# Workflow Generation with Planning

Dissertation zur Erlangung des Doktorgrades
der Technischen Fakultät der
Albert-Ludwigs-Universität Freiburg im
Breisgau

vorgelegt von
BENEDICT WRIGHT

Dekanin:
Prof. Dr. Hannah Bast

Erstgutachter und Betreuer der Arbeit:
Prof. Dr. Bernhard Nebel

Zweitgutachter:
Prof. Dr. Andreas Podelski

Datum der mündlichen Prüfung: 24.07.2019

# Abstract

Automated planning is the field of research with the goal of enabling agents to act in an intelligent way to reach a certain goal. In business applications, this corresponds to generating workflows, describing how objectives are to be reached. In this thesis, a method for creating such workflows automatically, from formal data specifications with the help of planning is described. Even though, this provides functional workflows for the developed digital preservation system, it lacks functionality considering usability issues, essential for human computer interaction. These usability constraints can be modeled using soft goals, which add optional constraints to the resulting plan. These constraints do not need to be fulfilled by the plan, however increase the quality of the result. A method for dealing with these soft constraints in classical planning is introduced, using conditional effects for tracking the constraints, and state-dependent action costs for guiding the search.

A prominent approach for solving such planning problems is heuristic search, which uses so called heuristic estimates to guide the search towards achieving the goal condition. When combining state-dependent action costs with conditional effects, some problems arise, which leads to the heuristic functions becoming relatively uninformed, providing inferior guidance. Therefore, a theory on treating both state-dependent action costs and conditional effects combined is introduced, which reduced the problems when dealing with them independently. This approach is based on an edge-valued multi-valued decision diagram (EVMDD) representation of the cost function and the conditional effects. Here, the existing theory of EVMDDs over arithmetic functions is generalized to EVMDDs over monoids, as to be able to represent the cost functions and conditional effects in one EVMDD combined.

As workflows involving human users, can consist of actions for which the outcome is not a priory known, soft trajectory constraints are introduced to the fully observable nondeterministic setting. This thesis provides a basic understanding on how these constraints can be interpreted in this setting, and how existing heuristic functions can be augmented, as to guide the search towards a goal, fulfilling the constraints.

# Zusammenfassung

Das Forschungsgebiet der automatisierten Handlungsplanung hat zum Ziel, Agenten das intelligente Handeln zu ermöglichen, um ein vorgeschriebenes Ziel zu erreichen. Im Bereich der Geschäftsanwendungen entspricht dies dem Erzeugen von Arbeitsprozessen, welche beschreiben, wie ein vorgegebenes Ziel erreicht werden soll. In dieser Arbeit wird eine neue Methode vorgestellt, welche es erlaubt, mit Hilfe einer formalen Datenspezifikation und automatischer Handlungsplanung solche Arbeitsprozesse zu generieren. Auch wen die vorgestellte Methode korrekte Arbeitsprozesse erzeugt, so mangelt es ihr an Funktionalität, um benutzerfreundliche Prozesse zu erstellen. Solch eine Funktionalität ist allerdings essenziell, wenn diese Prozesse Mensch-Maschinen Interaktionen beinhalten. Solche Benutzeranforderungen können jedoch mit Hilfe optionaler ziele modelliert werden, welche zusätzliche Anforderungen an den resultierenden Plan stellen. Diese Ziele müssen jedoch nicht erfüllt sein, tragen jedoch zur Qualität des Planes bei. Eine Methode solche optionalen Ziele zu erreichen wird in dieser Arbeit vorgestellt und basiert auf einer Kompilierung in konditionalen Effekten, um den Zustand der Ziele zu verfolgen, und zustandsabhängigen Kosten, um die Suche effizient zu leiten.

Ein gängiger Ansatz um solche Planungsprobleme zu lösen ist die heuristische Suche. Hierbei, wird ein Suchalgorithmus (z.B. $A^*$) mit Hilfe einer heuristischen Schätzfunktion in Richtung des Zieles geleitet. Wenn jedoch konditionale Effekte und zustandsabhängige Kosten gemeinsam verwendet werden, können Probleme auftreten, welche die Genauigkeit der heuristischen Schätzung negativ beeinträchtigt. Dies kann so weit gehen, dass die Heuristik vollkommen uninformativ wird. Zu diesem Zweck wird in dieser Arbeit eine Methode vorgestellt, um diese beiden Konzepte gemeinsam zu betrachten, um den eingeführten Fehler wieder zu beheben. Dieser Ansatz basiert auf der Repräsentation der konditionalen Effekte und der zustandsabhängigen Kosten als kanten-gewichteter mehrwertiger Entscheidungsdiagramme (englisch edge-valued multi-valued decision diagrams kurz EVMDD) dargestellt. Hierfür wird die existierende Theorie über EVMDDs für arithmetische Ausdrücke, auf Ausdrücke über Monoide erweitert.

Da Arbeitsprozesse, welche menschliche Interaktion berücksichtigen, oft auch Aktionen mit unbestimmtem Ausgang beinhalten, wird in dieser Arbeit das Konzept der optionalen Ziele auf das Planungsproblem mit nichtdeterministischen Aktionen erweitert. Hierfür, bietet diese Arbeit ein generelles Verständnis über die Interpretation solcher optionalen Ziele im nichtdeterministischen Planungsumfeld. Des weiteren wird eine Theorie präsentiert, wie bestehende heuristik Funktionen angepasst werden können, um die Suche in Richtung der Erfüllung der optionalen Ziele zu lenken.

## Acknowledgments

As with every work, a PhD thesis is not created by a single individual, but is rather the result of years of collaboration with many people. Additionally to professional collaborations, the support received by all sorts of people is essential for succeeding in such an endeavor. I therefore want to use this space to thank all those, without whom this work would not have been possible. First and foremost, I want to thank my supervisor Prof. Bernhard Nebel, for offering me a position in his research group, and provided me with all the resources required to do my research, as well as travel to conferences, and meetings with other research groups. Additionally, I want to thank him for providing an environment, in which I could develop and pursue my own Ideas, in an atmosphere of open mindedness and collaboration. I would also like to thank all of my colleagues for the insightful discussions, and making the department such a pleasurable place for research and education. Especially, I would like to thank Robert Mattmüller and Florian Geißer for the close cooperation on various research topics. My thanks towards Robert Mattmüller cannot be emphasized enough, as his endurance, patience, and feedback in teaching me the necessary formal skills in writing scientific work, was essential for me to be able to finish my research and publications. Thank you Johannes Aldinger, Yusra Alkhazraji, Thorsten Engesser, Florian Geißer, Andreas Hertle, Felix Lindner, Robert Mattmüller, Tim Schulte, David Speck, and Stephan Wölfl for their support and ongoing friendship. A special thanks goes to Uli Jakob and Petra Geiger who worked so hard over the years to keep the infrastructure up and running, and handling all sorts of office management and bureaucracy issues. Many thanks go to my friends, Michael Hödl, Mario Wellik, Wolfdietrich Aichelburg, and Andreas Huss, for their encouragement and support at all times. Thank you to my training partners, especially Christian Kehl, Yvonne Kehl, Michael Naroska, and Franziska Sauter at my karate club, and my piano teacher Mechthild Ehlich for the mental and physical compensation, during times of stress. Finally, I want to thank my family Helena Aspernig for all her support. Thank you to my mother Margret Holt, my father David Wright,and my sisters Claire and Suzanne, for supporting me during all the ups and downs throughout the years, not only during my research, but also in the years leading up to it. Without their support this would not have been possible, and I am very thankful for having such a great family.

Thank you!

## Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- oder Beratungsdiensten (Promotionsberaterinnen oder Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

Freiburg, *Januar 2019*. (Benedict Wright)

# Contents


**1 Introduction** **1**
- 1.1 Motivation . . . 2
- 1.2 Outline . . . 3
- 1.3 Contribution . . . 3

**2 Mathematical Foundations** **6**
- 2.1 Edge-Valued Multi-Valued Decision Diagrams . . . 7
- 2.2 Linear Temporal Logic . . . 21

**3 Planning Foundations** **24**
- 3.1 Classical Planning . . . 25
- 3.2 Planning with State-Dependent Action Costs . . . 35
- 3.3 Fully Observable Nondeterministic Planning . . . 43

**4 Planning with conditional effects and state-dependent action costs** **48**
- 4.1 Conditional Effects Revisited . . . 48
- 4.2 Combining State Dependent-Action Costs and Conditional Effects . . . 60

**5 Planning with soft trajectory constraints** **74**
- 5.1 State trajectory constraints . . . 75
- 5.2 Evaluation . . . 89

**6 Soft trajectory constraints in FOND planning** **99**

**7 Application in Digital Preservation** **110**
- 7.1 OntoRAIS . . . 112

**8 Future Work** **133**

**9 Conclusion** **135**

**List of Figures** **137**

**List of Tables** **138**

**List of Algorithms** **139**

**Bibliography** **140**

# Chapter 1

# Introduction

Artificial intelligence(AI) in computer science is the attempt of creating so called intelligent agents. These agent should be able to act in an environment pursuing an individual goal. The field of AI consist of multiple disciplines, such as reasoning and knowledge representation, machine learning, and planning.

This work is concerned with the field of automatic planning. In planning, an agent is tasked with reaching a goal, by applying a sequence of actions (Russell and Norvig, 2016). Even though this definition suggests, that planning requires the presence of a physical or virtual agent such as in robotics, basically any problem consisting of internal state representation and a set of actions transforming this state can be modeled. In the field of domain independent planning, general solving approaches are investigated, which do not rely on any knowledge of the underlying problem domain. In *classical* planning the assumption is, that a single agent acts in a fully deterministic environment. Thus, the system has full knowledge over its current and future states (what happens after executing a certain action). These kind of problems can be modeled using the *planning domain definition language* (PDDL) (McDermott et al., 1998). Additionally, to the classical setting, multiple extensions where proposed over the years, such as non-deterministic (Cimatti et al., 2003) or probabilistic planning, where the effects of an action may occur non deterministically, or with a certain probability. Many more formalisms exist such as for multi-agent planning or epistemic planning. However, this work focuses on classical planning and fully observable non-deterministic planning (FOND). In FOND planning, the agent has full knowledge over its internal state, and the environment (as in classical planning), however the effects of an action may occur non-deterministically.

Classical planning has been well analyzed over the years (Bäckström and Nebel, 1995; Bylander, 1994), and finding a solution is in general *PSPACE*-

*hard*. However, applying heuristic search yields satisfying results to many domains. In heuristic search, the problem of finding a plan is reduced to the search in a graph, where nodes represent states, and edges represent action outcomes. Often such actions also come with a cost, thus the edges are also labeled with the cost of applying the action. The problem of finding a plan is then solved by finding a path in the graph from an initial state to a goal state. In optimal planning, this is extended to finding the *shortest* or *cheapest* path. As the search space is exponential in the input size (the number of variables), classical depth-first or breadth first search is intractable. In planning a heuristic search is therefore used. This approach uses an estimator, or heuristic, annotating every node with an estimated distance to a goal state. Then the search can expand the most promising nodes first leading to a more goal oriented search.

Additionally, to reaching a certain goal, it is often desirable to be able to add optional requirements. These can be in the form of soft goals, or soft trajectory constraints (Gerevini and Long, 2005). Soft goals, are additional goal conditions, that an agent might fulfill. However, not doing so does not result in a faulty behavior (e.g. a service robot that tries to have the kitchen cleaned at the end of its task). On the other hand soft trajectory constraints add constraint on how the goal is reached (e.g. first serving dinner and then cleaning the kitchen). Again, not fulfilling the constraints does not result in faulty behavior.

## 1.1 Motivation

The main aspect of business applications such as digital preservation systems, is that of performing certain business processes, formally described in workflows. These workflows describe how a certain goal is reached, ensuring formal correctness, and the following of business policies. During the development of the research archival system OntoRAIS described in Chapter 7, it came apparent, that creating and maintaining all the required workflows manually was tedious and error prone. Therefore, a method of automatically generating workflows using planning was derived, and implemented (Section 6). However, the resulting workflows, although being correct, did not fulfill the users expectations on how these processes where executed. Adding constraints to how a goal is reached is the area of state trajectory constraints, in particular, soft trajectory constraints. Therefore, a theory of handling soft trajectory constraints in classical planning was derived, for which the foundations first had to be laid. This led to the development of the combined treatment of conditional effects and state-dependent action costs (described is Section 4.2), and the underlying data structure of EVMDDs over arbitrary monoids (described in Section 2.1).

## 1.2 Outline

Chapter 2 first introduces the mathematical foundations such as edge-valued multi-valued decision diagrams (EVMDD) and linear temporal logic (LTL) are introduced. Additionally, the theory of EVMDDs over monoids, developed by the author and other members of the work group, is presented. Then Chapter 3 introduces the basic planning formalism used throughout the rest of the Thesis, including classical planning, planning with state-dependent action costs, and fully observable nondeterministic planning.

Chapter 4 then revisits conditional effects presenting different methods of representing and compiling them away. Most notably, a theory on representing conditional effects as EVMDDs is presented in Section 4.1. Section 4.2 then discusses the problems that arise when dealing with conditional effects and state-dependent action costs combined. A solution based on the generalized theory of EVMDDs over effect cost tuples is then presented. Also a method of compiling away conditional effects with state dependent action costs based on these EVMDDs is presented.

Chapter 5 introduces state trajectory constraints to the classical planning setting. A method of compiling away these constraints is then presented, facilitating conditional effects and state-dependent action costs.

The notion of state trajectory constraints is then expanded to the fully observable nondeterministic setting in Chapter 6. Providing a basic theory on how to guide the planning process in to fulfilling these constraints, by defining constraint aware heuristics.

Finally, Chapter 7 introduces a digital preservation system, developed for the BrainLinks-BrainTools cluster of excellence at the Albert-Ludwigs University of Freiburg. The requirements towards a preservation system are introduced, and the developed system described. Additionally, a method for automatically generating workflows, for digital preservation, based on formal data specification and planning is introduced.

## 1.3 Contribution

This thesis presents the contributions by the author to the field of AI planning, and computer science in general:

- **EVMDDs**: Edge-Valued Multi-Valued Decision Diagrams(EVMDDs) are used to represent decision diagrams branching over multiple multi valued variables. Additionally, the edges are annotated by a label.

This thesis presents a novel theory on EVMDDs over Monoids, created in collaboration with Robert Mattmüller, Florian Geißer and Bernhard Nebel. This theory goes beyond EVMDDs for arithmetic expressions, and provides the required background for representing state-dependent action costs and conditional effects in a combined EVMDD. The work presented in Chapter 2 is based on the work presented in Mattmüller et al. (2018) and Mattmüller et al. (2017), but provides more details and proofs, omitted in the original publications.

- **Conditional effects and state-dependent action costs**: When dealing with conditional effects in a setting that also contains state-dependent action costs, one major issue arises, when treating these independently from each other. The problem being that in relaxed or abstracted planning tasks, the more expensive effect may be reached by the cheaper cost. This is the result of taking the minimum cost, and maximum effect when executing an action in the relaxed or abstracted task. The Section 4.1 describes how conditional effects can be represented using EVMDDS. Section 4.2 then discusses in depth how EVMDDs can be used represent conditional effects together with state-dependent action costs, as published in the conference papers Mattmüller et al. (2018) and Mattmüller et al. (2017).

- **Soft trajectory constraints**: Soft trajectory constraints in planning are used to specify additional optional constraints towards how a goal state is reached. In this thesis a theory on compiling tasks with soft trajectory constraints into tasks without soft trajectory constraints is presented. Section 5 present this theory first presented in the conference papers Wright et al., 2018b and Wright et al., 2018c together with an empirical evaluation. This work was done in collaboration with Robert Mattmüller and Bernhard Nebel. Additionally, a theory for soft trajectory constraints in fully observable nondeterministic planning is introduced. This theory has not been published prior to this work, and was done in collaboration with Robert Mattmüller.

- **Digital Preservation**: During the years leading up to this thesis, a digital preservation system was developed for the BrainLinks-BrainTools cluster of excellence at the Albert-Ludwigs University of Freiburg. This was done to tackle the increasing amount of research data created by researchers in all different fields of research. For this a method of automatically generating workflows was developed using formal data specifications from an ontology and fully observable nondeterministic planning, and is presented in Section 6. In the process of this project, a conference paper (Wright et al., 2018a) was produced, discussing the necessity and the benefits of such a digital preservation system in research facilities such as universities.

## References to Author's Contributions

Mattmüller, Robert, Geißer, Florian, Wright, Benedict, and Nebel, Bernhard (2018). "On the Relationship Between State-Dependent Action Costs and Conditional Effects in Planning." In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*.

Wright, Benedict, Brunner, Oliver, and Nebel, Bernhard (2018a). "On the Importance of a Research Data Archive". In: *Eighth AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18)*.

Wright, Benedict, Mattmüller, Robert, and Nebel, Bernhard (2018b). "Compiling Away Soft Trajectory Constraints in Planning". In: *Proceedings of the 2018 Conference on Principles of Knowledge Representation and Reasoing (KR 2018)*.

Wright, Benedict, Mattmüller, Robert, and Nebel, Bernhard (2018c). "Compiling Away Soft Trajectory Constraints in Planning". In: *Proceedings of the 2018 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2018)*.

Mattmüller, Robert, Geißer, Florian, Wright, Benedict, and Nebel, Bernhard (2017). "On the Relationship Between State-Dependent Action Costs and Conditional Effects in Planning." In: *Proceedings of the 9th Workshop on Heuristics and Search for Domain-Independent Planning (HSDIP 2017)*.

Wright, Benedict and Mattmüller, Robert (2016). "Automated Data Management Workflow Generation with Ontologies and Planning." In: *Proceedings of the 30th Workshop on Planen/Scheduling und Konfigurieren/Entwerfen (PUK 2016)*.

# Chapter 2

# Mathematical Foundations

In this chapter some mathematical foundations and terminologies are introduced, which is used throughout the rest of this thesis. As this work is concerned with planning as heuristic search over the state space, first the concept of *state* is introduced.

**Definition 1 (State).** Let $\mathcal{V} = \{v_1, \ldots, v_m\}$ be a set of multi valued variables where $\langle d_1, \ldots, d_n \rangle$ is called the domain $\mathcal{D}_v$ of $v$. The notation $|\mathcal{D}_v|$ is used meaning the domain size $n$ of the variable $v$. Then $s$ is a state if it assigns each variable $v \in \mathcal{V}$ exactly one value of its domain.

In the following chapters (Chapter 3.2, Chapter 5) arithmetic expressions are augmented by logical expressions. This adds flexibility to defining cost functions (Chapter 3.2) over state variables, as it adds the functionality of conditional arithmetic terms.

**Definition 2 (Iverson Brackets).** Let $P$ be a logic expression over facts, then the interpretation of Iverson brackets in state $s$ is as follows:

$$[P]_s = \begin{cases} 1 \texttt{ if } s \models P \\ 0 \texttt{ if } s \not\models P \end{cases}$$

For example let $x = \{0, 1, 2\}$ and $y = \{0, 1, 2, 3\}$ be two multi valued variables, then using Iverson brackets the expression

$$[x = 1] \cdot y^2$$

can be defined. This evaluates to $y^2$ only if $x = 1$ and $0$ otherwise. In planning this could correspond to: "If condition $x = 0$ holds the cost of this action is $y^2$ otherwise $0$".

## 2.1 Edge-Valued Multi-Valued Decision Diagrams

In general an Edge-Valued Multi-Valued Decision Diagram (EVMDD from here on) is a decision diagram, which can branch over multiple variables with multiple values. Each outgoing edge $e$ from a node **v** corresponds to a decision over the variable value. Additionally, each edge $e$ is annotated by a label.

Here the definition of EVMDDs over integers (Ciardo and Siminiceanu, 2002) is generalized using monoids for more general use in different applications.

**Definition 3 (Monoid).** Let $T$ be a set of elements and $\bullet$ a binary operator $T \times T \rightarrow T$ and $\mathrm{e} \in T$ then $M = \langle T, \bullet, \mathrm{e}\rangle$ is a monoid if:

$$\forall a, b, c \in T : (a \bullet b) \bullet c = a \bullet (b \bullet c)$$
$$\forall a \in T : \mathrm{e} \bullet a = a \bullet \mathrm{e} = a$$

In words $\mathrm{e}$ is the neutral element with regards to $T$ and the associative operator $\bullet$. Additionally, if for any two elements $x, y \in T$, $x \bullet y = y \bullet x$ then $M$ us a commutative monoid (CM).

**Definition 4 (Partially ordered set).** Let $\leq$ be a binary relation over a set $P$ such that:

1. $a \leq a$
2. $a \leq b$ and $b \leq a$ then $a = b$
3. $a \leq b$ and $b \leq c$ then $a \leq c$

Then $\leq$ is called reflexive, antisymmetric, and transitive. If such a $\leq$ exists for $P$, then $P$ is a partially ordered set.

**Definition 5 (Greatest lower bound).** Let $S$ be a subset of a partially ordered set $(P, \leq)$, then $a \in P$ is a lower bound of $S$ if $a \leq x$ for all $x \in S$. Such an $a$ is a greatest lower bound if for all lower bounds $y$ from $S$, $y \leq a$.

**Definition 6 (Meet-semilattice ordered monoid (MSM)).** Let $M = \langle T, \bullet, \mathrm{e}, \leq\rangle$ be a monoid with a partially ordering relation $\leq$, then $T$ is a meet semilattice if $\leq$ on $T$ has a *greatest lower bound* for any non empty finite subset $T' \subseteq T$, denoted as $\bigwedge T'$.

**Definition 7 (Monus Operator $\dot{-}$ (Amer, 1984)).** Let $M = \langle T, \bullet, \mathrm{e}\rangle$ be a commutative monoid. Let $a, b \in T$ and a relation $\leq$ on $T$ defined as $a \leq b$ iff there exists a $c \in T$ with $c \neq \mathrm{e}$ such that $a + c = b$. Then $\leq$ is reflexive and transitive, and antisymmetric by definition, resulting in a partial order over $T$. If for all $a, b \in T$ there exists a unique smallest $c \in T$ such that $a \leq b + c$ then $M$ is called a *monoid with monus (MM)* and $c$ is this monus with $a \dot{-} b = c$.

**Definition 8 (Meet-semilattice ordered commutative monoid with monus (MCMM)).** Let $M = \langle T, \bullet, \mathrm{e}\rangle$ be a meet semilattice ordered monoid, $\bullet$ is commutative and there exists a monus operator $\dot{-}$. If $\bullet$ is distributable over the greatest lower bound operator $\bigwedge$ i. e., $a \bullet (b \bigwedge c) = (a \bullet b) \bigwedge (a \bullet c)$ and $(b \bigwedge c) \bullet a = (b \bullet a) \bigwedge (c \bullet a)$ for all $a, b, c \in T$, and $\bigwedge T = \mathrm{e}$, then $M = \langle T, \bullet, \mathrm{e}, \dot{-}\rangle$ is a meet-semilattice ordered commutative monoid with monus.

In the following work the monus operator is often omitted in the definition as it is given by the set $T$ and the operator $\bullet$.

**Example 1 (MCMM).** Let $M = \langle \mathbb{N}, +, 0\rangle$ be a monoid with the natural order $\leq$, the greatest lower bound defined by the minimum (min), and the monus operator $-$ the clamped minus operator such that for two elements $a, b \in \mathbb{N}$ with $b > a$ then $a - b = 0$. Then $M$ is a meet semilattice ordered commutative monoid with monus.

**Definition 9 (EVMDD over meet semilattice ordered MCMM).** An EVMDD over the variables $\mathcal{V}$ and the MCMM $M = \langle T, \bullet, \mathrm{e}\rangle$ is a tuple $\mathcal{E} = \langle \kappa, \mathbf{f}\rangle$, where $\kappa \in T$ and $\mathbf{f}$ is a directed acyclic graph consisting of two types of nodes:

1. The single terminal node denoted with **0**.
2. Nonterminal nodes $\mathbf{v} = \langle v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k\rangle$, where $v \in \mathcal{V}$, $k = |\mathcal{D}_v| - 1$, children $\chi_0, \ldots, \chi_k$ are terminal or non terminal nodes of $\mathcal{E}$ and $w_0, \ldots, w_k \in T$ and $\bigwedge_{i=0,\ldots,k} w_i = \mathrm{e}$.

The elements from $\mathbf{v}$ are referred to as $v(\mathbf{v})$, $\chi_i(\mathbf{v})$ and $w_i(\mathbf{v})$. Edges between nodes and their children are implicitly given by the nonterminal nodes and the weight of an edge from $\mathbf{v}$ to $\chi_i(\mathbf{v})$ is $w_i(\mathbf{v})$. Note that this is a real generalization of EVMDDs over groups (Roux and Siminiceanu, 2010).

An EVMDD over a monoid $M = \langle T, \bullet, \mathrm{e}\rangle$ and variables $\mathcal{V}$ denotes a function $\mathcal{S} \to T$ where $\mathcal{S}$ is a set of states, where each state defines a unique valuation of variables $v \in \mathcal{V}$. Evaluating the EVMDD for a given state $s \in \mathcal{S}$ is done by following the unique path given by the values of $v$ and applying the operator $\bullet$ to the encountered edge labels. For an EVMDD over the monoid $M = \langle \mathbb{N}, +, 0\rangle$ this means adding up all values in the edge labels.

**Definition 10.** An EVMDD $\mathcal{E} = \langle \kappa, \mathbf{f}\rangle$ over a MCMM $M = \langle T, \bullet, \mathrm{e}\rangle$ and variables $\mathcal{V}$ represents the function $\kappa + f$ from states over $\mathcal{V}$ to $T$ where $f$ is the function denoted by $\mathbf{f}$. The terminal node **0** denotes the constant function $\mathrm{e}$, and a node $\mathbf{v} = \langle v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k\rangle$ represents the function $f(s) = w_{s(\mathbf{v})} + f_{s(\mathbf{v})}(s)$, where $f_{s(\mathbf{v})}$ is the function represented by the child $\chi_{s(\mathbf{v})}$. In the rest of this thesis the notation $\mathcal{E}(s)$ representing $\kappa + f(s)$ is used.

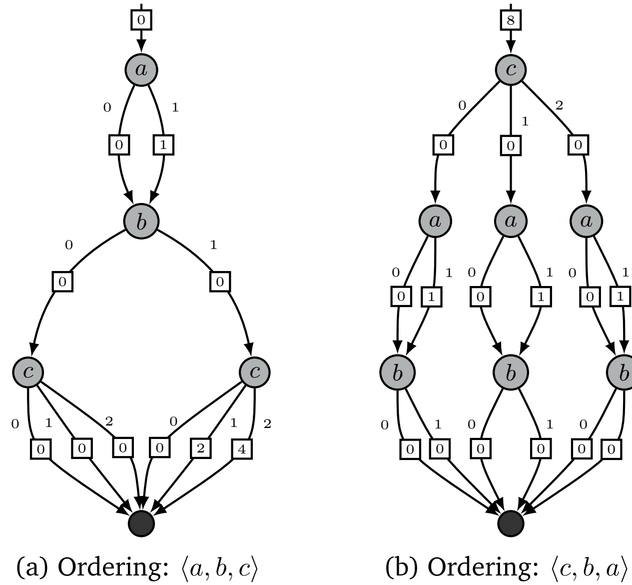
**Definition 11 (Reduced EVMDD).** An EVMDD $\mathcal{E}$ is Shannon-reduced if there exists no internal node $\mathbf{v} = \langle v, \chi_0, \ldots, \chi_k, 0 \ldots, 0\rangle$, such that $\chi_0 = \ldots = \chi_k$. If $\mathcal{E}$ is Shannon-reduced and it is isomorphism reduced, meaning that there are no two nonterminal nodes $\mathbf{v}_1, \mathbf{v}_2$ such that $\mathbf{v}_1 = \mathbf{v}_2$, then $\mathcal{E}$ is called reduced.

**Definition 12 (Quasi-Reduced EVMDD (Ciardo and Siminiceanu, 2002)).** An EVMDD $\mathcal{E}$ is quasi-reduced if on every path from root to terminal node no variable is skipped.

In the remainder of this thesis it is assumed that quasi-reduced EVMDDs are also isomorphically reduced.

**Definition 13 (Ordered EVMDD).** An EVMDD $\mathcal{E}$ is ordered if on every path from root to terminal all variables appear in the same order.

The position of a variable in the ordering is hereafter referred to by its *level*. As a shortcut the level of an EVMDD will refer to the level of the top most variable. The EVMDD in Figure 2.1a represents the arithmetic



(a) Ordering: $\langle a, b, c\rangle$ (b) Ordering: $\langle c, b, a\rangle$

Figure 2.1: Visual representation of the quasi-reduced EVMDDs over the monoid $M = \langle \mathbb{N}, +, 0\rangle$ representing the arithmetic function $a^2 + 2bc + 8$, and variable orderings (top to bottom) $\langle a, b, c\rangle$, and $\langle c, b, a\rangle$.

function $a^2 + 2bc + 8$ with the variable domains $|\mathcal{D}_a| = 2$, $|\mathcal{D}_b| = 2$, and $|\mathcal{D}_c| = 3$. The variable ordering(top to bottom) is $\langle a, b, c\rangle$. The graph is then interpreted in a way such that grey nodes are decisions over variable valuations. Each node together with its incoming edge corresponds to a sub

EVMDD $\mathcal{E} = \langle \kappa, \mathbf{f} \rangle$ where the value in the white box corresponds to $\kappa$ and the remaining graph corresponds to $\mathbf{f}$. Each outgoing edge is labeled with its corresponding variable valuation $k$ and leads to the child node $\chi_k$ with the weight $w_k$ in the white box. Given a variable valuation $a = 1, b = 1, c = 1$, by following the corresponding edges and summing up all the weights $w$ on the way, the result is $8 + 1 + 0 + 2$ which is equivalent to $a^2 + 2bc + 8 = 11$. As can be seen Figure 2.1b, representing the same arithmetic function with the same domains, but with a different variable ordering $\langle c, a, b \rangle$, has great influence on the number of nodes in the EVMDD. This in turn has a great effect on the time required for evaluating the EVMDD for a given variable valuation.

### 2.1.1 EVMDD construction

The construction of EVMDDs is based on the *apply* (Lai et al., 1996) algorithm, which was originally developed for binary decision diagrams (EVBDD) over a single monoid $\langle \mathbb{N}, \bullet, \mathrm{e}_\bullet \rangle$. The idea behind the algorithm is to take two EVBDDs $\langle c_f, f \rangle, \langle c_g, g \rangle$ and an operator $\bullet$ and returns a new EVBDD $\langle c_h, h \rangle$ such that $c_h + h = (c_f + f) \bullet (c_g + g)$, where $c_f, c_g, c_h$ are constants and $f, g, h$ are functions denoted by the EVBDD over the same monoid. This notion is extended to EVMDDs with two arbitrary monoids and an binary operator $\otimes_\mathcal{E}$. Let $\mathcal{L} = \langle L, \bullet_L, \mathrm{e}_L \rangle$, $\mathcal{R} = \langle R, \bullet_R, \mathrm{e}_R \rangle$, and $\mathcal{T} = \langle T, \bullet_T, \mathrm{e}_T \rangle$ be three MCMMs and let $\otimes : L \times R \to T$ be an operator. Furthermore, let $f : \mathcal{S} \to L$ and $g : \mathcal{S} \to R$ be two functions over a fixed state space $\mathcal{S}$, by slight abuse of notation $\otimes$ is also an operation on these functions such that $(f \otimes g)(s) = f(s) \otimes g(s)$. Now let $\mathcal{E}_{(.)}$ be the construction of an reduced ordered EVMDD $\mathcal{E}_f$ from a given function $f$. Let $\otimes_\mathcal{E}$ be a operation on EVMDDs that mimics the $\otimes$ such that $\mathcal{E}_{f \otimes g} = \mathcal{E}_f \otimes_\mathcal{E} \mathcal{E}_g$. This $\otimes_\mathcal{E}$ operation is achieved by the extended notion of the *apply* procedure usually referred to in the literature as *apply*$(\otimes_\mathcal{E}, \mathcal{E}_f, \mathcal{E}_g)$. This *apply* procedure produces a new EVMDD $\mathcal{E}_t$ by traversing both input EVMDDs $\mathcal{E}_f, \mathcal{E}_g$ synchronously from top to bottom, propagating edge labels down, applying *apply* recursively to sub graphs. In the base case where the EVMDDs only consist of an edge to the terminal node, the $\otimes_\mathcal{E}$ operator is applied to the edge labels such that $w_t = w_l \otimes_\mathcal{E} w_r$ with $w_l$ the incoming edge label of $\mathcal{E}_l$ and $w_r$ the incoming edge label of $\mathcal{E}_r$ . After the recursive call, the greatest lower bound $\bigwedge$ of all outgoing edge labels is subtracted from the edge labels using the $\dot{-}$ operation, and applied to the incoming edge again using the $\otimes_\mathcal{E}$ operator. This results in that for all nodes in the resulting EVMDD $\mathcal{E}_t$ the greatest lower bound of all outgoing edges is the neutral element $\mathrm{e}_t$. Finally, the Shannon-reduction is applied , which, for an EVMDD $\mathcal{E}\langle \kappa, \mathbf{f} \rangle$ with $\mathbf{f} = (v, \chi_0, \dots, \chi_{|\mathcal{D}_v|-1}, w_0, \dots, w_{|\mathcal{D}_v|-1})$ returns the EVMDD as follows:

- if $\forall 0 \leq i \leq |\mathcal{D}_v| - 1 : w_i = \mathrm{e}$ and each $\exists k : \chi_i = k$ is the same, return $\langle \mathrm{e}, \chi_k \rangle$
- else return $\mathcal{E}$

In words the Shannon-reduction returns the single sub-EVMDD if all outgoing edges lead to the same child and have the same edge weights.

