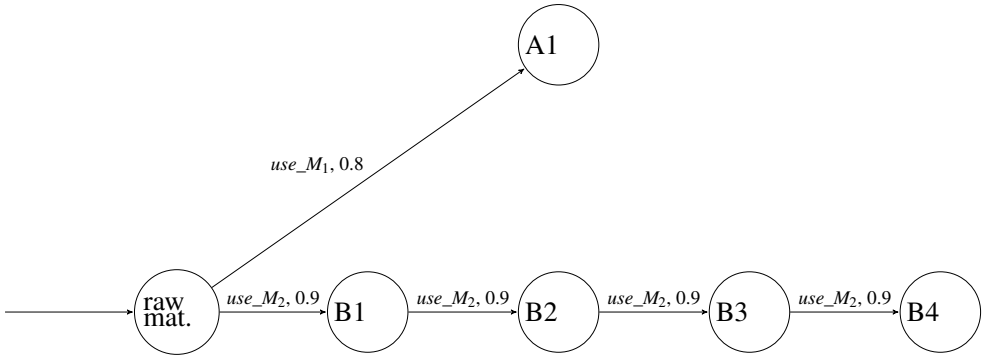

Figure A.6: s = 4

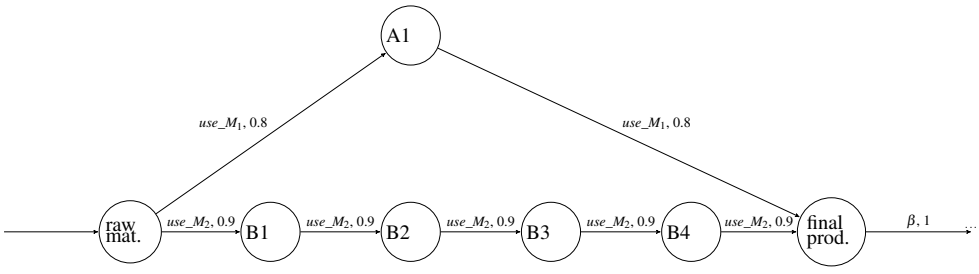

Figure A.7: s = 5

A.2.2 Algorithm 2

The second algorithm now iterates through all arrays with three entries where each entry can have a value between 0 and 5. These arrays are sorted by probability and then checked for satisfiability if their probability lies in $[0.9^5, 0.9^4)$. The first satisfiable array is returned and can be used as frequencies of occurrence to obtain a maximum plan. In this case, these match the upper path of Figure A.1.

```
1
2  ...
3  all_combinations ← [[1,0,0],[2,0,0],...,[5,0,0],[0,1,0],[0,2,0],...,[2,1,0],...]
4  all_combinations ← [[1,0,0],[2,0,0],...,[2,1,0],[0,2,0],[5,0,0],...] //after sorting them
5  ... //only the latter three arrays are within the probability range and are checked for satisfiability
6  [2, 1, 0] is not satisfiable //there is no path from the initial to the goal state with different transitions with
       probabilities 0.9 and 0.8
7  [0, 2, 0] is satisfiable −> return //only satisfied by the upper path from the initial to the goal state
```

As Figure A.1 includes the same path from the initial to the goal state as Figure 5.1, this is the most probable path in the MDP.

A.3 First Algorithm Idea

Definition A.3.1 (Search Table)
Let $M = (S, Act, P, t_{\text{init}}, AP, L)$ be an MDP, `horizon` the horizon of the search problem and $p_1, \ldots, p_n \in (0,1)$ be the values of all probabilities that occur in $\mathcal{M}$ lower than 1.0, sorted from highest to lowest. Then, $T_{p \geq p_j} \subseteq P$ denotes the set of all transitions in $\mathcal{M}$ that have a probability of success that is higher than or equal to p_j, given $j \in \{1, \ldots, n\}$.

Given some $m \in \mathbb{N}$, the table

step	p_1	...	p_j	...
0	0	...	0	...
1	1	...	$n_{1,j}$	...
⋮	⋮	⋮	⋮	⋮
$i-1$	$i-1$	...	$n_{i-1,j}$	...
i	i	...	$n_{i,j}$	...
⋮	⋮	⋮	⋮	⋮
m	`horizon`	...	`horizon`	...

is the *search table* S_n, where each row represents a step in the search, and each column (titled with p_j) represents how many transitions $t_j \in T_{p \geq p_j}$ can be taken in step i in $\mathcal{M}$, so that $\sum_{t_j \in T_{p \geq p_j}}$ transition t_j was taken $\leq n_{i,j}$. The set of all sets of transitions $t_j \in T_{p \geq p_j}$ that fulfill this condition is called $T_{i,j}$.

In each step, $n_{i,1}$ is incremented by one, whereas $n_{i,j}$ is only incremented by one if

$$p_j^{n_{i-1,j}+1} \geq p_1^{n_{i,1}}$$

and else remains unchanged. As soon as $n_{i,1} = horizon$, $n_{i,2}$ is incremented in each step, as incrementing $n_{i,1}$ does not make sense in a non-concurrent domain when the transition cannot be taken more often due to the constraint given by `horizon`. The same behavior applies to all following situations where $n_{i,j} = horizon$ for any $j \in \{1,\ldots,n\}$, until, in the final row of S, $n_{i,j} = horizon$ for all $j \in \{1,\ldots,n\}$.

The initial algorithm idea was to dynamically create the search table until a solution is found in a row. Each row would have been traversed from right to left, and $T_{i,j}$ would have been used instead of the frequencies of occurrence. As soon as one of the constraints given by $n_{i,j}, i \in [1,m], j \in [1,n]$ was satisfiable, the second algorithm Algorithm 2 could have been used with $T_{i,j}$ to find a maximum path. This original idea did not always return a maximum path. There could not be found a change to the idea of the search table that would have allowed it to operate similarly to Algorithm 1.

A.4 IPPC RDDL Domains

These domains were taken from the domain download links in [ippc] [ippb] [ippa] or the example domains in [rdd] and changed so that no underscores are used but for those that are part of the RDDL syntax. Shortly after the download, the links in [ippb] and [ippa] stopped working. Thus, all used domains and their instances are shown in this part of the appendix.

A.4.1 Academic Advising (see [ippc])

Domain

```
1  ////////////////////////////////////////////////////////////////////////////
2  // //
3  // //
4  // RDDL MDP version of the IPC 2018 Academic Advising domain. //
5  // //
6  // //
7  // Created and modified for the probabilistic tracks of IPC 2014 by //
8  // //
9  // Libby Ferland (libby.knouse [at] uky.edu) and //
10 // Scott Sanner (ssanner [at] mie.utoronto.ca) //
11 // //
12 // and modified for the probabilistic tracks of IPC 2018 by //
13 // Thomas Keller (tho.keller [at] unibas.ch). //
14 // //
15 // //
16 // In this domain, a student may take courses at a given cost and passes the //
17 // course with a probability determined by how many of the prerequisites they //
18 // have successfully passed. A student also receives a penalty at each time //
19 // step if they have not yet graduated from their program (i.e., completed all //
20 // required courses). We allow multiple courses to be taken in a semester in //
21 // some instances. //
22 // //
```

```
// //
////////////////////////////////////////////////////////////////////////////

domain academic-advising_mdp {
    requirements {
        reward-deterministic,
        preconditions
    };


    types {
        course : object;
    };


    pvariables {
        ////////////////////// non-fluents ///////////////////////

        // number of courses that can be taken in parallel, (introduced for IPC
        // 2018, replaces max-nondef-actions of IPC 2014 domain)
        COURSES-PER-SEMESTER : { non-fluent, int, default = 1 };

        // first argument is a prerequisite of second argument
        PREREQ(course, course) : { non-fluent, bool, default = false };

        // probability of passing a course without prerequisites
        PRIOR-PROB-PASS-NO-PREREQ(course) : { non-fluent, real, default = 0.8 };

        // base probability of passing a course with prerequisites (if no
        // prerequisite has been passed)
        PRIOR-PROB-PASS(course) : { non-fluent, real, default = 0.2 };

        // program requirements for graduation
        PROGRAM-REQUIREMENT(course) : { non-fluent, bool, default = false };

        // cost for taking a course the first time
        COURSE-COST : { non-fluent, real, default = -1 };

        // cost for taking a course except the first time
        COURSE-RETAKE-COST : { non-fluent, real, default = -2 };

        // penalty per step for incomplete program
        PROGRAM-INCOMPLETE-PENALTY : { non-fluent, real, default = -5 };


        ////////////////////// state-fluents ///////////////////////

        // course has been taken successfully
        passed(course) : { state-fluent, bool, default = false };

        // course has been taken at least once
        taken(course) : { state-fluent, bool, default = false };
```

```
75
76
77          ////////////////////// action−fluents //////////////////////
78
79          // take a course
80          take−course(course) : { action−fluent, bool, default = false };
81      };
82
83
84      cpfs {
85          passed'(?c) =
86              // if ?c is taken and has no prerequisites, it's passed according to
87              // a prior probability
88              if ( take−course(?c) & ~( exists_{?c2 : course} [ PREREQ(?c2,?c) ] ) )
89                  then Bernoulli( PRIOR−PROB−PASS−NO−PREREQ(?c) )
90
91              // if ?c is taken and has no prerequisites, it's passed according to
92              // a prior probability and a bonus depending on passed prerequisites
93              else if ( take−course(?c) )
94                  then Bernoulli( PRIOR−PROB−PASS(?c) +
95                                      ( (1 − PRIOR−PROB−PASS(?c) ) ∗ ( sum_{?c2 : course} [ PREREQ(?c2,?c)
                                            & passed(?c2) ] ) /
96                      (1 + sum_{?c2 : course} [ PREREQ(?c2,?c) ] ) ) )
97
98              // otherwise, the value persists
99              else passed(?c);
100
101
102         // ?c is taken if it has been taken earlier or is taken now
103         taken'(?c) = taken(?c) | take−course(?c);
104     };
105
106
107     reward =
108         // taking a course for the first time incurs a cost
109         ( sum_{ ?c : course } [ COURSE−COST ∗ ( take−course(?c) & ~taken(?c) ) ] ) +
110
111         // taking a course that has been taken before incurs a (usually higher) cost
112         ( sum_{ ?c : course } [ COURSE−RETAKE−COST ∗ ( take−course(?c) & taken(?c) ) ] ) +
113
114         // as long as the program is not completed, a penalty is incurred
115         ( PROGRAM−INCOMPLETE−PENALTY ∗ ~( forall_{ ?c : course } [ PROGRAM−REQUIREMENT
                (?c) => passed(?c) ] ) );
116
117
118     action−preconditions {
119         // only take courses that haven't been passed
120         ( forall_{ ?c : course } [ take−course(?c) => ~passed(?c) ] );
121
122         // replaces max−nondef−actions
123         ( ( sum_{ ?c : course } [take−course(?c)] ) <= COURSES−PER−SEMESTER );
124     };
```

```
}
```

Instance

```
////////////////////////////////////////////////////////////////////////
// //
// //
// RDDL MDP version of Academic Advising instance #01 for IPC 2018 by Thomas //
// Keller (tho.keller [at] unibas.ch), based on the IPC 2014 domain by Libby //
// Ferland (libby.knouse [at] uky.edu). //
// //
// //
// The naive policy that ignores all preconditions and takes courses that are //
// program requirements until it succeeds is expected to complete the program //
// after 15.00 steps with an expected reward of -75.00. //
// //
// //
////////////////////////////////////////////////////////////////////////

instance academic-advising_inst_mdp__01 {
    domain = academic-advising_mdp;
    objects {
        course : { c0000, c0001, c0002, c0003, c0004, c0100, c0101, c0102, c0103, c0200, c0201, c0202, c0300,
            c0301, c0302 };
    };

    non-fluents {
        COURSES-PER-SEMESTER = 1;

        // PRIOR PROBS
        PRIOR-PROB-PASS-NO-PREREQ(c0000) = 0.80;
        PRIOR-PROB-PASS-NO-PREREQ(c0001) = 0.55;
        PRIOR-PROB-PASS-NO-PREREQ(c0002) = 0.67;
        PRIOR-PROB-PASS-NO-PREREQ(c0003) = 0.78;
        PRIOR-PROB-PASS-NO-PREREQ(c0004) = 0.75;
        PRIOR-PROB-PASS(c0100) = 0.22;
        PRIOR-PROB-PASS(c0101) = 0.45;
        PRIOR-PROB-PASS(c0102) = 0.41;
        PRIOR-PROB-PASS(c0103) = 0.44;
        PRIOR-PROB-PASS(c0200) = 0.14;
        PRIOR-PROB-PASS(c0201) = 0.07;
        PRIOR-PROB-PASS(c0202) = 0.24;
        PRIOR-PROB-PASS(c0300) = 0.23;
        PRIOR-PROB-PASS(c0301) = 0.08;
        PRIOR-PROB-PASS(c0302) = 0.16;

        // PREREQ
        PREREQ(c0003, c0100);
        PREREQ(c0000, c0100);
        PREREQ(c0004, c0100);
```

```
        PREREQ(c0001, c0101);
        PREREQ(c0002, c0101);
        PREREQ(c0000, c0102);
        PREREQ(c0004, c0102);
        PREREQ(c0001, c0103);
        PREREQ(c0001, c0200);
        PREREQ(c0101, c0200);
        PREREQ(c0103, c0201);
        PREREQ(c0002, c0202);
        PREREQ(c0200, c0300);
        PREREQ(c0201, c0301);
        PREREQ(c0201, c0301);
        PREREQ(c0200, c0302);

        // PROGRAM REQUIREMENT
        PROGRAM-REQUIREMENT(c0300);
        PROGRAM-REQUIREMENT(c0202);
        PROGRAM-REQUIREMENT(c0101);
        PROGRAM-REQUIREMENT(c0002);
        PROGRAM-REQUIREMENT(c0001);

        // COURSE COSTS
        COURSE-COST = 0;
        COURSE-RETAKE-COST = 0;

    };

    init-state {
        ~passed(c0000);
    };

    horizon = 20;

    discount = 1.0;
}
```

A.4.2 Cooperative Recon (see [ippc])

Domain

```
////////////////////////////////////////////////////////////////////////////
// //
// //
// RDDL MDP version of the IPC 2018 Cooperative Recon domain. //
// //
// //
// Based on the Recon domain that has been created for the probabilistic //
// tracks of IPC 2011 by //
// //
// Tom Walsh (thomasjwalsh [at] gmail.com) //
```

```
// //
// and modified for the probabilistic tracks of IPC 2018 by //
// Thomas Keller (tho.keller [at] unibas.ch). //
// //
// //
// In the Cooperative Recon domain, the planner controls one or more planetary //
// rovers that examine objects of interest in order to detect life and take a //
// picture of it. As in the Recon domain of IPC 2011, first has to be detected //
// before life is detected, and negative results (one for life, two for water) //
// contaminate the object of interest such that no life can be detected. //
// //
// The main changes compared to the IPC 2011 Recon domain that have been //
// realized are as follows: //
// //
// 1. In the 2011 version, taking pictures with a damaged camera lead to a //
// negative reward, which is never a reasonable option (not taking a //
// picture at all is always better). Here, we grant a lower positive //
// reward instead, which makes for interesting decisions between returning //
// to the base to repair the camera or go with the lower reward. //
// //
// 2. Hazards are replaced by a more general mechanism where probabilities //
// that a tool is damaged are directly linked to the cell. However, the //
// instance generation script still distributed hazards over the grid to //
// compute these probabilities. The main difference is that hazards can //
// overlap in a way that the probabilities accumulate. //
// //
// 3. In the IPC 2011 instance, all rovers were equipped with a tool to detect //
// water, a tool to detect life and a camera. In the instances for IPC //
// 2018, some rovers are only partially equipped such that the rovers have //
// to collaborate to perform all required tests. //
// //
// 4. To emphasize colaboration even more, there is a novel support-agent //
// action that rovers can take to increase the probability for successfully //
// detecting life or water. This leads to interesting decisions between //
// optimizing the probability of successfully detecting life and the number //
// of objects of interest that can be examined within the finite horizon. //
// //
// //
////////////////////////////////////////////////////////////////////////////////

domain cooperative-recon_mdp {
    requirements {
        reward-deterministic,
        preconditions
    };


    types {
        xpos : object;
        ypos : object;
        object-of-interest : object;
        agent : object;
```

```
63          tool : object;
64      };
65
66
67      pvariables {
68          ////////////////////// non-fluents ///////////////////////
69
70          // connectivity of the grid
71          ADJACENT-UP(ypos,ypos) : { non-fluent, bool, default = false };
72          ADJACENT-DOWN(ypos,ypos) : { non-fluent, bool, default = false };
73          ADJACENT-RIGHT(xpos, xpos) : { non-fluent, bool, default = false };
74          ADJACENT-LEFT(xpos,xpos) : { non-fluent, bool, default = false };
75
76          // location of object of interest
77          OBJECT-AT(object-of-interest, xpos, ypos) : { non-fluent, bool, default = false };
78
79          // probability a tool is damaged at a given grid cell
80          DAMAGE-PROB(xpos, ypos) : { non-fluent, real, default = 0.0 };
81
82          // probability water or life is detected with non-damaged tool and without support
83          DETECT-PROB : { non-fluent, real, default = 0.6 };
84
85          // probability water or life is detected with damaged tool and without support
86          DETECT-PROB-DAMAGED : { non-fluent, real, default = 0.3 };
87
88          // probability water or life is detected with non-damaged tool and with support
89          DETECT-PROB-WITH-SUPPORT : { non-fluent, real, default = 0.8 };
90
91          // probability water or life is detected with damaged tool and with support
92          DETECT-PROB-DAMAGED-WITH-SUPPORT : { non-fluent, real, default = 0.5 };
93
94          // tool is a camera
95          CAMERA-TOOL(tool) : { non-fluent, bool, default = false };
96
97          // tool is a life-detector
98          LIFE-TOOL(tool) : { non-fluent, bool, default = false };
99
100         // tool is a water-detector
101         WATER-TOOL(tool) : { non-fluent, bool, default = false };
102
103         // tool is mounted on agent
104         HAS-TOOL(agent, tool) : { non-fluent, bool, default = false };
105
106         // grid cell is a base (where tools can be repaired)
107         BASE(xpos, ypos) : { non-fluent, bool, default = false };
108
109         // reward for taking a picture with a non-damaged camera
110         GOOD-PIC-REWARD(object-of-interest) : { non-fluent, real, default = 10.0 };
111
112         // reward for taking a picture with a damaged camera
113         BAD-PIC-REWARD(object-of-interest) : { non-fluent, real, default = 5.0 };
114
```

```
115
116         ////////////////////// state-fluents //////////////////////
117
118         // tool is damaged
119         damaged(tool) : { state-fluent, bool, default = false };
120
121         // water detector has been used on this object of interest (at least once)
122         waterChecked(object-of-interest) : { state-fluent, bool, default = false };
123
124         // water has been detected at this object of interest
125         waterDetected(object-of-interest) : { state-fluent, bool, default = false };
126
127         // life detector has been used on this object of interest (at least once)
128         lifeChecked(object-of-interest) : { state-fluent, bool, default = false };
129
130         // life detector has been used on this object of interest at least twice
131         lifeChecked2(object-of-interest) : { state-fluent, bool, default = false };
132
133         // life has been detected at this object of interest
134         lifeDetected(object-of-interest) : { state-fluent, bool, default = false };
135
136         // a picture of this object of interest has been taken
137         pictureTaken(object-of-interest) : { state-fluent, bool, default = false };
138
139         // the location of an agent
140         agent-at(agent, xpos, ypos) : { state-fluent, bool, default = false };
141
142
143         ////////////////////// action-fluents //////////////////////
144
145         // move an agent upwards
146         up(agent) : { action-fluent, bool, default = false };
147
148         // move an agent downwards
149         down(agent) : { action-fluent, bool, default = false };
150
151         // move an agent to the left
152         left(agent) : { action-fluent, bool, default = false };
153
154         // move an agent to the right
155         right(agent) : { action-fluent, bool, default = false };
156
157         // have an agent use a tool on an object of interest
158         use-tool-on(agent, tool, object-of-interest) : { action-fluent, bool, default = false };
159
160         // have an agent support another agent using a tool
161         support-agent(agent, agent) : { action-fluent, bool, default = false };
162
163         // repair a tool
164         repair(agent, tool) : { action-fluent, bool, default = false };
165     };
166
```

```
167
168     cpfs {
169         damaged'(?t) =
170             // if an agent repairs ?t, it's not damaged
171             if ( exists_{ ?a: agent } [ repair(?a, ?t) ] )
172                 then false
173             // if it was damaged and is not repaired, it remains damaged
174             else if ( damaged(?t) )
175                 then true
176             // otherwise, it becomes damaged with a probability given by the location
177             // of the agent carrying the tool
178             else Bernoulli( ( sum_{ ?a: agent, ?x : xpos, ?y : ypos } [ HAS-TOOL(?a, ?t) * agent-at(?a, ?x, ?y)
                    * DAMAGE-PROB(?x, ?y) ] ) );
179
180         waterChecked'(?o) =
181             // remains true of becomes true if a water detector is applied now
182             waterChecked(?o) |
183             exists_{ ?a: agent, ?t: tool } [ use-tool-on(?a, ?t, ?o) & WATER-TOOL(?t) ];
184
185         waterDetected'(?o) =
186             // once water is detected, it remains this way
187             if (waterDetected(?o))
188                 then true
189             // if checking for water fails once, no water is ever detected
190             else if (waterChecked(?o))
191                 then false
192             // an agent checks now with a damaged tool and with support
193             else if ( exists_{?t : tool, ?a1: agent, ?a2 : agent} [ use-tool-on(?a1, ?t, ?o) & support-agent(?a2, ?
                    a1) & WATER-TOOL(?t) & damaged(?t) ] )
194                 then Bernoulli(DETECT-PROB-DAMAGED-WITH-SUPPORT)
195             // an agent checks now with a damaged tool and without support
196             else if ( exists_{?t : tool, ?a: agent} [ use-tool-on(?a, ?t, ?o) & WATER-TOOL(?t) & damaged(?t)
                    ] )
197                 then Bernoulli(DETECT-PROB-DAMAGED)
198             // an agent checks now with a non-damaged tool and with support
199             else if ( exists_{?t : tool, ?a1: agent, ?a2 : agent} [ use-tool-on(?a1, ?t, ?o) & support-agent(?a2, ?
                    a1) & WATER-TOOL(?t) ] )
200                 then Bernoulli(DETECT-PROB-WITH-SUPPORT)
201             // an agent checks now with a non-damaged tool and without support
202             else if ( exists_{?t : tool, ?a: agent} [ use-tool-on(?a, ?t, ?o) & WATER-TOOL(?t) ] )
203                 then Bernoulli(DETECT-PROB)
204             // the value persists
205             else false;
206
207         lifeChecked'(?o) =
208             // remains true of becomes true if a life detector is applied now
209             lifeChecked(?o) |
210             exists_{ ?a: agent, ?t: tool } [ use-tool-on(?a, ?t, ?o) & LIFE-TOOL(?t) ];
211
212         lifeChecked2'(?o) =
213             // true if it was true before or if a life detector is applied now and was applied before
214             lifeChecked2(?o) |
```

```
215             ( lifeChecked(?o) & exists_{ ?a: agent, ?t: tool } [ use−tool−on(?a, ?t, ?o) & LIFE−TOOL(?t) ] );
216
217         lifeDetected'(?o) =
218             // once life is detected, it remains this way
219             if ( lifeDetected(?o) )
220                 then true
221             // if checking for life fails twice or there is no water, no life is ever detected
222             else if ( lifeChecked2(?o) | ~waterDetected(?o) ) // Never detect life after 2nd try or if no water
223                 then false
224             // an agent checks now with a damaged tool and with support
225             else if ( exists_{?t : tool, ?a1: agent, ?a2 : agent} [ use−tool−on(?a1, ?t, ?o) & support−agent(?a2, ?
                    a1) & LIFE−TOOL(?t) & damaged(?t) ] )
226                 then Bernoulli(DETECT−PROB−DAMAGED−WITH−SUPPORT)
227             // an agent checks now with a damaged tool and without support
228             else if ( exists_{?t : tool, ?a: agent} [ use−tool−on(?a, ?t, ?o) & LIFE−TOOL(?t) & damaged(?t) ] )
229                 then Bernoulli(DETECT−PROB−DAMAGED)
230             // an agent checks now with a non−damaged tool and with support
231             else if ( exists_{?t : tool, ?a1: agent, ?a2 : agent} [ use−tool−on(?a1, ?t, ?o) & support−agent(?a2, ?
                    a1) & LIFE−TOOL(?t) ] )
232                 then Bernoulli(DETECT−PROB−WITH−SUPPORT)
233             // an agent checks now with a non−damaged tool and without support
234             else if ( exists_{?t : tool, ?a: agent} [ use−tool−on(?a, ?t, ?o) & LIFE−TOOL(?t) ] )
235                 then Bernoulli(DETECT−PROB)
236             // the value persists
237             else false;
238
239         pictureTaken'(?o) =
240             // remains true of becomes true if a picture os taken now
241             pictureTaken(?o) |
242             ( exists_{ ?a: agent, ?t: tool } [ use−tool−on(?a, ?t, ?o) & CAMERA−TOOL(?t) ] );
243
244         agent−at'(?a, ?x, ?y) =
245             // agent moves to the left and ends up here
246             if ( left(?a) & ( exists_{ ?x2 : xpos } [ agent−at(?a, ?x2, ?y) & ADJACENT−LEFT(?x, ?x2) ] ) )
247                 then true
248             // agent moves to the right and ends up here
249             else if( right(?a) & ( exists_{ ?x2 : xpos } [ agent−at(?a, ?x2, ?y) & ADJACENT−RIGHT(?x, ?x2) ]
                    ) )
250                 then true
251             // agent moves upwards and ends up here
252             else if( up(?a) & ( exists_{ ?y2 : ypos } [ agent−at(?a, ?x, ?y2) & ADJACENT−UP(?y, ?y2) ] ) )
253                 then true
254             // agent moves downwards and ends up here
255             else if( down(?a) & ( exists_{ ?y2 : ypos } [ agent−at(?a, ?x, ?y2) & ADJACENT−DOWN(?y, ?y2) ]
                    ) )
256                 then true
257             // agent moves, but it doesn't end up here
258             else if ( left(?a) | right(?a) | up(?a) | down(?a) )
259                 then false
260             // agent doesn't move, so it is here only if it was already here
261             else agent−at(?a, ?x, ?y);
262     };
```

```
263
264
265     reward =
266         ( sum_{?o : object-of-interest}
267             // get reward for a picture of an object of interest with life with a non-damged camera
268             [ ( GOOD-PIC-REWARD(?o) * exists_{ ?a: agent, ?t: tool } [ use-tool-on(?a, ?t, ?o) &
                        CAMERA-TOOL(?t) & ~damaged(?t) ] ) +
269
270             // get reward for a picture of an object of interest with life with a damged camera
271               ( BAD-PIC-REWARD(?o) * exists_{ ?a: agent, ?t: tool } [ use-tool-on(?a, ?t, ?o) & CAMERA
                        -TOOL(?t) & damaged(?t) ] ) ] );
272
273
274     action-preconditions {
275         // dont move outside of the grid
276         forall_{ ?a : agent } [ left(?a) => exists_{ ?x1 : xpos, ?x2 : xpos, ?y : ypos } [ agent-at(?a, ?x1, ?y) &
                ADJACENT-LEFT(?x2, ?x1) ] ];
277         forall_{ ?a : agent } [ right(?a) => exists_{ ?x1 : xpos, ?x2 : xpos, ?y : ypos } [ agent-at(?a, ?x1, ?y) &
                ADJACENT-RIGHT(?x2, ?x1) ] ];
278         forall_{ ?a : agent } [ up(?a) => exists_{ ?x : xpos, ?y1 : ypos, ?y2 : ypos } [ agent-at(?a, ?x, ?y1) &
                ADJACENT-UP(?y2, ?y1) ] ];
279         forall_{ ?a : agent } [ down(?a) => exists_{ ?x : xpos, ?y1 : ypos, ?y2 : ypos } [ agent-at(?a, ?x, ?y1) &
                ADJACENT-DOWN(?y2, ?y1) ] ];
280
281         // only use tools on this agent
282         forall_{ ?a : agent, ?t : tool, ?o : object-of-interest }
283             [ use-tool-on(?a, ?t, ?o) => HAS-TOOL(?a, ?t) ];
284
285         // only use tool on objects at the same location
286         forall_{ ?a : agent, ?t : tool, ?o : object-of-interest }
287             [ use-tool-on(?a, ?t, ?o) => ( exists_{ ?x : xpos, ?y : ypos } [ agent-at(?a, ?x, ?y) & OBJECT-AT
                    (?o, ?x, ?y) ] ) ];
288
289         // only take pictures of objects that have not been photographed before
290         forall_{ ?a : agent, ?t : tool, ?o : object-of-interest }
291             [ use-tool-on(?a, ?t, ?o) => ( ~CAMERA-TOOL(?t) | ~pictureTaken(?o) ) ];
292
293         // only take a picture if life was detected
294         forall_{ ?a : agent, ?t : tool, ?o : object-of-interest }
295             [ use-tool-on(?a, ?t, ?o) => ( ~CAMERA-TOOL(?t) | lifeDetected(?o) ) ];
296
297         // repair is only possible at the base
298         forall_{ ?a : agent, ?t : tool } [ repair(?a, ?t) => ( exists_{ ?x : xpos, ?y : ypos } [ agent-at(?a, ?x, ?y) &
                BASE(?x, ?y) ] ) ];
299
300         // repair only damaged tool
301         forall_{ ?a : agent, ?t : tool } [ repair(?a, ?t) => damaged(?t) ];
302
303         // repair only tools on this agent
304         forall_{ ?a : agent, ?t : tool } [ repair(?a, ?t) => HAS-TOOL(?a, ?t) ];
305
306         // objects can only be investigated by one agent at a time
```

```
307         forall_{ ?o : object-of-interest } [ ( sum_{ ?a : agent, ?t : tool } [ use-tool-on(?a, ?t, ?o) ] ) <= 1 ];
308
309         // agents can support other agents if they are in the same location
310         forall_{ ?a1 : agent, ?a2 : agent } [ support-agent(?a1, ?a2) => ( exists_{ ?x : xpos, ?y : ypos } [ agent-
                at(?a1, ?x, ?y) & agent-at(?a2, ?x, ?y) ] ) ];
311
312         // agents can support other agents only if the other agent is using a tool
313         // NOTE: This one is not possible with ipc 2018 rules. As supporting doesn't have an
314         // effect if the other agent doesn't use a tool, it doesn't matter if we keep it commented, though.
315         // forall_{ ?a1 : agent, ?a2 : agent } [ support-agent(?a1, ?a2) => ( exists_{ ?t : tool, ?o : object-of-
                interest } [ use-tool-on(?a2, ?t, ?o) ] ) ];
316
317         // each agent may perform one action per step
318         forall_{ ?a : agent } [ ( left(?a) + right(?a) + up(?a) + down(?a) +
319                                   ( sum_{ ?t : tool, ?o : object-of-interest } [ use-tool-on(?a, ?t, ?o) ] ) +
320                                   ( sum_{ ?t : tool } [ repair(?a, ?t) ] ) +
321                                   ( sum_{ ?a2 : agent } [ support-agent(?a, ?a2) ] ) ) <= 1 ];
322     };
323 }
```