If, during the recursive traversal the two EVMDDs come out of sync, and the decision variables do not match (the decision variables in the nodes have different levels), the two EVMDDs need to be aligned. This is done once to each of the input EVMDDs and is depicted in Algorithm 2 and works as follows. Taking two EVMDDs $\mathcal{E}_f = \langle \kappa_f, \mathbf{f}_f \rangle$, $\mathcal{E}_g = \langle \kappa_g, \mathbf{f}_g \rangle$ as input if the level of $\mathcal{E}_f$ is higher or equal to the level of $\mathcal{E}_g$ the input weight $\kappa_f$ of $\mathcal{E}_f$ is pushed down, thus for each edge represented as child node and weight $\langle \chi, w \rangle$ a new EVMDD with $\kappa + w$ as input weight and $\chi$ as root node is created. If however, the level of $\mathcal{E}_g$ is higher than the level of $\mathcal{E}_f$ a copy of $\mathcal{E}_f$ is created for every child in $\mathcal{E}_g$. This can be seen as simply pushing down the weights of $\mathcal{E}_f$ until the both EVMDDs are aligned. The code from this *apply* operation is depicted in Algorithm 1.

In the rest of this work, it is assumed that the functions for which the EVMDDs are constructed are given as syntactic terms such as multivariate polynomials, or logical expressions, and can therefore be represented as an abstract syntax tree or AST. For a given function $f$ over the MCMM $M = \langle T, \bullet, \mathrm{e} \rangle$ the EVMDD is generated by first creating the AST. Then, for each leaf, one of two base EVMDDs is created. One representing constant, consisting of the terminal node and an edge with the constant $\kappa$ as its edge weight $\langle \kappa, \mathbf{0} \rangle$. For an AST leaf representing a decision variable $var$, a EVMDD consisting of a incoming edge with edge weight $\mathrm{e}$ leading to a decision node representing the variable $v$, and an edge for each domain value $d \in \mathcal{D}_v$ of $v$ with the domain value as edge weight leading to the terminal node is created $(\kappa, \mathbf{v})$. The decision node hereby consist of $\mathbf{v} = \langle v, \mathbf{0}, \ldots, \mathbf{0}, d_0, \ldots, d_{|\mathcal{D}_v|-1} \rangle$. Then these basic EVMDDs are combined using the *apply* operator and operators specified by the internal AST nodes.

**Example 2 (EVMDD construction).** Let $f = a^2 + 2bc + 8$ be a function over the monoid $M = \langle \mathbb{N}, +, 0 \rangle$ with variable domains $\mathcal{D}_a = \{0, 1\}$,$\mathcal{D}_b = \{0, 1\}$, and $\mathcal{D}_c = \{0, 1, 2\}$ and ordering $(a, b, c)$. First the AST is created as depicted in Figure 2.2. Then for each leaf of the AST, a base EVMDD is created by Algorithm 3 and Algorithm 4 as shown in Figure 2.3. Then, for each inner node of the AST, the EVMDDs from the child nodes are merged using the *apply* algorithm together with the operator from the AST node. For the inner node representing $b * c$, the two EVMDDs do not have the same level, therefore, they must first be aligned by Algorithm 2 called once for each direction (*align*($\mathcal{E}_b$,$\mathcal{E}_c$) and *align*($\mathcal{E}_c$,$\mathcal{E}_b$)). The results of the alignment

**Algorithm 1:** APPLY algorithm for two EVMDDs

**input :** Two EVMDDs $\mathcal{E}_f = \langle \kappa_f, \mathbf{f}_f \rangle$, $\mathcal{E}_g = \langle \kappa_g \mathbf{f}_g \rangle$, and an operator $\bullet$
**output:** A third EVMDD $\mathcal{E}$ such that $\mathcal{E} = \mathcal{E}_f \bullet \mathcal{E}_g = \mathcal{E}_{f \bullet g}$

```
1
2  // Terminal Case
3  if terminal(E_f, E_g, •):
4      return ⟨κ_f • κ_g, 0⟩
5
6  // Check if sub EVMDD already created (Isomorphism
       reduction)
7  if inCache(E_f, E_g, •)):
8      return cached EVMDD
9
10 // Allign the EVMDDs to have the same current level
11 maxlevel = max(level(f),level(g))
12 v = variableAtIndex(maxLevel)
13 d = |D_v| − 1
14 C_f = align(E_f,E_g)
15 C_g = align(E_g,E_f)
16
17 // Recursively call APPLY for all children
18 for i = 0 in d:
19     κ_i, f_i = APPLY(C_f^i,C_g^i,•)
20
21 // Calculate the greatest lower bound m and create a new
       EVMDD with κ = m and the weights to the children are
       κ_i ∸ m
22 m = ⋀(κ_1, ..., κ_d)
23 E = ⟨m, (v, f_0, ..., f_d, κ_0 ∸ m, ..., κ_d ∸ m)⟩
24 // Apply shannon reduction
25 E = shannon-reduce(E)
26
27 return E
```

**Algorithm 2:** ALIGN algorithm for two EVMDDs with same variable ordering

**input :** Two EVMDDs $\mathcal{E}_f = \langle \kappa_f, \mathbf{f}_f \rangle$, $\mathcal{E}_g = \langle \kappa_g \mathbf{f}_g \rangle$
**output:** A set of EVMDDs with the same level as $\mathcal{E}_f$

1
2 `//` $\mathcal{E}_f$ `has the same or higher variable level`
3 **if** *level*($\mathcal{E}_f$)$\geq$ *level*($\mathcal{E}_g$)**:**
4     `// Create an EVMDD` $\mathcal{E} = \langle \kappa, \mathbf{f} \rangle$ `for each child` $\mathbf{f}_i$ `of` $\mathbf{f}_f = (v, \mathbf{f}_0, \ldots, \mathbf{f}_d, w_0, \ldots, w_d)$
5     **return** $\langle \kappa_f \bullet w_0, \mathbf{f}_0 \rangle$,..., $\langle \kappa_f \bullet w_d, \mathbf{f}_d \rangle$
6 **else:**
7     `// create copies of` $\mathcal{E}_f$ `for each child of` $\mathbf{f}_g = (v, \mathbf{f}_0, \ldots, \mathbf{f}_k, w_0, \ldots, w_k)$
8     **return** $\underbrace{\mathcal{E}_f, \ldots, \mathcal{E}_f}_{k-times}$

**Algorithm 3:** Create an EVMDD for a constant

**input :** A constant $\kappa$
**output:** A EVMDD representing the constant $\kappa$

1 **return** $\langle \kappa, \mathbf{0} \rangle$

**Algorithm 4:** Create an EVMDD for a variable

**input :** A variable description $v$, and a domain $\mathcal{D} = \{d_0, \ldots, d_n\}$
**output:** A EVMDD representing the variable $v$

1 `// Create a child EVMDD for each value in` $\mathcal{D}$ `that leads to the terminal node`
2 **return** $\langle \mathrm{e}, (v, \underbrace{\mathbf{0}, \ldots, \mathbf{0}}_{n-times}, d_0, \ldots, d_n) \rangle$

is shown in Figure 2.4. The *apply* algorithm is called recursively until the terminal case is reached, which after aligning looks as shown in Figure 2.5. Then for every edge leading to the terminal node the $\times$ operator is applied pairwise to the edges $(0{,}0 \times 1{,}0 \times 2)$. Recovering from the recursive call and pushing the greatest lower bound $(0)$ up and repeating for the $1$ edge of $\mathcal{E}_b$ the resulting EVMDD is shown in Figure 2.6a. After repeating this process for every internal node of the AST, the resulting EVMDD is shown in Figure 2.6b.

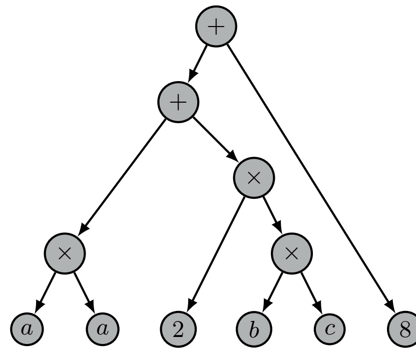



Figure 2.2: The AST over the expression $a^2 + 2bc + 8$

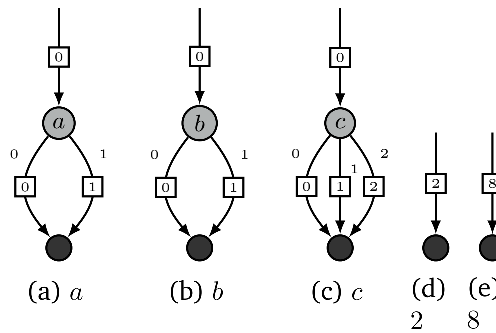
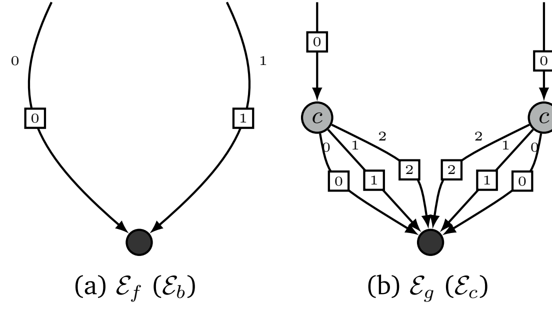



Figure 2.3: Visual representation of the EVMDDs created by Algorithm 3 and Algorithm 4.

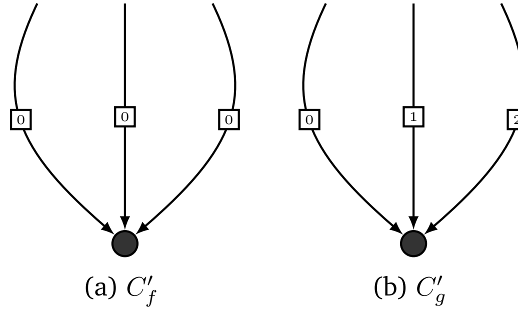
### 2.1.2 EVMDD evaluation

Given a EVMDD $\mathcal{E}\langle\kappa, \mathbf{f}\rangle$ over a monoid $M = \langle T, \bullet, \mathrm{e}\rangle$ evaluating over a given state $s$ is performed by traversing the graph top to bottom accumulating the edge labels by applying the operator $\bullet$, denoted as $\mathcal{E}(s)$. For arithmetic expressions this corresponds to adding up all edge values on the path from $\kappa$ to the terminal node. For every node $\mathbf{v} = (v, \mathbf{f}_0, \ldots, \mathbf{f}_k, w_0, \ldots, w_k)$ evaluate the child $\mathbf{f}_i$ such that $i = s[v]$ and return $\kappa + \textit{evaluate}(\langle w_i, \mathbf{f}_i\rangle, s)$.




(a) $\mathcal{E}_f$ ($\mathcal{E}_b$) (b) $\mathcal{E}_g$ ($\mathcal{E}_c$)

Figure 2.4: $\mathcal{E}_b$ and $\mathcal{E}_c$ aligned where the inputs are $\mathcal{E}_f = \mathcal{E}_b$ and $\mathcal{E}_g = \mathcal{E}_c$




(a) $C'_f$ (b) $C'_g$

Figure 2.5: Alignment for the last recursive call of *APPLY* with $\mathcal{E}_b$ and $\mathcal{E}_c$ and the operator $\times$ on the $0$ edge of $\mathcal{E}_b$

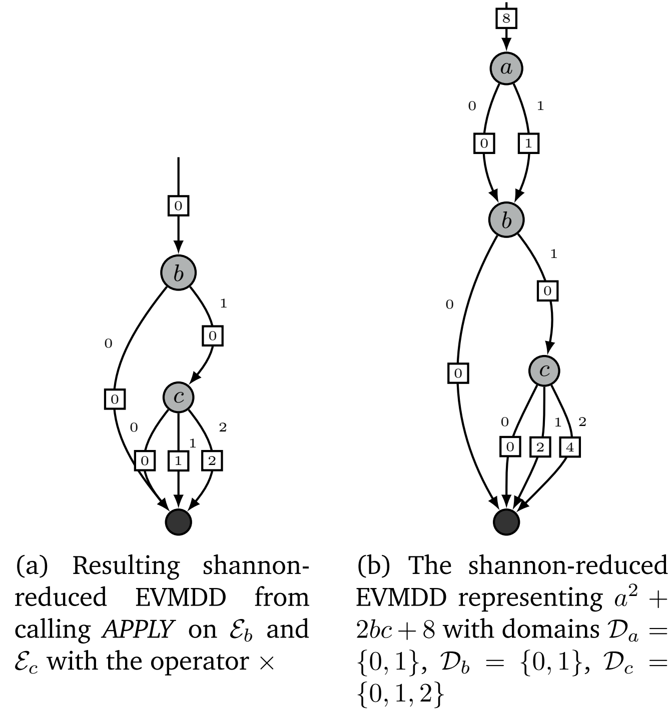
**Example 3 (EVMDD evaluation).** Recalling Example 2 evaluating the expression $f = a^2 + 2bc + 8$ represented by the EVMDD in Figure 2.6b over the state $s = \{a = 1, b = 1, c = 1\}$ is achieved by traversing the EVMDD top to bottom following the corresponding edges. This corresponds to the path highlighted red in Figure 2.7 resulting in the summation $8+1+0+2$, which corresponds to $1^2 + 2 * 1 * 1 + 8$.

**Theorem 1.** Let $\mathcal{E}_l = \langle \kappa_l, \mathbf{f}_l \rangle$, and $\mathcal{E}_r = \langle \kappa_r, \mathbf{f}_r \rangle$ be two EVMDD and $\bullet$ the applied operator. Then $\mathcal{E}_{\{l \bullet r\}} = \mathit{apply}(\mathcal{E}_l, \mathcal{E}_r, \bullet)$

*Proof.* The proof over the correctness goes inductively over the base case with $\mathbf{f}_l = \mathbf{0}$ and $\mathbf{f}_r = \mathbf{0}$ where both EVMDDs represent a constant function. In this case a new base EVMDD $\langle \kappa_l \bullet \kappa_r, \mathbf{0} \rangle$ is created. For the case of two EVMDDs consisting of one decision node only, such that

$$\mathbf{f}_l = (v, \mathbf{0}, \ldots, \mathbf{0}, w_{l_0}, \ldots, w_{l_{|\mathcal{D}_v|-1}})$$
$$\mathbf{f}_r = (v, \mathbf{0}, \ldots, \mathbf{0}, w_{r_0}, \ldots, w_{r_{|\mathcal{D}_v|-1}})$$

Then the resulting EVMDD $\mathcal{E}$ is a result of the pairwise application of $\bullet$ such



(a) Resulting shannon-reduced EVMDD from calling *APPLY* on $\mathcal{E}_b$ and $\mathcal{E}_c$ with the operator $\times$

(b) The shannon-reduced EVMDD representing $a^2 + 2bc + 8$ with domains $\mathcal{D}_a = \{0, 1\}$, $\mathcal{D}_b = \{0, 1\}$, $\mathcal{D}_c = \{0, 1, 2\}$

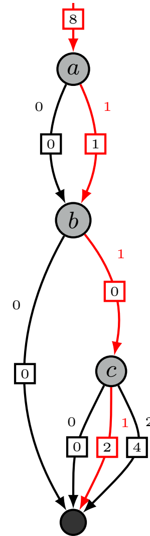


Figure 2.7: Evaluation of the EVMDD over state $s = \{a = 1, b = 1, c = 1\}$

that $\mathcal{E} = \langle \kappa_l \bullet \kappa_r, \mathbf{f} \rangle$ with

$$\mathbf{f}_l = (v, \mathbf{0}, \ldots, \mathbf{0}, w_{l_0} \bullet w_{r_0}, \ldots, w_{l_{|\mathcal{D}v|-1}} \bullet w_{r_{|\mathcal{D}v|-1}})$$

For EVMDDs consisting of multiple variables the EVMDDs are assumed to be

**Algorithm 5:** Evaluate an EVMDD on a given state

**input :** An EVMDD $\mathcal{E} = \langle \kappa, \mathbf{f} \rangle$ over a monoid $M = \langle T, \bullet, \mathrm{e} \rangle$, and a state $s$ over variables $\mathcal{V}$
**output:** An element $m$ from $T$

```
1
2 // Terminal Case
3 if f = 0:
4     return κ
5
6 // Evaluate the child f_i of E with
      f = (v, f_0, ..., f_k, w_0, ..., w_k) such that i = s[v]
7 i = s[v]
8 return κ• evaluate(⟨w_i, f_i⟩, s)
```

aligned as this can be achieved using the *align* algorithm. While traversing both EVMDDs the weights are pushed down to the base case by applying the $\bullet$. Then the base EVMDDS are handled by the base-case. Finally the resulting edge weights are distributed over the edges of the EVMDD by pulling up the greatest lower bound of all outgoing edge weights at every node using the $\dot{-}$ operator on the edge weights $w$ and the $\bullet$ operator on the incoming edge-weights $\kappa$. □

**Theorem 2 (Correctness of construction).** Let $\mathcal{E}$ be an EVMDD over the monoid $M = \langle T, \bullet, \mathrm{e} \rangle$, constructed as shown above, representing the function $f$, and a given state space $\mathcal{S}$, then $f(s) = \mathcal{E}(s)$ for every $s \in \mathcal{S}$.

*Proof.* By Definition 10 an EVMDD $\mathcal{E} = \langle \kappa, \mathbf{f} \rangle$ represents the function $f$ over the MCMM $M = \langle T, \bullet, \mathrm{e} \rangle$. It is easy to see that for the base cases where $f$ is either a constant or a single variable this is true, as they are represented by the EVMDDs $\langle \kappa, \mathbf{0} \rangle$ for constants and $\langle \mathrm{e}, (v, \mathbf{0}, \ldots, \mathbf{0}, d_0, \ldots, d_{|\mathcal{D}_v|-1}) \rangle$. The inductive case where two EVMDDs $\mathcal{E}_l$,$\mathcal{E}_r$ are combined using an operator $\otimes$ defined on $T$ follows the correctness of the *apply* algorithm (Lai et al., 1996). Hereby, the *apply* algorithm is applied for every internal AST node $n$ which represents an operator in $f$ on the EVMDDs created by the child nodes of $n$. □

**Definition 14 (Canonicity of a node).** A node $\mathbf{v}$ in $\mathcal{E}$ is canonical iff $\mathbf{v}$ is the terminal node $\mathbf{0}$, or has the form $\mathbf{v} = (v, \chi_0, \ldots, \chi_{|\mathcal{D}_v|-1}, w_0, \ldots, w_{|\mathcal{D}_v|-1})$ such that the greatest lower bound over all outgoing edge weights is the neutral element:

$$\bigwedge_{i=0}^{|\mathcal{D}_v|-1} w_i = \mathrm{e} \; .$$

**Definition 15 (Canonicity).** An EVMDD $\mathcal{E} = \langle \kappa, \mathbf{f} \rangle$ is canonical iff all nodes are canonical.

**Theorem 3.** All EMVDDs constructed by above process are canonical.

*Proof.* In above construction all nodes are canonical, as for every node $\mathbf{v} = (v, \chi_0, \ldots, \chi_{|\mathcal{D}_v|-1}, w_0, \ldots, w_{|\mathcal{D}_v|-1})$ in $\mathbf{f}$ the greatest lower bound

$$\bigwedge_{i=0}^{|\mathcal{D}_v|-1} w_i = m$$

over all outgoing edge weights is "removed" from the edge weights by $w_i \dot{-} m$ (Algorithm 1 line 23) and pushed up to the incoming edge weight $\kappa$. Therefore the resulting EVMDD is also canonical. □

**Theorem 4 (Reduced).** All EMVDDs constructed by above process are isomorphism and shannon-reduced.

*Proof.* Note that the definition of the EVMDDs does not allow for multiple identical nodes, thus ensuring isomorphism, reduction. However during the execution of the algorithm such duplication can occur and thus must be accounted for. This is achieved by caching already created nodes and doing a simple lookup for already stored subgraphs (Algorithm 1 line 7). The shannon-reduction is achieved by calling the shannon-reduce algorithm in Algorithm 1 line 25 for every created sub-EVMDD. □

**Theorem 5 (Ordered).** All EMVDDs constructed by above process are ordered.

*Proof.* By requiring a fixed variable ordering as input to the construction algorithm and the alignment of both EVMDDs in the *apply* algorithm (Algorithm 1 line 14 and line 15) ordering is ensured during construction of the EVMDD. □

**Definition 16 (Redundancy freedom).** The EVMDD $\mathcal{E} = \langle \kappa, \mathbf{f} \rangle$ over MCMM $M = \langle T, \bullet, \mathrm{e} \rangle$ with the monus operator $\bullet$ is redundancy free iff for every pair of edge weights $(a, b)$ on any path from root to terminal node the condition $(a \bullet b) \dot{-} b = a$ holds.

**Theorem 6 (Redundancy freedom).** All EMVDDs constructed by above process are redundancy free.

*Proof.* In Algorithm 2 line 5 weights are pushed down towards the terminal node, where the operator $\bullet$ is applied to two consecutive edges $a, b$. In the recovery of the recursion, the greatest lower bound over all outgoing edge weights $\bigwedge_{\{w_0, \ldots, w_{|\mathcal{D}_v|-1}\}}$ is pushed back up using the $\dot{-}$ operator on the

outgoing edge weights(see Canonicity). This ensures that $(a \bullet b) \dot{-} b = a$ holds for every pair of edges. $\square$

**Theorem 7 (Uniqueness).** Let $\mathcal{E}_f = \langle \kappa_f, \mathbf{f}_f \rangle$, $\mathcal{E}_g = \langle \kappa_g, \mathbf{f}_g \rangle$ be reduced ordered redundancy free EVMDDs over the same monoid $M = \langle T, \bullet, \mathrm{e} \rangle$ and a fixed variable ordering *level*, representing the functions $f : \mathcal{S} \rightarrow T$ and $g : \mathcal{S} \rightarrow T$ respectively and where constructed as described above. Then $f = g$ if and only if $\mathcal{E}_f$ and $\mathcal{E}_g$ are isomorphic.

The following lemma and proof is a generalization of the one given by Geißer (2018) which considers the special case of the MCMM $M = \langle \mathbb{Q}^+, +, 0 \rangle$ and strongly follows the same structure.

**Lemma 1.** Let $\mathbf{v}_f$ and $\mathbf{v}_g$ be two reduced, ordered, redundancy free EVMDD nodes with the same level, resulting from above construction. Furthermore, let $f : \mathcal{S} \rightarrow T$ be the function denoted by $\mathbf{v}_f$ and $g : \mathcal{S} \rightarrow T$ be the functions denoted by $\mathbf{v}_g$, then $f = g$ iff $\mathbf{v}_f$ and $\mathbf{v}_g$ are isomorphic.

*Proof.* **Base case:** Let both nodes $\mathbf{v}_f$ and $\mathbf{v}_g$ be terminal nodes with

$$\mathit{level}(\mathbf{v}_f) = \mathit{level}(\mathbf{v}_g) = 0 \; .$$

Then the encoded function is $f = g = \mathrm{e}$, and by definition the nodes are isomorphic.
**Inductive case** *level*$(\mathbf{v}_f)$=*level*$(\mathbf{v}_g)$= $l > 0$: Assume Lemma 1 holds for all nodes $\mathbf{v}$ with *level*$(\mathbf{v}) \leq l - 1$. Let $\mathcal{S}$ be the set of states $s$ such that $s(v) = i$ with $i = \{0, \ldots, k\}$ then the notation $\chi_f$ for the function denoted by $\chi_{f,i}$, $w_f$ for $w_{f,i}$, $\chi_g$ for the function denoted by $\chi_{g,i}$, and $w_g$ for $w_{g,i}$ are used. First assume $f = g$, then $\chi_f(s) \bullet w_f = \chi_g(s) \bullet w_g$ for $s \in \mathcal{S}$, as $\chi_f(s) \bullet w_f$ denotes $f$ and $\chi_g(s) \bullet w_g$ denotes $g$. Since every node is canonical (Theorem 3) and the EVMDD is redundancy free (Theorem 6), it holds that $\bigwedge_{s \in \mathcal{S}} \chi_f(s) = \mathrm{e} = \bigwedge_{s \in \mathcal{S}} \chi_g(s)$. From this follows that $\bigwedge_{s \in \mathcal{S}} \chi_f(s) \bullet w_f = w_f$ and $\bigwedge_{s \in \mathcal{S}} \chi_g(s) \bullet w_g = w_g$. This implies that $\chi_f = \chi_g$ and by the induction hypothesis that $\chi_{f,i}$ and $\chi_{g,i}$ are isomorph. As $i$ can be have any arbitrary value in $0 \leq i \leq k$ it follows that $\mathbf{v}_f$ and $\mathbf{v}_g$ are isomorph. Secondly, assume $\mathbf{v}_f$ and $\mathbf{v}_g$ are isomorphic, then $w_f = w_g$. Since $\chi_{f,i}$ and $\chi_{g,i}$ are isomorphic, it follows by the induction hypothesis that $\chi_f = \chi_g$ and thus $\chi_f \bullet w_f = \chi_g \bullet w_g$. As $i$ is arbitrary it follows that $f = g$. $\square$

*Proof of Theorem 7.* Assume the $f = g$. As every node is canonical and the EVMDD is redundancy free, it follows that there exists a state $s$ such that $\bigwedge_{s \in \mathcal{S}} \mathbf{f}_f(s) \bullet \kappa_f = f(s) = g(s) = \bigwedge_{s \in \mathcal{S}} \mathbf{f}_g(s) \bullet \kappa_g$. Therefore, $\mathbf{f}_f$ and $\mathbf{f}_g$ must encode the same function $f \dot{-} \kappa_f$ and from Lemma 1 follows that they are isomorph.
Now assume $\kappa_f = \kappa_g$, and $\mathbf{f}_f$ and $\mathbf{f}_g$ are isomorphic. $\mathcal{E}_f$ encodes the function $f = \kappa_f \bullet f'$ where $f'$ is the function encoded by $\mathbf{f}_f$ and $\mathcal{E}_g$ encodes the

function $g = \kappa_g \bullet g'$ where $g'$ is the function encoded by $\mathbf{f}_g$. Following Lemma 1 $\mathbf{f}_f$ and $\mathbf{f}_g$ encode the same functions, thus $f' = g'$, and since $\kappa_f = \kappa_g$, $g = \kappa_g \bullet \kappa'_g = \kappa_f \bullet \kappa'_f = f$. □

## 2.2 Linear Temporal Logic

Linear Temporal Logic (LTL) (Pnueli, 1977) allows reasoning over discrete time series or state trajectories. For this a LTL formula $\varphi$ is interpreted over a (infinite) sequence of states. In the general literature these states are also called worlds w and the state trajectories infinite discrete time series. In addition to the logic operators $\wedge, \neg$, LTL adds the unary operator $\bigcirc$ and the binary operator $\mathcal{U}$. The unary *NEXT* operator $\bigcirc\varphi$ states that in the next world $\varphi$ must hold. *UNTIL* $\varphi\mathcal{U}\psi$ on the other hand states that in the future $\psi$ will hold and until then $\varphi$ holds. Let $\mu$ be a infinite state sequence $\mu = (s_0, s_1, \ldots)$, then the truth value of a given LTL formula $\varphi$ over a set of propositional symbols P at a given time instance $i = 0 - \infty$ written as $\mu, i \models \varphi$ is given by the following interpretation:

$$
\begin{aligned}
&\mu, i \models p \in \mathsf{P} \text{ iff } p \in \mu(i) \\
&\mu, i \models \varphi \wedge \psi \text{ iff } \mu, i \models \varphi \text{ and } \mu, i \models \psi \\
&\mu, i \models \neg\varphi \text{ iff not} \mu, i \models \varphi \\
&\mu, i \models \bigcirc \varphi \text{ iff } \mu, i+1 \models \varphi \\
&\mu, i \models \varphi\mathcal{U}\psi \text{ iff } \exists j \geq i,\ \mu, j \models \psi \text{ and } \forall k | i \leq k < j,\ \mu, k \models \varphi
\end{aligned}
$$

Additionally to the standard abbreviations $\vee$ (or), $\rightarrow$(implies), $\top$(true), and $\bot$(false), it is common to use the following abbreviations:

$$
\begin{aligned}
\Diamond\varphi &= \top\mathcal{U}\varphi, \varphi \text{ will eventually be true (finally } \varphi) \\
\Box\varphi &= \neg\Diamond\neg\varphi, \varphi \text{ will be true in every future world. (globally } \varphi) \\
\varphi\mathcal{R}\psi &= \neg(\neg\varphi\mathcal{U}\neg\psi), \psi \text{ holds until } \varphi \text{ holds, or forever. (} \varphi \text{ releases } \psi) \\
\varphi\mathcal{W}\psi &= (\varphi\mathcal{U}\psi \vee \Box\varphi), \varphi \text{ holds until } \psi \text{ or forever(} \varphi \text{ weak until } \psi).
\end{aligned}
$$

**Definition 17 (Büchi automaton (BA)).** A Büchi automaton (Büchi, 1990) is a tuple $\mathscr{B} = (\mathsf{Q}, \Sigma, \Delta, Q_0, F)$:

- Q the finite set of states.
- $\Sigma$ the finite alphabet.
- $\Delta : \mathsf{Q} \times \Sigma \rightarrow 2^{\mathsf{Q}}$ a transition relation.
- $Q_0$ the set of initial state.
- $F \subseteq \mathsf{Q}$ the set of accepting states.

An input $\alpha$ for $\mathscr{B}$ is then an infinite sequence over the alphabet $\Sigma$, $\alpha = (a_1, a_2, a_3, \ldots)$. A run of $\mathscr{B}$ on an input $\alpha$ is an infinite sequence $\rho = (r_0, r_1, r_1, r_2, \ldots)$ of states if $r_0 \in Q_0$ and for each $r_i, i \geq 1, r_i \in \Delta(r_{i-1}, a_i)$. The automaton accepts an input word $\alpha$ if at least on state $a \in F$ is visited infinitely often by the run over the word.

Gerth et al. (1995) introduced an algorithm for creating such a Büchi automaton for a given LTL formula.

**Theorem 8.** (Gerth et al., 1995) For any LTL formula $\varphi$ over the propositions P there exists a Büchi automaton $\mathscr{B}$ as constructed by Gerth et al. (1995) that accepts exactly all the sequences over $(2^{\mathsf{P}})^{\alpha}$ that satisfy $\varphi$. □

### 2.2.1 Linear Temporal Logic on Finite Traces

$\mathsf{LTL_f}$ (De Giacomo and Vardi, 2013) extends the notion of linear temporal logic to that over finite traces, thus the sequence of worlds is finite. The formula share the same syntax but $\mathsf{LTL_f}$ adds the shortcut $\lambda = \neg \bigcirc \top$, which becomes true in the last state of the finite sequence of states. Let $\mu$ be a sequence of states then the length of the sequence is denoted as *length*$(\mu)$ and *last* $=$ *length*$(\mu) - 1$, then the interpretation for a time step $i$ $(0 \leq i \leq$ *last*$)$ is defined as:

$\mu, i \models p \in \mathsf{P}$ iff $p \in \mu(i)$
$\mu, i \models \varphi \wedge \psi$ iff $\mu, i \models \varphi$ and $\mu, i \models \psi$
$\mu, i \models \neg\varphi$ iff not $\mu, i \models \varphi$
$\mu, i \models \bigcirc \varphi$ iff $i <$ *last* and $\mu, i+1 \models \varphi$
$\mu, i \models \varphi \mathcal{U} \psi$ iff $\exists j | i \leq j \leq$ *last*, $\mu, j \models \psi$ and $\forall k | i \leq k < j$, $\mu, k \models \varphi$
$\mu, i \models \lambda\varphi$ iff $i =$ *last* and $\mu, i \models \varphi$

Additionally one more $\mathsf{LTL_f}$ specific abbreviation is introduced:
$\bullet\varphi = \neg \bigcirc \neg\varphi$ if $\lambda$ does not hold in the next state then $\varphi$ must hold in the next state (weak next $\varphi$).

**Definition 18 (Nondeterministic finite automaton (NFA)).** A NFA is a tuple $\mathcal{N} = \langle \mathsf{Q}, \Sigma, \Delta, q_0, F \rangle$ consisting of:

- $\mathsf{Q}$ the set of states.
- $\Sigma$ the alphabet consisting of a finite set of input symbols.
- $\Delta : \mathsf{Q} \times \Sigma \rightarrow 2^{\mathsf{Q}}$ the transition function.
- $q_0$ the initial state.
- $F \subseteq \mathsf{Q}$ the set of accepting states.

Let $\mu = a_1, \ldots, a_n$ be a word over the alphabet $\Sigma$, then the automaton $\mathcal{N}$ accepts the word $\omega$ if there exists a sequence of states $r_0, \ldots, r_n$ in Q with $r_0 = q_0$, $r_{i+1} \in \Delta(r_i, a_{i+1})$ for $i = 0, \ldots, n-1$, and $r_n \in F$. Thus, the

difference to above Büchi automata is the acceptance condition, where here only the last state must be accepting.

De Giacomo et al. (2014) provide an algorithm for constructing such a NFA for any $\mathsf{LTL_f}$ formula.

**Theorem 9.** Let $\varphi$ be an $\mathsf{LTL_f}$ formula and $\mathcal{N}_\varphi$ the NFA constructed as described in De Giacomo et al. (2014), then for every finite trace $\pi$ over $\Sigma$ satisfies $\varphi$, iff the automaton $\mathcal{N}_\varphi$ accepts the input $\pi$ (De Giacomo et al., 2014).

# Chapter 3

# Planning Foundations

The main target of planning is to decide which action to take in a given situation. The main application areas hereby cover high-level planning of intelligent agents, autonomous systems, and problem solving. In contrast to specialized algorithms, planning utilizes a general-purpose problem description, and special algorithms for solving problems formalized in this fashion. Planning can be categorized depending on if the actions are deterministic, non-deterministic or probabilistic, and if the state of the world is fully or only partially observable, thus if the agent has full knowledge of its context. Often planning with deterministic actions in a fully observable setting is called classical planning. In this setting the agent starts in a fully specified initial state and has a goal description, together with actions it can take to reach the goal. The aim of planning is then to find a, possibly optimal, sequence of actions that changes the initial state step by step to reach the goal. When non-deterministic actions are present, planning produces a policy stating in which state which action should be applied, as the nondeterminism introduces branchings during execution.

This work focuses on classical planning (deterministic actions, fully observable world) and FOND planning (Fully observable non-deterministic). The following section formalizes planning using the classical setting. Then this setting is extended to add expressiveness. Section 3.2 introduces state-dependent action costs, where the cost of an action depends on the state in which it is executed. Section 4.1 then adds conditional effects, similar to state-dependent action costs, only that the outcome of an action depends on the state in which it is executed. The combination of conditional effects and state-dependent action costs is then discussed in Section 4.2. Section 5 then adds optional constraints towards the plan, adding controll over how goals are achieved. Finally, Section 3.3 introduces non-determinism, where the outcome of an action is not deterministic.

## 3.1 Classical Planning

In classical planning full knowledge of the world is assumed, thus the planner has full knowledge over the initial state, the current state, the goal state, and the precondition and outcome of each action. No unforeseen action can manipulate any part of the planning task. Any interaction with a changing environment is disregarded in this setting. Additionally every action is deterministic, meaning that they have one distinct set of effects in every state.

A fact is a pair $(v, d)$ where $v \in \mathcal{V}$ and $d \in \mathcal{D}_v$ and $\mathcal{F}$ is the set of all facts. $\mathcal{F}$ is considered consistent if no two facts $v = d$ and $v = d'$ such that $d \neq d'$ are in $\mathcal{F}$.

**Definition 19 (Partial Variable Assignment).** Given the set of all facts $\mathcal{F}$, a *partial variable assignment* is a consistent set $\xi \subseteq \mathcal{F}$ assigning values to variables for some variables $v \in \mathcal{V}$.

**Definition 20 (State).** If the *partial variable assignment* $s$ assigns a value to each $v \in \mathcal{V}$, then $s$ is called a state.

Let $\mathcal{S}$ denote the set of all states in $\Pi$, and $s(v)$ denote the value of the variable $v$ in state $s$.

**Definition 21 (Action).** An action $a \in \mathcal{A}$ is a tuple $a = \langle pre, eff\rangle$ of preconditions and effects, where *pre* is a *partial variable assignment* and $eff = \{eff_0, \dots eff_n\}$ is a conjunction of conditional effects $eff_i = \langle \varphi_i \rhd (v := d)\rangle$ where the effect condition $\varphi_i$ is a propositional formula over $\mathcal{V}$ and $v := d$ a fact with $v \in \mathcal{V}$ and $d \in \mathcal{D}_v$. An action $a$ is applicable in a state $s$ if $s \models pre$. The changes obtained by applying $a$ in $s$ is then defined in the form of the changeset (Rintanen, 2003) over the actions effect.

If every atomic effect in *eff* occurs at most once, and there exist no nested effects or conditions then this is called *effect normal form* or ENF. An additional requirement towards conditional effects is that they are non contradicting thus if there are two effects $e, e' \in eff$ with $e = \varphi_i \rhd (w := d')$ and $e' = \varphi_j \rhd (w := d'')$ and $d' \neq d''$ then there exists no reachable state $s$ such that $s \models \varphi_i \wedge s \models \varphi_j$

**Definition 22 (Classical Planning Task).** A classical planning task $\Pi$ is a tuple $\langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a\rangle$, where $\mathcal{V}$ is a set of multi valued variables each with a finite domain $\mathcal{D}_v$, $\mathcal{A}$ is a set of actions, $s_0$ the initial state, and $s_\star$ a goal condition, and $c : \mathcal{A} \to \mathbb{N}$ a mapping function from actions to costs.

**Definition 23 (Changeset).** Let $s$ be a state in $\mathcal{S}$ and $e$ an effect in *ENF* over $v \in \mathcal{V}$. Then the changeset denoted as $[eff]_s$ (effect applied in state $s$) is

1. $[eff_1 \wedge \cdots \wedge eff_n]_s = [eff_1]_s \cup \cdots \cup [eff_n]_s$ ,

2. $[\varphi \triangleright f]_s = \{f\}$ if $s \models \varphi$, and $[\varphi \triangleright f]_s = \emptyset$, otherwise.

Applying an action $a$ in state $s$ then yields a new state $s'$ where $s(v)' = [\mathit{eff}]_s(v)$ for all $v$ where $[\mathit{eff}]_s(v)$ is defined, and $s(v)' = s(v)$ for all $v$ where $[\mathit{eff}]_s(v)$ is undefined. The notation $s[a]$ for $s'$, denoting the state after applying changes from $a$ is used throughout the rest of this thesis. Additionally, let $\delta(s, (a_0, \dots, a_n))$ be the state after applying the sequence of actions $(a_0, \dots, a_n)$ in state $s$, defined as follows:

$$\delta(s, []) = s$$
$$\delta(s, [(a_1, \dots, a_n), a_0]) = \delta(s, (a_1, \dots, a_n))[a_0]$$

Let $\pi = (a_0, \dots, a_{n-1})$ be a sequence of actions from the set of actions $\mathcal{A}$, then $\pi$ is a plan for $\Pi$ if $a_0$ is applicable in $s_0$ and there exists a set of states $s_i \dots s_n$ such that $a_i$ is applicable in $s_i$ and $s_{i+1} = s_i[a_i]$ for all $i = 0, \dots, n-1$, and $s_n$ is a goal state. A state $s$ is a goal state if $s \models s_\star$. The corresponding sequence of states $\mu_\pi = (s_0, s_1, \dots, s_n)$ created by sequentially applying actions $a$ in $\pi$ with $s_{i+1} = s_i[a_i]$ is then called the *state trajectory* induced by $\pi$.

**Plan quality** Two kinds of planning problems can be distinguished: Satisficing and optimal planning. In satisficing planing, the aim is to find a plan that satisfies the goal condition, plan length or plan costs are not considered. Plan length corresponds to the number of actions taken to reach the goal state $s_n \models s_\star$ from the initial state $s_0$. Given a plan $\pi = (a_0, \dots, a_n)$ The plan cost is then defined as $c = \sum_{i=0}^{n} c_{a_i}$. In optimal planning the goal is to find the *optimal* plan regarding some metric.

**Definition 24 (Satisficing Plan).** A satisficing plan is any plan that reaches the goal state. The quality of the plan is given by total cost $c(\pi)$ of the plan.

**Definition 25 (Optimal Plan).** A plan $\pi$ for planning task $\Pi$ with some $c(\pi)$ is optimal if there exists no other plan $\pi'$ for the same planning task $\Pi$ with $c(\pi')$ such that $c(\pi') < c(\pi)$.

**Example 4 (Logistics Domain Example).** Illustrated in Figure 3.1 is a short example of the logistics domain, with $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a \rangle$ formalized as follows:

- There exist 3 locations {Freiburg, Vienna, Graz} 2 packages {PkgRed, PkgGreen} and one truck.
- For every package $p \in${PkgRed, PkgGreen} a variable *atLocPkg*$(p)$ with domain {Freiburg, Vienna ,Graz, Truck} is created.
- For the truck a variable *atLocTruck* with domain {Freiburg, Vienna, Graz} is created.

The possible actions $\mathcal{A}$ are then:

- For every “connected” location a move action is created:
  - *move(Freiburg,Vienna)* = $\langle$*atLocTruck* = *Freiburg*, *atLocTruck*=*Vienna*$\rangle$
  - *move(Vienna,Freiburg)* = $\langle$*atLocTruck* = *Vienna*, *atLocTruck*=*Freiburg*$\rangle$
  - *move(Vienna,Graz)* = $\langle$*atLocTruck* = *Vienna*, *atLocTruck*=*Graz*$\rangle$
  - *move(Graz,Vienna)* = $\langle$*atLocTruck* = *Graz*, *atLocTruck*=*Vienna*$\rangle$
- For every package and location pair a pickup and drop off action is created:
  - *pickup(PkgRed,Freiburg)* = $\langle$*atLocTruck* = *Freiburg*$\wedge$ *atLocPkg(PkgRed)* = *Freiburg*, *atLocPkg(PkgRed)* = *Truck*$\rangle$
  - *dropoff(PkgRed,Freiburg)* = $\langle$*atLocTruck* = *Freiburg*$\wedge$ *atLocPkg(PkgRed)* = *Truck*, *atLocPkg(PkgRed)* = *Freiburg*$\rangle$
  - ...

All action costs $c_a$ are set to unit costs.

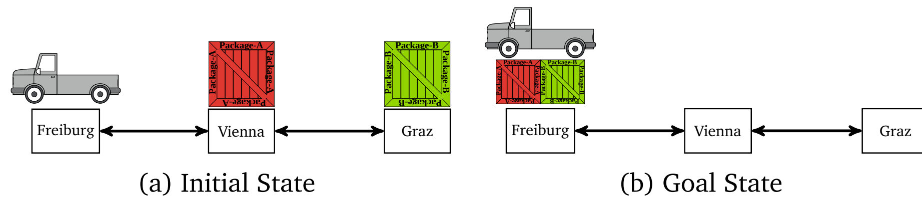
In the initial state, as shown in Figure 3.1a, the package *PkgRed* is in *Vienna* and the package *PkgGreen* is in *Graz* and the truck is in *Freiburg*. The goal is to deliver both packages to *Freiburg*, as shown in Figure 3.1b. Note that there exist multiple solutions to this task including the following:

1: *move(Freiburg,Vienna)* → *pickup(PkgRed)* → *move(Vienna,Graz)* → *pickup(PkgGreen)* → *move(Graz, Vienna)* → *move(Vienna, Freiburg)* → *dropoff(PkgRed)* → *dropoff(PkgGreen)*

2: *move(Freiburg,Vienna)* → *move(Vienna,Graz)* → *pickup(PkgGreen)* → *move(Graz, Vienna)* → *pickup(PkgRed)* → *move(Vienna, Freiburg)* → *dropoff(PkgRed)* → *dropoff(PkgGreen)*

The search tree corresponding to this task is shown in Figure 3.2. Irrelevant paths, such as moving back and forth between two locations, in the search are removed.

As shown by Bylander (1994) determining whether a plan exists is *PSPACE-complete*. Even if the restriction to the formalism is made, that actions can




(a) Initial State (b) Goal State

Figure 3.1: Logistics domain example with three cities one truck and two packages.

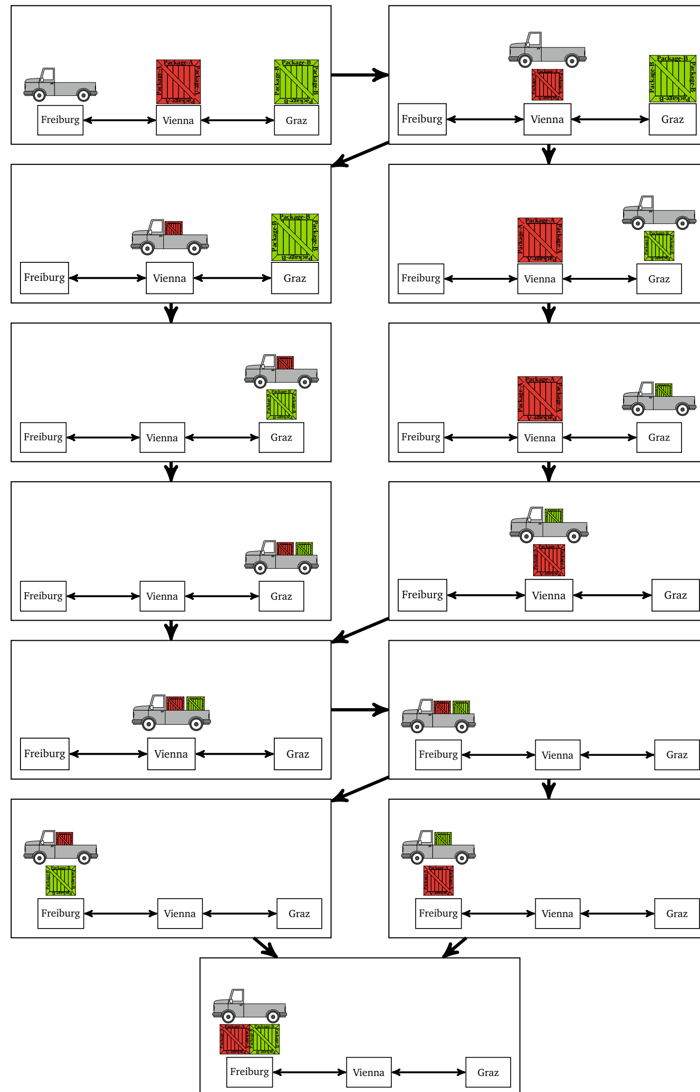



Figure 3.2: Search graph: Irrelevant paths disregarded

not have negative effects (facts that have been achieved stay achieved), the problem of finding a solution is still *NP-complete* (Bylander, 1994). Du to this complexity, naive approaches exhaustively searching the state space are not feasible for any kind of problem apart from the very smallest.

**Planning as heuristic search.** One approach of finding a plan $\pi$ for a given planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a \rangle$ is to apply a heuristic search algorithm over the state space $\mathcal{S}$. A heuristic search algorithm uses a so called heuristic function $h : \mathcal{S} \rightarrow \mathbb{N}$ providing a numeric quality measure to a state $s$. The value of a given state is also referred to as the *h-value*. One search algorithm using this *h-value* is greedy best first search. A second measure for evaluating the quality of a given state is the *g-value* (abbreviated with $g$) indicating the cost of reaching the current state. Note that each state can have multiple *g-values* as they may be reached by multiple paths in the search. A well known algorithm using this *g-value* only is Dijkstra's algorithm (Dijkstra, 1959). The most common algorithm for heuristic search uses a combination of *h-value* and *g-value* is the A* algorithm (Hart et al., 1968) using $f = g + h$ as the states quality measure. During the search a graph consisting of nodes $n = \langle s, g, h \rangle$ storing a planing state $s$, a *g-value* and a *h-value* are created. Edges in the graph $e = \langle n, a, c_a, n' \rangle$ with $n = \langle s, g, h \rangle$ and $n' = \langle s', g', h' \rangle$ are labeled with the action $a$ such that $s' = s[a]$, and $g' = g + c_a$.

Given a node $n = \langle s, g, h \rangle$ the search algorithm then *expands* this node by applying every action applicable in $s$ resulting in new states $\mathcal{S}'$. For each $s' \in \mathcal{S}'$ reached from applying action $a$ a new node $n' = \langle s', g', h' \rangle$ is created and connected to its predecessor $n$ via the edge $e = \langle n, a, c_a n' \rangle$. This is done until a state is reached that fulfills the goal condition $s_\star$. Some algorithms then continue the search to prove that the found plan is optimal, or to try and improve the plan quality. This however, is dependent on the actual algorithm. The main difference in the above algorithms is how they select the next node to *expand*. Greedy best first search expands the node with the lowest *h-value*, whereas Dijkstra's algorithm expands the node with the lowest *g-value*. A* then combines the both by selecting the node with the lowest $f = g + h$ value. Note that this description is a simplification of the actual algorithms, but provide some intuition on how heuristic search works in general. Also, these three algorithms are by no means all possible algorithms, nor is this forward search approach the only possible way of finding solutions to planning tasks. Well known planners working with this paradigm are the *Fast Forward Planning System* (Hoffmann and Nebel, 2001) and the *Fast Downward Planning System* (Helmert, 2006).

A classical example illustrating such a planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a \rangle$ is the logistics domain (Helmert, 2008), where a sequence of actions is re-

quired to move a truck from one location to another, loading and unloading packages, with the goal being that all packages are at their destination.

As mentioned before, heuristics guiding the search towards a goal state are required. Such heuristics give an estimate of how *far* a given state is to fulfilling the goal condition (reaching a goal state). A simple heuristic for path planning could be the Euclidean distance between the current position(current state) and destination(goal state). It is clear to see that this does not represent the exact distance, due to the fact that roads do not connect all locations by a straight line. However, states in which the Euclidean distance to the destination is smaller are more likely to lead to the actual destination. In domain independent planning, a more general definition of a heuristic is required:

**Definition 26 (Heuristic).** A heuristic function $h : \mathcal{S} \to \mathbb{R} \cup \{\infty\}$ maps a state to a heuristic value, estimating the distance from the state $s$ to the goal $s_\star$.

Many heuristic functions exist for classical planning, and are an intense field of research. Some of the more common heuristics are briefly discussed here, as to give a better understanding, and intuition of what heuristics are and how they work. The baseline heuristic used is the so called blind heuristic $h^{\text{blind}}$ which, as the name already suggests, corresponds to the search without heuristic estimate, apart for a goal state. This heuristic assigns a value of $1$ to all states that do not fulfill the goal condition and $0$ to states fulfilling the goal condition.

More complex heuristics work on the basis of first reducing the search space, thus making the problem easier, and from this easier problem, calculating estimates for the original planning task. Two methods are introduced in the next few paragraphs. The first being task relaxation, where the idea is to keep every fact that has become true once true for ever, and the second is task abstraction where variables and values are merged and treated as one variable value.

**Definition 27 (Relaxed planning task).** Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a \rangle$ be a planning task and Π' be the relaxed task with $\Pi' = \Pi$. A relaxed state $s^+$ then assigns to each $v \in \mathcal{V}$ a non empty set $s^+(v) \subseteq \mathcal{D}_v$. Let $\mathcal{S}^+$ be the set of all relaxed states. $s^+$ *subsumes* $s$ if for all $v \in \mathcal{V}$, $s(v) \in s^+(v)$. An action $a = \langle \mathit{pre}, \mathit{eff} \rangle$ is applicable in the relaxed state if $\mathit{pre}(v) \in s^+(v)$ for all $v \in \mathcal{V}$ where $\mathit{pre}(v)$ is defined. Applying $a$ in $s^+$ results in $s^{+'} = s^+ \cup \{\mathit{eff}(v)\}$ [1].

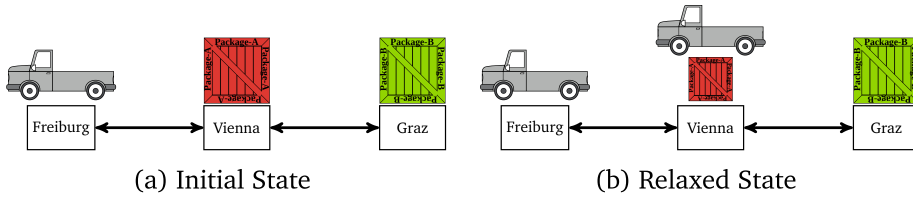
**Definition 28 (Relaxed reachability).** A fact $v = d$ is relaxed reachable from a state $s^+$ in a relaxed planning task if there exists a possible empty

[1] It is not entirely clear as to whom this formulation can be attributed, but it can be traced down to Helmert (2006) or Kupferschmid et al. (2006).

sequence of actions $(a_0, \ldots, a_n)$ applicable in $s^+$, with the induced state trajectory $(s^+, \ldots, s_n)$ such that $v = d$ is in $s_n$.

In words, states in a relaxed plan assign a set of values to each variable, thus each value that was achieved once stays achieved, variables in the relaxed task therefore, can reside in multiple states at once. This reduces the search space, as each value only needs to be reached once. This can be exploited to generate a heuristic estimate of the original task such as $h^{\text{max}}$ and $h^{\text{add}}$ (Bonet and H. Geffner, 2001).

**Example 5 (Relaxed Planning Task).** Recalling the Example 4 if in the initial state the action *move(Freiburg, Vienna)* is applied, the effect will be that the *Truck* is now in two locations simultaneous *Freiburg* and *Vienna*:




(a) Initial State (b) Relaxed State

Figure 3.3: Relaxed planning task. After applying *move(Freiburg, Vienna)* the *Truck* now is in two locations simultaneous.

**$h^{\text{max}}$ and $h^{\text{add}}$ Heuristics:** The basic idea of $h^{\text{max}}$ and $h^{\text{add}}$ (Bonet and H. Geffner, 2001) is to estimate the cost of achieving each fact $f \in s_\star$ from any state $s$. Let $\mathcal{A}(f)$ be the set of achievers of $f$, i.e. the actions $a = \langle pre, eff \rangle$ such that $f$ is made true in *eff*.

$h^{\text{max}}$ is defined by the recursive function (Equation 3.1-Equation 3.3). For a set of facts $F$ the $h^{\text{max}}$ value is the maximum over all $h^{\text{max}}$ values of the individual facts (Equation 3.2). For a fact $f$ the value of $h^{\text{max}}$ is either $0$ if the fact is already in the set of achieved facts (initially $s_0$) or for every achiever $a = \langle pre, eff \rangle \in \mathcal{A}(f)$ of $f$, the $h^{\text{max}}$ value of the facts in *pre* are calculated recursively and added to the cost of $c_a$. Then from this set the minimum is taken (Equation 3.3). To calculate $h^{\text{max}}$ for any given state $s$, this is done by setting the set of initially achieved facts to $s$ (Equation 3.1).

$$h^{\text{max}}(s) = h_s^{\text{max}}(s_\star) \tag{3.1}$$

$$h_s^{\text{max}}(s_p) = \max_{f \in s_p}(h_s^{\text{max}}(f)) \tag{3.2}$$

$$h_s^{\text{max}}(f) = \begin{cases} 0 & \texttt{if } f \in s \\ \min\limits_{a \in \mathcal{A}(f)} [h_s^{\text{max}}(pre(a)) + c_a] & \texttt{if } f \notin s \end{cases} \tag{3.3}$$

For $h^{\text{add}}$ Equation 3.2 is simply replaced by $h_s^{\text{add}}(s_p) = \sum_{f \in s_p} h_s^{\text{add}}(f)$.

Alternatively to task relaxation, tasks can also be abstracted. In general, this approach tries to reduce the complexity of the search by abstracting the variables merging variable values and treating them as one. Thus for a given abstract state $s'$ the different variable values are treated as one.

When all the values of a single variable are merged to one value (the different states of the variable become indistinguishable, thus the variable can be completely removed), this is called projection abstraction (Figure 3.4a). When only some values of a variable are merged in to one, this is called domain abstraction (the domain of a variable is abstracted Figure 3.4b), and if the abstraction is a Cartesian projection over the variables, then this is called Cartesian abstraction (Figure 3.4c). Alternatively, values from different variables can also be merged in to one value. This is the most general case of abstraction (Figure 3.4d).

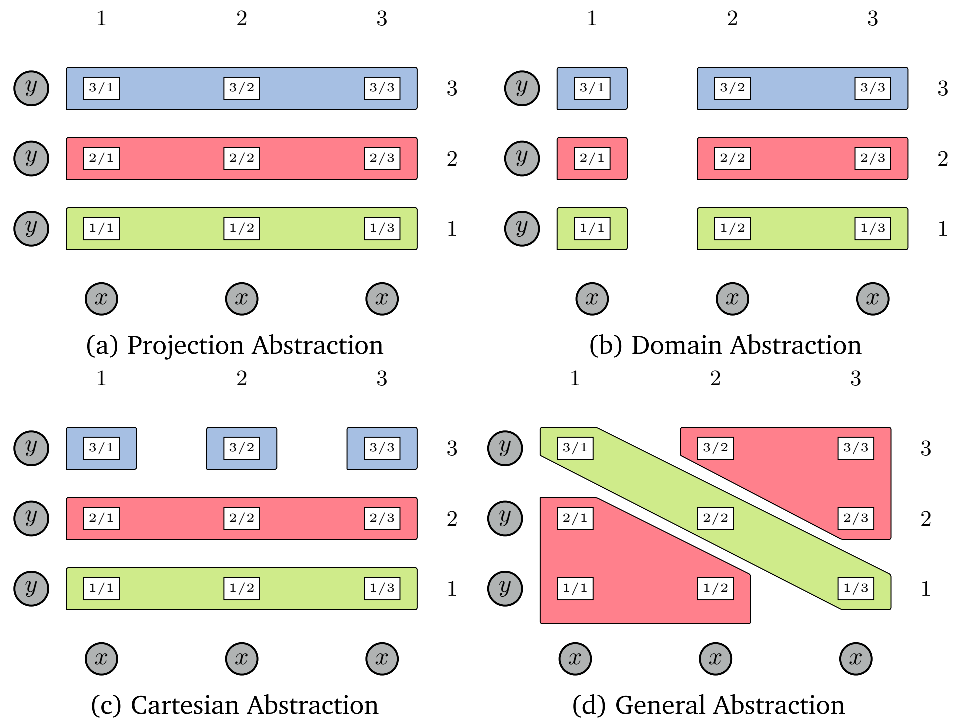



Figure 3.4: Different types of task abstractions

In the following projection abstraction is described, as it is commonly used in pattern database heuristics (introduced later on in this section).

**Definition 29 (Projection abstraction).** Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a \rangle$ be a planning task with states $\mathcal{S}$, and let $\mathcal{V}' \subseteq \mathcal{V}$ be a subset of the variables. Then let $s_0' = \{v = s_0(v) | v \in \mathcal{V}'\}$ and $s_\star' = \{v = s_\star(v) | v \in \mathcal{V}'\}$. An action

$a = \langle pre, eff\rangle$ is applicable in a abstract state $s^a$ iff $pre(v) = s^a(v)$ for all $v \in \mathcal{V}'$ where $pre(v)$ is defined. The effect of the action is then $s^a(v)' = [eff]_{s^a}(v)$ for all $v \in \mathcal{V}'$ where $[eff]_{s^a}(v)$ is defined, and $s^a(v)' = s^a(v)$ for all $v$ where $[eff]_{s^a}(v)$ is undefined.

This task abstraction reduces the overall state space making the search for a solution in $\Pi'$ easier. Even though the solution to the abstract task $\Pi'$ is not a correct solution to the original task $\Pi$, this abstraction can be used to calculate heuristic values for the original task, by searching for a solution in the abstracted task and using the cost of the solution as heuristic value for the original task.

$h^{\mathbf{cpdb}}$ **Heuristic:** Alternatively task abstraction can be used to reduce the search space of the original task, making it easy to search for a solution in the abstract task. The $c(\mu_\pi)$ of the abstracted task solution is then used as heuristic value for the original task. A common class of heuristics, which utilizes task abstraction are *pattern database* heuristics. Let $P$ be an abstraction from $\Pi$, yielding $\Pi^P$. If $P$ is small enough it is tractable to calculate the optimal cost of achieving the goal condition in any state of $\Pi^P$, by breadth-first search, as the state space is bounded by $\prod_{v \in P} |\mathcal{D}_v|$ the product of the domains of all variables in the pattern $P$. These costs are then stored in a table called the pattern database for pattern $P$. Multiple such pattern databases can then be combined to calculate a heuristic value for the original task. Such a combination can be taking the maximum or the sum of the values, but depend on the patterns and the actual heuristic. One such pattern database heuristic is the *canonical pattern database heuristic*($h^{\mathrm{cpdb}}$) (Haslum et al., 2007). The heuristic function defined by $h^{\mathrm{cpdb}}$ is

$$h^C(s) = \max_{S \in A} \sum_{P \in S} h^P(s) \tag{3.4}$$

where $C = \{P_1, \ldots, P_n\}$ is a collection of patterns and $A$ the collection of all additive subsets of $C$. Two patterns $P_i$ and $P_j$ are additive if the set of actions that effect a variable in $P_i$ is disjoint from the set of actions that affect any variable in $P_j$. The main issue that remains is then how to actually create the patterns, as this determines the quality of the heuristic value, and the time it takes to calculate this value. $h^{\mathrm{cpdb}}$ constructs the patterns by a search over nodes where a node is a pattern collection, and its neighbors are modifications to this collection. Initially, the pattern collection $C$ consists of one pattern for each variable from the goal condition containing only this variable $C = \bigcup_{v \in s_\star} \{\{v\}\}$. From a given collection $C$ a new collection $C'$(and neighbor of $C$) is created by selecting a single pattern $P \in C$ and a variable $v \notin P$ and adding a new pattern to $C' = C \cup \{P \cup v\}$. After expanding the current collection $C'$ the *best* neighbor is selected for the next

iteration, and terminates when now significant improvement is detected, or a given storage limit is exceeded. The corresponding pattern databases are calculated in each iterative step and stored in a global table, for later lookup. What is still missing now is the calculation of the *best* neighbor of a collection $C$. For this Haslum et al. (2007) introduce a approximating function

$$\frac{1}{m} \sum_{\{n_i | h^C(n_i) < h^{C'}(n_i)\}} N_{cb - h^c(n_i)} \tag{3.5}$$

where $m \in \mathbb{N}$ and $n$ a sample of $m$ nodes from the search tree, $cb$ a cost bound, and $N_k$ the number of nodes in the search tree with accumulated cost (g-value) of at most $k$. This formula gives a measure of how much the search improves if the pattern collection $C'$ is used instead of the collection $C$.

For finding optimal plans, heuristics in general must be *admissible*.

**Definition 30 (Admissible Heuristic).** Let $h^\star(s)$ be the true cost to the goal, then a heuristic $h$ is admissible if for all states $s \in \mathcal{S}$, $h(s) \leq h^\star(s)$, thus it never overestimates the costs to the goal.

In some special cases like the *A** algorithm without revisiting already processed nodes, for optimality the heuristic must not only be admissible, but also consistent.

**Definition 31 (Consistent Heuristic).** A heuristic $h$ is consistent if for all states $s \in \mathcal{S}$ and all actions $a \in \mathcal{A}$ applicable in $s$, $h(s) \leq c_a + h(s')$ where $s' = s[a]$.

The consistency requirement is a form of the triangle inequality, which in this case says that the path from $s$ to $s_\star$ can not be longer than the path from $s$ to $s_\star$ via $s'$.

**Planning formalism PDDL** After introducing the basic principles of AI planning in the sections above, a more practical definition of a planning task is required. For this reason, the planning community has introduced PDDL (*P*lanning *D*omain *D*efinition *L*anguage) as a language for describing planning tasks in a more abstract way (PDDL 1.2 (McDermott et al., 1998), PDDL 2.1 (Fox and Long, 2003), PDDL 2.2 (Edelkamp and Hoffmann, 2004), PDDL 3.0 (Gerevini and Long, 2005)). Even though, this is not the only language for specifying planning tasks, it is used by almost all implemented modern systems, especially Fast-Downward(Helmert, 2006) which is used as a platform to implement and test work presented later on in this theses.