Instance

```
1  ////////////////////////////////////////////////////////////////////////////
2  // //
3  // //
4  // RDDL MDP version of Cooperative Recon instance #01 for IPC 2018 by Thomas //
5  // Keller (tho.keller [at] unibas.ch), based on the IPC 2011 domain by Tom //
6  // Walsh (thomasjwalsh [at] gmail.com). //
7  // //
8  // //
9  ////////////////////////////////////////////////////////////////////////////
10
11 instance cooperative-recon_inst_mdp__01 {
12     domain = cooperative-recon_mdp;
13
14     objects {
15         xpos : { x00, x01, x02 };
16         ypos : { y00, y01, y02 };
17         object-of-interest : { obj00, obj01 };
18         agent : { a00, a01 };
19         tool : { w00, l00, c00, w01, l01, c01 };
20     };
21
22     non-fluents {
23         // ADJACENCY
24         ADJACENT-LEFT(x00, x01);
25         ADJACENT-RIGHT(x01, x00);
26         ADJACENT-LEFT(x01, x02);
27         ADJACENT-RIGHT(x02, x01);
28         ADJACENT-DOWN(y00, y01);
```

```
29          ADJACENT-UP(y01, y00);
30          ADJACENT-DOWN(y01, y02);
31          ADJACENT-UP(y02, y01);
32
33          // BASE
34          BASE(x00, y00);
35
36          // TOOLS
37          WATER-TOOL(w00);
38          HAS-TOOL(a00, w00);
39          LIFE-TOOL(l00);
40          HAS-TOOL(a00, l00);
41          CAMERA-TOOL(c00);
42          HAS-TOOL(a00, c00);
43          WATER-TOOL(w01);
44          HAS-TOOL(a01, w01);
45          LIFE-TOOL(l01);
46          HAS-TOOL(a01, l01);
47          CAMERA-TOOL(c01);
48          HAS-TOOL(a01, c01);
49
50          // DAMAGE-PROBS
51          DAMAGE-PROB(x01, y00) = 0.52;
52          DAMAGE-PROB(x01, y01) = 0.27;
53          DAMAGE-PROB(x02, y00) = 0.27;
54
55          // DETECT-PROBS
56          DETECT-PROB = 0.6;
57          DETECT-PROB-DAMAGED = 0.3;
58          DETECT-PROB-WITH-SUPPORT = 0.8;
59          DETECT-PROB-DAMAGED-WITH-SUPPORT = 0.4;
60
61          // OBJECT-AT
62          OBJECT-AT(obj00, x02, y01);
63          OBJECT-AT(obj01, x02, y01);
64
65          // REWARDS
66          GOOD-PIC-REWARD(obj00) = 11.63;
67          BAD-PIC-REWARD(obj00) = 8.21;
68          GOOD-PIC-REWARD(obj01) = 1.50;
69          BAD-PIC-REWARD(obj01) = 0.30;
70
71      };
72
73      init-state {
74          agent-at(a00, x01, y02);
75          agent-at(a01, x01, y02);
76
77      };
78
79      horizon = 30;
80
```

```
81      discount = 1.0;
82  }
```

A.4.3 Crossing Traffic (see [ippb])

Domain

```
1   ////////////////////////////////////////////////////////////////////
2   //
3   // Crossing Traffic Robot Navigation
4   //
5   // Author: Sungwook Yoon (sungwook.yoon [at] gmail.com)
6   //
7   // Modified for competition and translation purposes by Scott Sanner.
8   //
9   // In a grid, a robot (R) must get to a goal (G) and avoid obstacles (O)
10  // arriving randomly and moving left. If an obstacle overlaps with the
11  // robot, the robot disappears and can no longer move around. The robot
12  // can "duck" underneath a car by deliberately moving right/east when
13  // a car is to the right of it (this can make the solution interesting...
14  // the robot should start at the left side of the screen then). The robot
15  // receives −1 for every time step it has not reached the goal. The goal
16  // state is absorbing with 0 reward.
17  //
18  // ****************
19  // * R *
20  // * <−O <−O <−O *
21  // * <−O <−O *
22  // * <−O <−O *
23  // * <−O <−O *
24  // * G *
25  // ****************
26  //
27  // You can think of this as the RDDL version of Frogger:
28  //
29  // http://en.wikipedia.org/wiki/Frogger
30  //
31  ////////////////////////////////////////////////////////////////////
32
33  domain crossing−traffic_mdp {
34    requirements = {
35  //  constrained−state,
36        reward−deterministic
37    };
38
39    types {
40      xpos : object;
41      ypos : object;
42    };
43
```

```
44  pvariables {
45
46    NORTH(ypos, ypos) : {non−fluent, bool, default = false};
47    SOUTH(ypos, ypos) : {non−fluent, bool, default = false};
48    EAST(xpos, xpos) : {non−fluent, bool, default = false};
49    WEST(xpos, xpos) : {non−fluent, bool, default = false};
50
51    MIN−XPOS(xpos) : {non−fluent, bool, default = false};
52    MAX−XPOS(xpos) : {non−fluent, bool, default = false};
53    MIN−YPOS(ypos) : {non−fluent, bool, default = false};
54    MAX−YPOS(ypos) : {non−fluent, bool, default = false};
55
56    INPUT−RATE : {non−fluent, real, default = 0.2};
57
58    GOAL(xpos,ypos) : {non−fluent, bool, default = false};
59
60    // Fluents
61    robot−at(xpos, ypos) : {state−fluent, bool, default = false};
62    obstacle−at(xpos, ypos) : {state−fluent, bool, default = false};
63
64    // Actions
65    move−north : {action−fluent, bool, default = false};
66    move−south : {action−fluent, bool, default = false};
67    move−east : {action−fluent, bool, default = false};
68    move−west : {action−fluent, bool, default = false};
69  };
70
71  cpfs {
72
73    robot−at'(?x,?y) =
74
75      // Goal is absorbing so robot stays put
76      if ( GOAL(?x,?y) ^ robot−at(?x,?y) )
77      then
78        KronDelta(true)
79      else if ( exists_{?x2 : xpos, ?y2 : ypos} [ GOAL(?x2,?y2) ^ robot−at(?x2,?y2) ] )
80      then
81        KronDelta(false) // because of fall−through we know (?x,y) != (?x2,?y2)
82
83      // Check for legal robot movement (robot disappears if at an obstacle)
84      else if ( move−north ^ exists_{?y2 : ypos} [ NORTH(?y2,?y) ^ robot−at(?x,?y2) ^ ~obstacle−at(?x,?y2) ] )
85      then
86        KronDelta(true) // robot moves to this location
87      else if ( move−north ^ exists_{?y2 : ypos} [ NORTH(?y,?y2) ^ robot−at(?x,?y) ] )
88      then
89        KronDelta(false) // robot leaves this location
90      else if ( move−south ^ exists_{?y2 : ypos} [ SOUTH(?y2,?y) ^ robot−at(?x,?y2) ^ ~obstacle−at(?x,?y2) ] )
91      then
92        KronDelta(true) // robot moves to this location
93      else if ( move−south ^ exists_{?y2 : ypos} [ SOUTH(?y,?y2) ^ robot−at(?x,?y) ] )
94      then
95        KronDelta(false) // robot leaves this location
```

```
96          else if ( move−east ^ exists_{?x2 : xpos} [ EAST(?x2,?x) ^ robot−at(?x2,?y) ^ ~obstacle−at(?x2,?y) ] )
97          then
98            KronDelta(true) // robot moves to this location
99          else if ( move−east ^ exists_{?x2 : xpos} [ EAST(?x,?x2) ^ robot−at(?x,?y) ] )
100         then
101           KronDelta(false) // robot leaves this location
102         else if ( move−west ^ exists_{?x2 : xpos} [ WEST(?x2,?x) ^ robot−at(?x2,?y) ^ ~obstacle−at(?x2,?y) ] )
103         then
104           KronDelta(true) // robot moves to this location
105         else if ( move−west ^ exists_{?x2 : xpos} [ WEST(?x,?x2) ^ robot−at(?x,?y) ] )
106         then
107           KronDelta(false) // robot leaves this location
108
109         // A noop or illegal movement, so state unchanged
110         else
111           KronDelta( robot−at(?x,?y) ^ ~obstacle−at(?x,?y) );
112
113       obstacle−at'(?x, ?y) =
114
115         // No obstacles in top or bottom row (these rows are safe havens)
116         if ( MIN−YPOS(?y) | MAX−YPOS(?y) )
117           then KronDelta( false )
118
119         // Check for RHS border input cell
120         else if ( MAX−XPOS(?x) )
121           then Bernoulli( INPUT−RATE )
122
123         // Not a top or bottom row and not a border input cell −− inherits obstacle to east
124         else
125           KronDelta( exists_{?x2 : xpos} [EAST(?x,?x2) ^ obstacle−at(?x2,?y)] );
126
127     };
128
129     // 0 reward for reaching goal, −1 in all other cases
130     reward = [sum_{?x : xpos, ?y : ypos} −(GOAL(?x,?y) ^ ~robot−at(?x,?y))];
131
132   // state−action−constraints {
133   //
134   //  // Robot at exactly one position
135   //  [sum_{?x : xpos, ?y : ypos} robot−at(?x,?y)] <= 1;
136   //
137   //  // EAST, WEST, NORTH, SOUTH defined properly (unique and symmetric)
138   //  forall_{?x1 : xpos} [(sum_{?x2 : xpos} WEST(?x1,?x2)) <= 1];
139   //  forall_{?x1 : xpos} [(sum_{?x2 : xpos} EAST(?x1,?x2)) <= 1];
140   //  forall_{?y1 : ypos} [(sum_{?y2 : ypos} NORTH(?y1,?y2)) <= 1];
141   //  forall_{?y1 : ypos} [(sum_{?y2 : ypos} SOUTH(?y1,?y2)) <= 1];
142   //  forall_{?x1 : xpos, ?x2 : xpos} [ EAST(?x1,?x2) <=> WEST(?x2,?x1) ];
143   //  forall_{?y1 : ypos, ?y2 : ypos} [ SOUTH(?y1,?y2) <=> NORTH(?y2,?y1) ];
144   //
145   //  // Definition verification
146   //  [ sum_{?x : xpos} MIN−XPOS(?x) ] == 1;
147   //  [ sum_{?x : xpos} MAX−XPOS(?x) ] == 1;
```

```
148 // [ sum_{?y : ypos} MIN-YPOS(?y) ] == 1;
149 // [ sum_{?y : ypos} MAX-YPOS(?y) ] == 1;
150 // [ sum_{?x : xpos, ?y : ypos} GOAL(?x,?y) ] == 1;
151 //
152 // };
153
154 }
```

Instance

```
1  non-fluents nf-crossing-traffic_inst_mdp__1 {
2    domain = crossing-traffic_mdp;
3    objects {
4      xpos : {x1,x2,x3};
5      ypos : {y1,y2,y3};
6    };
7    non-fluents {
8      NORTH(y1,y2);
9      SOUTH(y2,y1);
10     NORTH(y2,y3);
11     SOUTH(y3,y2);
12
13     EAST(x1,x2);
14     WEST(x2,x1);
15     EAST(x2,x3);
16     WEST(x3,x2);
17
18     MIN-XPOS(x1);
19     MAX-XPOS(x3);
20     MIN-YPOS(y1);
21     MAX-YPOS(y3);
22
23     GOAL(x3,y3);
24
25     INPUT-RATE = 0.3;
26   };
27 }
28
29 instance crossing-traffic_inst_mdp__1 {
30   domain = crossing-traffic_mdp;
31   non-fluents = nf-crossing-traffic_inst_mdp__1;
32   init-state {
33     robot-at(x3,y1);
34     obstacle-at(x1,y2);
35     obstacle-at(x3,y2);
36   };
37   max-nondef-actions = 1;
38   horizon = 40;
39   discount = 1.0;
40 }
```

A.4.4 Prop DBN (see dbn_prop.rddl in [rdd])

Domain

```
////////////////////////////////////////////////////////////////////
// A simple propositional 2−slice DBN (variables are not parameterized).
//
// Author: Scott Sanner (ssanner [at] gmail.com)
////////////////////////////////////////////////////////////////////
domain prop−dbn_mdp {

    requirements = { reward−deterministic };

    pvariables {
        p : { state−fluent, bool, default = false };
        q : { state−fluent, bool, default = false };
        r : { state−fluent, bool, default = false };
        a : { action−fluent, bool, default = false };
    };

    cpfs {
        // Some standard Bernoulli conditional probability tables
        p' = if (p ^ r) then Bernoulli(.9) else Bernoulli(.3);

        q' = if (q ^ r) then Bernoulli(.9)
                    else if (a) then Bernoulli(.3) else Bernoulli(.8);

        // KronDelta is like a DiracDelta, but for discrete data (boolean or int)
        r' = if (~q) then KronDelta(r) else KronDelta(r <=> q);
    };

    // A boolean functions as a 0/1 integer when a numerical value is needed
    reward = p + q − r; // a boolean functions as a 0/1 integer when a numerical value is needed
}
```

Instance 1

```
instance prop−dbn_inst_mdp {

    domain = prop−dbn_mdp;
    init−state {
        p = true; // could also just say 'p' by itself
        q = false; // default so unnecessary, could also say '~q' by itself
        r; // same as r = true
    };

    max−nondef−actions = 1;
    horizon = 20;
    discount = 0.9;
}
```

Instance 2 - Modified Version of Instance 1

```
instance prop−dbn_inst_mdp_2 {

  domain = prop−dbn_mdp;
  init−state {
    p = true;
    q = true;
    r = false;
  };

  max−nondef−actions = 1;
  horizon = 20;
  discount = 0.9;
}
```

A.4.5 Navigation (see [ippa])

Domain

```
////////////////////////////////////////////////////////////////////
//
// Navigation MDP
//
// Author: Scott Sanner (ssanner [at] gmail.com)
//
// In a grid, a robot (R) must get to a goal (G). Every cell offers
// the robot a (different) chance of disappearing. The robot needs
// to choose a path which gets it to the goal most reliably within
// the finite horizon time.
//
// ************************
// * 0 0 0 0 R *
// * .1 .3 .5 .7 .9 *
// * .1 .3 .5 .7 .9 *
// * .1 .3 .5 .7 .9 *
// * .1 .3 .5 .7 .9 *
// * 0 0 0 0 G *
// ************************
//
// This is a good domain to test deteminized planners because
// one can see here that the path using the .3 chance of failure
// leads to a 1−step most likely outcome of survival, but
// a poor 4−step change of survival (.7^(.4)) whereas the path
// using a .1 chance of failure is much more safe.
//
// The domain generators for navigation have a flag to produce slightly
// obfuscated files to discourage hand−coded policies, but
// rddl.viz.NavigationDisplay can properly display these grids, e.g.,
//
```

```
31 // ./run rddl.sim.Simulator files/final-comp/rddl rddl.policy.RandomBoolPolicy
32 // navigation-inst-mdp--1 rddl.viz.NavigationDisplay
33 //
34 // (Movement was not made stochastic due to the lack of intermediate
35 // variables and synchronic arcs to support both the PPDDL and SPUDD
36 // translations.)
37 //
38 ////////////////////////////////////////////////////////////////////
39
40 domain navigation_mdp {
41   requirements = {
42 //  constrained-state,
43     reward-deterministic
44   };
45
46   types {
47     xpos : object;
48     ypos : object;
49   };
50
51   pvariables {
52
53     NORTH(ypos, ypos) : {non-fluent, bool, default = false};
54     SOUTH(ypos, ypos) : {non-fluent, bool, default = false};
55     EAST(xpos, xpos) : {non-fluent, bool, default = false};
56     WEST(xpos, xpos) : {non-fluent, bool, default = false};
57
58     MIN-XPOS(xpos) : {non-fluent, bool, default = false};
59     MAX-XPOS(xpos) : {non-fluent, bool, default = false};
60     MIN-YPOS(ypos) : {non-fluent, bool, default = false};
61     MAX-YPOS(ypos) : {non-fluent, bool, default = false};
62
63     P(xpos, ypos) : {non-fluent, real, default = 0.0};
64
65     GOAL(xpos,ypos) : {non-fluent, bool, default = false};
66
67     // Fluents
68     robot-at(xpos, ypos) : {state-fluent, bool, default = false};
69
70     // Actions
71     move-north : {action-fluent, bool, default = false};
72     move-south : {action-fluent, bool, default = false};
73     move-east : {action-fluent, bool, default = false};
74     move-west : {action-fluent, bool, default = false};
75   };
76
77   cpfs {
78
79     robot-at'(?x,?y) =
80
81       if ( GOAL(?x,?y) ^ robot-at(?x,?y) )
82       then
```

```
83            KronDelta(true)
84          else if (( exists_{?x2 : xpos, ?y2 : ypos} [ GOAL(?x2,?y2) ^ robot−at(?x2,?y2) ] )
85                | ( move−north ^ exists_{?y2 : ypos} [ NORTH(?y,?y2) ^ robot−at(?x,?y) ] )
86                | ( move−south ^ exists_{?y2 : ypos} [ SOUTH(?y,?y2) ^ robot−at(?x,?y) ] )
87                | ( move−east ^ exists_{?x2 : xpos} [ EAST(?x,?x2) ^ robot−at(?x,?y) ] )
88                | ( move−west ^ exists_{?x2 : xpos} [ WEST(?x,?x2) ^ robot−at(?x,?y) ] ))
89          then
90            KronDelta(false)
91          else if (( move−north ^ exists_{?y2 : ypos} [ NORTH(?y2,?y) ^ robot−at(?x,?y2) ] )
92                | ( move−south ^ exists_{?y2 : ypos} [ SOUTH(?y2,?y) ^ robot−at(?x,?y2) ] )
93                | ( move−east ^ exists_{?x2 : xpos} [ EAST(?x2,?x) ^ robot−at(?x2,?y) ] )
94                | ( move−west ^ exists_{?x2 : xpos} [ WEST(?x2,?x) ^ robot−at(?x2,?y) ] ))
95          then
96            Bernoulli( 1.0 − P(?x, ?y) )
97          else
98            KronDelta( robot−at(?x,?y) );
99
100     };
101
102     // 0 reward for reaching goal, −1 in all other cases
103     reward = [sum_{?x : xpos, ?y : ypos} −(GOAL(?x,?y) ^ ~robot−at(?x,?y))];
104
105   // state−action−constraints {
106   //
107   //  // Robot at exactly one position
108   //  [sum_{?x : xpos, ?y : ypos} robot−at(?x,?y)] <= 1;
109   //
110   //  // EAST, WEST, NORTH, SOUTH defined properly (unique and symmetric)
111   //  forall_{?x1 : xpos} [(sum_{?x2 : xpos} WEST(?x1,?x2)) <= 1];
112   //  forall_{?x1 : xpos} [(sum_{?x2 : xpos} EAST(?x1,?x2)) <= 1];
113   //  forall_{?y1 : ypos} [(sum_{?y2 : ypos} NORTH(?y1,?y2)) <= 1];
114   //  forall_{?y1 : ypos} [(sum_{?y2 : ypos} SOUTH(?y1,?y2)) <= 1];
115   //  forall_{?x1 : xpos, ?x2 : xpos} [ EAST(?x1,?x2) <=> WEST(?x2,?x1) ];
116   //  forall_{?y1 : ypos, ?y2 : ypos} [ SOUTH(?y1,?y2) <=> NORTH(?y2,?y1) ];
117   //
118   //  // Definition verification
119   //  [ sum_{?x : xpos} MIN−XPOS(?x) ] == 1;
120   //  [ sum_{?x : xpos} MAX−XPOS(?x) ] == 1;
121   //  [ sum_{?y : ypos} MIN−YPOS(?y) ] == 1;
122   //  [ sum_{?y : ypos} MAX−YPOS(?y) ] == 1;
123   //  [ sum_{?x : xpos, ?y : ypos} GOAL(?x,?y) ] == 1;
124   //
125   // };
126
127   }
```