## 3.2 Planning with State-Dependent Action Costs

In real world problems, the execution of an action is often related to a cost of doing so. For instance driving from location A to location B requires the use of fuel, which comes at some cost. In the classical setting this can be modeled by the action costs $c_a$. However, in many situations, the cost of an action is not only dependent of the action itself, but also on the state in which it is executed. Returning to our Example 4, the cost of moving from any location to another may be dependent on whether a package is on the truck or not, as additional weight increases fuel consumption. Additionally the cost of moving from Vienna to Graz will be significantly lower than moving from Freiburg to Vienna, as the distances are very different. Although these state dependent costs can be removed by adding additional actions such as *Move-Freiburg-Vienna-No-Package*, *Move-Vienna-Freiburg-No-Package*, *Move-Vienna-Graz-No-Package*, and *Move-Graz-Vienna*, for each possible combination of the state variables, representing action costs in a state dependent way gives way to a more compact and natural representation of the domain.

State-dependent action costs can be represented as the sum over tuples $c_a = \{(\varphi_1, c_1) \dots (\varphi_n, c_n)\}$ where $\varphi_i$ is a logical formula over facts $f \in \mathcal{F}$ and $c_i$ is a constant (Ivankovic et al., 2014). The cost of applying an action $a$ in state $s$ is $c(a, s) = \sum_{(\varphi,c) \in c_a} [\varphi]_s * c$. This can be extended to cost functions that can represent arithmetic terms over state variables (Geißer et al., 2015) resulting in the following definitions.

**Definition 32 (Planning Task with State-Dependent Action Costs).** Let $\mathcal{V}$ be a set of variables, $\mathcal{A}$ a set of actions, $s_0$ an initial state, and $s_\star$ a goal condition. Furthermore, let $c_a$ be a cost function $c_a : \mathcal{S} \to \mathbb{N}$ assigning a cost for application of an action $a$ to each state where $a$ is applicable. Then $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a \rangle$ is a planning task with state-dependent action costs.

**Example 6.** Recalling the logistics task from Example 4 using state-dependent action costs, it is possible to model the fact that the cost of moving from one city to another is dependent of packages loaded in the truck. Let the predicates *location(Package-Red, Truck)* and *location(Package-Green, Truck)* state the facts, that the red and green packages are loaded in the truck respectively. Then the cost function for the move actions is

$$
\begin{aligned}
c_{\textit{move}(s)} = & \textit{[location(Package-Red,Truck)]} * 2 + \\
& \textit{[location(Package-Green,Truck)]} * 2 + 1
\end{aligned}
$$

This states, that for every package that is located in the truck the cost of applying the move action is increased by $2$, and the cost of moving without package is $1$.

### 3.2.1 Expressing SDAC as EVMDD

Let $c_a(s) : \mathcal{S} \to \mathbb{N}$ be a arithmetic cost function. The EVMDD representing this cost function is then constructed as follows (Mattmüller et al., 2018) (following the construction schema from Section 2.1.1):

1. Create an abstract syntax tree from $c_a(s)$ with inner nodes representing arithmetic, Iverson and Boolean operators, and leaf nodes representing variables and constants.
2. For each leaf node an EVMDD $\mathcal{E}_i$ is created: For a constant $a$ the EVMDD $\mathcal{E}_i = \langle a, \mathbf{0} \rangle$ is created. For a variable $v$ the EVMDD $\mathcal{E}_i = \langle 0, \mathbf{f} \rangle$ with $\mathbf{f} = (v, \mathbf{0}, \ldots, \mathbf{0}, 1, \ldots, k)$ and $k = |\mathcal{D}(v)| - 1$ is created.
3. For inner nodes representing an operator $\bullet$, the child EVMDDs $\mathcal{E}_i, \mathcal{E}_j$ are combined by applying the *APPLY* procedure with the $\bullet$ operator.
   **Iverson brackets:** If an inner node represents an Iverson bracket, the children of this node must be Boolean operators. In this case the EVMDD created for the inner node is on the monoid $\mathscr{B} = (\{\top, \bot\}, \vee, \bot)$ identical to the construction in Section 4.1. When evaluating the Iverson bracket node, which only has one direct child, *APPLY* is called with a special operator $\odot : \{\top, \bot\} \to \{1, 0\}$. This operator takes one EVMDD and maps $\top \to 1$ and $\bot \to 0$.

**Proposition 1.** Let $c_a : \mathcal{S} \to \mathbb{N}$ be an arithmetic cost function and $\mathcal{E}_c$ the EVMDD representation as constructed above, then $c_a(s) = \mathcal{E}_c(s)$ for each state $s \in \mathcal{S}$.

*Proof.* By definition of EVMDDs over the monoid $\mathscr{N} = \langle \mathbb{N}, +, 0 \rangle$ (Ciardo and Siminiceanu, 2002), $c_a(s) = \mathcal{E}_c(s)$. □

As was seen in Section 3.1, heuristics are used to solve planning tasks. However, up until now only state-independent action costs where considered. During the search this is no problem as the evaluation of the actions cost can be executed on the current state. However, when calculating the heuristic on abstracted or relaxed tasks, this is non-trivial, as variables may assume more than one value at the same time, making the evaluation of the cost function non trivial. In the following section a relaxation heuristic with state dependent action costs is discussed.

### 3.2.2 $h^{\text{add}}$ with State-Dependent Action Costs

Missing in the original definition of relaxed planning tasks (Definition 27), is the notion of state dependent action costs. As the idea of relaxation heuristics based on the accumulation semantics (Hoffmann, 2005) (facts that have

become true at one point stay true) is to estimate the cost of reaching the goal, and for admissibility (Definition 30) the cost may never be overestimated, the interpretation of $c_a(s^+)$ in the relaxed state $s^+$ is the minimum of $c_a(s)$ for all unrelaxed states $s$ that are subsumed by $s^+$. Let $\mathcal{S}_c$ be all possible evaluations of variables occurring in $c_a(s)$, and let $\mathcal{A}(f)$ be the set of actions that make the fact $f$ true. The new definition of $h^{\text{add}}$ is then:

$$\begin{aligned}
h^{\text{add}}(s) &= h_s^{\text{add}}(s_\star) \\
h_s^{\text{add}}(s_{\text{p}}) &= \sum_{f \in s_p} h_s^{\text{add}}(f) \\
h_s^{\text{add}}(f) &= \begin{cases} 0 & \text{if } f \in s \\ \min\limits_{a \in \mathcal{A}(f)} [h_s^{\text{add}}(\mathit{pre}(a)) + C_a^s] & \text{if } f \notin s \end{cases} \\
C_a^s &= \min_{\hat{s} \in \mathcal{S}_c} [c_a(\hat{s}) + h_s^{\text{add}}(\hat{s})]
\end{aligned}$$

$C_a^{s^+}$ is the evaluation of $C_a$ in state $s^+$ (Geißer et al., 2015). If $c_a(s)$ is a constant function then $C_s^a = c_a(\emptyset) + h^{\text{add}} s(\emptyset) = c_a$, this definition reduces to the original definition of $h^{\text{add}}$.

The major problem here is that the number of unrelaxed states $s$ subsumed by $s^+$ can be exponential in the number of state variables (Geißer et al., 2015), making the evaluation of $C_a^{s^+}$ intractable. A compact representation of $c_a$ is therefore essential, which can be achieved using EVMDDs.

Geißer et al. (2015) introduces two possibilities of calculating $C_a^s$ from $h^{\text{add}}$ with state dependent action costs . The first approach is to calculate the heuristic value on a transformed planning task without SDAC, and the second being a transformation of the relaxed planning graph. The heuristic based on task transformation will be introduced here, whereas for the relaxed planning graph transformation, the reader is referred to Florian Geißers PhD Thesis (Geißer, 2018b).

There exists multiple ways of compiling away SDAC, in order to calculate the correct $C_a^s$ value. The first, naive way is to create a new action for each possible variable valuation in $c_a$. Thus, for an action $a$ a set of new actions $a^{\tilde{s}}$ is created such that $a^{\tilde{s}_i} \in a^{\tilde{s}}$ is only applicable in state $\tilde{s}_i$. Formally: $a^{\tilde{s}_i} = \langle \mathit{pre} \wedge \bigwedge_{f \in \tilde{s}_i} f, \mathit{eff} \rangle$. It is simple to see, that this results in a exponential blow up, in the number of variables in $c_a$.

**Example 7 (Naive SDAC action compilation).** Let $a$ be an action with cost function $c_a = x^2 + 2yz + 8$ with the variable domains $\mathcal{D}_x = 2$, $\mathcal{D}_y = 2$, and

$\mathcal{D}_z = 3$. Then the resulting actions $a^{\tilde{s}}$ are:

$$
\begin{aligned}
a_{\{x=0,y=0,z=0\}} &= \langle pre \wedge x = 0 \wedge y = 0 \wedge z = 0, \mathit{eff}\rangle & cost = 8\\
a_{\{x=0,y=0,z=1\}} &= \langle pre \wedge x = 0 \wedge y = 0 \wedge z = 1, \mathit{eff}\rangle & cost = 8\\
a_{\{x=0,y=0,z=2\}} &= \langle pre \wedge x = 0 \wedge y = 0 \wedge z = 2, \mathit{eff}\rangle & cost = 8\\
a_{\{x=0,y=1,z=0\}} &= \langle pre \wedge x = 0 \wedge y = 1 \wedge z = 0, \mathit{eff}\rangle & cost = 8\\
a_{\{x=0,y=1,z=1\}} &= \langle pre \wedge x = 0 \wedge y = 1 \wedge z = 1, \mathit{eff}\rangle & cost = 10\\
a_{\{x=0,y=1,z=2\}} &= \langle pre \wedge x = 0 \wedge y = 1 \wedge z = 2, \mathit{eff}\rangle & cost = 12\\
a_{\{x=1,y=0,z=0\}} &= \langle pre \wedge x = 1 \wedge y = 0 \wedge z = 0, \mathit{eff}\rangle & cost = 9\\
a_{\{x=1,y=0,z=1\}} &= \langle pre \wedge x = 1 \wedge y = 0 \wedge z = 1, \mathit{eff}\rangle & cost = 9\\
a_{\{x=1,y=0,z=2\}} &= \langle pre \wedge x = 1 \wedge y = 0 \wedge z = 2, \mathit{eff}\rangle & cost = 9\\
a_{\{x=1,y=1,z=0\}} &= \langle pre \wedge x = 1 \wedge y = 1 \wedge z = 0, \mathit{eff}\rangle & cost = 9\\
a_{\{x=1,y=1,z=1\}} &= \langle pre \wedge x = 1 \wedge y = 1 \wedge z = 1, \mathit{eff}\rangle & cost = 11\\
a_{\{x=1,y=1,z=2\}} &= \langle pre \wedge x = 1 \wedge y = 1 \wedge z = 2, \mathit{eff}\rangle & cost = 13
\end{aligned}
$$

resulting in 12 new actions.

**Theorem 10.** (Geißer et al., 2015) Let $\Pi$ be a planning task with state dependent action costs and $\Pi'$ the naively compiled task without state dependent action costs. Furthermore, let $s$ be a state of $\Pi$, then $h^{\text{add}}_{\Pi}(s) = h^{\text{add}}_{\Pi'}(s)$.

Alternatively, the EVMDD structure can be exploited to create a more compact compilation resulting in less actions (Geißer et al., 2015; Mattmüller et al., 2018). In short this is similar to the conditional effects compilation introduced in Section 4.1. However, instead of applying effects in the compiled actions, the new actions collect partial costs. Note that *Nebel's* compilation on conditional effects is not generalized to state-dependent action costs, as it is unclear what the branching factors would be.

**EVMDD compilation of state-dependent action costs** Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_{a,s}\rangle$ be a planning task with state dependent action costs. Also, let *sem* be a semaphore variable with domain $\mathcal{D}_{\mathit{sem}} = \{0, 1\}$ and $\mathcal{V}_{\mathit{aux}} = \{\mathit{aux}_a | a \in \mathcal{A}\}$ with domains $\mathcal{D}_{\mathit{aux}_a} = \mathit{size}(\mathcal{E}_{c_a})$ the size of the actions cost functions EVMDD, then $\mathcal{V}' = \mathcal{V} \cup \{\mathit{sem}\} \cup \mathcal{V}_{\mathit{aux}}$. Let

$$f_\xi(s) = s \cup \{\mathit{aux}_a = 0 | \mathit{aux} \in \mathcal{V}_{\mathit{aux}}\} \cup \{\mathit{sem} = 0\}$$

be the mapping function from states in $\Pi$ to states in $\Pi'$, then $s'_0 = f_\xi(s_0)$ and $s'_\star = f_\xi(s_\star)$. The actions $\mathcal{A}'$ are then created as follows:

For every action $a = \langle pre, eff \rangle$ with cost function $c_{a,s}$ an EVMDD $\mathcal{E}_c = \langle \kappa, \mathbf{f} \rangle$ is constructed as shown in Section3.2.1 and all nodes in $\mathcal{E}_c$ are topological enumerated denoted as $idx(\mathbf{v})$. Then three types of new actions are created.

1. A *init* action $a_{init} = \langle pre \wedge sem = 0 \wedge aux = 0, aux = idx(\mathbf{v}_0) \wedge sem = 1 \rangle$ with cost $\kappa$ is created.

2. For each non terminal node $\mathbf{v} = (v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$ and all children $\chi_d$ with label $w_d$ and $0 \leq d \leq |\mathcal{D}(v)| - 1$, a new action $a_{v,d,idx} = \langle aux_a = idx(\mathbf{v}) \wedge v = d, aux_a = idx(\chi_d) \rangle$ with cost $w_d$ is created.

3. Finally the action $a_{final} = \langle aux_a = |\mathcal{E}_c| + 1, aux_a = 0 \wedge sem = 0 \wedge eff \rangle$ with cost $0$ resetting the auxiliary variable $aux_a$ and the semaphore variables $sem$ is created,

The new task without state dependent action costs is $\Pi = \langle \mathcal{V}', \mathcal{A}', s'_0, s'_\star, c_a \rangle$.

Note that for a correct calculation of the $h^{\text{add}}$ heuristic it is necessary to have branches over all variables in every branch, therefore quasi reduced EVMDDs (Ciardo and Siminiceanu, 2002) are used.

**Example 8 (SDAC EVMDD compilation).** Let $a$ be the same action with the same cost as in Example 7, the EVMDD for this cost function is then depicted in Figure 3.5. The EVMDD compilation then results in the following actions:

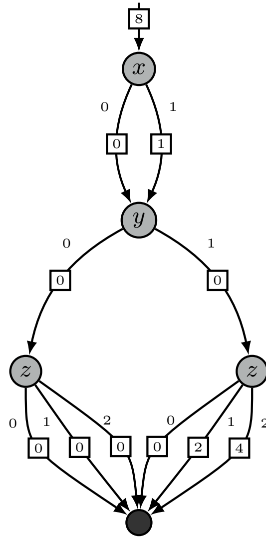



Figure 3.5: The EVMDD representing $x^2 + 2yz + 8$ with domains $(\mathcal{D}_x = 2, \mathcal{D}_y = 2, \mathcal{D}_z = 3)$

$$\begin{aligned}
a_{init} &:= \langle pre \wedge sem = 0 \wedge aux = 0, aux = 1 \wedge sem = 1\rangle & cost = 8\\
a_{x_1} &:= \langle aux = 1 \wedge x = 0, aux = 2\rangle & cost = 0\\
a_{x_2} &:= \langle aux = 1 \wedge x = 1, aux = 2\rangle & cost = 1\\
a_{y_1} &:= \langle aux = 2 \wedge y = 0, aux = 3\rangle & cost = 0\\
a_{y_2} &:= \langle aux = 2 \wedge y = 1, aux = 4\rangle & cost = 0\\
a_{z_{01}} &:= \langle aux = 3 \wedge z = 0, aux = 5\rangle & cost = 0\\
a_{z_{02}} &:= \langle aux = 3 \wedge z = 1, aux = 5\rangle & cost = 0\\
a_{z_{03}} &:= \langle aux = 3 \wedge z = 2, aux = 5\rangle & cost = 0\\
a_{z_{11}} &:= \langle aux = 4 \wedge z = 0, aux = 5\rangle & cost = 0\\
a_{z_{12}} &:= \langle aux = 4 \wedge z = 1, aux = 5\rangle & cost = 2\\
a_{z_{13}} &:= \langle aux = 4 \wedge z = 2, aux = 5\rangle & cost = 4\\
a_{final} &:= \langle aux = 5, sem = 0 \wedge aux = 0 \wedge eff\rangle & cost = 0
\end{aligned}$$

Evaluating this in state $s = \{x = 1, y = 1, z = 1\}$ results in the action sequence $(a_{init}, a_{x_1}, a_{y_1}, a_{z_{11}}, a_{final}$, resulting an the total cost of $1^2 + 2 * 1 * 1 + 8 = 11$

**Definition 33.** Given an action $a = \langle pre, eff\rangle$ with cost function $c_a$ and a set of EVMDD compiled actions $\hat{a}$ over the cost EVMDD $\mathcal{E}_c$, then $f_\pi(s, a)$ is the sequence of actions in $\hat{a}$ such that

$$f_\pi(s, a) = (f_\pi^{a,0}(s), f_\pi^{a,+}(s), f_\pi^{a,final}(s)) \tag{3.6}$$

with:

1. $f_\pi^{a,0}(s)$ is the *init* action from step 1 of the construction and is applicable in state $f_\xi(s)$.

2. $f_\pi^{a,+}(s)$ is defined recursively over the EVMDD $\mathcal{E}_{eff}$ and is the sequence of actions from step 2 of the construction. For the terminal node **0**, $f_\pi^{a,+}(s)$ denotes the empty sequence of actions. For each non terminal node $\mathbf{v} = (v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$, $f_\pi^{a,+}(s)$ is the action $a_{v,s(v),idx(\mathbf{v})}$ followed by the action sequence $f_{\pi,s(v)}^{a,+}(s)$ where $f_{\pi,s(v)}^{a,+}(s)$ is the function denoted by the child $\chi_{s(v)}$.

3. $f_\pi^{a,final}(s)$ is the *final* action from step 3 of the construction.

Additionally let $f_c(s, a) = \sum_{a' \in f_{\pi(s,a)}} c_{a'}$ be the total cost of the action sequence.

**Lemma 2.** For every action $a \in \mathcal{A}$ and every state $s \in \mathcal{S}$ such that $a$ is applicable in $s$, $\delta(f_\xi(s), f_\pi(s, a)) = f_\xi(\delta(s, a))$, and the total cost $f_c(s, a)$ is $c_a(s)$.

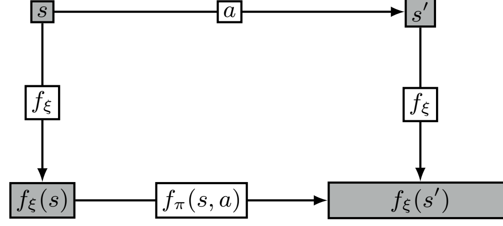



Figure 3.6: Applying the original action $a$ in state $s$ on the original task $\Pi$ and applying $f_\xi$ to the resulting state $s' = [a]_s$ is equivalent to applying the sequence of actions $f_\pi(s,a)$ to the transformed state $f_\xi(s)$.

*Proof.* Let $a = \langle pre, eff \rangle$ be an action in $\mathcal{A}$ with cost $c_a(s)$ applicable in state $s$, $t = s[a]$ and $\mathcal{E}_c = \langle \kappa, \mathbf{f} \rangle$ be the EVMDD representing the cost function $c_a(s)$. By construction, $f_\pi^{a,0}(s)$ is the action $a_{init}$, from step 1 of the EVMDD compilation, applicable in $f_\xi(s)$ as $f_\xi(s) \models pre \wedge sem = 0 \wedge aux = 0$ with the constant costs $\kappa$. Then $f_\pi^{a,+}(s)$ denotes the sequence of compiled actions consistent to state $s$, collecting the partial costs from $\mathbf{f}$. This is ensured by tracking the current node of $\mathcal{E}_c$ in *aux* and applying the edge labels as constant costs in the actions (step 2 of the compilation). Finally, $f_\pi^{a,final}(s)$ consists of the $a_{final}$ action applying the original action effects *eff* and resetting the *sem* and *aux* variable. Therefore, $f_\pi(s,a)$ is a sequence of actions corresponding to a path in $\mathcal{E}_c$ consistent to state $s$ gathering the partial costs of the visited edge labels. Thus, $\delta(f_\xi(s), f_\pi(s,a)) = f_\xi(\delta(s,a))$ holds and $f_c(s,a) = c_a(s) = \mathcal{E}_c(s)$. □

**Definition 34 (Plan equivalence).** Given a plan $\pi = (a_1, \dots, a_k)$ for $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a \rangle$ with the induced state sequence $\overline{s} = (s_0, \dots, s_k)$ such that $a_i$ is applicable in $s_{i-1}$ and results in state $s_i$ for all $i = 1, \dots, k$, and $s_k \models s_\star$. Given a second plan $\pi' = (a'_1 \dots, a'_l)$ for $\Pi' = \langle \mathcal{V}', \mathcal{A}', s'_0, s'_\star, c'_{a'} \rangle$ with the induced state sequence $\overline{s'} = (s'_0, \dots, s'_l)$ such that $a'_i$ is applicable in $s'_{i-1}$ and results in state $s'_i$ for all $i = 1, \dots, l$ and $s_l \models s'_\star$. Let $f_\xi : s \mapsto s'$ be a mapping from states in $\overline{s}$ to states in $\overline{s'}$ such that $s \subseteq f_\xi(s)$ for all states $s \in \overline{s}$. The plans $\pi$ and $\pi'$ are equivalent iff there exists an injective and strictly monotonous mapping $\sigma : \{0, \dots, k\} \to \{0, \dots, l\}$ such that for every $u = 0, \dots, k$, $s'_{\sigma(u)} = f_\xi(s_u)$ and $\sigma(0) = 0$ and $\sigma(k) = l$, and $c(\pi) = c(\pi')$. Additionally, for any state $s'_i$ with $\sigma(u) < i < \sigma(u+1)$ such that $s'_i \models s'_\star$ then $s'_{\sigma(u)} \models s'_\star$.

**Definition 35 (Task *plan-equivalence*).** Two planning tasks $\Pi$ and $\Pi'$ are *plan-equivalent* iff for every plan $\pi$ in $\Pi$ there exists an plan $\pi'$ in $\Pi'$ such that $\pi, \pi'$ are equivalent and for every plan $\pi'$ in $\Pi'$ there exists an plan $\pi$ in $\Pi$ such that $\pi, \pi'$ are equivalent.

**Proposition 2.** Every planning task $\Pi$ with SDAC is *plan-equivalent* to its

EVMDD compiled task $\Pi'$ without SDAC.

*Proof.* Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a \rangle$ be a planning task with SDAC and $\Pi = \langle \mathcal{V}', \mathcal{A}', s_0', s_\star', c_a' \rangle$ be its EVMDD compiled task without SDAC constructed as above. Furthermore, let $\pi = (a_1, \ldots, a_k)$ be a plan for $\Pi$ with the induced state sequence $\overline{s} = (s_0, \ldots, s_{k+1})$ with $s_{k+1} \models s_\star$ and total costs

$$c(\pi) = \sum_{i=1}^{k} c_{a_i}(s_{i-1})$$

Following Lemma 2, for each action $a$ in $\pi$, the state $s \in \overline{s}$ in which $a$ is executed, and $t = s[a]$, an action sequence $f_\pi(s, a)$ in $\mathcal{A}'$ exists, such that $\delta(f_\xi(s), f_\pi(s, a)) = f_\xi(\delta(s, a))$ and $f_c(s, a) = c_a(s)$. Applying this transformation for every action in $\pi$ results in a new equivalent plan $\pi'$ with identical total costs. Additionally, as the goal condition $s_\star'$ is augmented with *sem* $= 0$ no intermediate state $s'$ created by applying $f_\pi(s, a)$ in state $f_\xi(s)$ can exist such that $s' \models f_\xi(s_\star')$. Following this, and the fact that by construction every sequence of actions $f_\pi(s, a)$ can be associated with a action $a$ from the original task, it can be seen that for every plan $\pi'$ in $\Pi'$ there exists an equivalent plan $\pi$ in $\Pi$.

Showing the other that for every plan $\pi$ in $\Pi$ there exists an equivalent plan $\pi'$ in $\Pi'$ follows the same schema. □

## 3.3 Fully Observable Nondeterministic Planning

Up until now, the planning task $\Pi$ has always been fully observable and deterministic, meaning the planner has full knowledge over the state, and each action has a deterministic effect. However, in real life this is a vast oversimplification, as some actions may have multiple possible outcomes. One such action might be the act of a robot picking up an object from a table. As the robot gripper is a physical object interacting with its environment, a lot of things can go wrong. The gripper might not hold on to the object hard enough resulting in the robot dropping the object. For a planning task this results in the action effect either the object being in the robot gripper or not. This setting is called **fully observable non-deterministic planning** or **FOND planning** (Cimatti et al., 2003). This setting might be extended to partially observable worlds, where the agent also only has partial knowledge of the current state of the world. This is known as **partially observable non-deterministic planning** or **POND-planning**. Both these settings can be again extended to also incorporate probabilities of certain action outcomes also known as **Markov decision process planning** or **MDP planning**.

In systems in which users interact with software, the setting is exactly that of FOND, as the outcome of a users action is not known apriory. Such user actions can therefore be modeled as a non-deterministic action. On the other hand, the software knows its internal representation and thus has full knowledge of the current state it resides in. As Chapter 7 is concerned with generating workflows for user interaction with software systems, this chapter will only focus on FOND planning.

For the FOND setting, the definition of actions (Definition 21) must be revisited and adapted to the new setting.

**Definition 36 (FOND action).** An action $a$ is a tuple $a = \langle \mathit{pre}, \mathit{eff} \rangle$ where the precondition *pre* is a partial variable assignment and a non-deterministic effect $\mathit{eff} = \{\mathit{eff}_1, \dots, \mathit{eff}_n\}$. Each $\mathit{eff}_i$ is a deterministic effect in the form of $(w := d)$ with $w \in \mathcal{V}$ and $d \in \mathcal{D}_w$.

**Definition 37 (FOND action application).** An action $a = \langle \mathit{pre}, \mathit{eff} \rangle$ with $\mathit{eff} = \{\mathit{eff}_1, \dots, \mathit{eff}_n\}$ is applicable in state $s$ if $s \models \mathit{pre}$. Applying an action in state $s$, results in a set of new states $\mathcal{T} = \{s_1 = [\mathit{eff}_1]_s, \dots, s_n = [\mathit{eff}_n]_s\}$ (one for every non-deterministic effect), denoted as $\mathcal{T} = s[a]$

As actions might have multiple outcomes, the result of FOND planning task is not a sequence of actions, but rather a mapping from states to actions, also called policy. The following definitions are taken from Cimatti et al. (2003).

**Definition 38 (Policy).** A policy $\pi$ is a mapping $\pi : \mathcal{S} \to \mathcal{A} \cup \{\bot\}$ from states

to actions such that for all $s \in \mathcal{S}$, $\pi(s) \in \mathcal{A}(s) \cup \{\bot\}$, where $\mathcal{A}(s)$ is the set of actions applicable in state $s$ and $\bot$ means that the policy is undefined in that state. Let $\mathcal{S}_\pi = \{s \in \mathcal{S} | \pi(s) \neq \bot\}$ be the set states for which the policy $\pi$ is defined.

**Definition 39 (Policy properties).** Let $s, s'$ be two states in $\mathcal{S}$. Then $s'$ is reachable from state $s$ iff there is a sequence $s_0, \ldots, s_n$ with $s_0 = s$ and $s' = s_n$ such that for all $i = 0, \ldots, n-1$, $s_{i+1} \in s_i[a]$. A state $s'$ is called reachable from state $s$ following the policy $\pi$ if $s'$ is reachable in $n$ steps or less following the policy $\pi$. Let $\mathcal{S}_\pi(s) \subseteq \mathcal{S}$ be the set of states reachable from state $s$ that do not fulfill the goal condition.

A (partial) policy is *closed* with respect to state $s \in \mathcal{S}$ iff $\mathcal{S}_\pi(s) \subseteq \mathcal{S}_\pi$, or in words if every state reachable from $s$ following $\pi$ is defined in $\pi$.

A policy is *proper* with respect to a state $s$ iff a goal state can be reached from all states $s' \in \mathcal{S}_\pi(s)$, thus from any state reachable by the policy a goal state can be reached.

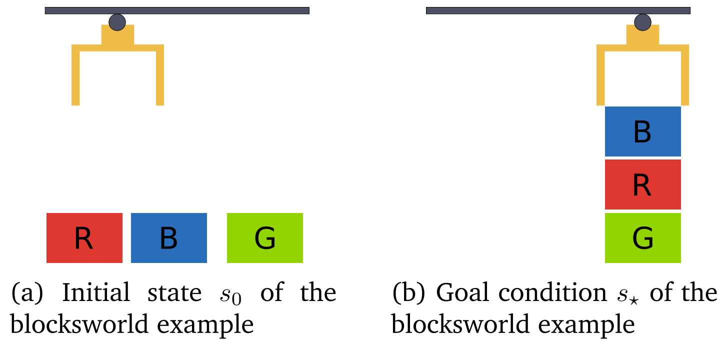
A policy is *acyclic* in respect to state $s$ iff there exists no state $s' \in \mathcal{S}_\pi(s)$ such that $s'$ is reachable from state $s'$ in $n > 0$ steps following the policy $\pi$ .

**Definition 40 (Weak, Strong Cyclic, and Strong policies).**

- A policy is *weak* if the goal state $s_\star$ can be reached from the initial state $s_0$ following $\pi$.
- A policy is *strong cyclic* if it is *closed* and *proper*.
- A policy is *strong* if it is *closed*, *proper*, and *acyclic*.

**Example 9 (FOND planning).**
Consider the FOND planning task $\Pi$ consisting of a set of blocks which should be rearranged in to stacks by robot gripper (also referred to in the literature as Blocksworld domain). The status of every block is described by its location *on(other-block)* on another block or *on-table* on the table, or *gripped* in the robot gripper. Additionally, the fact *clear* states if another block is on top of the block or not, and the variable *empty-gripper* states if the robot gripper is currently empty or not. For the three block example illustrated in Figure 3.7, this can be formalized as follows:

- There exist 3 blocks $\{R, G, B\}$.
- For each block $b \in \{R, G, B\}$ a position *pos* variable exists with domains $\mathcal{D}_{pos} = \{on(b') | b' \in \{R, G, B\} \setminus \{b\} \cup \{\textit{on-table}, \textit{in-gripper}\}$.




(a) Initial state $s_0$ of the blocksworld example

(b) Goal condition $s_\star$ of the blocksworld example

Figure 3.7: Blocksworld example

- For each block $b \in \{R, G, B\}$ a variable indicating if it is clear *clear(b)* with domain $\mathcal{D}_{clear} = \{\top, \bot\}$ exists.
- A Boolean variable *gripped* with domain $\mathcal{D}_{gripped} = \{\top, \bot\}$.