Instance 1

```
1   non−fluents nf−navigation_inst_mdp__1 {
2     domain = navigation_mdp;
```

```
  objects {
    xpos : {x6,x14,x21,x9};
    ypos : {y12,y20,y15};
  };
  non−fluents {
    SOUTH(y15,y12);
    GOAL(x21,y20);
    WEST(x14,x9);
    NORTH(y12,y15);
    MAX−YPOS(y20);
    P(x9,y15) = 0.34543713989357155;
    SOUTH(y20,y15);
    MIN−YPOS(y12);
    EAST(x14,x21);
    EAST(x9,x14);
    MAX−XPOS(x21);
    WEST(x9,x6);
    P(x6,y15) = 0.04896671138703823;
    P(x21,y15) = 0.928158446525534;
    EAST(x6,x9);
    P(x14,y15) = 0.6369951789577802;
    WEST(x21,x14);
    NORTH(y15,y20);
    MIN−XPOS(x6);
  };
}

instance navigation_inst_mdp__1 {
  domain = navigation_mdp;
  non−fluents = nf−navigation_inst_mdp__1;
  init−state {
    robot−at(x21,y12);
  };
  max−nondef−actions = 1;
  horizon = 40;
  discount = 1.0;
}
```

Instance 2

```
non−fluents nf−navigation_inst_mdp__2 {
  domain = navigation_mdp;
  objects {
    xpos : {x14,x9,x21,x6,x30};
    ypos : {y15,y12,y20};
  };
  non−fluents {
    SOUTH(y15,y12);
    MAX−XPOS(x30);
    EAST(x21,x30);
```

```
      P(x6,y15) = 0.0360226184129715;
      MAX-YPOS(y20);
      MIN-YPOS(y12);
      EAST(x9,x14);
      GOAL(x30,y20);
      WEST(x30,x21);
      P(x21,y15) = 0.6909389975480735;
      WEST(x9,x6);
      P(x30,y15) = 0.916325646918267;
      EAST(x6,x9);
      NORTH(y12,y15);
      WEST(x14,x9);
      WEST(x21,x14);
      P(x9,y15) = 0.23629253543913364;
      SOUTH(y20,y15);
      MIN-XPOS(x6);
      P(x14,y15) = 0.48970670998096466;
      NORTH(y15,y20);
      EAST(x14,x21);
    };
}

instance navigation_inst_mdp__2 {
  domain = navigation_mdp;
  non-fluents = nf-navigation_inst_mdp__2;
  init-state {
    robot-at(x30,y12);
  };
  max-nondef-actions = 1;
  horizon = 40;
  discount = 1.0;
}
```

Instance 3

```
non-fluents nf-navigation_inst_mdp__3 {
  domain = navigation_mdp;
  objects {
    xpos : {x14,x30,x9,x21,x6};
    ypos : {y20,y12,y27,y15};
  };
  non-fluents {
    P(x9,y20) = 0.2450880827382207;
    SOUTH(y20,y15);
    WEST(x21,x14);
    P(x6,y20) = 0.05156800337135792;
    GOAL(x30,y27);
    P(x30,y15) = 0.9280268289148808;
    SOUTH(y27,y20);
    WEST(x14,x9);
```

```
    WEST(x9,x6);
    P(x21,y15) = 0.7021599113941193;
    MIN−XPOS(x6);
    EAST(x21,x30);
    SOUTH(y15,y12);
    P(x14,y15) = 0.5013892482966185;
    MIN−YPOS(y12);
    P(x30,y20) = 0.9452640172094107;
    P(x21,y20) = 0.6855187271139584;
    NORTH(y20,y27);
    P(x9,y15) = 0.250524521805346;
    EAST(x14,x21);
    NORTH(y15,y20);
    MAX−YPOS(y27);
    P(x14,y20) = 0.48334174789488316;
    EAST(x6,x9);
    NORTH(y12,y15);
    P(x6,y15) = 0.03749256581068039;
    EAST(x9,x14);
    MAX−XPOS(x30);
    WEST(x30,x21);
  };
}

instance navigation_inst_mdp__3 {
  domain = navigation_mdp;
  non−fluents = nf−navigation_inst_mdp__3;
  init−state {
    robot−at(x30,y12);
  };
  max−nondef−actions = 1;
  horizon = 40;
  discount = 1.0;
}
```

Instance 4

```
non−fluents nf_navigation_inst_mdp__4 {
  domain = navigation_mdp;
  objects {
    xpos : {x21,x30,x9,x6,x14};
    ypos : {y47,y12,y36,y15,y27,y20};
  };
  non−fluents {
    SOUTH(y20,y15);
    SOUTH(y36,y27);
    P(x14,y27) = 0.4755020113661885;
    P(x6,y20) = 0.024376023560762405;
    P(x30,y36) = 0.9497081767767668;
    MIN−YPOS(y12);
```

```
14      P(x21,y15) = 0.7021266371011734;
15      SOUTH(y27,y20);
16      NORTH(y15,y20);
17      MAX-YPOS(y47);
18      SOUTH(y15,y12);
19      NORTH(y20,y27);
20      P(x30,y27) = 0.9107285801437683;
21      MAX-XPOS(x30);
22      P(x14,y36) = 0.4749935809522867;
23      SOUTH(y47,y36);
24      WEST(x9,x6);
25      P(x30,y20) = 0.9197213770821691;
26      P(x9,y15) = 0.23887041257694364;
27      MIN-XPOS(x6);
28      NORTH(y36,y47);
29      NORTH(y12,y15);
30      EAST(x9,x14);
31      P(x14,y15) = 0.5034075286239386;
32      WEST(x21,x14);
33      P(x30,y15) = 0.9199540922418237;
34      WEST(x30,x21);
35      P(x6,y36) = 0.05538930557668209;
36      EAST(x14,x21);
37      P(x6,y27) = 0.044584812596440315;
38      EAST(x6,x9);
39      P(x21,y27) = 0.7054883688688278;
40      P(x9,y20) = 0.27436082251369953;
41      P(x21,y36) = 0.7212282549589872;
42      GOAL(x30,y47);
43      P(x21,y20) = 0.7279216069728136;
44      P(x14,y20) = 0.49078163877129555;
45      P(x6,y15) = 0.013172781793400645;
46      NORTH(y27,y36);
47      P(x9,y27) = 0.24275896279141307;
48      WEST(x14,x9);
49      P(x9,y36) = 0.26499055325984955;
50      EAST(x21,x30);
51    };
52  }
53
54  instance navigation_inst_mdp__4 {
55    domain = navigation_mdp;
56    non-fluents = nf_navigation_inst_mdp__4;
57    init-state {
58      robot-at(x30,y12);
59    };
60    max-nondef-actions = 1;
61    horizon = 40;
62    discount = 1.0;
63  }
```

Instance 5

```
non−fluents nf_navigation_inst_mdp__5 {
  domain = navigation_mdp;
  objects {
    xpos : {x14,x54,x6,x86,x41,x9,x21,x30,x69,x105};
    ypos : {y12,y15,y20};
  };
  non−fluents {
    MIN−YPOS(y12);
    SOUTH(y15,y12);
    P(x6,y15) = 0.024014816619455814;
    P(x30,y15) = 0.42676472498310936;
    EAST(x9,x14);
    WEST(x41,x30);
    WEST(x21,x14);
    EAST(x69,x86);
    P(x105,y15) = 0.9336750203122696;
    WEST(x14,x9);
    EAST(x21,x30);
    WEST(x69,x54);
    GOAL(x105,y20);
    MAX−XPOS(x105);
    P(x54,y15) = 0.6266834967666202;
    EAST(x54,x69);
    P(x86,y15) = 0.8539166100737121;
    EAST(x30,x41);
    P(x69,y15) = 0.7261174401889244;
    WEST(x30,x21);
    P(x9,y15) = 0.1536506913188431;
    MAX−YPOS(y20);
    WEST(x105,x86);
    NORTH(y15,y20);
    MIN−XPOS(x6);
    NORTH(y12,y15);
    WEST(x9,x6);
    WEST(x86,x69);
    EAST(x86,x105);
    WEST(x54,x41);
    P(x41,y15) = 0.525113389827311;
    P(x21,y15) = 0.34208946157660747;
    P(x14,y15) = 0.21357332878849572;
    EAST(x14,x21);
    EAST(x6,x9);
    EAST(x41,x54);
    SOUTH(y20,y15);
  };
}

instance navigation_inst_mdp__5 {
  domain = navigation_mdp;
```

```
50    non−fluents = nf_navigation_inst_mdp__5;
51    init−state {
52       robot−at(x105,y12);
53    };
54    max−nondef−actions = 1;
55    horizon = 40;
56    discount = 1.0;
57 }
```

Instance 6

```
1  non−fluents nf_navigation_inst_mdp__6 {
2     domain = navigation_mdp;
3     objects {
4        xpos : {x54,x9,x86,x30,x105,x14,x69,x6,x41,x21};
5        ypos : {y12,y15,y27,y20};
6     };
7     non−fluents {
8        P(x6,y20) = 0.032087743282318115;
9        P(x69,y20) = 0.7137660816467056;
10       P(x41,y15) = 0.5501891952008009;
11       P(x41,y20) = 0.5153869516216218;
12       EAST(x30,x41);
13       EAST(x21,x30);
14       MAX−XPOS(x105);
15       WEST(x86,x69);
16       WEST(x30,x21);
17       WEST(x14,x9);
18       P(x86,y20) = 0.8570238709863689;
19       NORTH(y20,y27);
20       P(x69,y15) = 0.7465580261001984;
21       SOUTH(y27,y20);
22       NORTH(y15,y20);
23       P(x6,y15) = 0.03272361308336258;
24       P(x30,y20) = 0.43298558166457546;
25       P(x30,y15) = 0.4296299742741717;
26       P(x105,y20) = 0.9116935366376614;
27       WEST(x54,x41);
28       WEST(x69,x54);
29       SOUTH(y20,y15);
30       WEST(x21,x14);
31       EAST(x14,x21);
32       WEST(x9,x6);
33       MAX−YPOS(y27);
34       EAST(x86,x105);
35       P(x86,y15) = 0.8315923180845048;
36       SOUTH(y15,y12);
37       MIN−XPOS(x6);
38       EAST(x6,x9);
39       EAST(x41,x54);
```

```
    P(x105,y15) = 0.9274347008516391;
    EAST(x9,x14);
    GOAL(x105,y27);
    NORTH(y12,y15);
    P(x54,y20) = 0.6428951051914029;
    P(x14,y15) = 0.21247654371998376;
    P(x21,y15) = 0.33662864607241416;
    P(x14,y20) = 0.21680060737869805;
    MIN−YPOS(y12);
    EAST(x69,x86);
    P(x9,y20) = 0.15484551112684938;
    WEST(x105,x86);
    P(x54,y15) = 0.659609689273768;
    P(x9,y15) = 0.12321555666211578;
    P(x21,y20) = 0.3461974025186565;
    EAST(x54,x69);
    WEST(x41,x30);
  };
}

instance navigation_inst_mdp__6 {
  domain = navigation_mdp;
  non−fluents = nf_navigation_inst_mdp__6;
  init−state {
    robot−at(x105,y12);
  };
  max−nondef−actions = 1;
  horizon = 40;
  discount = 1.0;
}
```

Instance 7

```
non−fluents nf_navigation_inst_mdp__7 {
  domain = navigation_mdp;
  objects {
    xpos : {x9,x21,x54,x69,x30,x6,x14,x41,x86,x105};
    ypos : {y36,y27,y15,y12,y20};
  };
  non−fluents {
    P(x41,y15) = 0.5383095685392618;
    P(x54,y15) = 0.6193011131965451;
    EAST(x30,x41);
    SOUTH(y36,y27);
    GOAL(x105,y36);
    SOUTH(y20,y15);
    P(x30,y20) = 0.44101769311560524;
    P(x6,y15) = 0.023994946852326393;
    P(x14,y20) = 0.21105071109357393;
    WEST(x30,x21);
```

```
    EAST(x21,x30);
    NORTH(y27,y36);
    P(x54,y20) = 0.6595114645444684;
    WEST(x69,x54);
    SOUTH(y15,y12);
    MIN-XPOS(x6);
    MAX-XPOS(x105);
    P(x21,y20) = 0.33060312312510276;
    WEST(x21,x14);
    EAST(x9,x14);
    P(x86,y20) = 0.8134994268831279;
    NORTH(y15,y20);
    P(x41,y20) = 0.5119783256668597;
    WEST(x105,x86);
    P(x30,y27) = 0.4580075146837367;
    WEST(x41,x30);
    P(x6,y20) = 0.021341380663216114;
    P(x54,y27) = 0.642808957853251;
    P(x69,y20) = 0.7245696174601713;
    EAST(x86,x105);
    P(x14,y15) = 0.22739516860908932;
    P(x9,y15) = 0.15196049358281824;
    P(x86,y15) = 0.859581144940522;
    WEST(x9,x6);
    EAST(x41,x54);
    P(x69,y15) = 0.7486556774626175;
    P(x105,y20) = 0.9399694142242273;
    WEST(x54,x41);
    MAX-YPOS(y36);
    P(x21,y15) = 0.31251017162058914;
    P(x30,y15) = 0.43196028677953613;
    P(x14,y27) = 0.24023324913448757;
    P(x6,y27) = 0.011126482859253883;
    EAST(x54,x69);
    EAST(x14,x21);
    P(x105,y27) = 0.9386047304918369;
    P(x69,y27) = 0.7594633890936772;
    P(x9,y20) = 0.1492169178608391;
    SOUTH(y27,y20);
    EAST(x6,x9);
    P(x86,y27) = 0.833064128127363;
    EAST(x69,x86);
    P(x41,y27) = 0.5182037660852075;
    P(x21,y27) = 0.31010614638135947;
    WEST(x14,x9);
    WEST(x86,x69);
    NORTH(y20,y27);
    NORTH(y12,y15);
    P(x9,y27) = 0.14091320667001936;
    P(x105,y15) = 0.926005428036054;
    MIN-YPOS(y12);
};
```

```
}

instance navigation_inst_mdp__7 {
  domain = navigation_mdp;
  non−fluents = nf_navigation_inst_mdp__7;
  init−state {
    robot−at(x105,y12);
  };
  max−nondef−actions = 1;
  horizon = 40;
  discount = 1.0;
}
```

Instance 8

```
non−fluents nf_navigation_inst_mdp__8 {
  domain = navigation_mdp;
  objects {
    xpos : {x261,x149,x21,x69,x9,x174,x41,x86,x105,x329,x14,x366,x30,x405,x201,x294,x126,x6,x230,x54};
    ypos : {y12,y20,y15};
  };
  non−fluents {
    WEST(x14,x9);
    P(x30,y15) = 0.2347686960312881;
    EAST(x30,x41);
    WEST(x149,x126);
    WEST(x41,x30);
    EAST(x69,x86);
    GOAL(x405,y20);
    EAST(x14,x21);
    SOUTH(y15,y12);
    MAX−YPOS(y20);
    P(x126,y15) = 0.522427676441638;
    MIN−XPOS(x6);
    EAST(x9,x14);
    EAST(x21,x30);
    WEST(x366,x329);
    EAST(x329,x366);
    P(x69,y15) = 0.3703241362971695;
    EAST(x261,x294);
    P(x86,y15) = 0.4001688238135294;
    WEST(x230,x201);
    P(x329,y15) = 0.8186304341928151;
    WEST(x30,x21);
    NORTH(y15,y20);
    P(x201,y15) = 0.6305554033208051;
    P(x149,y15) = 0.5512959579692075;
    P(x105,y15) = 0.47063784213050414;
    P(x14,y15) = 0.12598508870915365;
    WEST(x54,x41);
```

```
36        EAST(x174,x201);
37        P(x21,y15) = 0.18618261039649187;
38        EAST(x201,x230);
39        MAX-XPOS(x405);
40        WEST(x174,x149);
41        EAST(x294,x329);
42        P(x6,y15) = 0.020123825408518314;
43        P(x41,y15) = 0.2575469744440756;
44        P(x261,y15) = 0.7651083509584791;
45        EAST(x6,x9);
46        WEST(x126,x105);
47        EAST(x230,x261);
48        P(x294,y15) = 0.7724588914578291;
49        EAST(x86,x105);
50        WEST(x294,x261);
51        WEST(x69,x54);
52        P(x366,y15) = 0.8710702612113795;
53        EAST(x41,x54);
54        EAST(x149,x174);
55        EAST(x126,x149);
56        EAST(x54,x69);
57        WEST(x201,x174);
58        P(x9,y15) = 0.08958914081909156;
59        WEST(x21,x14);
60        WEST(x105,x86);
61        WEST(x329,x294);
62        SOUTH(y20,y15);
63        WEST(x261,x230);
64        P(x230,y15) = 0.6802079464848104;
65        P(x54,y15) = 0.3001739060212123;
66        P(x174,y15) = 0.5920608441688513;
67        WEST(x86,x69);
68        NORTH(y12,y15);
69        EAST(x366,x405);
70        WEST(x9,x6);
71        WEST(x405,x366);
72        P(x405,y15) = 0.94463482979489;
73        MIN-YPOS(y12);
74        EAST(x105,x126);
75    };
76  }
77
78  instance navigation_inst_mdp__8 {
79    domain = navigation_mdp;
80    non-fluents = nf_navigation_inst_mdp__8;
81    init-state {
82      robot-at(x405,y12);
83    };
84    max-nondef-actions = 1;
85    horizon = 40;
86    discount = 1.0;
87  }
```

Instance 9

```
non−fluents nf_navigation_inst_mdp__9 {
  domain = navigation_mdp;
  objects {
    xpos : {x41,x174,x30,x294,x366,x54,x230,x105,x329,x14,x21,x126,x201,x149,x9,x6,x261,x69,x405,x86};
    ypos : {y15,y20,y12,y27};
  };
  non−fluents {
    P(x30,y20) = 0.22066297931106468;
    WEST(x261,x230);
    WEST(x21,x14);
    P(x230,y15) = 0.6826771881039205;
    WEST(x366,x329);
    SOUTH(y27,y20);
    P(x405,y15) = 0.9465039047951761;
    P(x405,y20) = 0.9567379925007883;
    MAX−YPOS(y27);
    NORTH(y12,y15);
    SOUTH(y20,y15);
    P(x329,y20) = 0.845591881165379;
    WEST(x86,x69);
    EAST(x6,x9);
    P(x126,y20) = 0.5255273675644083;
    EAST(x41,x54);
    P(x366,y15) = 0.9121467062321148;
    P(x86,y15) = 0.39646985175970356;
    P(x9,y20) = 0.07717571731068587;
    EAST(x30,x41);
    P(x174,y20) = 0.5823425541043674;
    EAST(x86,x105);
    P(x261,y20) = 0.7237653274236149;
    EAST(x149,x174);
    P(x6,y15) = 0.03873947076499462;
    WEST(x201,x174);
    P(x174,y15) = 0.6239512122579312;
    P(x201,y20) = 0.6425731274250307;
    P(x41,y15) = 0.2852499784019432;
    MAX−XPOS(x405);
    P(x329,y15) = 0.8237989943866667;
    EAST(x329,x366);
    WEST(x230,x201);
    WEST(x149,x126);
    P(x41,y20) = 0.24861652041344265;
    P(x149,y20) = 0.552126179890413;
    MIN−XPOS(x6);
    WEST(x54,x41);
    EAST(x126,x149);
    P(x149,y15) = 0.5461224238143155;
```

```
48        P(x14,y15) = 0.10511640175000618;
49        EAST(x261,x294);
50        WEST(x174,x149);
51        P(x261,y15) = 0.7466745006998903;
52        WEST(x14,x9);
53        P(x86,y20) = 0.4120094569301919;
54        MIN-YPOS(y12);
55        P(x105,y15) = 0.44468523473723937;
56        SOUTH(y15,y12);
57        P(x366,y20) = 0.8901059870657168;
58        WEST(x294,x261);
59        EAST(x54,x69);
60        WEST(x126,x105);
61        EAST(x201,x230);
62        P(x201,y15) = 0.668071996322588;
63        EAST(x69,x86);
64        P(x14,y20) = 0.11832347693607996;
65        WEST(x9,x6);
66        P(x21,y15) = 0.1869948428908461;
67        EAST(x230,x261);
68        EAST(x174,x201);
69        P(x69,y15) = 0.3620691891563566;
70        GOAL(x405,y27);
71        P(x30,y15) = 0.20788565547646662;
72        EAST(x294,x329);
73        EAST(x105,x126);
74        P(x126,y15) = 0.5183540826761408;
75        P(x54,y15) = 0.30929778271207686;
76        P(x21,y20) = 0.15904580260087786;
77        P(x294,y15) = 0.793770940582219;
78        WEST(x30,x21);
79        EAST(x14,x21);
80        P(x9,y15) = 0.0694843544379661;
81        NORTH(y20,y27);
82        P(x54,y20) = 0.29557425737699594;
83        P(x230,y20) = 0.6812154801170293;
84        WEST(x105,x86);
85        WEST(x329,x294);
86        WEST(x405,x366);
87        P(x105,y20) = 0.48526905761345435;
88        NORTH(y15,y20);
89        WEST(x69,x54);
90        EAST(x366,x405);
91        P(x6,y20) = 0.05842727981507778;
92        P(x294,y20) = 0.8174746013981732;
93        P(x69,y20) = 0.3763092361194523;
94        WEST(x41,x30);
95        EAST(x21,x30);
96        EAST(x9,x14);
97    };
98  }
99
```

```
instance navigation_inst_mdp__9 {
  domain = navigation_mdp;
  non−fluents = nf_navigation_inst_mdp__9;
  init−state {
    robot−at(x405,y12);
  };
  max−nondef−actions = 1;
  horizon = 40;
  discount = 1.0;
}
```