The initial state $s_0$ is

$$\begin{aligned} s_0 = \{&pos(R) = \textit{on-table}, pos(G) = \textit{on-table}, pos(B) = \textit{on-table}, \\ &clear(R) = \top, clear(G) = \top, clear(B) = \top, \\ &\textit{gripped} = \bot\} \end{aligned}$$

The goal condition is then

$$s_\star = \{pos(R) = \textit{on(G)}, pos(B) = \textit{on(R)}, pos(G) = \textit{on-table}\}$$

The set of nondeterministic actions are:

- For every pair of blocks $b, c \in \{R, G, B\}$ there is a *pickup(b,c)* action with two possible effects, one being that the block $b$ is held by the gripper, or the block $b$ falling on to the table:

$$\begin{aligned} \textit{pre} &= \{gripped = \bot, clear(b) = \top, pos(b) = \textit{on}(c)\} \\ \textit{eff}_1 &= \{gripped = \top, clear(c) = \top, pos(b) = \textit{in-gripper}\} \\ \textit{eff}_2 &= \{clear(c) = \top, pos(b) = \textit{on-table}\} \end{aligned}$$

- For every block $b \in \{R, G, B\}$ there is a *pickup-from-table(b)* action, where the first effect is that the gripper holds the block. The second effect is empty and represents the failure of picking up the block:

$$\begin{aligned} \textit{pre} &= \{gripped = \bot, clear(b) = \top, pos(b) = \textit{on-table}\} \\ \textit{eff}_1 &= \{gripped = \top, pos(b) = \textit{in-gripper}\} \\ \textit{eff}_2 &= \emptyset \end{aligned}$$

- For every pair of blocks $b, c \in \{R, G, B\}$ there is a *put-on-block(b,c)* action, where one effect represents the successful placing of block $b$ on block $c$ and the second effect is the failure of doing so resulting in block $b$ falling on to the table.

$$\begin{aligned}\textit{pre} &= \{gripped = \top, clear(c) = \top, pos(b) = \textit{in-gripper}\}\\ \textit{eff}_1 &= \{gripped = \bot, clear(c) = \bot, pos(b) = \textit{on}(c)\}\\ \textit{eff}_2 &= \{gripped = \bot, pos(b) = \textit{on-table}\}\end{aligned}$$

- For every block $b \in \{R, G, B\}$ there is a *put-on-table(b)* action, which can not fail as, dropping the block $b$ also result sin the block being on the table:

$$\begin{aligned}\textit{pre} &= \{gripped = \top, pos(b) = \textit{in-gripper}\}\\ \textit{eff}_1 &= \{gripped = \bot, pos(b) = \textit{on-table}\}\end{aligned}$$

Using these actions a possible policy for reaching a goal state is (noted as a state action mapping $s \rightarrow a$):

$\{pos(R) = \text{on-table}, pos(G) = \text{on-table}, pos(B) = \text{on-table},$
$clear(R) = \top, clear(G) = \top, clear(B) = \top, gripped = \bot\}$
$\rightarrow$ *pick-up(R)*
$\{pos(R) = \text{in-gripper}, pos(G) = \text{on-table}, pos(B) = \text{on-table},$
$clear(R) = \top, clear(G) = \top, clear(B) = \top, gripped = \top\}$
$\rightarrow$ *put-on(G)*
$\{pos(R) = \text{on(G)}, pos(G) = \text{on-table}, pos(B) = \text{on-table},$
$clear(R) = \top, clear(G) = \bot, clear(B) = \top, gripped = \bot\}$
$\rightarrow$ *pick-up(B)*
$\{pos(R) = \text{on(G)}, pos(G) = \text{on-table}, pos(B) = \text{in-gripper},$
$clear(R) = \top, clear(G) = \bot, clear(B) = \top, gripped = \top\}$
$\rightarrow$ *put-on(R)*

There exist multiple approaches for solving such a problem. Three categories of such planners can be identified (T. Geffner and H. Geffner, 2018), Decision diagram based planners such as Gamer (Kissmann and Edelkamp, 2009), graph search planner such as MyND (Mattmüller, 2013), and systems using classical approaches with determinization such as PRP (Muise et al., 2012)

As Chapter 7 is only concerned with user interaction, and thus every policy must result in a successful execution, for the rest of this thesis only *strong*

*cyclic policies* are considered. As plans generated by planning systems, although being correct, might not correspond to a users expectations towards how a goal is reached (more on this in Section 6), soft trajectory constraints are introduced to FOND planning. These, constraints help in producing policies more in line with the users expectations.

# Chapter 4

# Planning with conditional effects and state-dependent action costs

## 4.1 Conditional Effects Revisited

As introduced in Section 3.1 the effects of an action can be expressed as conditional effects. These effects do not always trigger when the corresponding action is executed, but rather when additional conditions hold. Often this makes modeling actions more natural and easier to conceive of. Take the scenario for example, where an agent can move on a grid given by $x$ and $y$ coordinates one step at a time. Instead of defining an action to move right for every possible position in which the agent is able to do so, it would be nicer to define an action *move-right* whose outcome depends on the current position of the agent.

**Example 10 (Conditional effects).** Let the planning task $\Pi$ be the task of an agent moving on a grid trying to reach some target location. Let the variables in $\Pi$ be *pos-x={0,1,2}* and *pos-y={0,1,2}*, with *pos-x=0* and *pos-y=0* be the top right corner in Figure4.1. The *move-right-down* action can now be defined as *move-right-down*= $\langle \{\}, \mathit{eff} \rangle$ with the effect *eff* being the

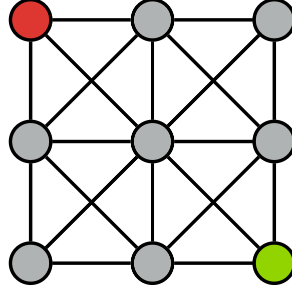


Figure 4.1: Agent moving on a grid layout trying to reach the goal destination. The agents location is indicated in red, whereas the goal location is indicated in green.

conjunction of the following conditional effects:

$$\langle \text{pos-x} = 0 \wedge \text{pos-y} = 0 \rhd \text{pos-x} = 1 \rangle$$
$$\langle \text{pos-x} = 0 \wedge \text{pos-y} = 0 \rhd \text{pos-y} = 1 \rangle$$
$$\langle \text{pos-x} = 1 \wedge \text{pos-y} = 0 \rhd \text{pos-x} = 2 \rangle$$
$$\langle \text{pos-x} = 1 \wedge \text{pos-y} = 0 \rhd \text{pos-y} = 1 \rangle$$
$$\langle \text{pos-x} = 0 \wedge \text{pos-y} = 1 \rhd \text{pos-x} = 1 \rangle$$
$$\langle \text{pos-x} = 0 \wedge \text{pos-y} = 1 \rhd \text{pos-y} = 2 \rangle$$
$$\langle \text{pos-x} = 1 \wedge \text{pos-y} = 1 \rhd \text{pos-x} = 2 \rangle$$
$$\langle \text{pos-x} = 1 \wedge \text{pos-y} = 1 \rhd \text{pos-y} = 2 \rangle$$

As can be seen the *move-right-down* action is always applicable (even when the agent is at the right most position) but has effects that depend on the current position of the agent. Similar, the actions *move-left*, *move-down*, *move-up*, and the remaining diagonal move actions can be defined.

As shown, the effect of an action with conditional effects can be calculated using the changeset from Definition 23. Here now an alternative method is presented using EVMDDs. This EVMDD representation is especially important when combining with state dependent action costs introduced in Section 3.2 and Section 4.2. An EVMDD representing conditional effects *eff* is constructed over the monoid $\mathscr{F} = (2^{\mathcal{F}}, \cup, \emptyset)$ such that $[\mathit{eff}]_s = \mathcal{E}_{\mathit{eff}}(s)$. The EVMDD representing the conditional effects from Example 4.1 is depicted in Figure 4.2.

**Calculating the changeset with EVMDDs.** One question is how to calculate the changeset $[e]_s$ in a given state $s$. Here a method using EVMDDs is introduced. The EVMDD $\mathcal{E}_e = \langle \kappa, \mathbf{f} \rangle$ over the monoid $\mathscr{F} = (2^{\mathcal{F}}, \cup, \emptyset)$, where

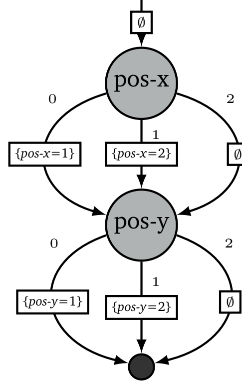



Figure 4.2: The EVMDD representing the conditional effects from Example 4.1

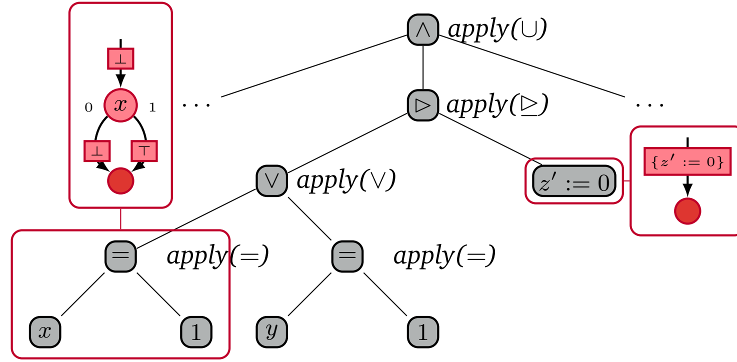



Figure 4.3: The abstract syntax tree for the conditional effects $(\ldots \wedge \langle (x = 1 \vee y = 1) \rhd z = 0 \rangle \wedge \ldots)$, annotated with EVMDD constructions.

$\kappa$ represents unconditional effects and $\mathbf{f}$ represents the function over conditional effects $\mathit{eff} = (\varphi_1 \rhd f_1) \wedge \ldots \wedge (\varphi_n \rhd f_n)$ is created in the following way (Mattmüller et al., 2018):

1. For each $\varphi_i$ an abstract syntax tree (AST) is generated with inner nodes representing Boolean connectives, and leaf nodes representing constants $\top$ and $\bot$ or facts $v = d$ with $v \in \mathcal{V}$ and $d \in \mathcal{D}(v)$, and is visualized in Figure 4.3.

2. For each leaf node an EVMDD over the monoid $\mathscr{B} = (\{\top, \bot\}, \vee, \bot)$ is created: For constants the EVMDD is either $\mathcal{E} = \langle \top, \mathbf{0} \rangle$ for true or $\mathcal{E} = \langle \bot, \mathbf{0} \rangle$ for false. For a fact $v = d$ the EVMDD is $\mathcal{E} = \langle \bot, \mathbf{f} \rangle$ where $\mathbf{f} = (v, \mathbf{0} \ldots \mathbf{0}, w_0, \ldots, w_k)$ with $k = |\mathcal{D}(v)| - 1$ and $w_d = \top$ and $w_i = \bot$ for all $i \neq d$.

3. Then leaf EVMDDs are combined using the *APPLY* procedure (Algo-

rithm 1) with the boolean connective operators from the inner nodes from the expression tree. The resulting EVMDD $\mathcal{E}_{\varphi_i}$ then correctly represents the Boolean expression $\varphi$ and evaluating the EVMDD (Algorithm 5) in a state $s$ for which $\varphi$ is true, results in a path through the EVMDD with exactly one edge with label $\top$ resulting in $s \models \varphi$ iff $\mathcal{E}_{\varphi_i}(s) = \top$.

4. For each effect fact $f$ a constant EVMDD $\mathcal{E}_f = \langle\{f'\}, \mathbf{0}\rangle$ is generated (Figure 4.2 $z' := 0$).

5. The conditional effect EVMDD $\mathcal{E}_{e_i}$ for conditional effect $\varphi_i \triangleright f_i$ is created by applying the operator $\trianglerighteq : \{\top, \bot\} \times 2^{\mathcal{F}} \rightarrow 2^{\mathcal{F}}$ to $\mathcal{E}_{\varphi_i}$ and $\mathcal{E}_{f_i}$ where $\top \trianglerighteq \mathcal{F}' = \mathcal{F}'$ and $\bot \trianglerighteq \mathcal{F}' = \emptyset$ for $\mathcal{F}' \subseteq \mathcal{F}$. In words, the edge with label $\top$ is replaced by the label representing the fact $f'$ and edges with label $\bot$ are replaced by the empty set $\emptyset$.

6. Finally all conditional effect EVMDDs are combined using the union operator $\cup$ resulting in an EVMDD $\mathcal{E}_{\mathit{eff}}$ representing the actions effects.

**Proposition 3.** Let *eff* be an effect in ENF, and let $\mathcal{E}_{\mathit{eff}}$ be the EVMDD representing *eff* constructed as above, and let $s$ be a state. Then $[\mathit{eff}]_s = \mathcal{E}_{\mathit{eff}}(s)$.

*Proof.* Let $s$ be a state and $\varphi \triangleright f$ a conditional effect. Then by Definition 23 $[\mathit{eff}]_s = \{f\}$ iff $s \models \varphi$, and $\emptyset$ otherwise. Let $\mathcal{E}_{\mathit{eff}}$ be the EVMDD representing the conditional effect $\varphi \triangleright f$, then by construction of $\mathcal{E}_\varphi$, $\mathcal{E}_\varphi(s) = \top$ if $s \models \varphi$ and $\bot$ otherwise. Following above construction, this results in $\mathcal{E}_{\mathit{eff}}(s) = \{f\}$ iff $s \models \varphi$ and $\emptyset$ otherwise. Therefore, $[\mathit{eff}]_s = \mathcal{E}_{\mathit{eff}}(s)$. For non conditional effects this obviously also holds, as $\varphi = \top$. For conjunctions of effects, $\mathit{eff} = (\mathit{eff}_0, \ldots, \mathit{eff}_n)$ the combined EVMDD is constructed by combining each individual effect EVMDD with the $\cup$ operator. By construction, this implies that for conjunctions of effects *eff*, $[\mathit{eff}]_s = \mathcal{E}_{\mathit{eff}}(s)$. □

Alternatively to calculating the changeset, a common approach of dealing with conditional effects is to simply compile them away. In the following, three schemata for compiling planning tasks with conditional effects into tasks without conditional effects are presented:

**Exponential Compilation.** The naive approach to compiling away conditional effects is to simply create a new action for each possible combination of the conditional effects. This, however, leads to exponentially many actions in the number of the variables in the effect preconditions ($\prod_{v \in \mathit{vars}(\varphi)} \mathcal{D}_v$), with $\mathit{vars}(\varphi)$ the variables occurring in $\varphi$.

**Example 11 (Exponential compilation of *move-right-down* from Example 4.1).** Given the *move-right-down* action from Example 4.1 creating an

action for each possible variable evaluation, leads to the following sets of actions:

$$
\begin{aligned}
a_1 &:= \langle \text{pos-x} = 0 \wedge \text{pos-y} = 0, \text{pos-x} = 1 \wedge \text{pos-y} = 1 \rangle \\
a_2 &:= \langle \text{pos-x} = 0 \wedge \text{pos-y} = 1, \text{pos-x} = 1 \wedge \text{pos-y} = 2 \rangle \\
a_3 &:= \langle \text{pos-x} = 0 \wedge \text{pos-y} = 2, \emptyset \rangle \\
a_4 &:= \langle \text{pos-x} = 1 \wedge \text{pos-y} = 0, \text{pos-x} = 2 \wedge \text{pos-y} = 1 \rangle \\
a_5 &:= \langle \text{pos-x} = 1 \wedge \text{pos-y} = 1, \text{pos-x} = 2 \wedge \text{pos-y} = 2 \rangle \\
a_6 &:= \langle \text{pos-x} = 1 \wedge \text{pos-y} = 2, \emptyset \rangle \\
a_7 &:= \langle \text{pos-x} = 2 \wedge \text{pos-y} = 0, \emptyset \rangle \\
a_8 &:= \langle \text{pos-x} = 2 \wedge \text{pos-y} = 1, \emptyset \rangle \\
a_9 &:= \langle \text{pos-x} = 2 \wedge \text{pos-y} = 2, \emptyset \rangle
\end{aligned}
$$

In this case $9$ new action where created.

**Nebel's Compilation.** An alternative compilation reducing the number of newly created actions was introduced by Nebel (2000). Here, the original action $a$ is converted to ENF and split into multiple new actions, two for each conditional effect ($a_s$ and $a_f$). The action $a_s$ corresponds to the successful application of the effect (the effect condition $\varphi$ evaluates to true in the current state) and $a_f$ corresponds to not applying the effect ($\varphi$ was false). Additionally, a tracking variable is introduced, enforcing one of the action pairs to be performed for each pair of new actions. To ensure that earlier actions do not affect the outcome of a later action, the effects are first applied to a copy of the original variables, and later applied to the original variables. For this, another set of actions is added, one for each effect variable. Additionally to ensure that all conditional effects are applied, a *aux* variable in introduced, enumerating every conditional effect and ensuring correct progression. As can be seen this compilation has three times as many actions as conditional effects in the original action. In our toy example (Example 12), this however, is significantly more than the exponential compilation. The benefit of Nebel's compilation is only apparent in more complex effect conditions.

**Example 12 (Nebel's compilation of *move-right-down* from Example 4.1).** Converting the *move-right-down* action from Example 4.1 in to ENF results

in:

$$\langle \text{pos-x} = 0 \wedge (\text{pos-y} = 0 \vee \text{pos-y} = 1), \text{pos-x} = 1\rangle$$
$$\langle \text{pos-x} = 1 \wedge (\text{pos-y} = 0 \vee \text{pos-y} = 1), \text{pos-x} = 2\rangle$$
$$\langle (\text{pos-x} = 0 \vee \text{pos-x} = 1) \wedge \text{pos-y} = 0, \text{pos-y} = 1\rangle$$
$$\langle (\text{pos-x} = 0 \vee \text{pos-x} = 1) \wedge \text{pos-y} = 1, \text{pos-y} = 2\rangle$$

Then the Nebel's compilation results in the following actions:

$$\begin{aligned}
a_{init} :=& \langle \textit{pre} \wedge \text{aux} = 0, \text{aux}{=}1\rangle \\
a_{1,s} :=& \langle \text{aux} = 1 \wedge \text{pos-x} = 0 \wedge (\text{pos-y} = 0 \vee \text{pos-y} = 1), \\
& \text{aux} = 2 \wedge \text{pos-x}' = 1\rangle \\
a_{1,f} :=& \langle \text{aux} = 1 \wedge \neg(\text{pos-x} = 0 \wedge (\text{pos-y} = 0 \vee \text{pos-y} = 1)), \\
& \text{aux} = 2\rangle \\
a_{2,s} :=& \langle \text{aux} = 2 \wedge \text{pos-x} = 1 \wedge (\text{pos-y} = 0 \vee \text{pos-y} = 1), \\
& \text{aux} = 3 \wedge \text{pos-x}' = 2\rangle \\
a_{2,f} :=& \langle \text{aux} = 2 \wedge \neg(\text{pos-x} = 1 \wedge (\text{pos-y} = 0 \vee \text{pos-y} = 1)), \\
& \text{aux} = 3\rangle \\
a_{3,s} :=& \langle \text{aux} = 3 \wedge (\text{pos-x} = 0 \vee \text{pos-x} = 1) \wedge \text{pos-y} = 0, \\
& \text{aux} = 4 \wedge \text{pos-y}' = 1\rangle \\
a_{3,f} :=& \langle \text{aux} = 3 \wedge \neg((\text{pos-x} = 0 \vee \text{pos-x} = 1) \wedge \text{pos-y} = 0), \\
& \text{aux} = 4\rangle \\
a_{4,s} :=& \langle \text{aux} = 4 \wedge (\text{pos-x} = 0 \vee \text{pos-x} = 1) \wedge \text{pos-y} = 1, \\
& \text{aux} = 5 \wedge \text{pos-y}' = 2\rangle \\
a_{4,f} :=& \langle \text{aux} = 4 \wedge \neg((\text{pos-x} = 0 \vee \text{pos-x} = 1) \wedge \text{pos-y} = 1), \\
& \text{aux} = 5\rangle \\
a_{x=1} :=& \langle \text{aux} = 5 \wedge \text{pos-x}' = 1, \text{aux} = 6 \wedge \text{pos-x} = 1\rangle \\
a_{x=2} :=& \langle \text{aux} = 5 \wedge \text{pos-x}' = 2, \text{aux} = 6 \wedge \text{pos-x} = 2\rangle \\
a_{y=1} :=& \langle \text{aux} = 6 \wedge \text{pos-y}' = 1, \text{aux} = 7 \wedge \text{pos-y} = 1\rangle \\
a_{y=2} :=& \langle \text{aux} = 6 \wedge \text{pos-y}' = 2, \text{aux} = 7 \wedge \text{pos-y} = 2\rangle \\
a_{final} :=& \langle \text{aux} = 7, \text{aux} = 0\rangle
\end{aligned}$$

The first action $a_{init}$ starts the actions sequence by checking the original precondition and setting the auxiliary variable. The following action $a_{1,s}$ to $a_{4,f}$ are the new action tuples each covering one of the original conditional effects. Finally, actions $a_{X=1}$ to $a_{y=2}$ copy the temporary variable values

back to the original variables. The final action $a_{final}$ then resets the auxiliary variable. Note that all other actions from the planning task may not be executed while the action sequence is executed. This can be archived by adding aux $= 0$ to the precondition of every other action.

**EVMDD Compilation.** Alternatively conditional effects can also be compiled away using the EVMDD representation as described above. Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a \rangle$ be the original task with conditional effects. Let *sem* be a semaphore variable with domain $\mathcal{D}_{sem} = \{0, 1\}$, $\mathcal{V}_{aux} = \{aux_a | a \in \mathcal{A}\}$ indexing variables with domains $\mathcal{D}_{aux_a} = \{0, \ldots, \text{size}(\mathcal{E}_a)\}$ and $\mathcal{V}_c = \{v' | v \in \mathcal{V}\}$ copies of the original variables with domains $\mathcal{D}_{v'} = \mathcal{D}_v$. Let $\mathcal{V}' = \mathcal{V} \cup \mathcal{V}_c \cup \mathcal{V}_{aux} \cup \{sem\}$ be the new state variables.

Let $f_\xi(s) = s \cup \{aux = 0 | aux \in \mathcal{V}_{aux}\} \cup \{v' = s(v) | v' \in \mathcal{V}_c\} \cup \{sem = 0\}$ be a mapping from states in $\Pi$ to states in $\Pi'$, then let $s'_0 = f_\xi(s_0)$ be the new initial state. Let $s'_\star = s_\star \cup \{sem = 0\}$ be the augmented goal state. The actions $\mathcal{A}'$ are then created as follows:

For every action $a = \langle pre, eff \rangle$ in $\mathcal{A}$, an EVMDD $\mathcal{E}_{eff} = \langle \kappa, \mathbf{f} \rangle$ is created and the nodes in $\mathcal{E}_{eff}$ are topologically enumerated denoted as $idx(\mathbf{v})$. The notion of copied variables is extended to facts such that $f = (v = d)$ corresponds to $f' = (v' = d)$ and sets of facts $w = \{f_1, \ldots f_n\}$ correspond to $w' = \{f'_1, \ldots f'_n\}$.

Four types of new actions are created, where for the first three action cost $0$ is assumed:

1. An init action $a_{init} = \langle pre \wedge sem = 0 \wedge aux_a = 0, aux_a = idx(\mathbf{v}_0) \wedge sem = 1 \wedge \kappa' \rangle$ is created. This action ensures, that the precondition of the original action holds, that no other compiled action is executed, and that the execution starts with the first action in this sequence (action corresponding to the incoming edge from $\mathcal{E}_{eff}$). The effect of this action sets the indexing variable to the index of the first node $\mathbf{v}_0$ in $\mathbf{f}$, sets the semaphore variable $sem = 1$ and applies any unconditional effect stated by the label $\kappa$ of the incoming edge.
2. For each nonterminal node $\mathbf{v} = (v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$ and all children $\chi_d$ with label $w_d$ and $0 \leq d \leq |\mathcal{D}(v)| - 1$, a new action $a_{v,d,idx(\mathbf{v})} = \langle aux_a = idx(\mathbf{v}) \wedge v = d, w'_d \wedge aux_a = idx(\chi_d) \rangle$ is created.
3. Let $\mathcal{V}_{eff} = \{v_0, \ldots, v_n\} \subseteq \mathcal{V}$ be the variables changed in *eff*, then a new action for every variable $v_i \in \mathcal{V}_{eff}$ and for every value $d \in \mathcal{D}(v_i)$ is created: $a_{v=d} = \langle aux_a = |\mathcal{E}_{eff}| + i \wedge v' = d, v = d \wedge aux_a = |\mathcal{E}_{eff}| + i + 1 \rangle$.
4. Finally the action $a_{final} = \langle aux_a = |\mathcal{E}_{eff}| + n + 1, aux_a = 0 \wedge sem = 0 \rangle$ with cost $c(a_{final}) = c(a)$ is added, ensuring that $aux_a$ is reset to $0$, the

semaphore variable *sem* is set to $0$, and the original action costs are applied.

The new task without conditional effects is then $\Pi' = \langle \mathcal{V}', \mathcal{A}', s_0', s_\star', c_a' \rangle$.

Note that during the copy actions (3), the value of any variable $v$ is only set to the value of its copy $v'$. As $v'$ is initialized to the value $v$ in the initial state it always holds that $v = v'$ before action $a_{init}$ is executed and after action $a_{\mathit{final}}$ is executed.

Depending on the structure of the original actions, sometimes the Nebel compilation is preferable and sometimes the EVMDD compilation. Analyzing this in depth is left for future work.

**Example 13 (EVMDD compilation of *move-right-down* action).** Given the operator *move-right-down* from Example 4.1 the conditional effects can be represented as the logical expression

$$\begin{aligned}
&(\text{pos-x} = 0 \triangleright \text{pos-x} = 1)\wedge \\
&(\text{pos-x} = 1 \triangleright \text{pos-x} = 2)\wedge \\
&(\text{pos-y} = 0 \triangleright \text{pos-y} = 1)\wedge \\
&(\text{pos-y} = 1 \triangleright \text{pos-y}{=}2)
\end{aligned}$$

From this the EVMDD shown in Figure 4.2 can be constructed.

$$\begin{aligned}
a_{init} &:= \langle \{\mathit{pre} \wedge \text{aux} == 0 \wedge \mathit{sem} = 0\}, \{aux = 1 \wedge \mathit{sem} = 1\}\rangle \\
a_{x,0,1} &:= \langle \{\text{aux} == 1 \wedge \text{pos-x} == 0\}, \{\text{aux} = 2 \wedge \text{pos-x}' = 1\}\rangle \\
a_{x,1,1} &:= \langle \{\text{aux} == 1 \wedge \text{pos-x} == 1\}, \{\text{aux} = 2 \wedge \text{pos-x}' = 2\}\rangle \\
a_{x,2,1} &:= \langle \{\text{aux} == 1 \wedge \text{pos-x} == 2\}, \{\text{aux} = 2\}\rangle \\
a_{y,0,2} &:= \langle \{\text{aux} == 2 \wedge \text{pos-y} == 0\}, \{\text{aux} = 3 \wedge \text{pos-y}' = 1\}\rangle \\
a_{y,1,2} &:= \langle \{\text{aux} == 2 \wedge \text{pos-y} == 1\}, \{\text{aux} = 3 \wedge \text{pos-y}' = 2\}\rangle \\
a_{y,2,2} &:= \langle \{\text{aux} == 2 \wedge \text{pos-y} == 2\}, \{\text{aux} = 3\}\rangle \\
a_{x=0} &:= \langle \{\text{aux} == 3 \wedge \text{pos-x}' == 0\}, \{\text{aux} = 4 \wedge \text{pos-x} = 0\}\rangle \\
a_{x=1} &:= \langle \{\text{aux} == 3 \wedge \text{pos-x}' == 1\}, \{\text{aux} = 4 \wedge \text{pos-x} = 1\}\rangle \\
a_{x=2} &:= \langle \{\text{aux} == 3 \wedge \text{pos-x}' == 2\}, \{\text{aux} = 4 \wedge \text{pos-x} = 2\}\rangle \\
a_{y=0} &:= \langle \{\text{aux} == 4 \wedge \text{pos-y}' == 0\}, \{\text{aux} = 5 \wedge \text{pos-y} = 0\}\rangle \\
a_{y=1} &:= \langle \{\text{aux} == 4 \wedge \text{pos-y}' == 1\}, \{\text{aux} = 5 \wedge \text{pos-y} = 1\}\rangle \\
a_{y=2} &:= \langle \{\text{aux} == 4 \wedge \text{pos-y}' == 2\}, \{\text{aux} = 5 \wedge \text{pos-y} = 2\}\rangle \\
a_{final} &:= \langle \{\text{aux} == 5\}, \{\text{aux} = 0, \mathit{sem} = 0\}\rangle
\end{aligned}$$

From the actions above, the ones copying the values from the primed variables $v'$ back to the unprimed variables $v$ can be optimized. This can be achieved, by removing all those where the value of $v'$ was not changed by the action. in this case this would result in removing actions $a_{x=0}$ and $a_{y=0}$.

**Definition 41.** Given an action $a = \langle pre, eff\rangle$ in $\mathcal{A}$ and a set of EVMDD compiled actions $\hat{a}$ over the effect EVMDD $\mathcal{E}_{eff}$. Then $f_\pi(s,a)$ is a sequence of actions $\overline{a}$ in $\hat{a}$ applicable in $f_\xi(s)$, as follows: $f_\pi(s,a) = (f_\pi^{a,0}(s)$, $f_\pi^{a,+}(s)$, $f_\pi^{a,-}(s)$, $f_\pi^{a,final}(s))$

1. $f_\pi^{a,0}(s)$ is the *init* action from step 1 of the construction.
2. $f_\pi^{a,+}(s)$ is defined recursively over the EVMDD $\mathcal{E}_{eff}$ and returns the sequence of actions from step 2 of the construction applicable in state $s$. For the terminal node $\mathbf{0}$, $f_\pi^{a,+}(s)$ denotes the empty sequence of actions. For each non terminal node $\mathbf{v} = (v, \chi_0, \ldots, \chi_k, w_0, \ldots, w_k)$, $f_\pi^{a,+}(s)$ is the action $a_{v,s(v),idx(\mathbf{v})}$ followed by the action sequence $f_{\pi,s(v)}^{a,+}(s)$ where $f_{\pi,s(v)}^{a,+}(s)$ is the function denoted by the child $\chi_{s(v)}$.
3. $f_\pi^{a,-}(s)$ denotes the sequence of copy actions from step 3 of the construction dependent of the values in $\mathcal{V}_c$.
4. $f_\pi^{a,final}(s)$ is the *final* action from step 4 of the construction.

Recalling Example 13, the parts from above definition correspond to the actions: $f_\pi^{a,0}(s)$ is $a_{init}$. Then $f_\pi^{a,+}(s)$ is the sequence of actions corresponding to the EVMDD edges $a_{x,0,1}, \ldots, a_{y,2,2}$, and $f_\pi^{a,-}(s)$ is the sequence of copy actions $a_{x=1}, \ldots, a_{y=2}$. Finally, $f_\pi^{a,final}(s)$ is $a_{final}$.

The following lemma states that the transformed state $f_\xi(t)$ reached by applying the unary sequence of actions $[a]$ in state $s$ from the original task is equal to applying the sequence of actions $f_\pi(s,a)$ retrieved from the compilation in the transformed state $f_\xi(s)$. Better illustrated by the commutative diagram in Figure 4.4.

**Lemma 3.** For every action $a \in \mathcal{A}$ and every state $s \in \mathcal{S}$ such that $a$ is applicable in $s$, $\delta(f_\xi(s), f_\pi(s,a)) = f_\xi(\delta(s,a))$.

*Proof.* Let $a = \langle pre, eff\rangle$ be an action from $\mathcal{A}$ applicable in state $s$, $t = s[a]$ and the EVMDD $\mathcal{E}_{eff} = \langle \kappa, \mathbf{f}\rangle$ representing the effects. By construction $f_\pi^0(s,a)$ is the action $a_{init} = \langle pre_{init}, eff_{init}\rangle$ form step 1 of the EVMDD compilation. This action is applicable in $f_\xi(s)$ as $f_\xi(s) \models pre_{init} = pre \wedge sem = 0 \wedge aux_a = 0$. Additionally all effects from $\kappa$ are applied in $\kappa'$ (recall the notation $\kappa'$ meaning the copies of the set of facts). By definition $f_\pi^{a,+}(s)$ denotes the sequence of compiled actions from step 2 consistent to state $s$. This is

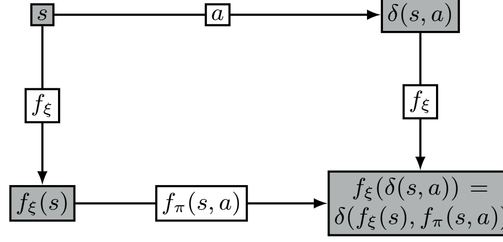



Figure 4.4: Transition diagram of first applying action and transforming the results to the compiled state space, or first transforming the state in to the compiled state space and applying the compiled action sequence.

ensured by tracking the current node of $\mathcal{E}_{eff}$ the action sequence is in, using the *aux* variable, and applying the correct edge label effects by checking the value of $v$. The sequence of actions then corresponds to the path through the EVMDD consistent with state $s$ where each edge is represented by an action. As the edge label effects are only applied to copies of the variables $\mathcal{V}_c$ changing the value of a variable earlier in the sequence does not interfere with precondition evaluations later on. Only once the copy actions $f_\pi^{a,-}(s)$ are executed, are the values copied to the actual variables $\mathcal{V}$. Following this action sequence and the correctness of the EVMDD evaluation, the changeset of the sequence of actions $f_\pi^{a,+}(s), f_\pi^{a,-}(s)$ without $\mathcal{V}_c$,$\mathcal{V}_{aux}$, and *sem*, corresponds to the changeset of the original action $[a]_s$. Finally $f_\pi^{a,final}(s)$ is the last action $a_{final}$ from the construction step 4, resetting the $aux_a$ and *sem* variable. The resulting sequence of actions $(f_\pi^{a,0}(s), f_\pi^{a,+}(s), f_\pi^{a,-}(s), f_\pi^{a,final}(s))$ therefore fulfills $\delta(f_\xi(s), f_\pi(s,a)) = f_\xi(\delta(s,a))$. □

**Proposition 4.** Every planning task $\Pi$ with conditional effects is *plan-equivalent* to its EVMDD compiled task $\Pi'$ without conditional effects.