Instance 10

```
non−fluents nf_navigation_inst_mdp__10 {
  domain = navigation_mdp;
  objects {
    xpos : {x86,x9,x201,x69,x329,x126,x261,x54,x230,x30,x6,x174,x149,x366,x21,x405,x14,x294,x41,x105};
    ypos : {y20,y36,y12,y27,y15};
  };
  non−fluents {
    P(x14,y15) = 0.15249802260414552;
    SOUTH(y15,y12);
    WEST(x69,x54);
    P(x294,y27) = 0.7871350846988591;
    EAST(x54,x69);
    P(x261,y20) = 0.7218886631920836;
    P(x201,y20) = 0.6473250686142006;
    EAST(x21,x30);
    EAST(x41,x54);
    P(x69,y20) = 0.3765070266825588;
    MIN−YPOS(y12);
    NORTH(y12,y15);
    WEST(x41,x30);
    EAST(x6,x9);
    P(x201,y27) = 0.6546238641205587;
    WEST(x230,x201);
    EAST(x261,x294);
    P(x14,y27) = 0.1155198830621023;
    MIN−XPOS(x6);
    P(x30,y20) = 0.20109588794385722;
    P(x405,y27) = 0.9236670966799322;
    P(x21,y15) = 0.17413296334837614;
    NORTH(y20,y27);
    P(x30,y27) = 0.2127137481185951;
    P(x366,y20) = 0.9017004185405216;
    SOUTH(y27,y20);
    P(x41,y15) = 0.2896668278661213;
    P(x126,y15) = 0.5218342786752863;
    P(x105,y20) = 0.47504829154594946;
    P(x105,y27) = 0.46703882240935374;
```

```
38      P(x329,y20) = 0.8276717586834964;
39      SOUTH(y36,y27);
40      EAST(x126,x149);
41      P(x329,y15) = 0.8344797723387417;
42      P(x261,y15) = 0.743265128057254;
43      P(x230,y27) = 0.721857582365996;
44      WEST(x366,x329);
45      WEST(x86,x69);
46      GOAL(x405,y36);
47      P(x329,y27) = 0.8417070795242724;
48      P(x174,y20) = 0.57908649514368;
49      NORTH(y27,y36);
50      P(x9,y20) = 0.06489074041478729;
51      P(x30,y15) = 0.24802985405059239;
52      WEST(x14,x9);
53      P(x86,y15) = 0.4203020499921159;
54      P(x126,y27) = 0.5217572396719141;
55      P(x230,y15) = 0.6967564664388958;
56      P(x405,y20) = 0.9195293152312699;
57      MAX−XPOS(x405);
58      WEST(x405,x366);
59      EAST(x149,x174);
60      WEST(x201,x174);
61      WEST(x294,x261);
62      P(x126,y20) = 0.502970067980258;
63      EAST(x86,x105);
64      WEST(x30,x21);
65      P(x174,y27) = 0.6242552704520916;
66      P(x6,y27) = 0.05393376015126705;
67      EAST(x366,x405);
68      P(x149,y20) = 0.5566534208820054;
69      P(x105,y15) = 0.45070689548983384;
70      SOUTH(y20,y15);
71      P(x9,y27) = 0.09803082372405028;
72      P(x6,y20) = 0.05381382070481777;
73      P(x21,y20) = 0.1901519668141478;
74      P(x54,y27) = 0.30268661153355714;
75      EAST(x294,x329);
76      EAST(x69,x86);
77      P(x149,y15) = 0.5480713560000846;
78      NORTH(y15,y20);
79      WEST(x174,x149);
80      P(x261,y27) = 0.7239234615730024;
81      EAST(x174,x201);
82      P(x69,y27) = 0.3881094917458923;
83      P(x405,y15) = 0.9321193240190807;
84      WEST(x149,x126);
85      P(x294,y15) = 0.7941499553424748;
86      P(x6,y15) = 0.049380214884877205;
87      P(x230,y20) = 0.6796873716245356;
88      WEST(x261,x230);
89      WEST(x105,x86);
```

```
90      P(x366,y15) = 0.8813777306166134;
91      P(x9,y15) = 0.09127656693913436;
92      P(x366,y27) = 0.8738803072881541;
93      P(x201,y15) = 0.6594119965049782;
94      P(x149,y27) = 0.5589389553979823;
95      WEST(x21,x14);
96      P(x54,y20) = 0.33979411550650473;
97      P(x86,y27) = 0.40163065473500054;
98      P(x14,y20) = 0.14583939402119112;
99      P(x174,y15) = 0.6133839792915081;
100     P(x54,y15) = 0.33987255149373885;
101     P(x294,y20) = 0.7689352319146948;
102     WEST(x329,x294);
103     EAST(x201,x230);
104     EAST(x230,x261);
105     WEST(x54,x41);
106     EAST(x30,x41);
107     EAST(x9,x14);
108     P(x21,y27) = 0.17168686323260007;
109     EAST(x105,x126);
110     MAX-YPOS(y36);
111     P(x69,y15) = 0.3854513592821987;
112     WEST(x126,x105);
113     P(x41,y27) = 0.2793932707097969;
114     EAST(x14,x21);
115     EAST(x329,x366);
116     P(x41,y20) = 0.25024425783684767;
117     P(x86,y20) = 0.43350079449775974;
118     WEST(x9,x6);
119   };
120 }
121
122 instance navigation_inst_mdp__10 {
123   domain = navigation_mdp;
124   non-fluents = nf_navigation_inst_mdp__10;
125   init-state {
126     robot-at(x405,y12);
127   };
128   max-nondef-actions = 1;
129   horizon = 40;
130   discount = 1.0;
131 }
```

A.4.6 Skill Teaching (see [ippb])

Domain

```
1 ////////////////////////////////////////////////////////////////
2 // Skill Teaching Domain
3 //
```

```
// Author: Tom Walsh (thomasjwalsh [at] gmail.com)
// Special thanks to Derek Green and Paul Cohen at
// University of Arizona for help with the design.
//
// In the SkillTeaching MDP domain, the agent is trying to teach a series
// of skills to a student through the use of hints and multiple choice
// questions. The student has a proficiency level for each skill, which
// indicates his ability to answer questions of that skill and positive
// reward is given for high proficiency on skills while negative reward
// is given for low proficiency. Each skill also has a weight on
// how much it is worth.
//
// Many of the skills are connected in that some are
// ``pre-conditions'' of others. If all of a skill's
// pre-conditions are learned, the student has some probability
// of answering questions about it right, and each precondition
// that is at high proficiency adds to the probability though
// knowing all of them can lead to a probability higher than the sum
// of the components. Hints only work if all the preconditions
// are known and can only get you to medium proficiency.
//
// student proficiency increases with questions answered right and
// decreases with questions about a skill answered wrong and
// sometimes decreases by chance.
//
// To model the teacher-student interaction, every other step in the
// domain is the student's turn, where they answer a question.
//
// The planning problems here are:
// 1) Whether or not to teach all the prerequisites of a skill before
// teaching it.
// 2) What skill to focus on next
// 3) When to give hints and when to use multiple choice problems
//
////////////////////////////////////////////////////////////////////

domain skill-teaching_mdp {

  requirements = {
    reward-deterministic
  };

  types {
    skill : object;
  };

  pvariables {

    //how valuable is this skill?
    SKILL-WEIGHT(skill) : { non-fluent, real, default = 1.0 };

    //some skills are pre-reqs for others. Your ability to achiev a higher level skill is dependent on how
```

```
56      //many of the pre−reqs you have mastered
57      PRE−REQ(skill, skill) : { non−fluent, bool, default = false };
58
59      //probability of getting a question right if you have all the pre−reqs
60      PROB−ALL−PRE(skill) : { non−fluent, real, default = 0.8 };
61      //if you don't have all the pre−cons, probaility mass is summed using these individual pieces
62      PROB−PER−PRE(skill) : { non−fluent, real, default = 0.1 };
63
64      PROB−ALL−PRE−MED(skill) : { non−fluent, real, default = 1.0 };
65      //if you don't have all the pre−cons, probaility mass is summed using these individual pieces
66      PROB−PER−PRE−MED(skill) : { non−fluent, real, default = 0.3 };
67
68      PROB−HIGH(skill) : { non−fluent, real, default = 0.9 };
69
70      LOSE−PROB(skill) : { non−fluent, real, default = 0.02 };
71
72      //proficiency values, they accumulate so low and med can be on at the same time and only high will turn off
73      proficiencyMed(skill) : { state−fluent, bool, default = false };
74      proficiencyHigh(skill) : { state−fluent, bool, default = false };
75
76      updateTurn(skill) : {state−fluent, bool, default = false};
77
78      answeredRight(skill): {state−fluent, bool, default = false};
79      hintedRight(skill): {state−fluent, bool, default = false};
80      hintDelayVar(skill) : {state−fluent, bool, default = false};
81
82      //two actions. Hint can get you directly to proficiencyMed, but only if all the pre−reqs are on
83      askProb(skill) : {action−fluent, bool, default = false};
84      giveHint(skill) : {action−fluent, bool, default = false};
85    };
86
87    cpfs {
88
89      updateTurn'(?s) =
90        KronDelta( [forall_{?s2: skill} ~updateTurn(?s2)] ^ (askProb(?s) | giveHint(?s)) );
91
92      //without intermediate nodes, we need to keep ''on'' all proficiency levels that have been attained
93
94      answeredRight'(?s) =
95        if ([forall_{?s2: skill} ~updateTurn(?s2)] ^ askProb(?s) ^ proficiencyHigh(?s))
96          then Bernoulli(PROB−HIGH(?s))
97        else if ([forall_{?s2: skill} ~updateTurn(?s2)] ^ askProb(?s) ^ proficiencyMed(?s) ^forall_{?s3: skill}[PRE
              −REQ(?s3, ?s) => proficiencyHigh(?s3)])
98          then Bernoulli(PROB−ALL−PRE−MED(?s))
99          else if ([forall_{?s2: skill} ~updateTurn(?s2)] ^ askProb(?s) ^proficiencyMed(?s) ^ askProb(?s))
100           then Bernoulli(sum_{?s2: skill}[PRE−REQ(?s2, ?s) * PROB−PER−PRE−MED(?s)])
101       else if ([forall_{?s3: skill} ~updateTurn(?s3)] ^ askProb(?s) ^forall_{?s2: skill}[PRE−REQ(?s2, ?s) =>
              proficiencyHigh(?s2)])
102         then Bernoulli(PROB−ALL−PRE(?s))
103         else if ([forall_{?s2: skill} ~updateTurn(?s2)] ^ askProb(?s) ^ askProb(?s))
104           then Bernoulli(sum_{?s2: skill}[PRE−REQ(?s2, ?s) * PROB−PER−PRE(?s)])
105       else
```

```
106             KronDelta( false );
107
108       hintedRight'(?s) =
109         KronDelta( [forall_{?s3: skill} ~updateTurn(?s3)] ^ giveHint(?s) ^ forall_{?s2: skill}[PRE-REQ(?s2, ?s)
                => proficiencyHigh(?s2)] );
110
111       hintDelayVar'(?s) =
112         KronDelta( [forall_{?s2: skill} ~updateTurn(?s2)] ^ giveHint(?s) );
113
114       //proficiencyMed can be reached through a hint if all preconditions are known or by a problem answered
              correctly
115       proficiencyMed'(?s) =
116           if (~updateTurn(?s) ^ proficiencyMed(?s))
117             then KronDelta( true )
118           else if (updateTurn(?s) ^ hintedRight(?s))
119             then KronDelta( true )
120           else if (updateTurn(?s) ^ answeredRight(?s))
121             then KronDelta( true )
122           else if (proficiencyHigh(?s)) //may come down
123             then KronDelta( true )
124           else if (proficiencyMed(?s) ^ updateTurn(?s) ^ hintDelayVar(?s))
125             then KronDelta( true ) //can't lose it on a hint
126             else
127               KronDelta( false );
128
129       //high proficiency is reached by getting a question and having proficiencyMed
130       //but you can lose it too if you get questions wrong
131       proficiencyHigh'(?s) =
132           if (forall_{?s2: skill}[~updateTurn(?s2)]) //student turn
133             then KronDelta( proficiencyHigh(?s) )
134           else if (~updateTurn(?s) ^ proficiencyHigh(?s))
135             then Bernoulli(1.0 - LOSE-PROB(?s))
136           else if (proficiencyMed(?s) ^ updateTurn(?s) ^ answeredRight(?s))
137             then KronDelta( true )
138           else if (proficiencyHigh(?s) ^ updateTurn(?s) ^ (hintDelayVar(?s) | answeredRight(?s))) //can't lose it on a
                  hint
139             then KronDelta( true )
140           else KronDelta( false );
141
142     };
143
144     reward = [sum_{?s : skill} [SKILL-WEIGHT(?s) * proficiencyHigh(?s)]] + [sum_{?s : skill} -[SKILL-
            WEIGHT(?s) * ~proficiencyMed(?s)]];
145
146 }
```

Instance 1

```
1 non-fluents nf-skill-teaching_inst_mdp__1 {
2   domain = skill-teaching_mdp;
```

```
  objects {
    skill : {s0,s1};

  };
  non−fluents {
    PROB−ALL−PRE(s0) = 0.56987906;
    PROB−ALL−PRE−MED(s0) = 0.71801746;
    PROB−HIGH(s0) = 0.9066789;
    SKILL−WEIGHT(s0) = 1.1778302;
    LOSE−PROB(s0) = 0.04352919459342957;
    PROB−ALL−PRE(s1) = 0.7414986;
    PROB−ALL−PRE−MED(s1) = 0.7900833;
    PROB−HIGH(s1) = 0.9543038;
    SKILL−WEIGHT(s1) = 1.2346091;
    LOSE−PROB(s1) = 0.018769168853759767;
  };
}
instance skill−teaching_inst_mdp__1 {
  domain = skill−teaching_mdp;
  non−fluents = nf−skill−teaching_inst_mdp__1;
  max−nondef−actions = 1;
  horizon = 40;
  discount = 1.0;
}
```

Instance 2

```
non−fluents nf−skill−teaching_inst_mdp__2 {
  domain = skill−teaching_mdp;
  objects {
    skill : {s0,s1};

  };
  non−fluents {
    PROB−ALL−PRE(s0) = 0.6266419;
    PROB−ALL−PRE−MED(s0) = 0.78803456;
    PROB−HIGH(s0) = 0.8867099;
    SKILL−WEIGHT(s0) = 1.4431845;
    LOSE−PROB(s0) = 0.034901031851768495;
    PROB−ALL−PRE(s1) = 0.692982;
    PROB−ALL−PRE−MED(s1) = 0.6979286;
    PROB−HIGH(s1) = 0.882593;
    SKILL−WEIGHT(s1) = 1.4221066;
    LOSE−PROB(s1) = 0.028824603557586672;
  };
}
instance skill−teaching_inst_mdp__2 {
  domain = skill−teaching_mdp;
  non−fluents = nf−skill−teaching_inst_mdp__2;
  max−nondef−actions = 1;
```

```
  horizon = 40;
  discount = 1.0;
}
```

Instance 3

```
non−fluents nf−skill−teaching_inst_mdp__3 {
  domain = skill−teaching_mdp;
  objects {
    skill : {s0,s1,s2,s3};

  };
  non−fluents {
    PROB−ALL−PRE(s0) = 0.66335756;
    PROB−ALL−PRE−MED(s0) = 0.7459964;
    PROB−HIGH(s0) = 0.99047893;
    SKILL−WEIGHT(s0) = 1.0374055;
    LOSE−PROB(s0) = 0.012337374687194825;
    PROB−ALL−PRE(s1) = 0.55798024;
    PROB−ALL−PRE−MED(s1) = 0.7089525;
    PROB−HIGH(s1) = 0.8791513;
    SKILL−WEIGHT(s1) = 1.1605124;
    LOSE−PROB(s1) = 0.04907787442207337;
    PRE−REQ(s1, s2);
    PROB−ALL−PRE(s2) = 0.708089;
    PROB−PER−PRE(s2) = 0.6819602966308593;
    PROB−ALL−PRE−MED(s2) = 0.7432575;
    PROB−PER−PRE−MED(s2) = 0.6840869665145874;
    PROB−HIGH(s2) = 0.9442033;
    SKILL−WEIGHT(s2) = 2.058421;
    LOSE−PROB(s2) = 0.0229320228099823;
    PRE−REQ(s2, s3);
    PRE−REQ(s1, s3);
    PROB−ALL−PRE(s3) = 0.6968088;
    PROB−PER−PRE(s3) = 0.27056136131286623;
    PROB−ALL−PRE−MED(s3) = 0.6968088;
    PROB−PER−PRE−MED(s3) = 0.29863872528076174;
    PROB−HIGH(s3) = 0.9625534;
    SKILL−WEIGHT(s3) = 3.2540152;
    LOSE−PROB(s3) = 0.018247979879379272;
  };
}
instance skill−teaching_inst_mdp__3 {
  domain = skill−teaching_mdp;
  non−fluents = nf−skill−teaching_inst_mdp__3;
  max−nondef−actions = 1;
  horizon = 40;
  discount = 1.0;
}
```

A.4.7 Triangle Tireworld (see [ippb])

Domain

```
////////////////////////////////////////////////////////////
//
// Triangle Tireworld Domain from IPPC 2008
//
// This domain is taken from
//
// I. Little and S. Thiebaux.
// Probabilistic Planning vs Replanning.
// ICAPS Workshop International Planning Competition: Past, Present and Future, 2007.
// http://users.cecs.anu.edu.au/~thiebaux/papers/icaps07wksp.pdf
//
// who defined this as a "probabilistically interesting" problem
// (see Definition 1 in the above citation). In short, this problem
// was intended to be difficult for determinization/replanning approaches
// since the highest probability path to the goal is longer than other
// lower probability (but still possible) paths to the goal.
//
// This version is a direct translation of the version from the IPPC 2008
//
// http://ippc-2008.loria.fr/wiki/index.html
//
// run by Daniel Bryce and Olivier Buffet. See the results of IPPC 2008
// planners on this problem in Figure 1 here
//
// http://ippc-2008.loria.fr/wiki/images/0/03/Results.pdf
//
// taken from
//
// http://ippc-2008.loria.fr/wiki/index.php/Results.html
//
// RDDL translation by Scott Sanner (ssanner@gmail.com). The
// original PPDDL domain is included in comments at the end, which
// also provides a nice point of comparison between RDDL and PPDDL
// domain specification styles.
//
////////////////////////////////////////////////////////////

domain triangle_tireworld_mdp {

    types {
        location : object;
    };

    pvariables {

    // Nonfluents: probability constants
    FLAT-PROB : { non-fluent, real, default = 0.49 };

```

```
49      // Nonfluents: topology
50      road(location,location) : { non−fluent, bool, default = false }; // Road topology
51      goal−location(location) : { non−fluent, bool, default = false }; // Additional nonfluent to specify a goal
            location
52
53    // State
54      vehicle−at(location) : { state−fluent, bool, default = false };
55      spare−in(location) : { state−fluent, bool, default = false };
56      not−flattire : { state−fluent, bool, default = false }; // Not clear why negated
57      hasspare : { state−fluent, bool, default = false };
58      goal−reward−received : { state−fluent, bool, default = false }; // An additional fluent to enforce a goal
            reward is only received once
59
60    // Actions
61      move−car(location,location) : { action−fluent, bool, default = false }; // Not clear why we need from location
            parameter
62      loadtire(location) : { action−fluent, bool, default = false }; // Not clear to me why this requires location
            parameter
63      changetire : { action−fluent, bool, default = false };
64    };
65
66    cpfs {
67
68    // Some observations on PPDDL vs. RDDL:
69    //
70    // A domain like this is where PPDDL action−centric effects are more intuitive... in RDDL,
71    // transition specifications are fluent−centric and we have to explicitly define fluent
72    // values in the next state as a function of the previous state. These are essentially
73    // successor state axioms and can be compiled from effects using Ray Reiters default solution
74    // to the situation calculus... so it could be possible to automate translation from PPDDL to
75    // RDDL. Compiling successor state axioms back into effects (RDDL−>PPDDL) would be harder.
76    //
77    // So why not use PPDDL style action−centric effects in RDDL if they are more clear? PPDDL is
78    // more clear for domains like this, but when multiple independent probabilistic exogenous
79    // events act on a fluent, the action−centric PPDDL approach is not guaranteed to provide a
80    // consistent state update or probability distribution and hence simply cannot be used, hence
81    // my motivation for RDDL.
82
83    vehicle−at'(?l) =
84      // Did it move to ?l and become true?
85      if (exists_{?from : location} (move−car(?from,?l) ^ vehicle−at(?from) ^ road(?from,?l) ^ not−flattire))
86        then true
87
88      // Did it leave ?l and become false?
89      else if (exists_{?to : location} (move−car(?l,?to) ^ vehicle−at(?l) ^ road(?l,?to) ^ not−flattire))
90        then false
91
92      // It didn't move, so it's current value persists (frame axiom)
93      else
94        vehicle−at(?l);
95
96    spare−in'(?l) =
```

```
97      // Was the spare used?
98      if (loadtire(?l) ^ vehicle−at(?l) ^ spare−in(?l))
99        then false
100
101     // It was not used, so it's current value persists (frame axiom)
102     else
103       spare−in(?l);
104
105   not−flattire' =
106     // If the car moved there is a FLAT−PROB probability of getting a flat
107     if (exists_{?from : location, ?to : location} (move−car(?from,?to) ^ vehicle−at(?from) ^ road(?from,?to) ^ not
            −flattire))
108       then Bernoulli(FLAT−PROB)
109
110     // If the tire was changed, then the flat is fixed
111     else if (changetire ^ hasspare)
112       then true
113
114     // The car didn't move and the tire was not changed, so it's current value persists (frame axiom)
115     else
116       not−flattire;
117
118   hasspare' =
119     // If the tire was changed, then the spare is used
120     if (changetire ^ hasspare)
121       then false
122
123     // If the tire was loaded, then the spare is available
124     else if (exists_{?l : location} (loadtire(?l) ^ vehicle−at(?l) ^ spare−in(?l)))
125       then true
126
127     // The spare was not used or replaced, so it's current value persists (frame axiom)
128     else
129       hasspare;
130
131   goal−reward−received' = goal−reward−received | exists_{?l : location} (vehicle−at(?l) ^ goal−location(?l));
132
133   };
134
135   // We get a reward of 100 for reaching the goal and lose −1 on every iteration goal not reached
136   reward = if (~goal−reward−received ^ exists_{?l : location} (vehicle−at(?l) ^ goal−location(?l)))
137               then 100
138            else if (goal−reward−received) then 0
139            else −1; // Modified from IPPC 2008 to encourage shorter paths since we don't separately evaluate
                    plan length
140
141 }
142
143 // Original PPDDL version of "Triangle Tireworld" domain from IPPC 2008:
144 //
145 // http://ippc−2008.loria.fr/wiki/images/6/68/Benchmarks−FOP.tgz
146 //
```

```
// DOMAIN:
//
// (define (domain triangle-tire)
// (:requirements :typing :strips :equality :probabilistic-effects :rewards)
// (:types location)
// (:predicates (vehicle-at ?loc - location)
// (spare-in ?loc - location)
// (road ?from - location ?to - location)
// (not-flattire) (hasspare))
// (:action move-car
// :parameters (?from - location ?to - location)
// :precondition (and (vehicle-at ?from) (road ?from ?to) (not-flattire))
// :effect (and (vehicle-at ?to) (not (vehicle-at ?from))
// (probabilistic 0.5 (not (not-flattire)))))
// (:action loadtire
// :parameters (?loc - location)
// :precondition (and (vehicle-at ?loc) (spare-in ?loc))
// :effect (and (hasspare) (not (spare-in ?loc))))
// (:action changetire
// :precondition (hasspare)
// :effect (and (not (hasspare)) (not-flattire)))
//)
//
// INSTANCE OBJECTIVE EXAMPLE:
//
// (:goal (vehicle-at l-1-3)) (:goal-reward 100) (:metric maximize (reward)))
```