*Proof.* Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a \rangle$ be a planning task with conditional effects, and $\Pi' = \langle \mathcal{V}', \mathcal{A}', s'_0, s'_\star, c'_a \rangle$ the EVMDD compiled task without conditional effects created as described above. Furthermore, let $\pi = (a_1, \dots, a_k)$ be a plan for $\Pi$ with the induced state sequence $\overline{s} = (s_0, \dots, s_{k+1})$ with $s_{k+1} \models s_\star$ and total costs $c(\pi)$.

Following Lemma 3, for each action $a$ in $\pi$, the state $s \in \overline{s}$ in which it is executed and $t = [a]_s$, an action sequence $f_\pi(s,a)$ in $\mathcal{A}'$ exists such that $\delta(f_\xi(s), f_\pi(s,a)) = f_\xi(t)$. Applying this transformation for each action in $\pi$ results in the new equivalent plan $\pi'$ for $\Pi'$. Furthermore, the total cost from $\pi$ is preserved in $\pi'$, as the cost $c_a$ for every action $a$ in $\pi$ is only applied once in every sequence of actions $f_\pi(s,a)$ of the compiled task $\Pi'$.

Additionally, as the goal condition $s'_\star$ is augmented with $sem = 0$, no intermediate state $s'$ created by applying $f_\pi(s,a)$ in state $f_\xi(s)$ can exist such that

$s' \models f_\xi(s'_\star)$. Following this, and the fact that by construction every sequence of actions $f_\pi(s,a)$ can be associated with an action $a$ from the original task, it can be seen that for every plan $\pi'$ in $\Pi'$ there exists an equivalent plan $\pi$ in $\Pi$. □

### 4.1.1 Discussion

As planning is *PSPACE-hard* in the size of the input, the size of the task representation is critical. Adding only a small number of variables or actions can greatly increase the number of possible states the planner must search through. Therefore, when transforming a task with conditional effects into a task without conditional effects, one major criterion is the increase of the search space. However, a second major criterion is the *friendliness* towards the applied heuristic during search. *Friendliness* hereby relates to how much better or worse the heuristic value becomes in relation to the true cost of reaching the goal from a given state. In this section the three introduced compilations are theoretically analyzed and compared in regards of size and heuristic friendliness using the $h^{\text{add}}$ heuristic introduced earlier in this thesis. As the name implies the exponential compilation is exponential in the size of the input and therefore provides the largest increase in size of the three compilations. On the other hand, when using the $h^{\text{add}}$ heuristic the heuristic value $h^{\text{add}}_{exp}$ of the compiled task is equal to the heuristic value of the original task $h^{\text{add}}_{exp} = h^{\text{add}}$, as calculating $h^{\text{add}}$ value for the preconditions *pre*$(a)$ of an achiever of a given fact $f$ takes the effect preconditions into account. Due to the fact that the effect conditions where simply lifted to the actions precondition this presents no change when calculating the $h^{\text{add}}$ value. In contrast the Nebel compilation has a linear growth in the number of conditional effects, however multiple problems arise when it comes to the heuristic value. Recalling the nature of relaxation heuristics such as $h^{\text{add}}$ facts that have been reached once stay true during the calculation of the heuristic. The Nebel compilation introduces an *aux* variable which ensures that all conditional effects from the original action are handled in the compilation. However, once the action was applied once, the *aux* variable already has all possible values, thus in a repeated application of the same action the handling of every conditional effect can no longer be guaranteed. This leads to an unnecessary underestimation of the actual heuristic value. Recalling the example shown in the Nebel compilation, let the first application of the action sequence result in always applying $a_{i,f}$ for $1 \leq i \leq 4$, thus the conditional effects never triggered. In a second application of the same action the *aux* variable already holds all values from $0$ to $7$. Let *pos-y*$=1$ then only actions $a_{4,s}$ and $a_{y=2}$ will be executed resulting in *pos-y*$=2$ and all other actions will be ignored. Depending on the actual implementation, the original action costs will either be applied in $a_{init}$ or $a_{final}$. These actions

however are not reapplied, thus the cost of achieving *pos-y*$=2$ is $0$ instead of the original action costs. Additionally, recalling the definition of $h^{\text{add}}$ the cost of making a fact true is calculated using $\min[h^{\text{add}}s(pre) + c_a]$, where the cost of achieving the precondition is added to the cost of achieving the fact. As the Nebel compilation adds a new action for every conditional effect, and these conditions can reason over the same facts multiple times, calculating the cost over the sequence of actions might require adding $h^{\text{add}}s(pre)$ multiple times. This can result in an over approximation of the heuristic value. Both problem combined can lead to an arbitrarily uninformed heuristic. The EVMDD compilation tries to find a middle ground. It is worst case exponential, however in practice shows much smaller size increases, and does not suffer all the problems the Nebel compilation has. As every decision variable from the conditional effects are only checked once, the $h^{\text{add}}s(pre)$ of achieving a fact $f$ is also only calculated once. Even though the EVMDD compilation has a similar issue regarding the values of its *aux* variable, as already reached values are kept, the implication is not quite so bad. This is due to the fact, that *aux* may only assume values that lie on the path from the root node to the terminal node of the corresponding EVMDD. Thus, actions corresponding to outgoing edges of not yet visited nodes require that the aux variable *aux* first assume the correct value. Additionally, both the Nebel and the EVMDD compilation suffer from the fact that the once a compiled action is executed the semaphore variable *sem* has both values $0$ and $1$. Thus, it can no longer be ensured that other actions can be executed during the execution of a compiled action sequence. One solution to the problem with the *aux* and *sem* variables however, is to forbid it's relaxation or abstraction in the heuristic calculation.In the relaxation case, this implies that any fact representing a value of the *sem* and *aux* variables do not remain achieved during the rest of the search, but rater keep the concrete value from the unrelaxed task. The same holds for the abstractions, where the two variable values may not be abstracted. in conclusion, the EVMDD compilation provides a compacter representation as the exponential compilation, and a more heuristic friendly structure than the Nebel compilation. However, it is still exponential in the worst case, and provides less accurate heuristic values than the exponential compilation.

## 4.2 Combining State Dependent-Action Costs and Conditional Effects

The previous two sections introduced conditional effects and state dependent action costs. This section focuses on how conditional effects and state-dependent action costs are related, and the issues that arise when using both together in a single planning task.

During the search for the plan, SDAC and conditional effects work nicely together, as the current state is fully specified. However, problems arise when using relaxation heuristics, where variables from the effect conditions and state-dependent action cost function may take on multiple values, and they have variables in common. Calculating the cost of an action will therefore result in choosing the variable values minimizing the cost function, whereas calculating the effect will take the union over all possible effects. This is best illustrated by the following example where the task is to climb a mountain, where the base of the mountain is at the bottom left of the $6 \times 6$ grid and peak is at the top right corner. The slope becomes steeper, the further up the mountain, thus the cost of moving becomes higher the higher up the mountain the agent is. There are only four actions

$$\begin{aligned}
\text{move-right} &= \langle \emptyset, (x = 0 \rhd x = 1) \wedge \ldots \wedge (x = 4 \rhd x = 5)\rangle, cost = x + 2y \\
\text{move-left} &= \langle \emptyset, (x = 5 \rhd x = 4) \wedge \ldots \wedge (x = 1 \rhd x = 0)\rangle, cost = x + 2y \\
\text{move-up} &= \langle \emptyset, (y = 0 \rhd y = 1) \wedge \ldots \wedge (y = 4 \rhd y = 5)\rangle, cost = x + 2y \\
\text{move-down} &= \langle \emptyset, (y = 5 \rhd y = 4) \wedge \ldots \wedge (y = 1 \rhd y = 0)\rangle, cost = x + 2y
\end{aligned}$$

These actions are always applicable and the effect is dependent on the current position. The goal is to move from the initial left bottom position to the right top goal position. This task is illustrated in Figure 4.5. A possible optimal solution is to first apply *move-right* five times followed by *move-up* five times. Both have a total plan cost of $75$.

Calculating the cost for applying *move-right* in a relaxed state $s^+ = \{x = 0, x = 1, x = 2, x = 3, x = 4, y = 0\}$ (highlighted in Figure 4.5) will add the fact $x = 5$ resulting in the new state $s^{+'} = \{x = 0, x = 1, x = 2, x = 3, x = 4, x = 5, y = 0\}$. The true cost for this action would be $4$ as applying this action in state $\{x = 4, y = 0\}$ for reaching the state where $x = 5$ is true is $x + 2y = 4$. However, as the action costs are minimized the cost function will use the variable values $\{x = 0, y = 0\}$ resulting in the cost of applying the action being $0$. The same behavior can be observed for all other actions, resulting in a initial heuristic value of $h^{\text{add}}(s_0) = 0$ instead of the optimal $h^\star_0 = 75$.

Additionally, the heuristic value may even increase towards the goal. Given

Figure 4.5: Climbing example. Initial position bottom left, goal position top right. Darker shades indicate higher costs to move.

the initial state $s_0 = \{x = 0, y = 0\}$. From above calculation, the heuristic $h^{\text{add}}(s_0)$ will be $0$, as applying *move-right* and *move-up* in the relaxed initial state always costs $0$. However, calculating the heuristic value $h^{\text{add}}(s_1)$ for the successor state $s_1 = \{x = 1, y = 1\}$ will result in $h^{\text{add}}(s_1) = 1$. Thus, the heuristic value actually increases, the closer to $s_\star$ the search gets, resulting in a completely uninformed heuristic.

A solution to this problem however exists. SDAC and conditional effects must not be treated separately by taking the minimum of the action cost, and the union of effects, but rather the interaction must be accounted for. For the above example this means, that the union of effects is taken, but a different cost is applied to each effect.

### 4.2.1 Relaxed planning with state-dependent action costs and conditional effects

As before in Section 3.2, dealing with the combination of conditional effects and state-dependent action costs is trivial during the search where each variable $v$ only consists of on value $d \in \mathcal{D}_v$ within the state $s \in \mathcal{S}$. However, problems arise when the state contains multiple values for each variable, as stated above.

First a new definition of the change set from Definition 27 is introduced, where instead of a set of facts, a set of pairs of facts and associated costs is used.

**Definition 42 (Changeset with SDAC and CE).** Let $s^+$ be a relaxed state

and $a = \langle pre, eff \rangle$ be an action with precondition *pre* and effect *eff* in ENF with a cost function $c : \mathcal{S} \to \mathbb{N}$. Then the changeset of *eff* in $s^+$ is $[eff]^c_{s^+} = \bigsqcup_{s \in \mathcal{S} : s \subseteq s^+} [\![eff]\!]^c_s$ with:

$$[\![eff_1 \wedge \ldots \wedge eff_n]\!]^c_s = [\![eff_1]\!]^c_s \cup \ldots \cup [\![eff_n]\!]^c_s$$
$$[\![\varphi \rhd f]\!]^c_s = \{(f, c(s))\} \text{ if } s \models \varphi$$
$$[\![\varphi \rhd f]\!]^c_s = \emptyset \text{ if } s \not\models \varphi$$
$$\bigsqcup_j E_j = \{(f, n) \in \bigcup_j E_j | \forall (f, l) \in \bigcup_j E_j : l \geq n\}.$$

where $e \in eff$ and $\bigsqcup$ the minimizing union.

The changeset $[eff]^c_{s^+}$ consists of all facts $f$ that can be achieved by applying $a$ in any state $s$ with $s \subseteq s^+$. Additionally every fact $f$ is associated with the minimal cost at which $f$ can be achieved.

Computing this changeset however comes at a high cost, as there may exist exponentially many states $s$ (in the number of the state variables $\mathcal{V}$) such that $s \subseteq s^+$. It can also be shown by reduction from SAT that computing the changeset is **NP-hard**.

*Proof.* Computing the change set from Definition 42 is **NP-hard**, even for unit costs, as a reduction from SAT shows: A propositional formula $\varphi$ is satisfiable iff $(f, 1) \in [\varphi \rhd f]^c_{s^+}$, where $c_a = 1$ for all $s$, and $s^+(v) = \mathcal{D}_v$ for all $v \in \mathcal{V}$. □

Recalling that SDAC can be represented as EVMDD over the monoid $\mathscr{N} = (\mathbb{N}, +, 0)$ and conditional effects as EVMDD over the monoid $\mathscr{F} = (2^{\mathcal{F}}, \cup, \emptyset)$. The combination of both EVMDDs may be constructed using the monoid $\mathscr{X} = \mathscr{N} \times \mathscr{F}$.

**Definition 43 (Product EVMDD).** Let $\mathcal{L}$ and $\mathcal{R}$ be two MSOCMMs, and $\mathcal{E}_f$ and $\mathcal{E}_g$ two EVMDDs representing the functions $f : \mathcal{S} \to \mathcal{L}$ and $g : \mathcal{S} \to \mathcal{R}$ respectively. Then $\mathcal{E}_{f,g}$ is defined as $\mathcal{E}_f \oplus \mathcal{E}_g$ with $\oplus : \mathcal{L} \times \mathcal{R} \to \mathcal{L} \times \mathcal{R}$ being the identity function $\oplus(l, r) = (l, r)$ for $(l, r) \in \mathcal{L} \times \mathcal{R}$.

**Proposition 5.** Given $f$ and $g$ as in the above definition. Assume a fixed variable ordering and let $s \in \mathcal{S}$. Then $\mathcal{E}_{f,g}(s) = (f(s), g(s))$.

*Proof.* Following the definition of Algorithm 1, *apply*$(\mathcal{E}_f, \mathcal{E}_g, \bullet) = \mathcal{E}_{f \bullet g}$. Using the the identity operator $\bullet = \oplus$, $\mathcal{E}_{f,g} = \mathcal{E}_f \oplus \mathcal{E}_g =$ *apply*$(\mathcal{E}_f, \mathcal{E}_g, \oplus) = \mathcal{E}_{f \oplus g}$. By definition of the EVMDD product (Definition 43) and the *apply* algorithm and its correctness follows: $\mathcal{E}_{f,g}(s) = \mathcal{E}_{f \oplus g}(s) = (f, g)(s) = (f(s), g(s))$ (Mattmüller et al., 2018). □

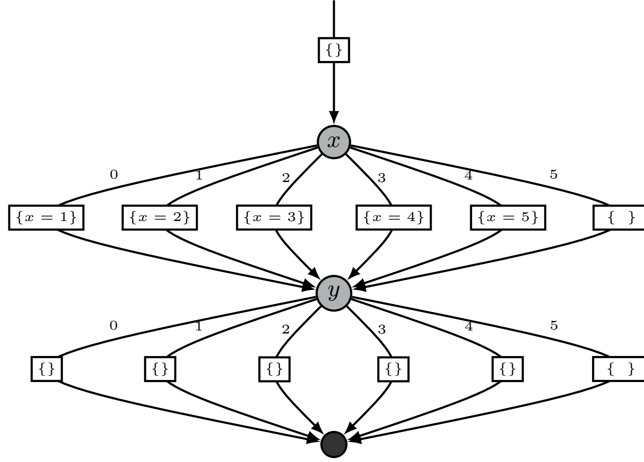



Figure 4.6: Shannon reduced conditional effect EVMDD for the *move-right* action.

**Corollary 1.** Let $c : \mathcal{S} \to \mathbb{N}$ be an arithmetic expression with $\mathcal{E}_c$ its representing EVMDD and $e : \mathcal{S} \to \mathcal{F}$ be a conditional effect in ENF represented by the EVMDD $\mathcal{E}_e$. If $\mathcal{E}_{c,e} = \mathcal{E}_c \oplus \mathcal{E}_e$ and $s \in \mathcal{S}$ a state, then $\mathcal{E}_{c,e}(s) = (\mathcal{E}_c(s), \mathcal{E}_e(s)) = (c(s), [e]_s)$. □

Reviewing the example action *move-right* from above, the two EVMDDs representing the conditional effects and the cost function are depicted in Figure 4.7 and Figure 4.6 respectively. The application of *apply* with the $\oplus$ operator on these two EVMDDs results in the combined EVMDD depicted in Figure 4.8.

The size of the product EVMDD $\mathcal{E}_{f,g} = \mathcal{E}_f \otimes_{\mathcal{E}} \mathcal{E}_g$ is limited by the product of the sizes of the factors of $\mathcal{E}_f$ and $\mathcal{E}_g$. However, in the case that both EVMDDs $\mathcal{E}_f$ and $\mathcal{E}_g$ share the same topology (as is the case in Figures 4.6, and 4.7), the combined EVMDD has the same topology as the two input EVMDDs. Additionally, if $\mathcal{E}_f$ and $\mathcal{E}_g$ do not share any variables,the resulting EVMDD is simply the second EVMDD ”glued“ to the end of the first EVMDD.

Using this combined EVMDD $\mathcal{E}_{(c,e)}$ a polynomial time (in the size of the EVMDD) algorithm, taking as input the relaxed state $s^+$ is presented for calculating the changeset (Mattmüller et al., 2018). For this the EVMDD $\mathcal{E}_{(c,e)}$ is traversed along the topological ordering from top to bottom (root node to terminal node) restricting the edges to those consistent to $s^+$. For each encountered node $\mathbf{v}$ a set $F$ of fact cost tuples $\langle f, c_f \rangle$ consisting of an entry for each achieved fact along any path from the root node to $\mathbf{v}$ together with its cheapest achieving cost $c_f$. Additionally the cost $n$ of the cheapest path to $\mathbf{v}$ is stored. The notation $\mathcal{E}_{(c,e)}(s^+)(\mathbf{v})$ is used denoting the

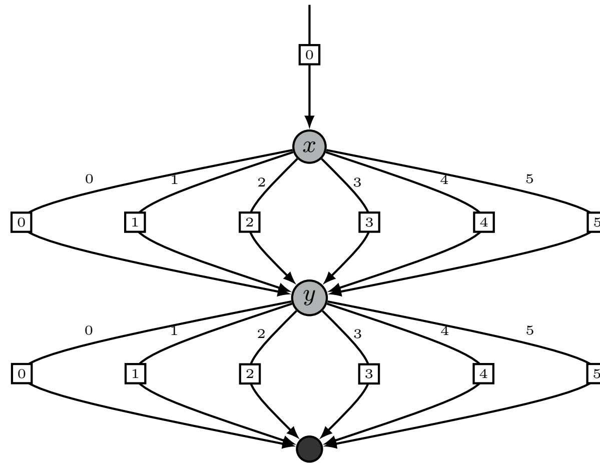



Figure 4.7: Cost function EVMDD for the *move-right* action.

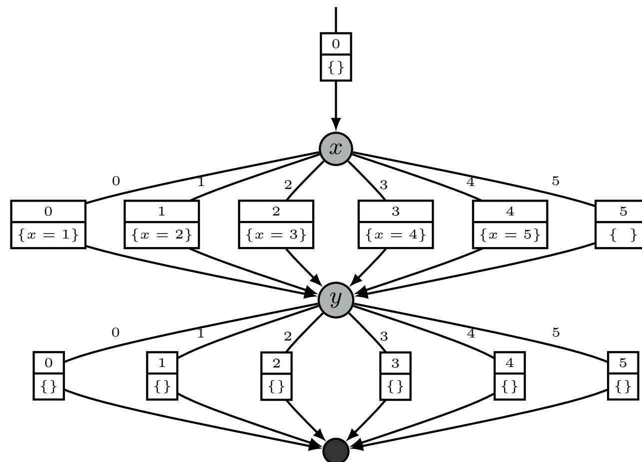



Figure 4.8: Combination of cost and effect EVMDD for the *move-right* action.

evaluation of $\mathcal{E}_{(c,e)}$ with the state $s^+$ up to the node $\mathbf{v}$. The evaluation of the whole EVMDD is then $\mathcal{E}_{(c,e)}(s^+)(\mathbf{0})$ with $\mathbf{0}$ the terminal node.

Formally, let $\mathbf{v}_1 \dots \mathbf{v}_t$ be a topological ordering, corresponding to the variable ordering from the construction, of $\mathcal{E}_{(c,e)}$ where $\mathbf{v}_t$ is the terminal node $\mathbf{0}$. For the base case $i = 1$ the node $\mathbf{v}_1$ has only the dangling incoming edge with the label $\kappa = (c, E)$ with $c \in \mathbb{N}$ the constant cost term, and $E \in \mathcal{F}$ the set of unconditional effects of the action. $\mathcal{E}_{(c,e)}(s^+)(\mathbf{v}_1) = (F, c)$ with $F = \{(f, c_f) | f \in E, c_f = c\}$.

For $i > 1$, let $\mathbf{v}_i$ be an interior node of $\mathcal{E}_{(c,e)}$, and $\mathbf{v}_j$ be a node corresponding to a decision variable $v_j$ in $\mathcal{E}_{(c,e)}$, with cost $c_j$ such that an edge $e$ with label $(c', E')$ with the edge constraint $v_j = d_j \in \mathcal{D}(v_j)$ with $(v_j = d_j) \in s^+$ exists from $\mathbf{v}_j$ to $\mathbf{v}_i$. Let $F^{old} = \{(f, c' + c) | (f, c) \in F_j\}$, the fact cost tuples of node $\mathbf{v}_j$ plus the cost from the edge label reaching $\mathbf{v}_i$. Then let $F^{new} = \{(f, c' + c_j) | f \in E'\}$ be the new facts added by the edge label from $e$. The new fact costs set is then $F_i = F^{old} \sqcup F^{new}$. Determining $n_i$ for node $\mathbf{v}_i$ the minimum of $n_j + c'$ and $n_i$ is calculated every time the node $\mathbf{v}_i$ is reached by some parent node $\mathbf{v}_k$. Obviously, on the first encounter of $\mathbf{v}_i$, $n_i = n_j + c$. Finally, the result of evaluating $\mathcal{E}_{(c,e)}$ in state $s^+$ at node $\mathbf{v}_i$ is $\mathcal{E}_{(c,e)}(s^+)(\mathbf{v}_i) = (F_i, n_i)$.

**Proposition 6.** Let $s^+$ be a relaxed state and $a = \langle pre, eff \rangle$ an action with effects in ENF and a cost function $c : \mathcal{S} \to \mathbb{N}$. Let $\mathcal{E}_{(c,e)}$ be the product of the cost function EVMDD $\mathcal{E}_c$ and the effect EVMDD $\mathcal{E}_{eff}$. With the above described evaluation procedure $[eff]^c_{s^+} = \mathcal{E}_{(c,e)}(s^+)$.

*Proof.* $[eff]^c_{s^+}$ and $\mathcal{E}_{(c,e)}(s^+)$ are by definition functional sets of fact cost pairs $(f, c)$ where each fact only occurs once. this follows from the usage of the minimizing union operator $\sqcup$ in both $[eff]^c_{s^+}$ and $\mathcal{E}_{(c,e)}(s^+)$ evaluations.

1. The sets of facts in $[eff]^c_{s^+}$ and $\mathcal{E}_{(c,e)}(s^+)$ are identical: By definition a fact $f$ is in $[eff]^c_{s^+}$ iff there exists an unrelaxed state $s \subseteq s^+$ such that the effect condition is satisfied in $s$, written $f \in [eff]_s$. This is equivalent to $f$ being in $\mathcal{E}_{eff}$ according to Proposition 3, which according to the product construction is equivalent to $f$ being in a label on the path of $\mathcal{E}_{(c,e)}$ corresponding to $s$. This in turn is equivalent to $f \in \mathcal{E}_{(c,e)}(s^+)$, since during the evaluation process of $\mathcal{E}_{(c,e)}$ the labels of all paths given by all $s \subseteq s^+$ are collected, and never discarded.

2. The costs assigned to the facts in $[eff]^c_{s^+}$ and $\mathcal{E}_{(c,e)}(s^+)$ are identical: In $[eff]^c_{s^+}$, for fact $f$, by definition this is the minimal cost $c(s)$ at which $f$ can be achieved in any state $s \subseteq s^+$. Let $s$ be such a state minimizing $c(s)$ at which fact $f$ is achieved, then $(f, c(s))$ is also in $\mathcal{E}_{(c,e)}(s^+)$:

From Proposition 1 it is known that $c(s)$ is the sum of edge labels in $\mathcal{E}_c$ for the path spanned by $s$. By definition of the product construction the same labels from $\mathcal{E}_c$ are also present in $\mathcal{E}_{(c,e)}$ for $s$. Moreover, evaluating $\mathcal{E}_{(c,e)}$ with $s$ the same path will be traversed (thus the same labels visited). The cost associated with $f$ in $\mathcal{E}_{(c,e)}$ along the path given by $s$ is first determined after the edge where $f$ appears as label, and the cost is the cost given by the label and the prefix (minimal cost $n_j$ of reaching the parents node $\mathbf{v}_j$) .From there whenever the fact $f$ is propagated, the cost $c'$ from the corresponding edge is added to the facts cost as given by the definition of $F_j^{old}$. The cost $c(s)$ coming from evaluating over $s$ is never removed in a minimizing union operation $\sqcup$, since $s$ is already the minimizing state. From this follows: $\mathcal{E}_{(c,e)}(s^+)(f) \leq [\mathit{eff}]^c_{s^+}(f)$. For $\mathcal{E}_{(c,e)}(s^+)(f)$ strictly smaller than $[\mathit{eff}]^c_{s^+}(f)$ there must exists a minimizing state $s' \subseteq s^+$ such that $\mathcal{E}_{(c,e)}(s^+)(f) < [\mathit{eff}]^c_{s^+}(f)$. However, this state $s'$ would also have to be taken in to account when calculating $[\mathit{eff}]^c_{s^+}(f)$, this is a contradiction, thus $\mathcal{E}_{(c,e)}(s^+)(f) = [\mathit{eff}]^c_{s^+}(f)$.

□

**Example 14 (Changeset for relaxed states with SDAC and CE).** Illustrating above procedure, recall the combined EVMDD from Figure 4.8, given a relaxed state $s^+ = \{x = \{1,2\}, y = \{3,4\}\}$, then at decision node $x$ the intermediate result is $\mathcal{E}_{(c,e)}(s^+)(\mathbf{v}_x) = (F_x, n_x)$, with $F_x = \emptyset$, and $n_x = 0$. In the decision node $y$ the intermediate result is: $\mathcal{E}_{(c,e)}(s^+)(\mathbf{v}_y) = (F_y, n_y)$, with $F_y = \emptyset \sqcup \{(x = 2, 1)\} \sqcup \{(x = 3, 2)\} = \{(x = 2, 1), (x = 3, 2)\}$, and $n_y = 1$, as the cheapest path in $s^+$ is the edge $e_1$ with cost label 1. Finally, the result in the terminal node $\mathbf{v_0}$ is $\mathcal{E}_{(c,e)}(s^+)(\mathbf{v_0}) = (F_\mathbf{0}, n_\mathbf{0})$ with $F_\mathbf{0} = \{(x = 2, 1+3)\} \sqcup \{(x = 3, 2+3)\} \sqcup \{(x = 2, 1+4)\} \sqcup \{(x = 3, 2+4)\} = \{(x = 2, 4), (x = 3, 5)\}$, and $n_\mathbf{0} = 4$.

**Compiling away SDAC together with CE.** Alternatively to calculating the changeset for relaxed state with conditional effects and state dependent action costs, it is also possible to use the EVMDD structure to compile away conditional effects and state dependent action costs together, similar to the approaches described in Sections 4.1 and 3.2.

Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a(s) \rangle$ be a planning task with conditional effects and state-dependent action cost. Let *sem* be a semaphore variable with domain $\mathcal{D} = \{0, 1\}$, $\mathcal{V}_{\mathit{aux}} = \{\mathit{aux}_a | a \in \mathcal{A}\}$ indexing variables with domains $\mathcal{D} = \{0, \dots, \mathit{size}(\mathcal{E}_a)\}$ and $\mathcal{V}_c = \{v' | v \in \mathcal{V}\}$ with $\mathcal{D}_{v'} = \mathcal{D}_v$. Then, $\mathcal{V}' = \mathcal{V} \cup \mathcal{V}_{\mathit{aux}} \cup \mathcal{V}_c \cup \{\mathit{sem}\}$ are the new state variables. Let $f_\xi(s) = s \cup \{\mathit{aux}_a = 0 | \mathit{aux}_a \in \mathcal{V}_{\mathit{aux}}\} \cup \{v' = s(v) | v' \in \mathcal{V}_c\} \cup \{\mathit{sem} = 0\}$ be the mapping from states in $\Pi$ to states in $\Pi'$, with $f_\xi(s_0) = s'_0$ and $s'_\star = s_\star \cup \{\mathit{sem} = 0\}$. For each

action $a = \langle \textit{pre}, \textit{eff} \rangle$ in $\mathcal{A}$ with cost function $c_a(s)$ a quasi-reduced product EVMDD $\mathcal{E}_{c,e} = \langle \kappa, \mathbf{f} \rangle$ with $\kappa = \langle c, e \rangle$ is created and its nodes topologically enumerated, denoted as $\textit{idx}(\mathbf{v})$.

Then four types of actions are created:

1. The action $a_{\textit{init}} = \langle \textit{pre} \wedge \textit{sem} = 0 \wedge \textit{aux}_a = 0, e' \wedge \textit{sem} = 1 \wedge \textit{aux}_a = \textit{idx}(\mathbf{v}_0) \rangle$ with cost $c$ is created.
2. For each non terminal node $\mathbf{v} = (v, \chi_0, \dots \chi_k, w_0, \dots, w_k)$ and all children $\chi_d$ with label $w_d = \langle c_d, e_d \rangle$ and $0 \leq d \leq |\mathcal{D}| - 1$, a new action $a_{v,d,\textit{idx}(\mathbf{v})} = \langle \textit{aux}_a = \textit{idx}(\mathbf{v}) \wedge v = d, e'_d \wedge \textit{aux}_a = \textit{idx}(\chi_d) \rangle$ with cost $c_d$ is created.
3. Let $\mathcal{V}_{\textit{eff}} = \{v_0, \dots v_n\} \subseteq \mathcal{V}$ be the variables changed in *eff*, then a new action for every variable $v_i \in \mathcal{V}_{\textit{eff}}$ and for every value $d \in \mathcal{D}(v_i)$ with cost $0$ is created: $a_{v=d} = \langle \textit{aux}_a = |\mathcal{E}_{c,e}| + i \wedge v' = d, v = d \wedge \textit{aux}_a = |\mathcal{E}_{c,e}| + i + 1 \rangle$.
4. Finally the action $a_{\textit{final}} = \langle \textit{aux}_a = |\mathcal{E}_{c,e}| + n + 1, \textit{aux}_a = 0 \wedge \textit{sem} = 0 \rangle$ with cost $0$ is added, resetting $\textit{aux}_a$ and *sem* to $0$.