Instance 1

```
non-fluents nf_triangle_tireworld_inst_mdp__1 {
  domain = triangle_tireworld_mdp;
  objects {
    location : {la1a1, la1a2, la1a3, la2a1, la2a2, la3a1};
  };

  non-fluents {
    FLAT-PROB = 0.4;
    road(la1a1,la1a2);
    road(la1a2,la1a3);
    road(la1a1,la2a1);
    road(la1a2,la2a2);
    road(la2a1,la1a2);
    road(la2a2,la1a3);
    road(la2a1,la3a1);
    road(la3a1,la2a2);

    goal-location(la1a3);
  };
}

```

```
instance triangle_tireworld_inst_mdp__1 {
   domain = triangle_tireworld_mdp;
   non−fluents = nf_triangle_tireworld_inst_mdp__1;
   init−state {
      vehicle−at(la1a1);
      spare−in(la2a1);
      spare−in(la2a2);
      spare−in(la3a1);
      spare−in(la3a1);
      not−flattire;
   };

   max−nondef−actions = 1;
   horizon = 40;
   discount = 1.0;
}
```

Instance 2

```
non−fluents nf_triangle_tireworld_inst_mdp__2 {
   domain = triangle_tireworld_mdp;
   objects {
      location : {la1a1, la1a2, la1a3, la2a1, la2a2, la3a1};
   };

   non−fluents {
      FLAT−PROB = 0.499;
      road(la1a1,la1a2);
      road(la1a2,la1a3);
      road(la1a1,la2a1);
      road(la1a2,la2a2);
      road(la2a1,la1a2);
      road(la2a2,la1a3);
      road(la2a1,la3a1);
      road(la3a1,la2a2);

      goal−location(la1a3);
   };
}

instance triangle_tireworld_inst_mdp__2 {
   domain = triangle_tireworld_mdp;
   non−fluents = nf_triangle_tireworld_inst_mdp__2;
   init−state {
      vehicle−at(la1a1);
      spare−in(la2a1);
      spare−in(la2a2);
      spare−in(la3a1);
      spare−in(la3a1);
      not−flattire;
```

```
    };

    max-nondef-actions = 1;
    horizon = 40;
    discount = 1.0;
}
```

Instance 3

```
non-fluents nf_triangle_tireworld_inst_mdp__3 {
    domain = triangle_tireworld_mdp;
    objects {
        location : {la1a1, la1a2, la1a3, la1a4, la1a5, la2a1, la2a2, la2a3, la2a4, la3a1, la3a2, la3a3, la4a1, la4a2, la5a1
                };
    };

    non-fluents {
        FLAT-PROB = 0.35;
        road(la1a1,la1a2);
        road(la1a2,la1a3);
        road(la1a3,la1a4);
        road(la1a4,la1a5);
        road(la1a1,la2a1);
        road(la1a2,la2a2);
        road(la1a3,la2a3);
        road(la1a4,la2a4);
        road(la2a1,la1a2);
        road(la2a2,la1a3);
        road(la2a3,la1a4);
        road(la2a4,la1a5);
        road(la3a1,la3a2);
        road(la3a2,la3a3);
        road(la2a1,la3a1);
        road(la2a3,la3a3);
        road(la3a1,la2a2);
        road(la3a3,la2a4);
        road(la3a1,la4a1);
        road(la3a2,la4a2);
        road(la4a1,la3a2);
        road(la4a2,la3a3);
        road(la4a1,la5a1);
        road(la5a1,la4a2);

        goal-location(la1a5);
    };
}

instance triangle_tireworld_inst_mdp__3 {
    domain = triangle_tireworld_mdp;
    non-fluents = nf_triangle_tireworld_inst_mdp__3;
```

```
41    init−state {
42      vehicle−at(la1a1);
43      spare−in(la2a1);
44      spare−in(la2a2);
45      spare−in(la2a3);
46      spare−in(la2a4);
47      spare−in(la3a1);
48      spare−in(la3a3);
49      spare−in(la4a1);
50      spare−in(la4a2);
51      spare−in(la5a1);
52      spare−in(la5a1);
53      not−flattire;
54    };
55
56    max−nondef−actions = 1;
57    horizon = 40;
58    discount = 1.0;
59  }
```

Instance 4

```
1   non−fluents nf_triangle_tireworld_inst_mdp__4 {
2     domain = triangle_tireworld_mdp;
3     objects {
4       location : {la1a1, la1a2, la1a3, la1a4, la1a5, la2a1, la2a2, la2a3, la2a4, la3a1, la3a2, la3a3, la4a1, la4a2, la5a1
                  };
5     };
6
7     non−fluents {
8       FLAT−PROB = 0.45;
9       road(la1a1,la1a2);
10      road(la1a2,la1a3);
11      road(la1a3,la1a4);
12      road(la1a4,la1a5);
13      road(la1a1,la2a1);
14      road(la1a2,la2a2);
15      road(la1a3,la2a3);
16      road(la1a4,la2a4);
17      road(la2a1,la1a2);
18      road(la2a2,la1a3);
19      road(la2a3,la1a4);
20      road(la2a4,la1a5);
21      road(la3a1,la3a2);
22      road(la3a2,la3a3);
23      road(la2a1,la3a1);
24      road(la2a3,la3a3);
25      road(la3a1,la2a2);
26      road(la3a3,la2a4);
27      road(la3a1,la4a1);
```

```
28      road(la3a2,la4a2);
29      road(la4a1,la3a2);
30      road(la4a2,la3a3);
31      road(la4a1,la5a1);
32      road(la5a1,la4a2);
33
34      goal-location(la1a5);
35    };
36  }
37
38  instance triangle_tireworld_inst_mdp__4 {
39    domain = triangle_tireworld_mdp;
40    non-fluents = nf_triangle_tireworld_inst_mdp__4;
41    init-state {
42      vehicle-at(la1a1);
43      spare-in(la2a1);
44      spare-in(la2a2);
45      spare-in(la2a3);
46      spare-in(la2a4);
47      spare-in(la3a1);
48      spare-in(la3a3);
49      spare-in(la4a1);
50      spare-in(la4a2);
51      spare-in(la5a1);
52      spare-in(la5a1);
53      not-flattire;
54    };
55
56    max-nondef-actions = 1;
57    horizon = 40;
58    discount = 1.0;
59  }
```

Instance 5

```
1   non-fluents nf_triangle_tireworld_inst_mdp__5 {
2     domain = triangle_tireworld_mdp;
3     objects {
4       location : {la1a1, la1a2, la1a3, la1a4, la1a5, la1a6, la1a7, la2a1, la2a2, la2a3, la2a4, la2a5, la2a6, la3a1, la3a2,
              la3a3, la3a4, la3a5, la4a1, la4a2, la4a3, la4a4, la5a1, la5a2, la5a3, la6a1, la6a2, la7a1};
5     };
6
7     non-fluents {
8       FLAT-PROB = 0.3;
9       road(la1a1,la1a2);
10      road(la1a2,la1a3);
11      road(la1a3,la1a4);
12      road(la1a4,la1a5);
13      road(la1a5,la1a6);
14      road(la1a6,la1a7);
```

```
15        road(la1a1,la2a1);
16        road(la1a2,la2a2);
17        road(la1a3,la2a3);
18        road(la1a4,la2a4);
19        road(la1a5,la2a5);
20        road(la1a6,la2a6);
21        road(la2a1,la1a2);
22        road(la2a2,la1a3);
23        road(la2a3,la1a4);
24        road(la2a4,la1a5);
25        road(la2a5,la1a6);
26        road(la2a6,la1a7);
27        road(la3a1,la3a2);
28        road(la3a2,la3a3);
29        road(la3a3,la3a4);
30        road(la3a4,la3a5);
31        road(la2a1,la3a1);
32        road(la2a3,la3a3);
33        road(la2a5,la3a5);
34        road(la3a1,la2a2);
35        road(la3a3,la2a4);
36        road(la3a5,la2a6);
37        road(la3a1,la4a1);
38        road(la3a2,la4a2);
39        road(la3a3,la4a3);
40        road(la3a4,la4a4);
41        road(la4a1,la3a2);
42        road(la4a2,la3a3);
43        road(la4a3,la3a4);
44        road(la4a4,la3a5);
45        road(la5a1,la5a2);
46        road(la5a2,la5a3);
47        road(la4a1,la5a1);
48        road(la4a3,la5a3);
49        road(la5a1,la4a2);
50        road(la5a3,la4a4);
51        road(la5a1,la6a1);
52        road(la5a2,la6a2);
53        road(la6a1,la5a2);
54        road(la6a2,la5a3);
55        road(la6a1,la7a1);
56        road(la7a1,la6a2);
57
58        goal-location(la1a7);
59    };
60  }
61
62  instance triangle_tireworld_inst_mdp__5 {
63    domain = triangle_tireworld_mdp;
64    non-fluents = nf_triangle_tireworld_inst_mdp__5;
65    init-state {
66      vehicle-at(la1a1);
```

```
67      spare-in(la2a1);
68      spare-in(la2a2);
69      spare-in(la2a3);
70      spare-in(la2a4);
71      spare-in(la2a5);
72      spare-in(la2a6);
73      spare-in(la3a1);
74      spare-in(la3a5);
75      spare-in(la4a1);
76      spare-in(la4a2);
77      spare-in(la4a3);
78      spare-in(la4a4);
79      spare-in(la5a1);
80      spare-in(la5a3);
81      spare-in(la6a1);
82      spare-in(la6a2);
83      spare-in(la7a1);
84      spare-in(la7a1);
85      not-flattire;
86    };
87
88    max-nondef-actions = 1;
89    horizon = 40;
90    discount = 1.0;
91  }
```

A.5 Analyzed Evaluation Results

A.5.1 Table Legend

Algorithm	Algorithm: Used algorithm for the presented data (reward [optimizer], probability [optimizer], search [algorithm], PROST, Random Bandit).
Interpreted Parameters	First Boolean value: Usage of interpreted parameters for all variables except for quantifiers / sums / products. Second value: Usage of interpreted parameters for all quantifiers / sums / ….
Adjust Plan	True if a new plan should be generated whenever the current plan fails.
Approximation	Approximation value for the reward approximation, see Subsection 5.2.3.
Probability	Probability value of the calculated plan for the first step.
Reward	Reward received after the simulation of a plan using RDDLsim.
Expected Reward	Expected reward, reward value of the first plan according to the algorithm (value in case of success).
Reward Expectation	Expected value, calculated using the expected reward, the probability of a plan and the worst reward, see Section 6.1 (rounded to the first decimal place).
Translation Time	Time taken to translate the domain.
Probability Translation Time	Additional time taken to translate probability constraints after the translation of the domain was finished.
Time Left	Time left in milliseconds, decreasing, starting at 3600000
SSD	Sample standard deviation (formula: $\sqrt{\frac{\sum_{i=0}^{n}(x_i-\bar{x})^2}{n-1}}$)

A.5.2 Academic Advising

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	true		0.03328046333333333	0.03328046333333333	0.0	0.03328046333333333	0.03328046333333333
probability	true + true	false	2.0	0.04291428166666667	0.04291428166666667	0.0	0.04291428166666667	0.04291428166666667
probability	true + true	true	2.0	0.04291428166666667	0.04291428166666667	0.0	0.04291428166666667	0.04291428166666667
probability	true + true	false	3.0	0.06875763525400001	0.06875763525400001	0.0	0.06875763525400001	0.06875763525400001
probability	true + true	true	3.0	0.0669203069304	0.06875763525400001	0.0070066935698763775	0.039481139133333336	0.06875763525400001

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	true		-56.333333333333336	-50.0	19.953970019528192	-100.0	-25.0	-97.5
probability	true + true	false	2.0	-98.16666666666667	-100.0	10.041580220928045	-100.0	-45.0	-97.6
probability	true + true	true	2.0	-98.33333333333333	-100.0	9.128709291752768	-100.0	-50.0	-97.6
probability	true + true	false	3.0	-96.66666666666667	-100.0	8.64364759104401	-100.0	-75.0	-98.3
probability	true + true	true	3.0	-91.33333333333333	-100.0	14.793599112881344	-100.0	-40.0	-98.3

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	true		-25.0	0.0	532.2ms	4239.3ms	3427847
probability	true + true	false	2.0	-45.0	0.0	521.4ms	4132.3ms	3413522
probability	true + true	true	2.0	-45.0	0.0	515.6ms	4029.4ms	3337007
probability	true + true	false	3.0	-75.0	0.0	516.6ms	4006.1ms	3414896
probability	true + true	true	3.0	-74.66666666666667	1.8257418583505538	524.1ms	4090.1ms	3335883
search	true + true	false	3.0					-1

Algorithm	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
PROST	442636	-45.33333333	-45	7.648904963	-60	-35
Random Bandit	76898	-54.83333333	-52.5	19.00468787	-100	-30

A.5.3 Cooperative Recon

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	false		8.542434412431E-8	3.644728564553E-8	2.1959148435219E-7	5.82E-18	1.13478573646297E-6
reward	true + true	true		3.086519264054E-8	3.644728564553E-8	1.292091328041E-8	4.11546156E-12	3.644728564553E-8
probability	true + true	false	0.25	null				
probability	true + true	false	0.1	0.11855084400005148	8.127243E-14	0.23463543464588715	8.127243E-14	0.64
probability	true + true	true	0.1	0.08222966763715826	8.127243E-14	0.21078663430379166	8.127243E-14	0.64
search	true + true	false	0.8	0.64	0.64	0.0	0.64	0.64

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	false		1.9300000000000002	0.0	4.028087592787974	0.0	13.13	0.0
reward	true + true	true		2.412	0.0	4.597774124133516	0.0	13.13	-
probability	true + true	false	0.25	0.0	0.0	0.0	0.0	0.0	-
probability	true + true	false	0.1	0.1	0.0	0.38056219755369364	0.0	1.5	0.4
probability	true + true	true	0.1	0.8113333333333334	0.0	2.557363584973229	0.0	11.63	-
search	true + true	false	0.8	1.9383333333333332	0.0	4.747927618094727	0.0	11.63	7.4

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	false		13.016	0.6244037155558894	256.2ms	210.4ms	3491461
reward	true + true	true		13.13	0.0	257.9ms	202.6ms	3361383
probability	true + true	false	0.25					
probability	true + true	false	0.1	3.7496666666666667	4.1890476930577725	274.4ms	213.7ms	3324751
probability	true + true	true	0.1	2.175333333333333	2.5700633741459447	270.8ms	207.1ms	753906
search	true + true	false	0.8	11.63	0.0	346.7ms	253.7ms	-1

Algorithm	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
PROST	40971	5.071666667	1.5	5.562218661	0	13.13
Random Bandit	76703	0	0	-	0	0

A.5.4 Crossing Traffic

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	true		9.20529422205E-9	9.20529422205E-9	0.0	9.20529422205E-9	9.20529422205E-9
probability	true + true	false	6.0	6.366805760909E-7	6.366805760909E-7	0.0	6.366805760909E-7	6.366805760909E-7
probability	true + true	true	6.0	6.366805760909E-7	6.366805760909E-7	0.0	6.366805760909E-7	6.366805760909E-7
probability	true + true	true	6.0	6.366805760909E-7	6.366805760909E-7	0.0	6.366805760909E-7	6.366805760909E-7
probability	false + true	false	6.0	9.0954368012986E-7	9.0954368012986E-7	0.0	9.0954368012986E-7	9.0954368012986E-7
probability	false + true	true	6.0	9.0954368012986E-7	9.0954368012986E-7	0.0	9.0954368012986E-7	9.0954368012986E-7

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	true		-9.6	-2.0	15.459847882554477	-40.0	-2.0	-
probability	true + true	false	6.0	-29.733333333333334	-40.0	13.723710499899708	-40.0	-12.0	-40
probability	true + true	true	6.0	-34.3	-40.0	11.600089179205359	-40.0	-10.0	-
probability	true + true	true	6.0	-28.8	-40.0	13.951640615428316	-40.0	-12.0	-
probability	false + true	false	6.0	-31.366666666666667	-40.0	13.415079866183875	-40.0	-11.0	-40
probability	false + true	true	6.0	-23.8	-12.0	15.586023801182728	-40.0	-2.0	-

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	true		-2.0	0.0	118.9ms	14.7ms	3552286
probability	true + true	false	6.0	-12.0	0.0	123.6ms	15.0ms	3565328
probability	true + true	true	6.0	-12.0	0.0	123.3ms	14.7ms	3556719
probability	true + true	true	6.0	-12.0	0.0	124.0ms	17.1ms	3552130
probability	false + true	false	6.0	-11.3	0.466091599699399	1721.2ms	123.4ms	3420450
probability	false + true	true	6.0	-11.133333333333333	1.8332810859848758	1695.2ms	119.1ms	1555663
search	true + true	false						-1

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Crossing	2258920	-4.566666667	-4	0.971430986	-8	-4

A.5.5 Navigation

Instance 1

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	true		0.071841553474466	0.071841553474466	0.0	0.071841553474466	0.071841553474466
reward	false + true	true		0.071841553474466	0.071841553474466	0.0	0.071841553474466	0.071841553474466
probability	true + true	false	4.0	0.9510332886129618	0.9510332886129618	0.0	0.9510332886129618	0.9510332886129618
probability	true + true	true	4.0	0.9510332886129618	0.9510332886129618	0.0	0.9510332886129618	0.9510332886129618
probability	false + true	false	4.0	0.9510332886129618	0.9510332886129618	0.0	0.9510332886129618	0.9510332886129618
probability	false + true	true	4.0	0.9510332886129618	0.9510332886129618	0.0	0.9510332886129618	0.9510332886129618
search	true + true	false	4.0	0.9510332886129618	0.9510332886129618	0.0	0.9510332886129618	0.9510332886129618
search	false + true	false	4.0	0.9510332886129618	0.9510332886129618	0.0	0.9510332886129618	0.9510332886129618

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	true		-38.733333333333334	-40.0	6.937819061732104	-40.0	-2.0	-
reward	false + true	true		-38.733333333333334	-40.0	6.937819061732104	-40.0	-2.0	-
probability	true + true	false	4.0	-10.133333333333333	-8.0	8.118660214478798	-40.0	-8.0	-9.6
probability	true + true	true	4.0	-9.066666666666666	-8.0	5.8423739467217715	-40.0	-8.0	-
probability	false + true	false	4.0	-11.2	-8.0	9.764114452139669	-40.0	-8.0	-9.6
probability	false + true	true	4.0	-9.066666666666666	-8.0	5.8423739467217715	-40.0	-8.0	-
search	true + true	false	4.0	-9.066666666666666	-8.0	5.8423739467217715	-40.0	-8.0	-9.6
search	false + true	false	4.0	-10.133333333333333	-8.0	8.118660214478798	-40.0	-8.0	-9.6

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	true		-2.0	0.0	133.1ms	22.6ms	3563697
reward	false + true	true		-2.0	0.0	1874.1ms	224.1ms	3474188
probability	true + true	false	4.0	-8.0	0.0	132.4ms	22.4ms	3560053
probability	true + true	true	4.0	-8.0	0.0	134.8ms	22.5ms	3559768
probability	false + true	false	4.0	-8.0	0.0	1871.7ms	227.5ms	3432726
probability	false + true	true	4.0	-8.0	0.0	1915.3ms	228.9ms	3428524
search	true + true	false	4.0	-8.0	0.0	135.0ms	22.8ms	3542010
search	false + true	false	4.0	-8.0	0.0	1882.4ms	229.0ms	3269728

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Navigation 1	3594721	-9.066666667	-8	5.842373947	-40	-8

Instance 2

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	true		0.083674353081733	0.083674353081733	0.0	0.083674353081733	0.083674353081733
probability	true + true	false	8.0	0.9616623822953189	0.9639773815870285	0.008810012177882725	0.9292523922113836	0.9639773815870285
search	true + true	false	8.0	0.9639773815870285	0.9639773815870285	0.0	0.9639773815870285	0.9639773815870285

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	true		-37.46666666666667	-40.0	9.640909004693572	-40.0	-2.0	-
probability	true + true	false	8.0	-13.333333333333334	-12.0	5.181754018924835	-40.0	-12.0	-13.46
search	true + true	false	8.0	-17.5	-16.0	6.1405941529891175	-40.0	-13.0	-16.77

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	true		-2.0	0.0	188.6ms	30.1ms	3549976
probability	true + true	false	8.0	-12.4	1.2205143065174586	191.0ms	24.7ms	3541923
search	true + true	false	8.0	-15.9	0.5477225575051661	188.1ms	24.9ms	3511802

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Navigation 2	3594611	-12	-10	7.611243951	-40	-10

Instance 3

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	false		0.003939522253899224	0.003939522253899224	0.0	0.003939522253899224	0.003939522253899224
probability	true + true	false	8.0	0.9128728475780877	0.9128728475780877	0.0	0.9128728475780877	0.9128728475780877
search	true + true	false	8.0	0.9128728475780877	0.9128728475780877	0.0	0.9128728475780877	0.9128728475780877

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	false		-40.0	-40.0	0.0	-40.0	-40.0	-39.85
probability	true + true	false	8.0	-24.533333333333335	-24.0	2.9211869733608857	-40.0	-24.0	-25.39
search	true + true	false	8.0	-17.4	-14.0	7.204404782899283	-40.0	-14.0	-17.79

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	false		-3.0	0.0	259.4ms	43.3ms	3528472
probability	true + true	false	8.0	-24.0	0.0	266.7ms	44.3ms	3489589
search	true + true	false	8.0	-15.666666666666666	3.790490217894517	268.1ms	40.1ms	3337505

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Navigation 3	3592722	-14.86666667	-11	10.02663121	-40	-11

Instance 4

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	false		2.885020282797116E-5	2.885020282797116E-5	0.0	2.885020282797116E-5	2.885020282797116E-5
probability	true + true	false	4.0	0.868897570701338	0.868897570701338	0.0	0.868897570701338	0.868897570701338
search	true + true	false	4.0	0.868897570701338	0.868897570701338	0.0	0.868897570701338	0.868897570701338

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	false		-40.0	-40.0	0.0	-40.0	-40.0	-40.0
probability	true + true	false	4.0	-25.166666666666668	-20.0	9.120206188433475	-40.0	-17.0	-22.48
search	true + true	false	4.0	-21.533333333333335	-20.0	6.388207526552253	-40.0	-14.0	-22.22

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	false		-5.0	0.0	504.6ms	78.2ms	3466120
probability	true + true	false	4.0	-19.833333333333332	0.6477192523656043	530.0ms	70.6ms	3273469
search	true + true	false	4.0	-19.533333333333335	1.279367659898984	515.4ms	73.4ms	1668107

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Navigation 4	3578894	-16.6	-13	9.335139398	-40	-13

Instance 5

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	false		0.0663249796877304	0.0663249796877304	0.0	0.0663249796877304	0.0663249796877304
probability	true + true	false	10.0	0.9759851833805442	0.9759851833805442	0.0	0.9759851833805442	0.9759851833805442
search	true + true	false	10.0	0.9759851833805442	0.9759851833805442	0.0	0.9759851833805442	0.9759851833805442

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	false		-38.733333333333334	-40.0	6.937819061732104	-40.0	-2.0	-37.48
probability	true + true	false	10.0	-21.333333333333332	-20.0	5.074162634049249	-40.0	-20.0	-20.48
search	true + true	false	10.0	-20.0	-20.0	0.0	-20.0	-20.0	-20.48

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	false		-2.0	0.0	647.0ms	47.6ms	3451148
probability	true + true	false	10.0	-20.0	0.0	602.0ms	48.7ms	3433072
search	true + true	false	10.0	-20.0	0.0	627.4ms	49.5ms	3108454

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Navigation 5	3527645	-20.33333333	-20	1.372973951	-27	-20

Instance 6

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	false		0.006407984930621872	0.006407984930621872	0.0	0.006407984930621872	0.006407984930621872
probability	true + true	false	8.0	0.9362386705302082	0.9362386705302082	0.0	0.9362386705302082	0.9362386705302082
search	true + true	false	8.0	0.9362386705302082	0.9362386705302082	0.0	0.9362386705302082	0.9362386705302082

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	false		-38.766666666666666	-40.0	6.7552448758970485	-40.0	-3.0	-39.76
probability	true + true	false	8.0	-24.0	-24.0	0.0	-24.0	-24.0	-25.02
search	true + true	false	8.0	-25.033333333333335	-24.0	4.072475594925569	-40.0	-23.0	-24.99

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	false		-3.0	0.0	916.2ms	108.9ms	3376911
probability	true + true	false	8.0	-24.0	0.0	940.4ms	83.0ms	3302836
search	true + true	false	8.0	-23.966666666666665	0.18257418583505536	940.1ms	90.7ms	1604850

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Navigation 6	3354634	-24.36666667	-21	6.77461455	-40	-21

Instance 7

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	false		2.727139499032272E-4	2.727139499032272E-4	0.0	2.727139499032272E-4	2.727139499032272E-4
probability	true + true	false	6.0	0.9445480110827212	0.9445480110827212	0.0	0.9445480110827212	0.9445480110827212
search	true + true	false	6.0	0.9445480110827212	0.9445480110827212	0.0	0.9445480110827212	0.9445480110827212

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	false		-40.0	-40.0	0.0	-40.0	-40.0	-40.0
probability	true + true	false	6.0	-25.066666666666666	-24.0	4.059330107239399	-40.0	-24.0	-24.9
search	true + true	false	6.0	-24.0	-24.0	0.0	-24.0	-24.0	-24.9

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	false		-4.0	0.0	1233.3ms	130.2ms	3292246
probability	true + true	false	6.0	-24.0	0.0	1267.0ms	129.3ms	3014400
search	true + true	false	6.0	-24.0	0.0	1737.7ms	125.9ms	-1

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Navigation 7	3150587	-24.96666667	-22	6.065637149	-40	-22

Instance 8

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	false		0.05536517020511	0.05536517020511	0.0	0.05536517020511	0.05536517020511
probability	true + true	false	12.0	0.5998311761864706	0.5998311761864706	0.0	0.5998311761864706	0.5998311761864706

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	false		-36.2	-40.0	11.594885911915856	-40.0	-2.0	-37.9
probability	true + true	false	12.0	-29.333333333333332	-24.0	7.671412823766146	-40.0	-24.0	-30.4

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left

reward	true + true	false		-2.0	0.0	4062.9ms	203.3ms	2963441
probability	true + true	false	12.0	-24.0	0.0	4114.8ms	232.5ms	2894284
search	true + true	false						-1

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Navigation 8	3267846	-32.66666667	-40	10.54819263	-40	-18

Instance 9

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	false		0.0023143484719296346	0.0023143484719296346	0.0	0.0023143484719296346	0.0023143484719296346
probability	true + true	false	10.0	0.48654842234440077	0.48654842234440077	0.0	0.48654842234440077	0.48654842234440077

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	false		-38.766666666666666	-40.0	6.7552448758970485	-40.0	-3.0	-39.9
probability	true + true	false	10.0	-35.0	-35.0	5.0854762771560775	-40.0	-30.0	-35.1

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	false		-3.0	0.0	5084.2ms	330.5ms	2706663
probability	true + true	false	10.0	-29.966666666666665	0.18257418583505536	6105.6ms	218.4ms	2434293

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Navigation 9	3378752	-35.43333333	-40	6.729673525	-40	-13

Instance 10

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	false		4.169611929714425E-4	4.169611929714425E-4	0.0	4.169611929714425E-4	4.169611929714425E-4
probability	true + true	false	8.0	0.3598156253383979	0.3837781194383407	0.03722905735943247	0.3039031391051981	0.3837781194383407

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	false		-40.0	-40.0	0.0	-40.0	-40.0	-40.0
probability	true + true	false	8.0	-37.06666666666667	-40.0	3.9210601428284875	-40.0	-32.0	-37.1

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	false		-4.0	0.0	7214.7ms	262.3ms	2387744
probability	true + true	false	8.0	-32.0	0.0	7859.7ms	335.7ms	1083358

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Navigation 10	2378944	-37.1	-40	4.028775804	-40	-29

A.5.6 Prop DBN

Instance 1

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	false		0.013138562614608527	0.013138562614608527	0.0	0.013138562614608527	0.013138562614608527
reward	true + true	true		0.013138562614608527	0.013138562614608527	0.0	0.013138562614608527	0.013138562614608527
reward	false + true	false		0.013138562614608527	0.013138562614608527	0.0	0.013138562614608527	0.013138562614608527
reward	false + true	true		0.013138562614608527	0.013138562614608527	0.0	0.013138562614608527	0.013138562614608527
probability	true + true	false	0.25	0.013138562614608527	0.013138562614608527	0.0	0.013138562614608527	0.013138562614608527
probability	false + true	false	0.25	0.013138562614608527	0.013138562614608527	0.0	0.013138562614608527	0.013138562614608527
probability	false + true	true	0.25	0.013138562614608527	0.013138562614608527	0.0	0.013138562614608527	0.013138562614608527
search	true + true	false	0.25	0.013138562614608527	0.013138562614608527	0.0	0.013138562614608527	0.013138562614608527

search	false + true	false	0.25	0.013138562614608527	0.013138562614608527	0.0	0.013138562614608527	0.013138562614608527
probability	false + true	false	0.125	0.013138562614608527	0.013138562614608527	0.0	0.013138562614608527	0.013138562614608527

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	false		5.444506670854235	5.682113578989544	1.611348569645841	1.2483759785904076	7.784233454094309	1.3
reward	true + true	true		5.074510235545573	5.512533772760831	1.8893683315587637	-0.38742048900000015	7.5554655295446995	-
reward	false + true	false		4.876005261593708	4.820057240895148	1.5323668602841962	0.7045973939776325	7.5554655295446995	-
reward	false + true	true		4.707193201355985	5.302105137530716	2.1221170508916223	-0.13804005089502444	7.5554655295446995	-
probability	true + true	false	0.25	5.822035859588707	6.301540521466206	1.8696248270119884	0.0	7.784233454094309	-
probability	false + true	false	0.25	5.155046509102228	5.112556884912078	1.5116108336894651	1.1254041106560086	7.598931435209125	-
probability	false + true	true	0.25	5.289433288957039	5.528232711101067	1.455829028359958	1.4703877909333438	7.784233454094309	-
search	true + true	false	0.25	3.4857867347314517	3.742506383627403	2.6535679327616513	0.0	6.458134171671002	-
search	false + true	false	0.25	5.160745962519301	5.478310955094308	2.28419040392423	0.0	7.193743454094308	-
probability	false + true	false	0.125	5.444506670854235	5.682113578989544	1.611348569645841	1.2483759785904076	7.784233454094309	-

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	false		7.905810108684878	0.0	27.8ms	5.8ms	3596377
reward	true + true	true		7.905810108684878	0.0	28.2ms	5.8ms	3593424
reward	false + true	false		7.905810108684878	0.0	31.8ms	3.7ms	3596541
reward	false + true	true		7.905810108684878	0.0	30.4ms	3.6ms	3593185
probability	true + true	false	0.25	7.905810108684878	0.0	30.2ms	6.0ms	-1
probability	false + true	false	0.25	7.905810108684878	0.0	28.2ms	3.6ms	3556013
probability	false + true	true	0.25	7.905810108684878	0.0	28.5ms	3.6ms	554130
search	true + true	false	0.25	7.905810108684878	0.0	46.7ms	9.2ms	-1
search	false + true	false	0.25	7.905810108684878	0.0	42.3ms	4.7ms	-1
probability	false + true	false	0.125	7.905810108684878	0.0	30.4ms	3.6ms	3549796

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Prop	3597202	5.44197083	5.693189668	1.240983065	3.18537587	7.305936554

Instance 2

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	false + true	true		1.572E-16	1.572E-16	0.0	1.572E-16	1.572E-16
reward	true + true	true		8.29E-18	8.29E-18	0.0	8.29E-18	8.29E-18
reward	true + true	true		8.29E-18	8.29E-18	0.0	8.29E-18	8.29E-18
reward	false + true	true		1.572E-16	1.572E-16	0.0	1.572E-16	1.572E-16
reward	true + true	true		8.29E-18	8.29E-18	0.0	8.29E-18	8.29E-18
reward	true + true	true		8.29E-18	8.29E-18	0.0	8.29E-18	8.29E-18
reward	true + true	true		8.29E-18	8.29E-18	0.0	8.29E-18	8.29E-18
reward	true + true	true		8.29E-18	8.29E-18	0.0	8.29E-18	8.29E-18
reward	false + true	true		1.572E-16	1.572E-16	0.0	1.572E-16	1.572E-16
reward	false + true	true		1.572E-16	1.572E-16	0.0	1.572E-16	1.572E-16
reward	false + true	true		1.572E-16	1.572E-16	0.0	1.572E-16	1.572E-16
reward	true + true	true		8.29E-18	8.29E-18	0.0	8.29E-18	8.29E-18
probability	true + true	false	0.25	2.76587798348395E-6	2.76587798348395E-6	0.0	2.76587798348395E-6	2.76587798348395E-6
probability	false + true	false	0.25	1.67263011131282E-6	1.62130689412719E-6	2.8110884013136E-7	1.62130689412719E-6	3.16100340969594E-6
probability	true + true	true	0.25	2.76587798348395E-6	2.76587798348395E-6	0.0	2.76587798348395E-6	2.76587798348395E-6
probability	false + true	true	0.25	1.87792298005532E-6	1.62130689412719E-6	5.8362045886004E-7	1.62130689412719E-6	3.16100340969594E-6
reward	true + true	false		8.29E-18	8.29E-18	0.0	8.29E-18	8.29E-18
reward	false + true	false		1.572E-16	1.572E-16	0.0	1.572E-16	1.572E-16
reward	true + true	false		8.29E-18	8.29E-18	0.0	8.29E-18	8.29E-18
reward	false + true	false		1.572E-16	1.572E-16	0.0	1.572E-16	1.572E-16

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	false + true	true		9.067621077873088	9.042742343046637	1.2813411648897732	6.478485660718657	11.63260584578822	6.5
reward	true + true	true		8.612570045222773	8.520312985539192	1.3177556413283003	5.785003504100652	11.171890813065476	-
reward	true + true	true		8.10138215662621	8.185373487990367	1.49499122032339	4.287419414992559	11.179349100095962	-
reward	false + true	true		8.911433385601901	8.874468077248158	1.1759697658173316	7.321225572420344	11.599221965543594	-
reward	true + true	true		7.9591249341063595	7.866433246958976	1.3994284892062376	5.876227171582244	11.161133171463254	-
reward	true + true	true		8.201160018052759	8.134350511772197	1.3052819809853335	5.334106159264298	10.852651564128537	-
reward	true + true	true		7.924722701773006	7.947786597015038	1.2514372072814748	5.755312413808558	10.607641379098963	-
reward	true + true	true		7.898249730267034	7.6779175789168415	1.2771389602348486	5.742650088934332	10.622295499981517	-
reward	false + true	true		8.956060779381588	8.688010132831353	1.264863346839125	6.431076334780408	13.293773857060811	-
reward	false + true	true		9.275487640100152	9.442926901975017	1.487520591795349	6.263250999101758	12.371639838678762	-
reward	false + true	true		9.03880698975411	9.190097865879428	1.3078614869617267	6.234692717924255	11.67962501044606	-
reward	true + true	true		8.038791754991887	8.186822181013248	1.305879420483443	4.259710950289558	10.408569231339884	-
probability	true + true	false	0.25	8.823963507627038	8.956323820354111	1.21463481720798	6.344131169682049	11.487584316540213	-
probability	false + true	false	0.25	7.775564450009903	7.449160350596753	1.5160841074592568	4.9882008684048	11.471798457083203	-

probability	true + true	true	0.25	10.597890372736538	10.72033716277329	1.4081595150238555	7.895719103377343	14.025719022671472	-
probability	false + true	true	0.25	9.371272293148499	9.529270788497483	1.5132825225123065	6.487585851795108	12.604874089339276	-
reward	true + true	false		7.802990362875172	7.870189826658334	1.1594217623559373	5.379234378663185	10.101293402079495	-
reward	false + true	false		8.871621342776352	8.604995972247874	1.2857979648080606	6.331857149707957	11.271346265283805	-
reward	true + true	false		7.84731410477811	7.435187486326667	1.6237500479728002	4.9872459155742765	11.631646545903116	-
reward	false + true	false		8.925798458477598	8.906698739829727	1.4928420249347458	6.239200300454743	12.27280773168296	-

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	false + true	true		17.811620217369757	0.0	30.1ms	3.6ms	3591256
reward	true + true	true		17.811620217369757	0.0	27.1ms	5.9ms	3591400
reward	true + true	true		17.811620217369757	0.0	27.3ms	5.9ms	3591285
reward	false + true	true		17.811620217369757	0.0	29.7ms	3.5ms	3591008
reward	true + true	true		17.811620217369757	0.0	27.3ms	5.8ms	3591294
reward	true + true	true		17.811620217369757	0.0	27.1ms	5.8ms	3591308
reward	true + true	true		17.811620217369757	0.0	27.7ms	5.8ms	3591178
reward	true + true	true		17.811620217369757	0.0	25.8ms	5.9ms	3591328
reward	false + true	true		17.811620217369757	0.0	29.9ms	3.6ms	3591198
reward	false + true	true		17.811620217369757	0.0	29.9ms	3.5ms	3591502
reward	false + true	true		17.811620217369757	0.0	29.7ms	3.5ms	3591462
reward	true + true	true		17.811620217369757	0.0	27.4ms	5.8ms	3591502
probability	true + true	false	0.25	7.0085222405820495	0.0	27.3ms	5.7ms	3564416
probability	false + true	false	0.25	4.757941102444115	0.3060086526719448	28.3ms	3.6ms	3572740
probability	true + true	true	0.25	7.0085222405820495	0.0	25.3ms	5.9ms	3505535
probability	false + true	true	0.25	4.981418224924366	0.6353158850134526	28.1ms	3.5ms	3476732
reward	true + true	false		17.811620217369757	0.0	28.4ms	5.9ms	3596579
reward	false + true	false		17.811620217369757	0.0	31.8ms	3.7ms	3596608
reward	true + true	false		17.811620217369757	0.0	28.6ms	6.0ms	3596573
reward	false + true	false		17.811620217369757	0.0	31.6ms	3.7ms	3596608
search	true + true	false						-1
search	false + true	false						-1

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Prop 2	3560079	11.05010366	11.21440429	1.349129816	7.190650229	14.40906962

A.5.7 Skill Teaching

Instance 1

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	true		0.2625719262500008	0.2625719262500008	0.0	0.2625719262500008	0.2625719262500008
reward	false + true	true		0.1615244678431959	0.15774537202520178	0.015291242723081569	0.15774537202520178	0.23421290673440887
probability	true + true	false	0.25	0.7900833	0.7900833	0.0	0.7900833	0.7900833
probability	true + true	true	0.25	0.7900833	0.7900833	0.0	0.7900833	0.7900833
probability	false + true	false	0.25	0.7739098520692193	0.7752540931337643	0.01618575545007361	0.71801746	0.7900833
probability	false + true	true	0.25	0.7748291260492624	0.7752540931337643	0.01164167660434826	0.71801746	0.7900833
probability	true + true	false	0.15	0.7900833	0.7900833	0.0	0.7900833	0.7900833
probability	true + true	false	0.35	0.7900833	0.7900833	0.0	0.7900833	0.7900833
probability	true + true	false	0.45	0.7900833	0.7900833	0.0	0.7900833	0.7900833
probability	true + true	false	0.55	0.5556961764162444	0.5566459748065088	0.003150124886990743	0.5461981925135998	0.5566459748065088
search	true + true	false	0.45	0.7900833	0.7900833	0.0	0.7900833	0.7900833
search	true + true	false	0.25	0.7900833	0.7900833	0.0	0.7900833	0.7900833

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	true		59.76652207333335	63.61735750000003	13.874946056409282	24.010835199999992	72.25962120000003	-
reward	false + true	true		57.93406321000002	59.41704680000002	13.653911457659675	0.7949045999999953	72.48673680000003	-
probability	true + true	false	0.25	9.318284043333334	17.511643	18.737202404672253	-31.872721000000002	18.6326943	8.15
probability	true + true	true	0.25	16.89288578	17.511643	4.772077175652397	-6.726307800000006	26.593611199999994	-
probability	false + true	false	0.25	9.033509623333334	17.5684219	15.282806616412959	-28.1688937	22.265216	8.4
probability	false + true	true	0.25	17.245750616666665	17.5684219	9.474187665635304	-28.2824515	25.415780999999992	-
probability	true + true	false	0.15	3.848635626666667	10.1039884	10.55066374984261	-13.3535845	10.1039884	6.2
probability	true + true	false	0.35	16.02632682	24.9192976	16.4865707177217	-13.3535845	26.0403489	17.9
probability	true + true	false	0.45	18.18383382	32.3837311	22.99755985257701	-47.9226393	32.3837311	16.5
probability	true + true	false	0.55	14.451589145454548	22.733973900000001	20.824747553136895	-20.4773446	39.2235967	14.4
search	true + true	false	0.45	23.383374642307693	33.4480035	19.067411709313827	-12.2325332	34.7961704	24.9
search	true + true	false	0.25	11.636136106666667	17.511643	12.544006820967656	-16.716738400000004	23.627909600000002	11.5

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	true		74.6720605	0.0	49.4ms	29.3ms	3583027

reward	false + true	true		74.8991761	0.029821739282795603	422.2ms	225.3ms	3505619
probability	true + true	false	0.25	18.783620476666666	0.2046750283768304	50.6ms	29.1ms	3574111
probability	true + true	true	0.25	18.783620476666666	0.2046750283768304	50.1ms	29.4ms	3569835
probability	false + true	false	0.25	19.07121171	0.889573826810795	431.1ms	233.9ms	3325187
probability	false + true	true	0.25	19.243881	2.449622891601054	421.6ms	228.2ms	3290015
probability	true + true	false	0.15	11.3385975	0.0	49.1ms	29.0ms	3574829
probability	true + true	false	0.35	26.191275076666667	0.2046750283768304	48.8ms	28.8ms	3575546
probability	true + true	false	0.45	33.65760120666667	0.21504138981694043	50.4ms	29.7ms	3568920
probability	true + true	false	0.55	42.2113865	2.0833526483054334	67.4ms	33.9ms	-1
search	true + true	false	0.45	34.78195013846154	0.3512974244039305	49.0ms	30.9ms	-1
search	true + true	false	0.25	19.043790706666666	1.212696215937316	47.5ms	29.8ms	116723

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Skill 1	3532726	68.13017347	72.4867368	5.337630408	58.3527744	72.4867368

Instance 2

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	true		0.10603091702089174	0.1034076865279631	0.015125286682626391	0.10043643503088072	0.18605816441374595
probability	true + true	false	0.25	0.700932132	0.6979286	0.016451022285886067	0.6979286	0.78803456
probability	true + true	true	0.25	0.703935664	0.6979286	0.022860614766856812	0.6979286	0.78803456
search	true + true	false	0.25	0.78803456	0.78803456	0.0	0.78803456	0.78803456

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	true		63.94831326000002	71.65335540000001	23.379000631028603	-11.587631800000004	85.95873300000005	-
probability	true + true	false	0.25	3.222652143333327	21.120819999999988	33.046992879567114	-55.67293639999999	21.732079100000007	-0.8
probability	true + true	true	0.25	18.84197989999999	21.120819999999988	12.869060261457507	-45.718190199999995	35.7212882	-
search	true + true	false	0.25	1.9197008433333331	23.13310780000001	30.413446266460134	-40.36701020000002	23.13310780000001	10.8

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	true		88.82964487333334	0.02139053851684533	49.7ms	28.9ms	3580911
probability	true + true	false	0.25	22.5640045	0.11544841294838142	49.8ms	29.0ms	3574000
probability	true + true	true	0.25	22.583677206666668	0.1551801591578483	49.6ms	29.0ms	3515190
search	true + true	false	0.25	24.529591343333333	0.25579167423404603	49.5ms	29.6ms	1437801

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Skill 2	3450090	77.53073686	77.4682492	9.191569418	60.3186584	86.0008888

Instance 3

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	true		0.005489888551797994	1.0872636315233462E-4	0.011362898426193115	1.0872636315233462E-4	0.03655883676510922
probability	true + true	false	0.25	0.2279681759671489	0.22040984079897502	0.01511667033634781	0.22040984079897502	0.25064318147167064

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	true		52.25025824666667	84.91383800000001	103.3308295263134	-235.99939480000006	183.52541600000004	-
probability	true + true	false	0.25	-110.42438025	-116.23543559999997	85.23249767609903	-201.71629570000005	-7.510354100000001	-144.7

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	true		190.91600325333334	3.934642470227925	86.1ms	86.3ms	3072946
probability	true + true	false	0.25	48.2940856	0.4035072	127.1ms	112.4ms	-1

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Skill 3	3325705	92.8994725	111.9004249	72.38114772	-46.0134138	182.262353

A.5.8 Triangle Tireworld

Instance 1

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	true		0.16	0.16	0.0	0.16	0.16
probability	true + true	false	0.25	0.23733333333333334	0.24	0.01460593486680443	0.16	0.24
probability	true + true	true	0.25	0.24	0.24	0.0	0.24	0.24
probability	false + true	false	0.25	0.24	0.24	0.0	0.24	0.24
probability	false + true	true	0.25	0.23466666666666666	0.24	0.020296650536196996	0.16	0.24
search	true + true	false	0.25	0.24	0.24	0.0	0.24	0.24
search	true + true	true	0.25	0.24	0.24	0.0	0.24	0.24
search	false + true	false	0.25	0.24	0.24	0.0	0.24	0.24
search (fixed)	true + true	true	0.25	0.24	0.24	0.0	0.24	0.24

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	true		15.2	-40.0	68.7616573188967	-40.0	98.0	-
probability	true + true	false	0.25	-1.9666666666666666	-40.0	50.9512839223246	-40.0	81.0	-15.5
probability	true + true	true	0.25	11.633333333333333	11.0	52.626519690291545	-40.0	81.0	-
probability	false + true	false	0.25	-5.333333333333333	-40.0	58.64759355578929	-40.0	95.0	-8.4
probability	false + true	true	0.25	4.333333333333333	-40.0	63.82915415028018	-40.0	95.0	-
search	true + true	false	0.25	13.533333333333333	-40.0	62.32657778869761	-40.0	88.0	-10.3
search	true + true	true	0.25	14.3	-40.0	63.19272655658381	-40.0	95.0	-
search	false + true	false	0.25	14.482758620689655	-40.0	61.88677489095664	-40.0	92.0	-10.5
search (fixed)	true + true	true	0.25	-14.566666666666666	-40.0	51.96927695404696	-40.0	92.0	-

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	true		98.0	0.0	83.7ms	4.0ms	3573750
probability	true + true	false	0.25	63.3	4.9490507623458315	83.2ms	4.1ms	3538771
probability	true + true	true	0.25	63.7	5.26635383910165	82.9ms	4.1ms	3533070
probability	false + true	false	0.25	91.46666666666667	8.629273886955492	1573.3ms	13.1ms	3248014
probability	false + true	true	0.25	91.0	8.948126369092101	1576.9ms	15.2ms	3201356
search	true + true	false	0.25	83.56666666666666	5.144017840193586	84.0ms	4.1ms	2401832
search	true + true	true	0.25	83.73333333333333	5.9592485833501625	103.3ms	4.0ms	304632
search	false + true	false	0.25	82.82758620689656	9.215937149142555	1643.4ms	1.8ms	-55881
search (fixed)	true + true	true	0.25	87.3	11.10808962749283	80.3ms	4.3ms	2318941