**Example 15.** Given the *move-right* action from above, and its product EVMDD $\mathcal{E}_{c,e}$ depicted in Figure 4.8, the compilation results in the following set of actions:

$$
\begin{aligned}
a_{init} &:= \langle \textit{pre} \wedge \textit{aux} = 0 \wedge \textit{sem} = 0, \textit{aux} = 1 \wedge \textit{sem} = 1 \rangle & \textit{cost} = 0 \\
a_{x_0} &:= \langle \textit{aux} = 1 \wedge x = 0, \textit{aux} = 2 \wedge x' = 1 \rangle & \textit{cost} = 0 \\
a_{x_1} &:= \langle \textit{aux} = 1 \wedge x = 1, \textit{aux} = 2 \wedge x' = 2 \rangle & \textit{cost} = 1 \\
a_{x_2} &:= \langle \textit{aux} = 1 \wedge x = 2, \textit{aux} = 2 \wedge x' = 3 \rangle & \textit{cost} = 2 \\
a_{x_3} &:= \langle \textit{aux} = 1 \wedge x = 3, \textit{aux} = 2 \wedge x' = 4 \rangle & \textit{cost} = 3 \\
a_{x_4} &:= \langle \textit{aux} = 1 \wedge x = 4, \textit{aux} = 2 \wedge x' = 5 \rangle & \textit{cost} = 4 \\
a_{x_5} &:= \langle \textit{aux} = 1 \wedge x = 5, \textit{aux} = 2 \rangle & \textit{cost} = 5 \\
a_{y_0} &:= \langle \textit{aux} = 2 \wedge y = 0, \textit{aux} = 3 \rangle & \textit{cost} = 0 \\
a_{y_1} &:= \langle \textit{aux} = 2 \wedge y = 1, \textit{aux} = 3 \rangle & \textit{cost} = 2 \\
a_{y_2} &:= \langle \textit{aux} = 2 \wedge y = 2, \textit{aux} = 3 \rangle & \textit{cost} = 4 \\
a_{y_3} &:= \langle \textit{aux} = 2 \wedge y = 3, \textit{aux} = 3 \rangle & \textit{cost} = 6 \\
a_{y_4} &:= \langle \textit{aux} = 2 \wedge y = 4, \textit{aux} = 3 \rangle & \textit{cost} = 8 \\
a_{y_5} &:= \langle \textit{aux} = 2 \wedge y = 5, \textit{aux} = 3 \rangle & \textit{cost} = 10 \\
a_{c_1} &:= \langle \textit{aux} = 3 \wedge x' = 1, \textit{aux} = 4 \wedge x = 1 \rangle & \textit{cost} = 0 \\
a_{c_2} &:= \langle \textit{aux} = 3 \wedge x' = 2, \textit{aux} = 4 \wedge x = 2 \rangle & \textit{cost} = 0 \\
a_{c_3} &:= \langle \textit{aux} = 3 \wedge x' = 3, \textit{aux} = 4 \wedge x = 3 \rangle & \textit{cost} = 0 \\
a_{c_4} &:= \langle \textit{aux} = 3 \wedge x' = 4, \textit{aux} = 4 \wedge x = 4 \rangle & \textit{cost} = 0
\end{aligned}
$$

$$a_{c_5} := \langle aux = 3 \wedge x' = 5, aux = 4 \wedge x = 5\rangle \qquad cost = 0$$
$$a_{c_6} := \langle aux = 4 \wedge y' = 1, aux = 5 \wedge y = 1\rangle \qquad cost = 0$$
$$a_{c_7} := \langle aux = 4 \wedge y' = 2, aux = 5 \wedge y = 2\rangle \qquad cost = 0$$
$$a_{c_8} := \langle aux = 4 \wedge y' = 3, aux = 5 \wedge y = 3\rangle \qquad cost = 0$$
$$a_{c_9} := \langle aux = 4 \wedge y' = 4, aux = 5 \wedge y = 4\rangle \qquad cost = 0$$
$$a_{c_10} := \langle aux = 4 \wedge y' = 5, aux = 5 \wedge y = 5\rangle \qquad cost = 0$$
$$a_{final} := \langle aux = 5, aux = 0 \wedge sem = 0\rangle \qquad cost = 0$$

As already mentioned in Example 13 the copy actions can again be optimized, as variable $y'$ is never altered by the actions. Thus, actions $a_{c_6}$ to $a_{c_10}$ can be removed.

**Definition 44.** Given an action $a = \langle pre, eff\rangle$ in $\mathcal{A}$ with cost $c_a$ and a set of EVMDD compiled actions $\hat{a}$ over the combined EVMDD $\mathcal{E}_{(c,e)}$, then $f_\pi(s, a)$ is a sequence of actions in $\hat{a}$ applicable in $f_\xi(s)$ analog to Definition 41.

The following lemma states that the transformed state $f_\xi(t)$ reached by applying the unary sequence of actions $[a]$ in state $s$ from the original task is equal to applying the sequence of actions $f_\pi(s, a)$ retrieved from the compilation in the transformed state $f_\xi(s)$. Also, the cost of applying action $a$ in state $s$ is equal to the total cost of applying the sequence $f_\pi(s, a)$ in state $f_\xi(s)$: $f_c(s, a) = c_a(s)$.

**Lemma 4.** For every action $a \in \mathcal{A}$ and every state $s \in \mathcal{S}$ such that $a$ is applicable in $s$, $\delta(f_\xi(s), f_\pi(s, a)) = f_\xi(\delta(s, a))$ and $f_c(s, a) = c_a(s)$.

*Proof.* Let $a = \langle pre, eff\rangle$ be an action from $\mathcal{A}$ with cost $c_a(s)$ applicable in state $s$ and $t = s[a]$ and $\mathcal{E}_{c,e} = \langle \kappa, \mathbf{f}\rangle$ its the product EVMDD representing $(c, eff)$. Then following Lemma 3, Lemma 2 and the correctness of the product over EVMDDs $\mathcal{E}_{eff}$ and $\mathcal{E}_c$, $\delta(f_\xi(s), f_\pi(s, a)) = f_\xi(\delta(s, a))$ and $f_c(s, a) = c_a(s)$. □

**Proposition 7.** Every planning task $\Pi$ with conditional effects and state-dependent action costs is *plan-equivalent* to its product EVMDD compiled task $\Pi'$ without conditional effects or state-dependent action costs.

*Proof.* Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a\rangle$ be a planning task with conditional effects and state-dependent action cost, and $\Pi' = \langle \mathcal{V}', \mathcal{A}', s'_0, s'_\star, c'_a\rangle$ the EVMDD compiled task without conditional effects and state-dependent action costs created as described above. Furthermore, let $\pi = (a_1, \ldots, a_k)$ be a plan for $\Pi$ with the induced state sequence $\overline{s} = (s_0, \ldots, s_{k+1})$ with $s_{k+1} \models s_\star$ and total costs $c(\pi)$. Following Lemma 4 for each action $a \in \pi$, the state $s \in \overline{s}$ in which it is executed and $t = [a]_s$, an action sequence $f_\pi(s, a)$ in

$\mathcal{A}'$ exists such that $\delta(f_\xi(s), f_\pi(s,a)) = f_\xi(t)$ and $f_c(s,a) = c_a(s)$. Applying this transformation for each action in $\pi$ results in the new equivalent plan $\pi'$ for $\Pi'$ with identical total costs. Additionally, as the goal condition $s'_\star$ is augmented with *sem* $= 0$ no intermediate state $s'$ created by applying $f_\pi(s,a)$ in state $f_\xi(s)$ can exist such that $s' \models f_\xi(s'_\star)$. Following this, and the fact that by construction every sequence of actions $f_\pi(s,a)$ can be associated with an action $a$ from the original task, it can be seen that for every plan $\pi'$ in $\Pi'$ there exists an equivalent plan $\pi$ in $\Pi$ with identical total cost. □

### 4.2.2 Discussion

As with the conditional effect compilation from Section 4.1, the EVMDD compilation is compared to the exponential compilation regarding size and heuristic friendliness. Let the exponential compilation create a new action for every possible variable value combination of the variables in the cost function and the conditional effects, as described in Section 4.1. As before, the exponential compilation, while being exponential in size, retains the original heuristic values for relaxation heuristics $h^{\text{max}}$ and $h^{\text{add}}$. This is due to the fact that the preconditions pre$(f)$ of a fact $f$, when calculating $h^{\text{max}}$ or $h^{\text{add}}$, already take both the effect conditions and the actions preconditions in to account. In contrast, the EVMDD compilation provides a more complex representation in practice, while being worst case exponential. It however trades the compact representation with heuristic friendliness. As before in Section 4.1 this compilation also suffers from abstraction or relaxation of the *aux* and *sem* variables, resulting in an underestimation of the actual heuristic value. Again this can be solved by ensuring that the *aux* and *sem* variables are not relaxed or abstracted (Geißer, 2018a).

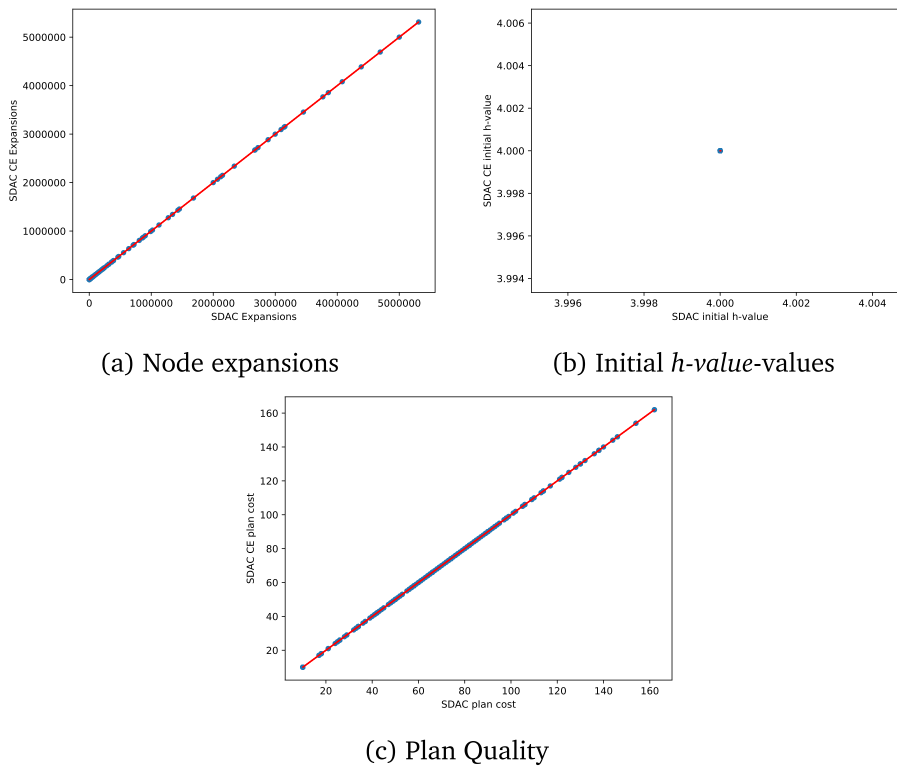
Moreover, this section shows that it is important not to treat all variables in a planning task as independent. Often some variable values will depend on other variable values, thus the search can be guided more efficiently if this correlation of variables is taken in to account. Frances and H. Geffner (2015) discuss such a correlation of variables in a scenario where $n$ variables $X_1, \ldots, X_n$ have an initial value of $0$ and the goal is, by applying an increment on individual variables, to reach the inequality $X_i < X_{i+1}$. Considering binary variables only one possible encoding for this problem would be as follows: One predicate *val*$(i,k)$ for each value $X_i = k$, and predicate *less*$(i,j)$ for each inequality $X_i < X_{i+1}$ and the goal defined by the conjunction of *less*$(i, i+1)$ for each variable $0 < i < n-1$. The single action available is *increment(v)* with cost $1$, which increments the value of any given variable. Calculating $h^{\text{max}}$ would result in an initial heuristic value of $1$ as each goal can be achieved by applying *increment(v)* once for every variable . In the relaxed task every variable then hast the values $\{0, 1\}$

fulfilling every inequality requirement. Taking the maximum cost over all these actions results in a heuristic value of $1$. Taking $h^{\text{add}}$ instead would result in an heuristic value of the sum of action costs being $n-1$. Whereas, the correct value for reaching the goal would be $i$ actions for each child $X_i$ : $1+2+3+\ldots+n-1 = n(n-1)/2$. It is clear to see, that these variables may not be treated separately, but rather in the context in which they are set by the goal condition. This shows, that there are many settings in which variables may not be treated as independent, but rather depend on each others values. One such correlation between variables in conditional effects and state-dependent action costs was shown in this section, together with a possible solution. However, as illustrated above, there are many more settings in which variables are interdependent. This work is seen as a step in the right direction, but more work is required to fully understand how correlating variables may be treated throughout in planning.

### 4.2.3 Empirical evaluation

The EVMDD compilation for state-dependent action costs ignoring conditional effects and the compilation combining the two concepts are evaluated using the *Asterix* domain (Speck, 2018). In this domain the comic character is tasked with bringing the *Edleweiss* to the village. This can be achieved in two ways. First by climbing the mountain with increasing cost the higher one gets ($x * y$ with $x$ and $y$ the coordinates on the mountain). Alternatively, Asterix can first go to the Forrest, fend of the Romans, gather some Mistletoe, and brew a magic potion. With this potion, climbing the mountain becomes much easier, as it reduces the cost for climbing to $1$. When using the $h^{\text{max}}$ heuristic (Figure 4.9), no difference between the two compilations can be detected. Even the initial heuristic value equal to $4$ for every instance. As expected, the plan quality is also identical in every instance, due to the fact that $h^{\text{max}}$ is admissible. This acts as a sanity check, to show that the properties of the heuristics are really preserved in compilation. Second the two heuristics $h^{\text{add}}$ (Figure 4.10) and $h^{\text{ff}}$ (Figure 4.11) are compared. Here one can see that when using $h^{\text{add}}$ the number of node expansions is much higher when using the combined compilation, as compared to the SDAC only compilation. Even though this might indicate worse performance in regards to the search time, one can see, that this results in better plan quality (lower total costs). On the other hand, when using the $h^{\text{ff}}$ heuristic, the number of node expansions is way lower when using the combined compilation. Here now significant difference in plan quality can be measured. The initial heuristic value for both heuristics, is higher when using the combined compilation. This emphasizes the initial claim, that without combined treatment, the heuristic provides high cost effects to low action costs. As a side mark, one should note, that only relaxation heuristics where used here, as

using abstraction heuristics with EVMDD compilations poses multiple challenges described in detail in Geißer (2018). In summary, one can say that with this approach, one can improve the quality of the resulting plan, or improve the runtime, depending on the chosen heuristic function.

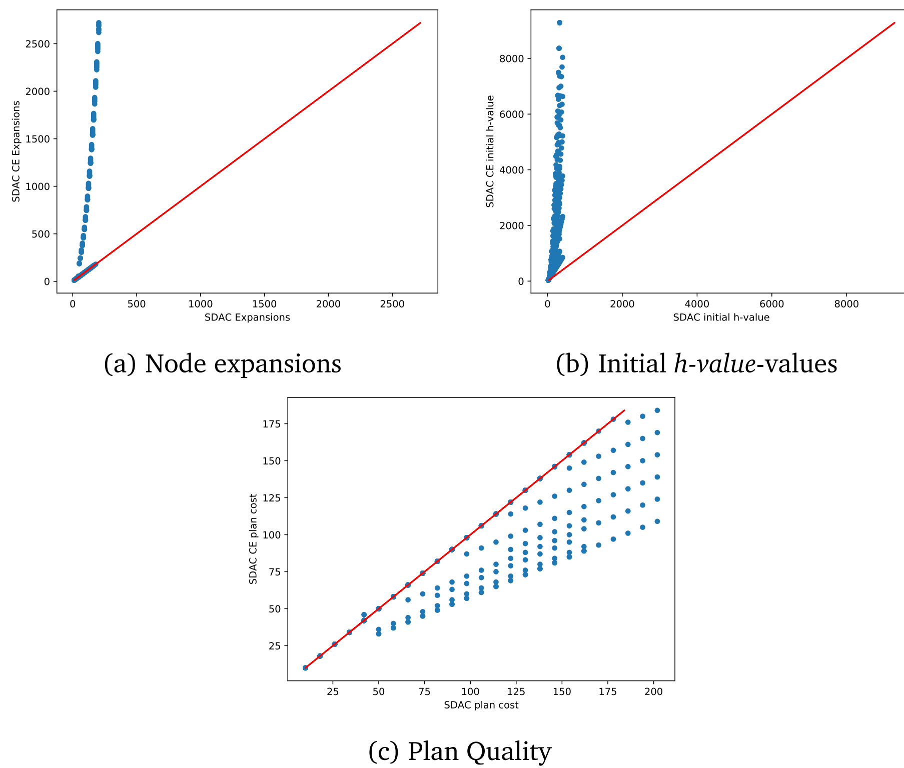



(a) Node expansions

(b) Initial *h-value*-values

(c) Plan Quality

Figure 4.9: Results for SDAC only compilation vs SDAC with CE compilation and $h^{\max}$ heuristic

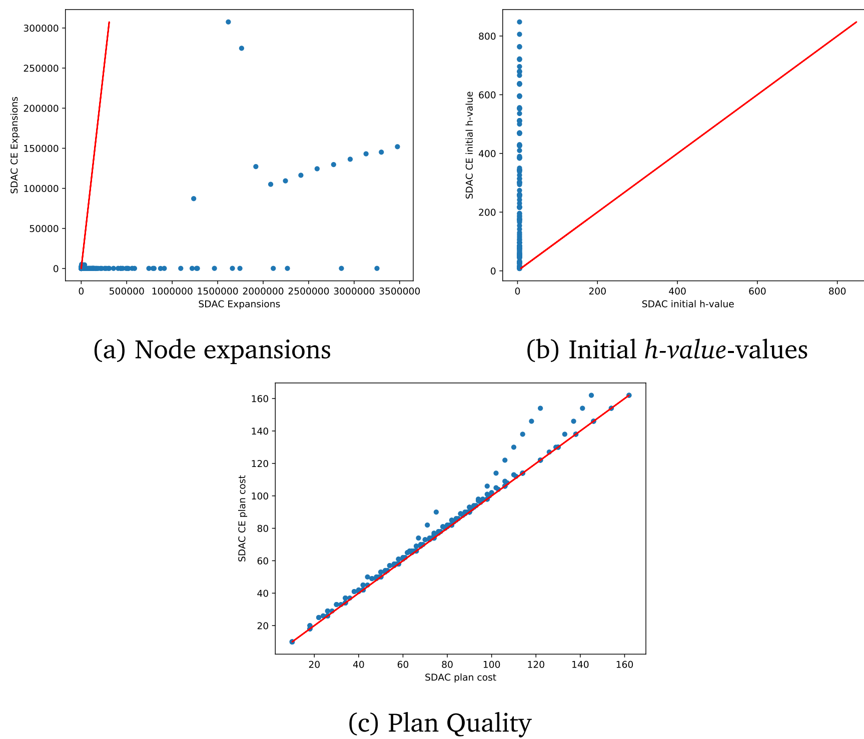



(a) Node expansions

(b) Initial *h-value*-values

(c) Plan Quality

Figure 4.10: Results for SDAC only compilation vs SDAC with CE compilation and $h^{\text{add}}$ heuristic




(a) Node expansions

(b) Initial *h-value*-values

(c) Plan Quality

Figure 4.11: Results for SDAC only compilation vs SDAC with CE compilation and $h^{\mathrm{ff}}$ heuristic

# Chapter 5

# Planning with soft trajectory constraints

In real world applications, it is often necessary to not only find a valid plan regarding the goal condition, but to add additional requirements towards how the plan is achieved.

**Example 16.** Recalling Example 4, let $\varphi$ be the requirement that the green package in Graz must be picked up first due to time constraints. Thus the planner would now disregard picking up the red package in Vienna first and drive directly to Graz to pick up the green package first. As can be seen, this not only adds flexibility to modeling how the goal is achieved, but in some cases even reduce the search space. In this example this is achieved by disregarding all plans that pick up the red package in Vienna before driving to Graz, basically removing half of the search tree from Figure 3.2.

This kind of constraint is considered to be *hard*, as it is a required condition for the plan to be valid. On the other hand, sometimes it is useful to be able to additionally be able to state optional or soft constraints towards the plan. These, in contrast to the hard constraints, are not required to be fulfilled by the plan, but rather have an impact on the resulting plan quality. Therefore soft constraints are usually associated with some kind of numeric value indicating how important the constraint is.

**Example 17.** Recalling Example 4 again, an optional goal might be to bring the red package to Freiburg as fast as possible. Thus, a more desired plan might be to first bring the red package to Freiburg and then drive to Graz. As can be seen, this is not an optimal plan for the original task without the constraints. However, depending on the importance of the constraint (the importance outweighs additional cost), this might be an optimal plan for the constrained task. If the importance is set too low the planner would most

likely disregard it, on the other hand increasing the importance will at some point result in the optimal plan fulfilling the constraint.

## 5.1 State trajectory constraints

PDDL 3.0 (Gerevini and Long, 2005) adds state trajectory constraints to the PDDL language expressed as temporal modal operators over first order formulas consisting of state predicates (facts). PDDL 3.0 introduces following modal operators, where $\varphi$ and $\psi$ are first order formulas over predicates and $n$ and $m$ are natural numbers:

- *at-end* $\varphi$: $\varphi$ must hold in the goal state (goal condition).
- *always* $\varphi$: $\varphi$ must hold in every state of the plan.
- *sometime* $\varphi$: $\varphi$ must hold at some point in the plan.
- *within* $n$ $\varphi$: $\varphi$ must hold within the next $n$ states of the plan.
- *at-most-once* $\varphi$: $\varphi$ must hold at most once during the whole plan.
- *sometime-after* $\varphi$ $\psi$: $\psi$ must hold at some point after $\varphi$ holds.
- *sometime-before* $\varphi$ $\psi$: $\psi$ must hold before $\varphi$ holds.
- *always-within* $n$ $\varphi$ $\psi$: $\varphi$ must hold within $n$ states after $\psi$ holds.
- *hold-during* $n$ $m$ $\varphi$: $\varphi$ must hold between the $n^{\text{th}}$ state and the $m^{\text{th}}$ state.
- *hold-after* $n$ $\varphi$: $\varphi$ must hold after the $n^{\text{th}}$ state.

**Definition 45.** Each PDDL 3.0 modal operator can be represented by an $\mathsf{LTL_f}$ formula over state predicates.

- *at-end* $\varphi := \lambda \wedge \varphi$
- *always* $\varphi := \Box\varphi$
- *sometime* $\varphi := \Diamond\varphi$
- *within* $m$ $\varphi := \bigvee_{0 \leq i \leq m} \underbrace{\bigcirc \ldots \bigcirc}_{i} \varphi$
- *at-most-once* $\varphi := \Box(\varphi \rightarrow \varphi \mathcal{W} \neg\varphi)$
- *sometime-after* $\varphi$ $\psi := \Box(\varphi \rightarrow \Diamond\psi)$
- *sometime-before* $\varphi$ $\psi := (\neg\varphi \wedge \neg\psi)\mathcal{W}(\neg\varphi \wedge \psi)$
- *always-within* $m$ $\varphi$ $\psi := \Box(\varphi \rightarrow \bigvee_{0 \leq i \leq m} \underbrace{\bigcirc \ldots \bigcirc}_{i} \psi)$

- *hold-during* $m\ k\ \varphi := \underbrace{\bigcirc \ldots \bigcirc}_{m}(\bigwedge_{0 \le i \le k} \underbrace{\bigcirc \ldots \bigcirc}_{i} \varphi)$
- *hold-after* $m\ \varphi := \underbrace{\bigcirc \ldots \bigcirc}_{m} \Diamond \varphi$

Note, that it might sometimes be beneficial to be able to reason about action trajectory constraints instead of state trajectory constraints (e.g action $a_a$ must be performed before action $a_b$). However, this can be easily transformed into state trajectory constraints by adding a predicates $exec_a$ and $exec_b$ that are only true if the action has been performed. Thus these constraints will not be considered in this work.

To be able to verify if a given plan is valid in regards to a trajectory constraints, it is no longer sufficient to check if a state (goal state) fulfills the goal condition, but rather if the whole state trajectory fulfills constraints, and the last state fulfills the goal condition.

**Definition 46 (Planning task with hard trajectory constrains).** Given a set of state trajectory constraints $\Phi = \langle \varphi_1, \ldots, \varphi_k \rangle$, a planning task with state trajectory constraint is then defined as $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a, \Phi \rangle$ with $\mathcal{V}, \mathcal{A}, s_0, s_\star$, and, $c_a$ as defined in the state-dependent action cost setting (Definition 32). A plan $\pi$ with its induced state trajectory $\mu_\pi = \langle s_0, s_1, s_2, \ldots, s_n \rangle$ with $\mu_\pi \models s_\star$ is then a valid plan for $\Pi$ if $s_n \models s_\star$ and $\mu_\pi \models \varphi$ for every $\varphi \in \Phi$.

Soft trajectory constraints are, as hard trajectory constraints, requirements towards the induced state trajectory. However, the fulfillment is optional, meaning a plan that does not fulfill the soft trajectory constraint does not become invalid. Not fulfilling such a constraint, may be the result of too high costs for doing so, or the specification of contradicting constraints. Recalling the Example 4 of the truck transporting packages from one location to another, a constraint might be to visit every city only once (*at-most-once at-truck(Vie)* and *at-most-once at-truck(Graz)* and *at-most-once at-truck(Fr)*). As in the example the city *Graz* is only reachable via *Vienna* it is not possible to bring the package from *Graz* to *Freiburg* without visiting *Vienna* twice. In other settings where *Graz* may be directly connected to *Freiburg* this constraint however might be satisfiable. A soft trajectory constraint therefore only impacts the quality of a plan rather than its validity. For this, constraints can be associated with a numeric value indicating its relative importance in respect to other constraints and general action costs.

**Definition 47 (Planning task with soft trajectory constraints).** Given a set of state trajectory constraint importance value tuples $\Phi = \langle (\varphi_1, \omega_1), \ldots, (\varphi_k, \omega_k) \rangle$ a planning task with state trajectory constraint is then defined as $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a, \Phi \rangle$ with $\mathcal{V}, \mathcal{A}, s_0, s_\star$, and, $c_a$ as defined in the state-

dependent action cost setting (Definition 32). A plan $\pi = \langle a_0, \ldots, a_{n-1} \rangle$ with its induced state trajectory $\mu_\pi = \langle s_0, s_1, \ldots, s_n \rangle$ is then a valid plan for $\Pi$ if $s_n \models s_\star$. The total cost $c(\mu_\pi)$ of $\pi$ is then the sum of actions plus the importance of every trajectory constraint as penalty for every constraint not fulfilled by $\mu_\pi$.

$$\sum_{i=0}^{n-1} c_{a_i} + \sum_{\varphi \in \Phi} (\omega_\varphi | \mu_\pi \not\models \varphi) \quad (5.1)$$

From this definition, it can be seen, that selecting the actual numeric value $\omega$ indicating the importance is crucial, as this influences if a soft trajectory constraint will be satisfied or not. If $\omega$ is chosen too low, any additional action taken to fulfill this will come with higher cost than the penalty for not achieving it. On the other hand scaling $\omega$ too high might result in very high plan costs as the planner will try to fulfill them no matter what the action costs for fulfilling it are.

A simple approach to dealing with trajectory constraints in planning is to track them during the search. For this an automaton for the corresponding LTL formula is created, and its state updated after every action. However, this requires that not only the planner be adapted, but also the heuristic functions must be redefined as to support trajectory constraints. Alternatively, trajectory constraints can be compiled away (Edelkamp, 2006), enabling of-the-shelf planners to plan with state trajectory constraints. Given a planning task with trajectory constraints $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a, \Phi \rangle$ a non-deterministic finite automaton $\mathcal{A}$ for every trajectory constraint $\varphi \in \Phi$ is created. Each edge of the NFA is hereby labeled with conditions over propositions of the planning task. The planning task is then augmented with the predicate *at(q,$\mathcal{A}$)* stating that automaton $\mathcal{A}$ is in state $q$, and a predicate *accepting*($\mathcal{A}$) which is true if $\mathcal{A}$ is in an accepting state. Then synchronization actions are added, which take the current state of the automaton, and the transition condition from the automaton edges and sets the new state of the automaton. Finally, the preconditions of the original actions are augmented by a *synchronize* flag, stating that these actions may only be executed after the synchronization actions where performed (one for every constraint). Thus the resulting plan with $n$ constraints is a alternating execution of original actions and synchronization actions: $(a_0, \mathrm{sync}_{0,\varphi_1}, \ldots, \mathrm{sync}_{0,\varphi_\mathrm{n}}, \ldots, a_m, \mathrm{sync}_{\mathrm{m},\varphi_1}, \ldots, \mathrm{sync}_{\mathrm{m},\varphi_\mathrm{n}})$. Finally the goal specification is augmented to also require every automaton to be in an accepting state. This approach can also be be extended to soft trajectory constraints by adding the penalty for not achieving the constraint to the plan metric instead of invalidating the plan (removing the *accepting ?a - automaton* from the goal condition).

Note that De Giacomo et al. (2014) provide an algorithm for directly con-

structing NFAs from the $\mathsf{LTL_f}$ formula, whereas the original algorithm by Edelkamp (2006) uses LTL formula and therefore must take a detour via Büchi automata. Also note that the NFA interpretation of LTL formula is not generally valid, but holds for the formula required for PDDL 3.0 constraints.

An alternative approach for tracking the state of trajectory constraints is to incorporate the tracking of the automatons state within the original actions. This can be achieved by adding conditional effects consisting of the current automatons state, and the transition conditions from the edges to the original actions (basically merging the transition actions from above in to the original actions) (J. A. Baier and McIlraith, 2006).

A special case of soft trajectory constraints are *soft goals*, considering only constraints towards the goal state. These can be either expressed as optional goal conditions, or using the PDDL 3.0 temporal modal operator *at-end* over facts: *at-end*$f$, where $f$ is a fact. For this special case Keyder and H. Geffner (2009) showed that soft goals can be compiled away. For a given soft goal $p = (\textit{at-end } f)$ two actions are added to the end of the action sequence, one called the $collect(p)$ action, applying zero action costs if the soft goal $p$ is fulfilled, or $forgo(p)$ with a numeric penalty cost if the soft goal was not fulfilled (the fact $f$ does not hold in the goal state). Additionally a new fact is added to the new goal condition indicating that one of the two actions was executed, thus adding the penalty of not achieving the soft goal to the total cost of the plan.

**Example 18.** Let $x = 1$ be a soft goal over the binary variable $x$ with assigned value $5$, and the goal condition $s_\star$. Then the Keyder and H. Geffner (2009) compilation adds two new actions:

$$\begin{aligned} \textit{collect}_{x=1} &: \langle x = 1, \textit{done}_{x=1} = 1 \rangle \qquad && 0 \\ \textit{forgo}_{x=1} &: \langle x = 0, \textit{done}_{x=1} = 1 \rangle \qquad && 5 \end{aligned}$$

The goal condition is then adapted to $s_\star = s_\star \cup \{\textit{done}_{x=1} = 1\}$. Whenever $x = 1$ is violated, the action $\textit{forgo}_{x=1}$ will add the penalty of $5$, and the action $\textit{collect}_{x=1}$ will add no penalty if the soft goal is achieved. Note that additional bookkeeping facts ensuring that the *collect* and *forgo* actions are executed last are required.

An alternative representation of soft trajectory constraints was introduced by J. A. Baier et al. (2009), where each constraint is represented by a parametrized non deterministic automaton. In contrast to the other approaches using NFAs or Büchi automata, PNFAs do not require the planning task to be grounded, but rather have lifted expressions on their edges(predicates over typed variables). The benefit of this approach being that for constraints over a whole type of objects, only a single automaton is constructed. Assuming an example with two objects $A$ and $B$ of type *package* and the constraint

*forall (p - package) sometime-after (loaded(p)) (delivered(p))* (Figure 5.1, stating that every package that has been loaded at some point must be unloaded at a future time step. Then a predicate *state(s,O,$\mathcal{A}$)* is added to the

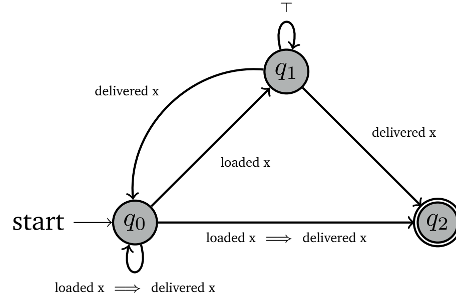



Figure 5.1: Parametrized NFA for the constraint *forall(x - package) sometime-after(loaded x) (delivered x)*. Adapted from J. A. Baier et al. (2009).

planning task indicating in which state $s$ of the automaton $\mathcal{A}$ the object $O$ resides in. Note, that due to the non-determinism, each object may reside in multiple states at once. In contrast to other approaches, state transitions are not coded in to the planning task, but are rather handled by the planner internally. This is achieved by applying the transitions of the automata (setting the predicates) independently after every planning action is executed. Thus every state of the planning task holds all information required to identify the state of every automaton object tuple, without requiring expensive synchronization actions (Edelkamp, 2006) or automaton tracking in conditional effects (J. A. Baier and McIlraith, 2006).

Here now a novel approach (first published in (Wright et al., 2018c; Wright et al., 2018b))for compiling away soft trajectory constraints utilizing conditional effects and state-dependent action costs is presented, providing tracking functionality with conditional effects and heuristic guidance using state-dependent action costs.

First an automata representation of the soft trajectory constraints in the planning task is introduced:

**Definition 48 (Automata semantics of planning tasks with state trajectory constraints).** The transition system of a planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a, \Phi \rangle$ can be represented as a DFA $\mathcal{A}(\Pi) = \langle \Sigma, Q, \Delta, q_0, Q_a \rangle$ with $\Sigma = \mathcal{S}(\mathcal{V}')$, where $\mathcal{V}' = \mathcal{V}(\Phi)$ are the variables relevant to the trajectory constraints only. The transition function $\Delta$ consists of transitions of the form $\langle s, s|\mathcal{V}', s' \rangle$, where $s|\mathcal{V}'$ is $s$ restricted to the variables in $\mathcal{V}'$.