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Triangle 1	3589931	93.73333333	94.5	2.14850924	90	96

Instance 2

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	true		0.24903426666666667	0.249001	1.8220903746337611E-4	0.249001	0.249999
probability	true + true	false	0.25	0.249999	0.249999	0.0	0.249999	0.249999
probability	true + true	true	0.25	0.24993246666666666	0.249999	2.532007154390564E-4	0.249001	0.249999
probability	false + true	false	0.25	0.24993246666666666	0.249999	2.532007154390564E-4	0.249001	0.249999
probability	false + true	true	0.25	0.2498992	0.249999	3.045183194761031E-4	0.249001	0.249999
search	true + true	false	0.25	0.249999	0.249999	0.0	0.249999	0.249999
search	true + true	true	0.25	0.249999	0.249999	0.0	0.249999	0.249999
search	false + true	false	0.25	0.249999	0.249999	0.0	0.249999	0.249999
search (fixed)	true + true	true	0.25	0.249999	0.249999	0.0	0.249999	0.249999

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	true		38.2	98.0	69.55295675313486	-40.0	98.0	-
probability	true + true	false	0.25	17.4	-40.0	62.41220039880576	-40.0	83.0	-9.3
probability	true + true	true	0.25	25.533333333333335	82.0	62.349810349147916	-40.0	83.0	-
probability	false + true	false	0.25	30.1	72.0	67.00764238451488	-40.0	95.0	-6.8
probability	false + true	true	0.25	36.833333333333336	69.5	64.4654857313176	-40.0	95.0	-
search	true + true	false	0.25	-2.966666666666667	-40.0	57.5814166222913	-40.0	95.0	-9.1
search	true + true	true	0.25	23.833333333333332	72.0	60.79705263576025	-40.0	82.0	-
search	false + true	false	0.25	21.178571428571427	11.5	62.80248495403758	-40.0	90.0	-9.0
search (fixed)	true + true	true	0.25	16.533333333333335	-40.0	65.85026553677949	-40.0	92.0	-

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left

reward	true + true	true		98.0	0.0	83.7ms	3.8ms	3574304
probability	true + true	false	0.25	82.86666666666666	2.674700541837487	84.1ms	3.9ms	3538201
probability	true + true	true	0.25	82.5	2.046864717663039	83.7ms	3.8ms	3521639
probability	false + true	false	0.25	92.76666666666667	6.831384639438334	1538.6ms	41.9ms	3252712
probability	false + true	true	0.25	90.56666666666666	9.884447321452763	1596.8ms	6.2ms	3218963
search	true + true	false	0.25	83.66666666666667	4.179864156622329	83.8ms	3.8ms	2466076
search	true + true	true	0.25	81.13333333333334	3.8928168449730265	92.8ms	4.0ms	529747
search	false + true	false	0.25	83.92857142857143	10.652520381402256	1596.9ms	6.5ms	-1
search (fixed)	true + true	true	0.25	89.93333333333334	5.030481798638852	87.5ms	3.9ms	2292373

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Triangle 2	3589854	94.2	95	2.006884702	90	96

Instance 3

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	true		0.01500625	0.01500625	0.0	0.01500625	0.01500625
probability	true + true	false	0.25	0.0318644812890625	0.0318644812890625	0.0	0.0318644812890625	0.0318644812890625

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	true		-35.46666666666667	-40.0	24.83008927356753	-40.0	96.0	-
probability	true + true	false	0.25	-36.0	-40.0	20.396078054371138	-40.0	64.0	-36.8

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	true		96.0	0.0	818.0ms	4.5ms	3447211
probability	true + true	false	0.25	60.15384615384615	0.7844645405527362	830.2ms	4.3ms	-1
search	true + true	false	0.25/0.75					-1

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Triangle 3	1565291	86.53333333	88	3.919300899	78	93

Instance 4

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	true		0.04131	0.04100625	0.0016637072684219422	0.04100625	0.05011875
probability	true + true	false	0.25	0.04505625	0.04100625	0.004614293741274956	0.04100625	0.05011875

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	true		-26.4	-40.0	41.49748642159359	-40.0	96.0	-
probability	true + true	false	0.25	-24.111111111111111	-40.0	38.88378476883028	-40.0	74.0	-35.1

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	true		96.0	0.0	813.6ms	4.2ms	3450895
probability	true + true	false	0.25	68.62962962962963	8.399396261033143	821.2ms	6.0ms	-1
search	true + true	false						-1

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Triangle 4	1625469	86.96666667	88	3.819038815	78	92

Instance 5

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Probability	Median Probability	SSD Probability	Min Probability	Max Probability
reward	true + true	true		8.262E-4	7.29E-4	2.9658497648373894E-4	7.29E-4	0.001701

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward	Reward Expectation
reward	true + true	true		-40.0	-40.0	0.0	-40.0	-40.0	-

Algorithm	Interpreted Parameters	Adjust Plan	Approximation	Mean Expected Reward	SSD Expected Reward	Mean Translation Time	Mean Probability Translation Time	Time Left
reward	true + true	true		94.0	0.0	8577.1ms	9.9ms	2807851
prob	true + true	false	0.25					-1

Domain	Time Left	Mean Reward	Median Reward	SSD Reward	Min Reward	Max Reward
Triangle 5	199464	70.86666667	71	2.344962809	67	75

A.6 Rddlsim Output For The Example Run in Triangle Tireworld

```
Connection from client at address localhost / 127.0.0.1
Client name: SMT − search
Instance requested: triangle_tireworld_inst_mdp__1
Round 1 / 1, time remaining: 3599311
[ Memory usage: 83,28Mb / 243,79Mb = 0,34 ]
** Actions received: [loadtire(la3a1);]
** Actions received: [loadtire(la1a2);]
** Actions received: [move−car(la2a1, la3a1);]
** Actions received: [move−car(la2a1, la3a1);]
** Actions received: [move−car(la3a1, la2a1);]
** Actions received: [move−car(la1a2, la3a1);]
** Actions received: [loadtire(la1a2);]
** Actions received: []
** Actions received: [move−car(la1a1, la1a2);]
** Actions received: []
** Actions received: [move−car(la1a2, la1a1);]
** Actions received: [move−car(la1a1, la2a1);]
** Actions received: [move−car(la2a2, la2a2);]
** Actions received: [move−car(la1a1, la1a3);]
** Actions received: [loadtire(la2a1);]
** Actions received: []
** Actions received: [loadtire(la3a1);]
** Actions received: [move−car(la2a2, la2a1);]
** Actions received: []
** Actions received: [loadtire(la1a3);]
** Actions received: [loadtire(la1a3);]
** Actions received: [move−car(la3a1, la2a2);]
** Actions received: [loadtire(la3a1);]
** Actions received: [loadtire(la1a3);]
** Actions received: [move−car(la2a2, la3a1);]
** Actions received: [loadtire(la2a1);]
** Actions received: [move−car(la3a1, la1a1);]
** Actions received: []
** Actions received: [move−car(la2a2, la1a2);]
** Actions received: [move−car(la1a2, la1a3);]
** Actions received: [loadtire(la2a2);]
```

```
** Actions received: [loadtire(la1a1);]
** Actions received: [move−car(la1a3, la2a1);]
** Actions received: [move−car(la3a1, la2a2);]
** Actions received: [move−car(la1a3, la1a3);]
** Actions received: [move−car(la2a1, la1a1);]
** Actions received: [move−car(la2a1, la2a2);]
** Actions received: [move−car(la3a1, la2a1);]
** Actions received: [move−car(la3a1, la1a2);]
** Actions received: [loadtire(la2a2);]
** Round reward: −40.0
Session finished successfully: SMT − search
Time left: 3561280
Number of simulations: 0
Number of runs: 1
Accumulated reward: −40.0
Average reward: −40.0
```

A.7 Code

Relevant parts of the algorithm implementation are shown in this section. Code parts that were not essential for the understanding of the implementation, for example self-explanatory object initializations or logging features, or simple code parts that can be replaced by a comment because they are not relevant for the overall functionality, were left out and are pointed out using "...". Some comments were added for better understanding.

The communication with RDDLsim, the extraction of actions from a Z3 model, the main file, storage classes, concurrency constraints and other code that is less problem-specific or easy to reproduce was left out. It is not required to understand the implementation of the algorithm and can either mostly be looked up in, for example, Z3's documentation or can be implemented intuitively.

A.7.1 Algorithm - Reward Approximation

```
//Returns false if UNSAT
private Boolean fulfillGoal(double approximation, Boolean firstTime) {
  if (opt.Check() == Status.SATISFIABLE) { //opt: Z3's optimization procedure
    Model model = opt.getModel();
    ...
    BoolExpr goalExpr = context.mkGe(context.mkAdd(allRewards.toArray(new ArithExpr[allRewards.size()])),
        context.mkReal(getReward(model, approximation)));
    //Add the reward approximation constraint to the search algorithm and probability optimizer
    solver.add(goalExpr);
    optProb.Add(goalExpr);
    return true;
  }
  else return false;
}
```

```
...
private String getReward(Model model, double approximation) {
  ...
  //Get the (optimal) reward value from the model
  Expr val = model.eval(context.mkAdd(allRewards.toArray(new ArithExpr[allRewards.size()])), false);
  //Calculate the minimum value using the approximation value
  Double approxReward = Double.parseDouble(((RatNum) val).toDecimalString(20)) * approximation;
  return "" + approxReward;
}
```

A.7.2 Weights, Frequencies of Occurrence, Weighted Sum

```
//Weights g_i, sorted according to the probabilities given in sortedProbabilityKeys
private void calculateWeights() {
  if (sortedProbabilityKeys.size() > 0) {
    Double maxProb = sortedProbabilityKeys.get(0);
    sortedWeights.add(1.0);

    for (int i = 1; i < sortedProbabilityKeys.size(); ++i) {
      sortedWeights.add(Math.log(sortedProbabilityKeys.get(i))/Math.log(maxProb));
    }
  }
}

//x_i: Sum of all satisfied action conditions with likelihood p_i
private void calculateX_i() {
  for (int i = 0; i < sortedProbabilityKeys.size(); ++i) {
    Double p_i = sortedProbabilityKeys.get(i);
    //Use sumActions to count the frequencies of occurrence (1 if true, 0 if false for each constraint)
    ArrayList<ArithExpr> sumActions = new ArrayList<>();
    //Get all translated Bernoulli conditions associated with the probability p_i
    ArrayList<BoolExpr> actions = contextVars.getBernoulliExpr(p_i);
    for (BoolExpr action : actions) {
      sumActions.add((ArithExpr) context.mkITE(action, context.mkInt(1), context.mkInt(0)));
    }
    ArithExpr addExpr = context.mkAdd(sumActions.toArray(new ArithExpr[sumActions.size()]));

    IntExpr x = context.mkIntConst("_b_" + i);
    solver.add(context.mkEq(x, addExpr));
    optProb.Add(context.mkEq(x, addExpr));
    x_i.put(p_i, x);
  }
}

//Uses x_i and g_i
private void calculateWeightedSum() {
  ArrayList<ArithExpr> weightedExpr = new ArrayList<>();
  for (int i = 0; i < sortedProbabilityKeys.size(); ++i) {
    weightedExpr.add(context.mkMul(x_i.get(sortedProbabilityKeys.get(i)), context.mkReal("" + sortedWeights.
        get(i))));
```

```
38      }
39
40      weightedSum = context.mkAdd(weightedExpr.toArray(new ArithExpr[weightedExpr.size()]));
41  }
```

A.7.3 Search Constraints

```
1   //Search constraints for the search algorithm
2   //Here, x_1 is s in the presented algorithm
3   private void addSearchConstraints(int x_1, Solver solverClone) {
4     //Sum less than or equal to x_1
5     BoolExpr e1 = context.mkLe(weightedSum, context.mkReal(x_1));
6     solverClone.add(e1);
7
8     //Sum greater than x_1 − 1
9     if (x_1 > 0) {
10      BoolExpr e2 = context.mkGt(weightedSum, context.mkInt(x_1 − 1));
11      solverClone.add(e2);
12    }
13
14    //Third implicit condition
15    for (int i = 0; i < sortedProbabilityKeys.size(); ++i) {
16      BoolExpr e3 = context.mkLe(context.mkMul(x_i.get(sortedProbabilityKeys.get(i)), context.mkReal("" +
              sortedWeights.get(i))), context.mkInt(x_1));
17      solverClone.add(e3);
18    }
19
20    //x_i >= 0
21    for (Double prob : sortedProbabilityKeys) {
22      solverClone.add(context.mkGe(x_i.get(prob), context.mkInt(0)));
23    }
24  }
25
26  //As described in the thesis: Add these constraints to search for a set of possible solutions to speed up the search
27  private void addSearchConstraints(Optimize optimize) {
28    //Sum less than or equal to x_1
29    optimize.MkMinimize(weightedSum);
30
31    //x_i >= 0
32    int maxSize = (new Double(Math.ceil(var_amount ∗ Math.log(sortedProbabilityKeys.get(sortedProbabilityKeys.
          size() − 1))/Math.log(sortedProbabilityKeys.get(0))))).intValue();
33    for (int i = 0; i < sortedProbabilityKeys.size(); ++i) {
34      BoolExpr e3 = context.mkLe(context.mkMul(x_i.get(sortedProbabilityKeys.get(i)), context.mkReal("" +
              sortedWeights.get(i))), context.mkInt("" + maxSize));
35      optimize.Add(e3);
36    }
37
38    for (Double prob : sortedProbabilityKeys) {
39      optimize.Add(context.mkGe(x_i.get(prob), context.mkInt(0)));
40      optimize.Add(context.mkLe(x_i.get(prob), context.mkInt("" + maxSize)));
```

```
41    }
42  }
```

A.7.4 Algorithm 1

```
1   //Still called binSearch because it was once a binary search
2   private int binSearchMinSATAmount() {
3     //Start with s = 0
4     int current_x_1 = 0;
5     Solver solverClone = solver.translate(context);
6
7     addSearchConstraints(current_x_1, solverClone);
8     ...
9     Status q = solverClone.check();
10
11    //Terminate when x_n > horizon * amount-of-state-variables (var_amount) or when SATISFIABLE
12    while (q != Status.SATISFIABLE && current_x_1 <= (new Double(Math.ceil(var_amount * Math.log(
            sortedProbabilityKeys.get(sortedProbabilityKeys.size() - 1))/Math.log(sortedProbabilityKeys.get(0))))).
            intValue()) {
13      current_x_1++;
14      solverClone = solver.translate(context);
15
16      addSearchConstraints(current_x_1, solverClone);
17      ...
18      q = solverClone.check();
19    }
20
21    ...
22    return current_x_1;
23  }
```

A.7.5 Algorithm 2

```
1   //minAmount is s in the presented algorithm
2   private ArrayList<Integer> findBestModel(int minAmount) {
3     ...
4     ArrayList<ArrayList<Integer>> nextAmounts = new ArrayList<>();
5     ...
6     if (nextAmounts != null) {
7       ...
8       while (q != Status.SATISFIABLE && nextAmounts != null) {
9         ...
10        //Get the most probable set of values that has not been tried out yet for x_i with this function (within the
              range given by s)
11        nextAmounts = getNextMostProbableAmountList(minAmount, relevantIndices, nextAmounts.get(0));
12
13        if (nextAmounts != null) {
14          System.out.println("Now checking " + calculateProbability(nextAmounts.get(0)) + " - " + nextAmounts.
                toString());
```

```
            //Check if any set of values for x_i (with the same probability) is satisfiable
            for (ArrayList<Integer> amountList : nextAmounts) {
              solverClone = solver.translate(context);
              solverClone.add(allowAmountOfActionsTaken(amountList));

              q = solverClone.check();

              if (q == Status.SATISFIABLE) {
                ...
                return amountList;
              }
            }
          }
        }
    }

    ...
    return null;
}
```

A.7.6 Finding a Model Depending on the Chosen Algorithm

```
public Model maximize(Boolean firstTime) {
    //In case no probabilities are part of the domain
    //1.0 − probabilities have been set to true and removed before
    if (sortedProbabilityKeys.size() <= 0) {
        //Use the reward optimizer
        ...
    }

    calculateWeights();
    calculateX_i();
    calculateWeightedSum();

    //opt: reward optimizer
    //optProb: probability optimizer
    //solver: search algorithm
    if (algorithm == Main.Algorithm.rewardOptimizer) {
        ...
        if (opt.Check() == Status.SATISFIABLE) {
            Model model = opt.getModel();
            ...
            return model;
        }
        else {
            ...
            return null;
        }
    }
    else {
```

```
29      //First, find out what states are supposed to be goal states using "fulfillGoal"−constraints
30      //States with a reward of rewardApproximation∗optimalReward are considered to be goal states
31      ...
32      Boolean satisfiable = fulfillGoal(rewardApproximation, firstTime);
33
34      if (satisfiable) {
35        if (algorithm == Main.Algorithm.probabilityOptimizer) {
36          ...
37          addSearchConstraints(optProb);
38
39          Status s = optProb.Check();
40          if (s == Status.SATISFIABLE) {
41            Model model = optProb.getModel();
42            ...
43            return model;
44          }
45          else {
46            ...
47            return null;
48          }
49        }
50        else {
51          //Use the search algorithm
52          ...
53          int minAmount = binSearchMinSATAmount();
54
55          ArrayList<Integer> modelList = findBestModel(minAmount);
56          if (modelList != null) {
57            solver.add(allowAmountOfActionsTaken(modelList));
58            solver.check();
59            Model model = solver.getModel();
60            ...
61            return model;
62          }
63
64          ...
65          return null;
66        }
67      }
68      else {
69        ...
70        return null;
71      }
72    }
73  }
```

A.7.7 Bernoulli Statement Translation - Placeholder Variable for Constraint Translation

```
//Save the Bernoulli statement and its context for later use − at this point, return a placeholder variable for the constraint that the Bernoulli statement is part of. This variable will be part of another constraint that can be used to control, connected to the probabilistic values, whether the placeholder variable becomes true or false. This constraint can also be ignored − the reward optimizer, for example, does not use the search constraints presented before.
public BoolExpr registerBernoulliStatement(Bernoulli bExpr, FuncDecl stateVar, ArrayList<Expr> stateParams, ArrayList<String> stateParamNames, ArrayList<ArrayList<Pair<String, Expr>>> quantifiedParams, Boolean isPrimed, BoolExpr conditionExpression, String conditionRDDLExpr) {
  //Bit blast was used before the word was replaced by the less misleading phrase "using interpreted parameters"
  if (doBitBlast) {
    //Create the placeholder variable
    String name = "_Bernoulli_" ... + name_addition;
    ++name_addition;
    BoolExpr newBernoulli = context.mkBoolConst(name);
    //Translate the Bernoulli expression for later use
    Expr valExpr = ConstraintTranslator.translateExpression(...);
    //Save the statement to create constraints for the search constraints later
    BernoulliStatements.add(new BernoulliStatement...));

    return newBernoulli;
  }
  else {
    //Similar behavior
    ...
  }
}
```

A.7.8 Bernoulli Statement Translation - Translation of Statements (Interpreted Case)

```
public void translateProbabilisticStatements(...) {
  ...
  for (BernoulliStatement BernoulliStatement : BernoulliStatements) {
    ...
    //Distinction between dynamic and static probability expression
    if (containsNoDynamicValues(BernoulliStatement.bExpr)) {
      ...
      //Get all possible probability values of the Bernoulli statement
      ArrayList<BigDecimal> probabilityValues = getProbabilityValues(...);

      for (BigDecimal val : probabilityValues) {
        //As described before: Values cannot be checked for equality, so use >= and <= with small epsilon
        Double value = val.doubleValue();
        Double valueLess = value − 0.00000000001;
        Double valueMore = value + 0.00000000001;
        //1.0 expressions − placeholder is always true
        if (value == 1.0) {
          ... .addBernoulli(BernoulliStatement.newBernoulli, (ArithExpr) BernoulliStatement.valExpr, 1.0);
        }
        //0.0 expressions − placeholder is always false
```

```
21            else if (value == 0.0) {
22              ... .addBernoulli(context.mkNot(BernoulliStatement.newBernoulli), (ArithExpr) BernoulliStatement.
                      valExpr, 1.0);
23            }
24            //Other expressions: Add p with true placeholder and (1 − p) with false placeholder
25            else {
26              //Statements for the weighted sum, used to count statements connected to their probability
27              BoolExpr statementTrue = context.mkAnd(BernoulliStatement.newBernoulli, context.mkGe((ArithExpr
                      ) BernoulliStatement.valExpr, context.mkReal("" + valueLess)), context.mkLe((ArithExpr)
                      BernoulliStatement.valExpr, context.mkReal("" + valueMore)));
28              BoolExpr statementFalse = context.mkAnd(context.mkNot(BernoulliStatement.newBernoulli), ...);
29              //Statements should only be counted if they are used − if they are only used under a certain condition,
                      that condition expression needs to be taken into account
30              if (BernoulliStatement.conditionExpression != null) {
31                statementTrue = context.mkAnd(BernoulliStatement.conditionExpression, BernoulliStatement.
                        newBernoulli, ...);
32                statementFalse = ...;
33              }
34              //Add the Bernoulli statements to an array depending on their probability (to later count x_i)
35              ... .addBernoulli(statementTrue, (ArithExpr) BernoulliStatement.valExpr, value);
36              ... .addBernoulli(statementFalse, (ArithExpr) BernoulliStatement.valExpr, (1.0 − value));
37            }
38          }
39        }
40        else {
41          ArrayList<BigDecimal> probabilityValues = getProbabilityValues(...);
42
43          for (BigDecimal _probValue : probabilityValues) {
44            //Similar behavior, only add valid probabilities (values in [0, 1] with epsilon−margin to catch at least
                    some rounding errors)
45            ...
46          }
47        }
48      ...
49      }
50    }
```

A.7.9 Bernoulli Statement Translation - Translation of Statements (Uninterpreted Case)

```
1   //The uninterpreted version does not regard dynamic probabilities
2   public BoolExpr createBernoulliConstraint(...) {
3     ...
4     //If any param is an uninterpreted variable, all possible interpretations are considered
5     //Furthermore, if e.g. state(_x_1, p) and state(_x_2, _x_3) are defined, then for _x_1=_x_2 and _x_3 = p the
            Bernoulli value must only count once, not twice, as both translations describe the same function for that
            parameter interpretation
6     //−> If the same function is translated with different parameters, consider this special case
7     if (...) {
8       ...
```

```
9       //Go through former translations of the same function
10      for(Expr func : ...) {
11        //Go through all function parameters and add a constraint that checks if they are equal (which will then be
              checked later by the solver, with concrete interpretations)
12        ArrayList<BoolExpr> sameParams = ...
13        //If the function has parameters, store the parameter−equal expression in parameters and the constraint to
              use the same function value as the one that has been translated before in useSameFunctionValue
14        if (sameParams.size() > 0) {
15          params.add(context.mkAnd(sameParams.toArray(new BoolExpr[sameParams.size()])));
16          useSameFunctionValue.add(context.mkAnd(context.mkAnd(sameParams.toArray(new BoolExpr[
                sameParams.size()])), (BoolExpr) func));
17        }
18      }
19      //Two final constraints: allParams [true if another former translated function with the same parameter
            interpretation exists] and otherFuncValue [the value of that function, if it exists]
20      ...
21        allParams = context.mkOr(params.toArray(new BoolExpr[params.size()]));
22      ...
23        otherFuncValue = context.mkOr(useSameFunctionValue.toArray(new BoolExpr[useSameFunctionValue.
              size()]));
24    }
25    //Add the current function to the set of former translated functions
26    ...
27
28    //Get all parameter interpretations for the uninterpreted parameters of the function
29    ...
30    //Store the set of possible parameters for the function
31    ...
32    //Go through all possible parameters, store the probability values which are calculated with the same function as
          in the interpreted case
33    for (ArrayList<Expr> stateParam : stateParams) {
34      ...
35    }
36    //Go through all probability values, similar to the interpreted case
37    for (Double probability : probabilityValues) {
38      //Create another placeholder variable for each probability value
39      BoolExpr currentBernoulli = context.mkBoolConst("_Bernoulli_" + name_addition);
40      ++name_addition;
41      //Store the placeholder variable in the context of its probability values for later use in this function
42      BernoulliValues.add(context.mkAnd(currentBernoulli, context.mkGe(...), context.mkLe(...)));
43      //Similar to the procedure in the interpreted case
44      if (probability == 1.0) {
45        ...
46      }
47      else if (probability == 0.0) {
48        ...
49      }
50      else {
51        //Now, do not only consider the condition expression as in the interpreted case, but also whether another
              former translated function, after interpreting the parameters, has the same set of parameters with
              allParams. Only if the placeholder variable refers to a function interpretation that has not referred to
              before, count the usage of the probabilistic transition for x_i (if used in the model) for this placeholder
```

```
            variable.
52        if (BernoulliStatement.conditionExpr != null) {
53          if (allParams != null) {
54            ... .addBernoulli(context.mkAnd(currentBernoulli, context.mkNot(allParams), BernoulliStatement.
                  conditionExpr, context.mkGe(...), context.mkLe(...)), ..., probability);
55            ... .addBernoulli(... context.mkNot(currentBernoulli), ... , (1.0 − probability));
56          }
57          else {
58            ... .addBernoulli(context.mkAnd(currentBernoulli, BernoulliStatement.conditionExpr ...), ..., probability
                  );
59            ...
60          }
61        }
62        else {
63          ...
64        }
65      }
66    }
67    //Finally, add a constraint that determines the value of the overall placeholder variable to either be equal to the
          probability placeholder variable of its probability (depending on the interpretation) that was stored before
          in BernoulliValues, or to the value of a former translated same function if it got the same parameter
          interpretation.
68    ...
69      return context.mkEq(BernoulliStatement.Bernoulli, context.mkOr(context.mkAnd(context.mkOr(
            BernoulliValues.toArray(new BoolExpr[BernoulliValues.size()])), context.mkNot(allParams)),
            otherFuncValue));
70    ...
71  }
```

A.7.10 Discrete Statement Translation

```
1   //Only simple discrete statements for the interpreted case are supported, which were used in the ‘‘original’’
        encodings of the files from the IPPC 2018 (other versions without enums and discrete statements were
        available as well). The code for Bernoulli expressions could be applied to discrete statements in a similar
        fashion for a broader support.
2   public Expr registerDiscreteStatement(Discrete dExpr, ... BoolExpr conditionExpression) {
3     ...
4     //Get the enum definition for this discrete statement and create a placeholder variable
5     EnumSort enumSort = varTranslator.getContextVars().getEnum(enumName);
6     Expr discreteVar = context.mkConst("_discrete_" + name_addition, enumSort);
7     ++name_addition;
8
9     //Register all probability−value pairs by iterating through all possible values of the enum
10    for (int i = 0; i + 1 < dExpr._exprProbs.size(); i += 2) {
11      //Get the current enum value
12      ENUM_VAL val = (ENUM_VAL) dExpr._exprProbs.get(i);
13      //Get the probability expression for this enum
14      EXPR prob = dExpr._exprProbs.get(i + 1);
15      //Get the enum value as Z3 expression
16      Expr value = ...;
```

```
17      ...
18      //Get the probability of the probabilistic statement (only supports one exact value in this case)
19      Double probability = getProbabilityValues(...).get(0).doubleValue();
20      //Similar behavior to Bernoulli, but assign a value to the placeholder variable instead of making it true or false
21      if (probability == 1.0) {
22        ... .addBernoulli(context.mkEq(discreteVar, value), ..., 1.0);
23      }
24      else if (probability == 0.0) {
25        ... .addBernoulli(context.mkNot(context.mkEq(discreteVar, value)), ..., 1.0);
26      }
27      else {
28        if (conditionExpression != null) {
29        ...
30      }
31    }
32
33    return discreteVar;
34    ...
35  }
```

A.7.11 Discrete and Bernoulli Statement Translation - getProbabilityValues and getVarValues

The function getProbabilityValues is used to get all possible probability values of a probabilistic statement. It traverses the statement recursively. In each function call, the set of possible values for the current expression is determined and returned. Expressions including calculations are handled using Java's BigDecimal class. If a Boolean expression is found in a step of the recursion, it is interpreted as being either 0 or 1. Function values are determined differently, using getVarValues. If they are not Boolean and not connected to enums, whose values can be determined easily, but non-fluent, then the default or initial values from the RDDL file can be used. Else, the value of the function might change over time, so, using Z3's solver class, all possible interpretations of the function are tried out and all satisfying interpretations are returned. This only works for integer-valued functions with a finite range.

Due to the length of getProbabilityValues, the function is not displayed in this thesis. Its actual implementation is less relevant than its idea.

A.7.12 Constraint Translation

The different constraint statements are translated iteratively for each step, for each set of parameters of the functions. The translation is similar for state constraints, action preconditions, cpfs and the reward function. It also relies on some storage classes, which are not described in the appendix. The class all translations are based on, translateExpression, which is used to translate a single RDDL constraint, is shown partially here.

```
1   public static Expr translateExpression(...) {
2     ...
```

```
3   //In each function call, one of the following expression types needs to be translated (recursively)
4   if (constraint instanceof Bernoulli) {
5     translation = BernoulliTranslator.registerBernoulliStatement(...);
6   }
7   else if (constraint instanceof PVAR_EXPR) {
8     PVAR_EXPR pExpr = (PVAR_EXPR)constraint;
9
10    String transVarName = pExpr._pName.toString();
11    //Get parameter values; some parameters might be quantified (or were part of a product or sum expression)
12    //Then, get the function expression in Z3, depending on whether its quantified or not, using a storage class
13    ...
14  }
15  else if (constraint instanceof QUANT_EXPR) {
16    //Quantifier expression
17    QUANT_EXPR qExpr = (QUANT_EXPR)constraint;
18
19    //Determine variables that are quantified
20    ArrayList<String> paramNames = ...;
21    ArrayList<String> paramTypes = ...;
22    ...
23
24    //Get all possible combinations of the quantified parameters
25    ArrayList<ArrayList<Pair<String, Expr>>> result = varTranslator.createQuantifiedParams(paramNames,
          paramTypes);
26    ...
27    //Get all possible expressions for all possible parameters
28    for (ArrayList<Pair<String, Expr>> replList : result) {
29      ...
30      expressions.add(translateExpression(..., qExpr._expr, ...));
31    }
32
33    if (qExpr._sQuantType.equals("forall")) {
34      translation = context.mkAnd(expressions ...);
35    }
36    else if (qExpr._sQuantType.equals("exists")) {
37      translation = context.mkOr(expressions ...);
38    }
39    ...
40  }
41  else if (constraint instanceof CONN_EXPR) {
42    //And, or, implication etc.
43    CONN_EXPR cExpr = (CONN_EXPR)constraint;
44
45    if (cExpr._alSubNodes != null) {
46      //Get a list of all translated expressions that are part of the CONN_EXPR, by calling translateExpression on
            each part again
47      ArrayList<Expr> expressions = ...;
48      ...
49
50      if(cExpr._sConn.equals("^")) {
51        translation = context.mkAnd(expressions ...);
52      }
```

```
53          else if(cExpr._sConn.equals("|")){
54            translation = context.mkOr(expressions ...);
55          }
56          else if(cExpr._sConn.equals("=>")){
57            translation = context.mkImplies(... expressions.get(0), ... expressions.get(1));
58          }
59          else if(cExpr._sConn.equals("<=>")){
60            translation = context.mkAnd(context.mkImplies(...), context.mkImplies(...));
61          }
62        }
63      }
64      else if (constraint instanceof NEG_EXPR) {
65        NEG_EXPR nExpr = (NEG_EXPR)constraint;
66
67        Expr expression = translateExpression(...);
68        translation = context.mkNot(... expression);
69      }
70      else if (constraint instanceof COMP_EXPR) {
71        //Comparator like >, < etc.
72        COMP_EXPR cExpr = (COMP_EXPR)constraint;
73
74        Expr expression1 = translateExpression(... , cExpr._e1, ...);
75        Expr expression2 = translateExpression(... , cExpr._e2, ...);
76        ...
77
78        if (cExpr._comp.equals("~=")) {
79          translation = context.mkNot(context.mkEq(expression1, expression2));
80        }
81        else if (cExpr._comp.equals("<=")) {
82          translation = context.mkLe((ArithExpr) expression1, (ArithExpr) expression2);
83        }
84        else if (cExpr._comp.equals("<")) {
85          translation = context.mkLt((ArithExpr) expression1, (ArithExpr) expression2);
86        }
87        ...
88      } ...
89      else if (constraint instanceof LVAR) {
90        //Used to get parameter representations in Z3 for PVAR_EXPR
91        ...
92      }
93      else if (constraint instanceof LTYPED_VAR) {
94        //Used to get parameter representations in Z3 for PVAR_EXPR
95        ...
96      }
97      else if (constraint instanceof TVAR_EXPR) {
98        //Not implemented
99      }
100     else if (constraint instanceof ENUM_VAL) {
101       translation = ... .getEnumExpr(((ENUM_VAL) constraint)._sConstValue);
102     }
103     else if (constraint instanceof OBJECT_VAL) {
104       //Not tested
```

```
105       translation = ... .getEnumExpr(((OBJECT_VAL) constraint)._sConstValue);
106     }
107     //DiracDelta and KronDelta are ‘‘ignored’’, as described in the solution chapter
108     else if (constraint instanceof DiracDelta) {
109       DiracDelta dDelta = (DiracDelta) constraint;
110       translation = translateExpression(..., dDelta._exprRealValue, ...);
111     }
112     else if (constraint instanceof KronDelta) { ...
113       KronDelta kDelta = (KronDelta) constraint;
114       translation = translateExpression(... , kDelta._exprIntValue, ...);
115     }
116     //Unsupported probability distributions
117     else if (constraint instanceof Uniform) { ...
118     else if (constraint instanceof Normal) { ...
119     else if (constraint instanceof Dirichlet) { ...
120     else if (constraint instanceof Multinomial) { ...
121     else if (constraint instanceof Discrete) {
122       translation = BernoulliTranslator.registerDiscreteStatement((Discrete) constraint, ...);
123     }
124     //Unsupported probability distributions
125     else if (constraint instanceof Exponential) { ...
126     else if (constraint instanceof Weibull) { ...
127     else if (constraint instanceof Gamma) { ...
128     else if (constraint instanceof Poisson) { ...
129     else if (constraint instanceof BOOL_CONST_EXPR) {
130       BOOL_CONST_EXPR bExpr = (BOOL_CONST_EXPR)constraint;
131       translation = context.mkBool(bExpr._bValue);
132     }
133     else if (constraint instanceof INT_CONST_EXPR) { ...
134       INT_CONST_EXPR iExpr = (INT_CONST_EXPR)constraint;
135       translation = context.mkInt(iExpr._nValue.toString());
136     }
137     else if (constraint instanceof REAL_CONST_EXPR) { ...
138       REAL_CONST_EXPR rExpr = (REAL_CONST_EXPR)constraint;
139       translation = context.mkReal(rExpr._dValue.toString());
140     }
141     //Unsupported expression
142     else if (constraint instanceof STRUCT_EXPR) { ...
143     else if (constraint instanceof OPER_EXPR) {
144       //Operator expression: ∗, +, /, − ...
145       OPER_EXPR oExpr = (OPER_EXPR) constraint;
146
147       Expr first = translateExpression(... , oExpr._e1, ...);
148       Expr second = translateExpression(... , oExpr._e2, ...);
149       ...
150
151       if (oExpr._op.toString().equals("+")) {
152         translation = context.mkAdd(expressions);
153       }
154       else if (oExpr._op.toString().equals("−")) {
155         translation = context.mkSub(expressions);
156       }
```

```
157     else if (oExpr._op.toString().equals("*")) {
158       translation = context.mkMul(expressions);
159     }
160     else if (oExpr._op.toString().equals("/")) {
161       translation = context.mkDiv((ArithExpr) first, (ArithExpr) second);
162     }
163     //Unsupported expressions, but support could be implemented if required
164     else if (oExpr._op.toString().equals("min")) { ...
165     else if (oExpr._op.toString().equals("max")) { ...
166   }
167   else if (constraint instanceof AGG_EXPR) {
168     //Aggregated expressions: Sum, product....
169     AGG_EXPR aExpr = (AGG_EXPR) constraint;
170
171     //Extract variables that need to be replaced for each possible value − code similar to QUANT_EXPR
172     ...
173     ArrayList<ArrayList<Pair<String, Expr>>> result = varTranslator.createQuantifiedParams(paramNames,
            paramTypes);
174     ...
175     for (ArrayList<Pair<String, Expr>> replList : result) {
176       ...
177       expressions.add(...);
178     }
179
180     if(aExpr._op == "sum") {
181       ...
182       translation = context.mkAdd(expressions.toArray(new ArithExpr[expressions.size()]));
183     }
184     else if(aExpr._op == "prod") {
185       ...
186       translation = context.mkMul(expressions.toArray(new ArithExpr[expressions.size()]));
187     }
188     //Unsupported expressions, but support could be implemented if required
189     else if(aExpr._op == "min") { ...
190     else if(aExpr._op == "max") { ...
191   }
192   //Unsupported expression
193   else if (constraint instanceof FUN_EXPR) { ...
194   else if (constraint instanceof IF_EXPR) {
195     IF_EXPR iExpr = (IF_EXPR) constraint;
196     //Condition
197     BoolExpr testExpr = (BoolExpr) translateExpression(... , iExpr._test, ...);
198     //In case it is a nested if expression: Refer to other conditions
199     ...
200     if (conditionExpression != null) {
201       conditionExprTrue = context.mkAnd(conditionExpression, testExpr);
202       conditionExprFalse = context.mkAnd(conditionExpression, context.mkNot(testExpr));
203     }
204     else {
205       conditionExprTrue = testExpr;
206       conditionExprFalse = context.mkNot(testExpr);
207     }
```

```
208     //Result if condition is true / false
209     Expr trueExpr = translateExpression(... , iExpr._trueBranch, ... , conditionExprTrue, ...);
210     Expr falseExpr = translateExpression(... , iExpr._falseBranch, ..., conditionExprFalse, ...);
211
212     translation = context.mkITE(testExpr, trueExpr, falseExpr);
213   }
214   //Unsupported expressions
215   else if (constraint instanceof SWITCH_EXPR) { ...
216   else { ...
217
218   return translation;
219 }
```

A.7.13 Function Definition

```
1  public VarTranslator(Context context_, RDDL dom_parse, RDDL inst_parse, Boolean bitBlast, Boolean
       quantifierBitBlast) {
2    instance_parse = inst_parse;
3    domain_parse = dom_parse;
4    ... mapActionVars ... mapNonFluentVars ... mapOtherVars ...
5    ...
6    horizon = instance_parse._tmInstanceNodes.firstEntry().getValue()._nHorizon;
7    ...
8    //Translate object enums to z3 sorts
9    for (Map.Entry<TYPE_NAME, OBJECTS_DEF> pVariable : instance_parse._tmNonFluentNodes.firstEntry().
         getValue()._hmObjects.entrySet()) {
10     ...
11     ArrayList<String> objectVal = new ArrayList<>();
12     for(LCONST val : pVariable.getValue()._alObjects) {
13       objectVal.add(val._sConstValue.toString());
14     }
15     mapSort.put(... ._sObjectClass.toString(), context_.mkEnumSort(... ._sObjectClass.toString(), objectVal.
           toArray(...)));
16   ...
17   }
18   ...
19   horizon = instance_parse._tmInstanceNodes.firstEntry().getValue()._nHorizon;
20   //Save all functions given by the RDDL domain description (pVariables) (this is not about the constraints)
21   for (PVARIABLE_DEF pvar : domain_parse._tmDomainNodes.firstEntry().getValue()._hmPVariables.values())
         {
22     String name = pvar._pvarName.toString();
23
24     if (pvar instanceof PVARIABLE_STATE_DEF) {
25       if (((PVARIABLE_STATE_DEF)pvar)._bNonFluent) {
26         mapNonFluentVars.put(name, pvar);
27       }
28       else {
29         mapOtherVars.put(name, pvar);
30       }
31     }
```

```
32        else if (pvar instanceof PVARIABLE_ACTION_DEF) {
33          mapActionVars.put(name, pvar);
34        }
35        else {
36          mapOtherVars.put(name, pvar);
37        }
38      }
39      //Add the reward
40      mapOtherVars.put("reward", new PVARIABLE_INTERM_DEF("reward", false, "real", new ArrayList<>(), 1));
41      ...
42
43      //Create the initial set of functions (either, if uninterpreted parameters were chosen, for the last step without
            possible later combinations that arise from the constraint translation through quantifiers/sums/products, or
            all possible combinations)
44      for (PVARIABLE_DEF pvar : domain_parse._tmDomainNodes.firstEntry().getValue()._hmPVariables.values())
            {
45        String name = pvar._pvarName.toString();
46        createVar(name);
47      }
48      createVar("reward", false);
49    }
50    ...
51
52    private void createVar(...) {
53      PVARIABLE_DEF pvar = ... .getVar(originalName);
54      ...
55      //Get parameters
56      ArrayList<EnumSort> params = new ArrayList<>();
57      if (pvar._alParamTypes != null) {
58        params = ... .getEnums(originalName);
59      }
60      //Create new consts if parameter interpretation is not desired, else go through all possible combinations
61      ArrayList<ArrayList<Expr>> concreteParameters = new ArrayList<>();
62      if (doBitBlast) {
63        concreteParameters = generateCombinationArray(params);
64      }
65      else {
66        ...
67        for (EnumSort param : params) {
68          Expr uninterpretedConstant = context.mkConst("_x_" + paramCount, param);
69          concreteParameters.get(0).add(uninterpretedConstant);
70          ...
71        }
72      }
73
74      for (String name : names) {
75        //Define the function according to its return type
76        FuncDecl decl = null;
77        if (pvar._typeRange._STypeName == "bool") {
78          decl = context.mkFuncDecl(name, params.toArray(new EnumSort[params.size()]), context.mkBoolSort());
79        }
80        else if (pvar._typeRange._STypeName == "int") {
```

```
81          ...
82        }
83        else if (pvar._typeRange._STypeName == "real") {
84          ...
85        }
86        else {
87          for (String enumName : contextVars.getAllEnumNames()) {
88            if (pvar._typeRange._STypeName.equals(enumName)) {
89              decl = context.mkFuncDecl(name, params.toArray(new EnumSort[params.size()]), contextVars.
                        getEnum(enumName));
90              break;
91            }
92          }
93        }
94        ...
95        ... .addVar(originalName, name, decl, concreteParameters, addParams);
96      }
97    }
```

A.7.14 Translation of Initial and Default Values

```
1   public HashMap<Expr, ArrayList<Expr>> initialiseVars(Context context, RDDL domain_parse, RDDL
        instance_parse, VarTranslator varTranslator) {
2     ...
3     //Initialize all non-fluents with values given by the RDDL instance description - other non-fluents need to be
          initialized using their default values given in the domain description
4     HashMap<String, ArrayList<Pair<Expr, Expr>>> nonDefaultValues = new HashMap<>(); //Stores all variables
          (name = key) with the condition when to set a non-default value (pair: precond, precond->effect)
5     ...
6     for (PVAR_INST_DEF initValue : instance_parse._tmNonFluentNodes.firstEntry().getValue()._alNonFluents) {
7       ...
8       //Get the param-z3-translation (params), use the string description of the param values given by the RDDL
            file to store them as Expr in paramValues and then define the constraint for the initial value given by the
            RDDL file
9       ArrayList<String> functions = ... .getAllVarsWithName(initValue._sPredName.toString(), 0);
10      for (String functionName : functions) {
11        ArrayList<ArrayList<Expr>> params = ... .getParameters(functionName);
12        ArrayList<EnumSort> enums = ... .getEnums(initValue._sPredName.toString());
13        for (ArrayList<Expr> parameters : params) { ...
14            ...
15            paramConditions.add(context.mkEq(param, paramValue));
16          }
17
18          //Translate the initial value to z3 - at this point, only booleans or integers/reals or enums are expected
19          BoolExpr exprEquals = context.mkEq(translatedVars.getFuncDecl(functionName).apply(parameters.
                toArray(new Expr[parameters.size()])), getValueExpr(context, translatedVars, initValue._oValue));
20
21          //Store given initial value for later access
22          ...
```

```
23          //Store the set of parameters for which the initial value is defined to later decide if a default value must be
                assigned or not (made for the uninterpreted approach, but also used in the interpreted case)
24          if (paramConditions.size() > 0) {
25            precondition = context.mkAnd(paramConditions.toArray(new BoolExpr[paramConditions.size()]));
26            implication = context.mkImplies(precondition, exprEquals);
27          }
28          else {
29            implication = exprEquals;
30          }
31          ...
32          nonDefaultValues.get(functionName + "_" + parameters.toString()).add(new Pair<Expr, Expr>(
                precondition, implication));
33        }
34      }
35    }
36    ...
37    //Use default values or the given non-default values for all defined non-fluents
38    for (Map.Entry<String, ArrayList<String>> entry : ... .getAllDefaultNonFluentVars().entrySet()) {
39      //Translate the default value to z3
40      Expr exprEqualsDefault = getValueExpr(context, translatedVars, translatedVars.getDefaultValue(entry.getKey
            ()));
41      ...
42      for (String functionName : entry.getValue()) {
43        ...
44        for (ArrayList<Expr> parameters : params) {
45          Expr funcExpr = translatedVars.getFuncDecl(functionName).apply(parameters.toArray(new Expr[
                parameters.size()]));
46
47          //Store the default value
48          ...
49          //Only use the default value if no other initial values are given
50          if (nonDefaultValues.get(functionName + "_" + parameters.toString()) == null) { ...
51            ...
52            initialConstraints.get(funcExpr).add(context.mkEq(funcExpr, exprEqualsDefault));
53          }
54          //Else: Use the default value if no other value has been specified for the function with the actual parameter
                interpretation
55          else {
56            ...
57            for (Pair<Expr, Expr> pair : nonDefaultValues.get(functionName + "_" + parameters.toString())) {
58              if (pair._o1 != null) {
59                nonDefaultConditions.add((BoolExpr) pair._o1);
60              }
61              if (initialConstraints.get(funcExpr) == null) {
62                initialConstraints.put(funcExpr, new ArrayList<>());
63              }
64              initialConstraints.get(funcExpr).add(pair._o2);
65            }
66            if (nonDefaultConditions.size() > 0) {
67              initialConstraints.get(funcExpr).add(context.mkImplies(context.mkNot(context.mkOr(
                      nonDefaultConditions.toArray(new BoolExpr[nonDefaultConditions.size()]))), context.mkEq(
                      funcExpr, exprEqualsDefault)));
```

```
68                }
69              }
70            }
71          }
72        }
73
74        //Do the same for state−fluents
75        ...
76
77        return initialConstraints;
78    }
```

A.7.15 Setting Program Parameters

```
1   public enum Algorithm {searchConstraints, rewardOptimizer, probabilityOptimizer};
2
3   //All parameters can be set in the main file
4   public static void main(String[] args) {
5     ...
6     double rewardApproximation = ...
7     //false if uninterpreted parameters should be used
8     Boolean doBitBlast = ...
9     Boolean planEveryStep = ...
10    //Go through all possible combinations either by iterating through them "manually" (exact solution) or by using a
          linear equation system to iterate from the optimal to the worst solution − this option has not been
          mentioned in the thesis because it did not work as expected, so keep this false
11    Boolean final_search_with_linear_equation_system = false;
12    //Set the algorithm using an enum called Algorithm
13    Algorithm algorithm = ...
14
15    //Only set this to false for domains with only existential (no sum, for all etc) quantifiers that are NOT negated, in
          combination with doBitBlast = false
16    Boolean doQuantifierBitBlast = true;
17
18    //Possible RDDL domains − set the file by commenting out the desired file variable and adding the instance
          number
19    String inst_number = "__1";
20    //String file = "prop−dbn_inst_mdp"; //works
21    String file = "navigation_inst_mdp"; //works
22    //String file = "triangle_tireworld_inst_mdp"; //works
23    ...
```