As can be seen, the resulting automaton lacks some essential information. For reconstructing the plan from a given trace of the automaton, information

about the actions is required. This can be achieved by simply adding the action to the label of each transition. Same holds for action costs, which can also be simply added as label to the transition. Also missing is the semantic representation of the trajectory constraints which is added in the form of the product of the task automaton and a DFA representing the trajectory constraint. Let $\mathcal{N}(\varphi)$ be an NFA representing the trajectory constraint $\varphi$ (De Giacomo et al., 2014). Then this NFA can be transformed into a DFA using the standard powerset construction (Rabin and Scott, 1959). Note that this results in a double exponential blowup, as the NFA can already be exponential in the size of $\varphi$ and the determinization of an NFA adds another exponential increase in size. However, note that for the $\mathsf{LTL_f}$ subset used in PDDL 3.0 constraints, the exponential blowup for the NFA cannot occur (J. A. Baier et al., 2009). Then the DFA $\mathcal{A}(\varphi) = \langle \Sigma, Q, \Delta, q_0, Q_a \rangle$ represents $\varphi$ where the input alphabet $\Sigma$ is the set of all states $\mathcal{S}(\mathcal{V}')$, where again $\mathcal{V}' = \mathcal{V}(\varphi)$ are the variables relevant to $\varphi$. The DFA $\mathcal{A}(\varphi)$ accepts a finite input trace $\mu_\pi$ over $\Sigma$ with, $\lambda$ being true exactly in the last state of $\mu_\pi$, iff the induced state trajectory $\mu_\pi$ of a plan $\pi$ fulfills $\varphi(\mu_\pi \models \varphi)$. For a hard trajectory constraint $\varphi$, a planning task $\Pi$ and the product automaton $\mathcal{A}^\times = \mathcal{A}(\Pi) \times \mathcal{A}(\varphi)$ the induced state trajectory $\mu_\pi$ from plan $\pi$ satisfies $\varphi$ iff $\mu_\pi$ is accepted by $\mathcal{A}^\times$. For soft trajectory constraints the same automaton construction can be used, however the product automaton must also accept traces for which the trajectory constraints are not fulfilled. Rather than not accepting traces where the soft trajectory constraints are violated, this should be represented in the plan costs, and thus in the plan quality. In the following, a method using such a product construction for compiling away soft trajectory constraints is presented.

**Tracking soft trajectory constraints.** Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a, \Phi \rangle$ be a planning task with soft trajectory constraints, and a numeric importance $\omega_\varphi$ for every $\varphi \in \Phi$, and the metric function $c(\mu_\pi)$. Ignoring transition costs and action names the semantics of $\Pi$ is captured by the product automaton $\mathcal{A}^\times = \mathcal{A}(\Pi) \times \prod_{\varphi \in \Phi} \mathcal{A}(\varphi)$ as described above. However, for compiling away soft trajectory constraints, the automaton is not explicitly constructed, but rather a new planning task $\Pi'$ such that $\mathcal{A}(\Pi')$ is isomorphic to $\mathcal{A}^\times$ is created. The planning task $\Pi'$ is constructed by adding a tracking variable $\tau_\varphi$ to $\Pi$ for every $\varphi \in \Phi$. This variable keeps track of the current state of $\mathcal{A}(\varphi)$. The actions of $\Pi'$ are those from $\Pi$ augmented by conditional effects taking care of the correct evolution of $\tau_\varphi$, thus encoding the soft trajectory constraints in to the actions.

Formally, let $\mathcal{A}(\varphi) = \langle \Sigma, Q, \Delta, q_0, Q_a \rangle$ be the DFA for a soft trajectory constraint $\varphi$, and $\tau$ a variable tracking the state of $\mathcal{A}(\varphi)$ with domain $\mathcal{D}_\tau = Q$. Then $\Pi' = \langle \mathcal{V}', \mathcal{A}', s'_0, s'_\star, c', \emptyset \rangle$ is the new planning task with $\mathcal{V}' = \mathcal{V} \cup$

$\{(\tau_\varphi | \varphi \in \Phi)\}$, $s'_0 = s_0 \cup \{(\tau_\varphi = q_0 | \varphi \in \Phi)\}$, and $s'_\star = s_\star$. The actions $\mathcal{A}' = \{a' | a \in \mathcal{A}\}$ where $a' = \langle pre', eff' \rangle$ are constructed as:

$$pre' = pre$$
$$eff' = eff \wedge \bigwedge_{\langle q,\sigma,q' \rangle \in \Delta} (((\tau_\varphi = q) \wedge \sigma) \rhd \tau_\varphi := q')$$

In words, the actions from $\Pi$ are augmented such that conditional effects tracking the state of $\tau_\varphi$ are added for every transition in $\mathcal{A}(\varphi)$. States in $\mathcal{A}(\Pi')$ are then isomorphic to pairs $(s, q)$ where $s$ is a state from $\mathcal{A}(\Pi)$ and $q$ is the state of $\mathcal{A}(\varphi)$ before reading $s$. Finally, an artificial last action $lastop$ is required, reading the last state of $\mathcal{A}(\Pi)$ and progressing $\mathcal{A}(\varphi)$ accordingly. This last action is the equivalent to reading the $\lambda$ symbol from $\mathsf{LTL_f}$.

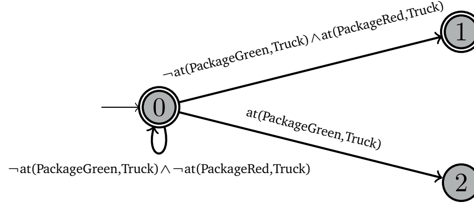



Figure 5.2: DFA of *sometime-before at(PackageGreen,Truck) (at(PackageRed,Truck)*.

**Example 19.** Recalling the logistics example (Example 4), a soft trajectory constraint might be to first pickup the green package before the red package (Figure 5.2):

*sometime-beforeat(PackageGreen,Truck)at(PackageRed,Truck)*

Let $\tau$ be the tracking variable for the soft trajectory constraint, then the compilation create a new action $a' = \langle pre', eff' \rangle$from an action $a = \langle pre, eff \rangle$ as follows:

$$\begin{aligned} pre' =& pre \\ eff' =& eff \wedge \\ & ((\tau = 0 \wedge \neg \text{at(PackageGreen,Truck)} \wedge \\ & \quad \text{at(PackageRed,Truck)}) \rhd \tau = 1) \wedge \\ & ((\tau = 0 \wedge \text{at(PackageGreen,Truck)}) \rhd \tau = 2) \end{aligned}$$

As can be seen, edges that lead to the same node as they originate can be ignored.

**Definition 49.** Two automata

$$\mathcal{A} = \langle \Sigma, Q, \Delta, q_0, Q_a \rangle \text{ and}$$
$$\mathcal{A}' = \langle \Sigma, Q', \Delta', q_0', Q_a' \rangle$$

over the same alphabet $\Sigma$ are isomorphic iff there exists a structure preserving bijection $\beta : Q \to Q'$ such that $\beta(q_0) = q_0'$, and $q \in Q_a$ iff $\beta(q) \in Q_a'$ for all $q \in Q$, and that $\langle q, \sigma, q' \rangle \in \Delta$ iff $\langle \beta(q), \sigma, \beta(q') \rangle \in \Delta'$ for all $q, q' \in Q$, $\sigma \in \Sigma$.

**Proposition 8.** Up to preservation of accepting states, $\mathcal{A}(\Pi')$ is isomorphic to $\mathcal{A}(\Pi) \times \mathcal{A}(\varphi)$.

*Proof.* Let $\mathcal{A}(\Pi)$ be the automaton representation of planning task $\Pi = \langle \Sigma, Q, \Delta, q_0, Q_a \rangle$. Let $\mathcal{A}(\varphi) = \langle \Sigma, Q^\varphi, \Delta^\varphi, q_0^\varphi, Q_a^\varphi \rangle$ be the soft trajectory constrain automaton representing the $\mathsf{LTL_f}$ formula. Finally let $\mathcal{A}(\Pi') = \langle \Sigma, Q', \Delta', q_0', Q_a' \rangle$ be the product automaton all with the same alphabet $\Sigma = \mathcal{S}(\mathcal{V}(\varphi))$. Then $\mathcal{A}^\times = \mathcal{A}(\Pi) \times \mathcal{A}(\varphi) = \langle \Sigma, Q^\times, \Delta^\times, q_0^\times, Q_a^\times \rangle$, with $Q^\times = Q \times Q^\varphi$, $q_0^\times = (q_0, q_0^\varphi)$, $Q_a^\times = Q_a \times Q_a^\varphi$, and a transition $\langle (q, q^\varphi), \sigma, (q', q^{\varphi\prime}) \rangle \in \Delta^\times$ iff $(q, \sigma, q') \in \Delta$ and $(q^\varphi, \sigma, q^{\varphi\prime}) \in \Delta^\varphi$. The bijection $\beta : Q^\times \to Q'$ is then given by $\beta((s, q)) = s \cup \{\tau_\varphi = q\}$. $\beta$ by definition preserves the initial state as $\beta((s_0, q_0)) = s_0 \cup \{\tau_\varphi = q_0\}$. The accepting states are not preserved in $\Pi'$ as the acceptance or violation of $\varphi$ should only be reflected in the cost of the plan. Therefore, for goal states, it holds that $q^\times = (q, q^\varphi) \in Q_a^\times$ iff $q \in Q_a$.

There is a transition $((q, q^\varphi), \sigma, (q', q^{\varphi\prime})) \in \Delta^\times$ iff $(q, \sigma, q') \in \Delta$ and $(q^\varphi, \sigma, q^{\varphi\prime}) \in \Delta^\varphi$. Since $(q, \sigma, q') \in \Delta$ there must be an action $a$ applicable in $q$ and $q' = q[a]$. With the construction of $\Pi'$ from $\Pi$ and the definition of the changeset (Definition 23), this implies that the modified action $a'$ is applicable in $\beta(q, q^\varphi)$ and because $(q^\varphi, \sigma, q^{\varphi\prime}) \in \Delta^\varphi$, $\beta(q', q^{\varphi\prime}) = \beta(q, q^\varphi)[a']$ this means that $(\beta(q, q^\varphi), \sigma, \beta(q', q^{\varphi\prime})) \in \Delta'$.

For the other direction the proof follows the same structure: There is a transition $(\beta(q, q^\varphi), \sigma, \beta(q', q^{\varphi\prime})) \in \Delta'$ iff $(q, \sigma, q') \in \Delta$ and $(q^\varphi, \sigma, q^{\varphi\prime}) \in \Delta^\varphi$. Since $(\beta(q, q^\varphi), \sigma, \beta(q', q^{\varphi\prime})) \in \Delta'$ there is an action $a' \in \mathcal{A}'$ such that $\beta(q', q^{\varphi\prime}) = \beta(q, q^\varphi)[a']$. Removing the conditional effects introduced in the construction of $\Pi'$ the original action from $\Pi$ is retrieved, thus there is a transition $(q, \sigma, q') \in \Delta$. Similar, as the construction only added conditional effects tracking the state of the automaton $\mathcal{A}(\varphi)$ there is also the transition $(q^\varphi, \sigma, q^{\varphi\prime}) \in \Delta^\varphi$. □

Up until now the main focus was on tracking the soft trajectory constraint within the planning task using conditional effects. One part still missing is the adaptation of the task such that the objective function introduced above is properly represented in the plans total cost. For this, two approaches are

introduced. The first, called *Goal action penalty compilation*, a generalization of Keyder and H. Geffner (2009) to soft trajectory constraints, adding a penalize action to the end of the plan. The second, called *General action penalty compilation*, then takes greater advantage of state-dependent action costs to guide the search towards fulfilling the constraints.

**Goal action penalty compilation.** To be able to evaluate the quality of a plan in regards to the soft trajectory constraints, penalties are added for every constraint not fulfilled by the plan. This is achieved by adding a new propositional variable penalized to $\Pi'$ that is initially false and required to be true in the goal condition $s'_\star = s_\star \cup \{\mathsf{penalized}\}$. Additionally, a new action $penalize = \langle s_\star \wedge \neg\mathsf{penalized}, \mathsf{penalized}\rangle$ is introduced requiring the original goal to be satisfied, and the penalize action not yet executed. The cost function of this $penalize$ action then determines the penalty value of the plan by adding all soft trajectory constraint values $\omega_\varphi$ as its action cost for which the plan $\pi \not\models \varphi$. This is achieved by simply testing if the tracking variables $\tau$ encode accepting or non accepting DFA states in the current planning state. Formally: $c_{penalize} = \sum_{\varphi\in\Phi}[\tau_\varphi \notin Q^\varphi_a]\omega_\varphi$ where $[\tau_\varphi \notin Q^\varphi_a] = 1$ if $\tau_\varphi = q$ and $q \notin Q^\varphi_a$ for some $q \in Q^\varphi$, and $0$ otherwise. $[\tau_\varphi \notin Q^\varphi_a]$ can be easily evaluated by rewriting it as $\sum_{q\in Q^\varphi\setminus Q^\varphi_a}[\tau_\varphi = q]$, where $[\tau_\varphi = q]$ is $1$ if $s(\tau_\varphi) = q$, and $0$ otherwise.

**Example 20.** Recalling above Example 19, let the value of the soft trajectory constraint be $\omega = 5$ and the original goal be to bring both packages to Freiburg, then the penalize action is as follows:

$$\begin{aligned} pre =& \text{(at-location PackageGreen Freiburg)} \\ & \wedge \text{(at-location PackageRed Freiburg)} \\ eff =& \mathsf{penalized} \end{aligned}$$

With cost function: $c = [\tau = 2] * 5$ stating that if the tracking variable $\tau$ has the value $2$ representing a non-accepting state of the DFA from Figure 5.2, the $penalize$ operator adds the soft trajectory penalty value of $5$. Note that if $\tau = \{1, 0\}$ the action has cost $0$ adding no additional penalties to the plans total cost.

**Proposition 9.** Let $\Pi'$ be the compiled task from $\Pi$. Then an optimal plan for $\Pi'$ is also an optimal plan for $\Pi$ disregarding the $penalize$ action.

*Proof.* Proposition 8 shows that the compilation is sound and complete. The only thing missing for optimality preserving is the objective function equality. The objective function can be seen as the sum of two parts $c(\pi) + penalty(\pi)$, where $c(\pi)$ is the sum of action costs and $penalty(\pi)$ the penalty for not achieved soft trajectory constraints. Up until the $penalize$ action the

objective function sums up all action costs identical to the original task, as the action costs are not altered in the compilation, $c(a) = \sum_{a\in\pi} c(a)$. The *penalize* action then adds a penalty for each soft trajectory constraint not fulfilled by $\pi$, *penalty*$(\pi) = \sum_{\varphi\in\Phi}[\tau_\varphi \notin Q_a^\varphi]\omega_\varphi$. This results in the same objective function for $\Pi'$ and $\Pi$ (Wright et al., 2018b). □

As an alternative representation of the cost function, a new propositional variable is_violated can be introduced, which is true if $\tau_\varphi$ represents a non-accepting state of the DFA. The cost function can then be rewritten as

$$\sum_{\varphi\in\Phi}[\mathsf{is_violated}_\varphi]w_\varphi \ .$$

This is briefly introduced to show the relation with the compilation of Keyder and H. Geffner (2009). As shown earlier, state-dependent action costs and conditional effects can be compiled away. If this EVMDD compilation on the *penalize* action is analyzed, it can be shown, that the resulting auxiliary actions are identical to the *collect*, *forgo* and *end* actions of the Keyder and H. Geffner (2009) compilation. This shows, that the *goal action compilation* extends the soft goal compilation of Keyder and H. Geffner (2009) to soft trajectory constraints.

**Proposition 10.** The EVMDD-based action compilation of the *penalize* action, generalizes the Keyder and H. Geffner (2009) compilation to soft state trajectory constraints.

*Proof.* For every soft trajectory constraint $\varphi \in \Phi$ a new variable $\mathsf{is_violated}_\varphi$ is introduced, which is true iff the corresponding tracking variable $\tau$ represents a non-accepting state of the DFA. The *penalize* then has the cost function $\sum_{\varphi\in\Phi}[\mathsf{is_violated}_\varphi]w_\varphi$. The EVMDD representation of this cost function is then depicted in Figure 5.3. The EVMDD compilation of *penalize* then creates a new action for every edge of the EVMDD and adds some bookkeeping machinery for ensuring the right action sequence is executed (see EVMDD compilation Section 3.2.1). These new actions then correspond exactly to the *collect*, *forgo* and *end* actions of the Keyder and H. Geffner (2009) compilation as annotated in Figure 5.3. (Wright et al., 2018b) □

Independently of which of the two cost functions is used, it can be seen that the *penalize* guides the search towards fulfilling the soft trajectory constraints, as states that accept the trajectory constraint will have lower heuristic values(*h-value*). This is due to the fact, that the *penalize* operation has lower costs of application if the constraints are satisfied. However, as the *penalize* is always the last action of the plan, the search algorithm is only informed on the acceptance of a trajectory constraint when applying the last action, even though the constraint might already be fulfilled earlier on in

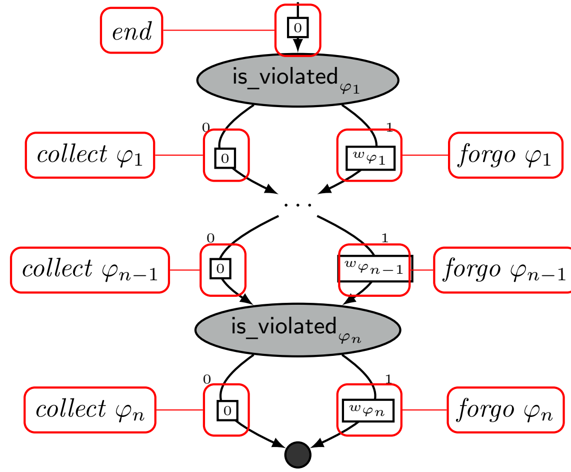



Figure 5.3: EVMDD compilation of the *penalize* action and the Keyder and H. Geffner (2009) *collect*,*forgo* and *end* actions annotated in red

the plan without the possibility of it becoming false again. This results in the heuristic value being unnecessarily high even when already close to the solution. Therefore, an alternative approach adding penalties, or rewards (negative penalties) as soon as they occur is desirable, resulting in a more informed heuristic value, and a more informed search.

**General action penalty compilation.** The previous approach showed how the search can be guided towards fulfilling soft trajectory constraints. However, the achievement of these is only represented in the final *penalize* action. Thus, up to the final action, the heuristic is relatively uninformed. A more desirable, approach would be to apply a penalty as soon as a soft trajectory constraint is violated, thus guiding the search more effective. In the following, a method is introduced, that adds rewards or penalties to actions whenever the DFA of a soft trajectory constraint enters or leaves an accepting state. This not only results in an informed *h-value* but also a more informative *g-value*, as the penalties and rewards are represented in it.

Let $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, c_a \rangle$ be a planning task with soft trajectory constraints. Furthermore, let $\Pi' = \langle \mathcal{V}', \mathcal{A}', s'_0, c'_a \rangle$ be a compiled task that only tracks the states of the soft trajectory constraints $\varphi \in \Phi$ as constructed above, $\mathcal{A}_\varphi = \langle \Sigma, Q, \Delta, q_0, Q_a \rangle$ be the DFA for soft trajectory constraint $\varphi \in \Phi$, and $\tau_\varphi$ be the tracking variable in $\mathcal{V}$ tracking the state of $\mathcal{A}_\varphi$. Then, information about the acceptance of a soft trajectory constraint $\varphi$ in a state $s$ is already present, as $\tau_\varphi$ represents an accepting or non-accepting state in $\mathcal{A}_\varphi$.

Whenever an action $a \in \mathcal{A}'$ changes the value of $\tau_\varphi$ transitioning from a state

$q$ to a state $q'$ in $\mathcal{A}(\varphi)$, a reward or a penalty can be added to the cost of $a$. If $q$ is an accepting state and $q'$ is a non-accepting state, a penalty is added. On the other hand, if $q$ is a non-accepting state and $q$ is an accepting state, a reward is added. The partial cost of transitioning from $s$ to $s'$ in the planning task with trajectory constraint $\varphi \in \Phi$ is then $\omega(s(\tau_\varphi), s'(\tau_\varphi))$, where $\omega : Q \times Q \to \mathbb{Z}$ is a function mapping state transitions to reward or penalty values. In the case of a transition from an accepting to a non-accepting state $\omega(s(\tau_\varphi), s'(\tau_\varphi))$ is set to a positive penalty value. In the case of a transition from a non-accepting to an accepting state $\omega(s(\tau_\varphi), s'(\tau_\varphi))$ is set to reward, or negative value. For the concrete value (positive or negative)of $\omega(s(\tau_\varphi), s'(\tau_\varphi))$ the original soft trajectory constraints weight $\omega$ is used.

The new cost function $c'_a$ of $a = \langle pre, eff \rangle$ is then

$$p_\varphi = \sum_{\langle q,\delta,q' \rangle \in \mathcal{A}_\varphi} [\tau_\varphi = q \wedge \delta] * \omega(q, q')$$
$$c'_a = c_a + \sum_{\varphi \in \Phi} p_\varphi$$

Here transitions $\langle q, \delta, q' \rangle \in \mathcal{A}_\varphi$ with $q = q'$ can be ignored.

Note that to ensure the correct application of penalties and rewards, the initial state of the automata needs to be taken in to account. If the initial state of the automaton is non-accepting, transitioning to an accepting state will apply a negative action cost(reward) without applying the penalty for entering a non-accepting state first, resulting in the total reward/penalty of this soft trajectory constraint being $\leq 0$. The problem here is, that entering the initial state is not taken in to account. Therefore a new initial action $init$ is introduced to $\mathcal{A}'$ applying penalties $\omega$ for each $\varphi \in \Phi$ with its DFA $\mathcal{A}_\varphi = \langle \Sigma, Q, \Delta, q_0, Q_a \rangle$ where $s_0(\tau_\varphi) \neq q_0$. Formally the action costs of this initial action is

$$c = \sum_{\varphi \in \Phi} (\omega | s_0(\tau_\varphi) \neq q_0)$$

**Example 21 (General action cost function).** Recalling the above logistics example (Example 19) with the action $a = \langle pre, eff \rangle$ and its compilation:

$$\begin{aligned} pre' =& pre \\ eff' =& eff \wedge \\ & ((\tau = 0 \wedge \neg(\text{at(PackageGreen,Truck)} \wedge \\ & \quad (\text{at(PackageRed,Truck)}) \rhd \tau = 1) \wedge \\ & ((\tau = 0 \wedge (\text{at(PackageGreen,Truck)}) \rhd \tau = 2) \end{aligned}$$

A soft trajectory constraint $\varphi$ = *sometime-before (at(PackageGreen,Truck) (at(PackageRed,Truck)* (Figure 5.2) with value $\omega = 5$, then the new cost function is:

$$c_a^{\prime} = c_a + [\tau == 0 \wedge \text{(at(PackageGreen,Truck)}] * 5$$

One major issue now is however, that negative action costs can arise due to the fact of entering accepting states. This is that case when the original action cost $c_a$ is less than the value of the reward function $\omega(s(\tau), s'(\tau))$ the cost of applying the action in state $s$ is then $c - \omega(q, q') \leq 0$.

As $\omega(s(\tau), s'(\tau))$ can not be applied twice in a row without negating it in between (leaving or entering an accepting state) it is guaranteed that no negative action cycles can occur during the search, resulting in non negative total costs $c(\mu_\pi) \geq 0$. Having such negative action cost cycles would result in non-termination of the search, as some states can be reached by ever cheaper paths.

Note that the fact that $\omega(s(\tau), s'(\tau))$ can be applied only once before subtracting it again (and vice versa) also ensures that minimizing over the total costs $c(\mu_\pi)$ in this compiled task amounts to the same as minimizing $c(\mu_\pi)$ in the original task.

The absence of negative action cost cycles enables this approach to be applied in any existing planning system, supporting negative action costs. Currently, Fast Downward (Helmert, 2006) with blind heuristic supports such negative action costs. However, it is desirable to be able to use more informed heuristics, or planners that do not support negative action costs. Therefore removing the negative action costs is desirable. To remove negative action costs, a more flexible way of specifying the $\omega(s(\tau), s'(\tau))$ value is introduced in the form of a transition cost table (Table 5.1). The above described transition costs are hereby captured by Table 5.1a, where state transitions from accepting to non accepting are set to the original $\omega$, and the transition from a non-accepting state to an accepting state is set to $-\omega$. All other transitions are set to $0$.

Table 5.1b shows the cost function shifted by $\omega$ (adding it to every entry of the table) resulting in no negative action costs. Leaving an accepting state is now penalized harder, as it is now $2\omega$. Remaining in a non-accepting state is now also penalized by $\omega$. This turns out to be beneficial, as this incentivises early fulfillment of soft trajectory constraints, as remaining in non-accepting states is penalized. However, remaining in an accepting state is now also penalized providing the unwanted effect of penalizing the remaining in an accepting state. This negative behavior is fixed by Table 5.1c, where this malicious entry is simply replaced by $0$.

Table 5.1: State Transition Costs

(a) Metric Preserving Costs

| From \ To | Accepting | ¬ Accepting |
|---|---|---|
| Accepting | $0$ | $w_\varphi$ |
| ¬ Accepting | $-w_\varphi$ | $0$ |

(b) Positively Shifted Costs

| From \ To | Accepting | ¬ Accepting |
|---|---|---|
| Accepting | $w_\varphi$ | $2w_\varphi$ |
| ¬ Accepting | $0$ | $w_\varphi$ |

(c) Adapted Positively Shifted Costs

| From \ To | Accepting | ¬ Accepting |
|---|---|---|
| Accepting | $0$ | $2w_\varphi$ |
| ¬ Accepting | $0$ | $w_\varphi$ |

The resulting cost function (regardless of which concrete transition cost table is used) is informative regarding both the *h-value* and the *g-value*. However, choosing cost transition tables 5.1b or 5.1c will result in higher total costs $c(\mu_\pi)$, as the penalties are added multiple times instead of only once for every time an accepting state is left.

**Proposition 11.** Let $\Pi'$ be the compiled task from $\Pi$ with metric preserving costs (Table 5.1a). Then an optimal plan for $\Pi'$ is also an optimal plan for $\Pi$ (without $penalize$ and $init$ action).

*Proof.* Proposition 8 shows that the compilation is sound and complete. The objective function of the original task is *penalty*$(\pi)$ + *cost*$(\pi)$, where *cost*$(\pi)$ is the sum of all original action costs and *penalty*$(\pi)$ is the sum of all penalties of not achieved soft trajectory constraints. Using metric preserving costs (Table 5.1a) it follows that at each step of the plan the current $c(\mu_\pi)$ is the sum of all applied action costs plus the sum of all penalties for leaving an accepting state minus the rewards for entering an accepting state of the corresponding soft trajectory constraint automata $\mathcal{A}$. Thus for any state the current total cost is the sum of action costs applied up until the state plus the penalties for any currently not achieved soft trajectory constraint. This obviously holds in the first state after applying the $init$ action as this simply adds the penalties for any soft trajectory constraint not in an accepting state in the initial state of the plan. Any subsequent action then adds a penalty for each transition from an accepting state to a non-accepting state, in the underlying $\mathcal{A}$(s), or subtracts the same value if the transition is from

a non-accepting state to an accepting state. Thus after applying an action, the current total cost $c(\mu_\pi)$ is updated by adding the original action cost and applying the penalty or reward, reflecting the current state of the soft trajectory constraints. In the goal state, this is equal to $\mathit{penalty}(\pi) + \mathit{cost}(\pi)$, the original metric of the task $\Pi$. $\square$

Equally, from Proposition 11 it is clear, that the other state transition cost tables (Table 5.1b and Table 5.1c) do not preserve this property, as at every step in which the tracking variable $\tau$ represents a non-accepting state, the penalty is applied. Thus, optimality is no longer preserved. This is especially critical in planning tasks without any hard goals, as doing nothing results in collecting every penalty only once (in the $\mathit{init}$ action). For any plan with length longer than $1$ and for which the soft trajectory constraint can not be achieved in one action this results in the optimal plan being the empty plan (disregarding $\mathit{init}$ and $\mathit{penalize}$ actions). Whereas, the expected plan would try and minimize the penalties.

The above approach using metric preserving costs (Table 5.1a) can also be seen as a form of potential-based reward shaping (Ng et al., 1999) from Markov decision processes, where $R(s, a, s')$ is the reward gathered from going from state $s$ to $s'$ with action $a$. In this case, negative action costs represent rewards and positive costs represent penalties. The shaped reward function is then $R'(s, a, s') = R(s, a, s') + F(s, a, s')$ where $F$ is the shaping function defined as $F(s, a, s') = \gamma \cdot \theta(s') - \theta(s)$, where $\theta$ is a potential function defined over states. In this case, $\theta(s) = \sum_{\varphi \in \Phi} [\mathsf{is_violated}_\varphi] w_\varphi$, with the discount factor $\gamma = 1$.

## 5.2 Evaluation

The compilation was implemented using the translate scripts from the Fast-Downward planning system (Helmert, 2006), producing an intermediate file that is then used as input to the Fast Downward planner and the Symple planning system (Speck et al., 2018b; Speck et al., 2018a). The experiments where then executed on a subset of the benchmark problems from the fifth international planning competition (IPC-5 (Dimopoulos et al., 2006)) with the added *Rovers* domain from the IPC-3. The original competitions where for satisficing planning only, thus the benchmarks where not designed for optimal planners. Instead of developing own benchmarks, the original settings where sampled, such that for every planning task 7 new instances where generated. For this, only a subset of the soft trajectory constraints where considered (0%, 5%, 10%, 20%, 30%, 50%, 100% respectively), whereas the original hard goals where not altered.

**Optimal solutions**

In Table 5.2 the coverage for optimal planning with the goal action compilation using the blind heuristic $h^{\text{blind}}$, the canonical pattern database heuristic $h^{\text{cpdb}}$ (Haslum et al., 2007), the maximum heuristic $h^{\text{max}}$ (Bonet and H. Geffner, 2001), and the merge and shrink heuristic $h^{\text{ms}}$ (Sievers et al., 2014) using Fast-Downward is presented along side the coverage for the symbolic planner Symple. As can be seen, for the Fast-Downward planner, the $h^{\text{max}}$ heuristic performs slightly better than the other heuristics, with exception of open stacks where the merge and shrink heuristic $h^{\text{ms}}$ and the pattern database heuristic $h^{\text{cpdb}}$ outperform the maximal heuristic. A reason for the overall bad performance when using the $h^{\text{ms}}$ heuristic is due to the conditional effects not being factored, leading to bad heuristic estimates (Helmert et al., 2014). Table 5.2 also shows that for the goal action compilation the symbolic planner *Symple* outperforms all other approaches, emphasizing the benefit of using compilations over native implementations. Nevertheless, compilations also introduce a certain overhead. In the case of the goal action compilation, this comes in the form of the conditional effects used for tracking the status of every soft trajectory constraint. In some cases this consist of tracking over 1600 soft trajectory constraints. These tasks however, where not solved, as even building all the automata, and compiling the tracking effects is infeasible. In contrast the general action

| | Fast Downward | | | | Symple |
|---|---|---|---|---|---|
| | $h^{\text{blind}}$ | $h^{\text{cpdb}}$ | $h^{\text{max}}$ | $h^{\text{ms}}$ | |
| open stacks | 5.00% | 16.43% | 7.14% | 13.93% | 32.14% |
| pathways | 13.81% | 13.33% | 19.52% | 0.00% | 40.95% |
| rovers | 19.29% | 18.57% | 22.86% | 7.86% | 34.29% |
| storage | 12.36% | 8.88% | 14.09% | 0.00% | 21.81% |
| trucks | 22.14% | 0.00% | 22.14% | 0.00% | 0.00% |

Table 5.2: Goal action compilation results for optimal planning

compilation with metric preserving penalty function (Table 5.1a) was executed only on the blind $h^{\text{blind}}$ heuristic, as it is the only admissible heuristic supporting negative action costs. This is due to the fact that it simply assigns 1 to non goal states and 0 to goal states. Therefore, the heuristic is relatively uninformed only guided by the *g-value* of the nodes. This is reflected in the coverage (Table 5.3), where the only improvement was made in the *pathways* domain. On the other hand, it does provide promising results, when comparing the number of expanded nodes (Figure 5.4) which are slightly lower in the general action compilation. Combined with heuristics, supporting negative action costs, this promises to provide better results.

As the shifted and adapted penalty functions (Table 5.1b and Table 5.1c)

| | Fast Downward |
|---|---|
| | $h^{\text{blind}}$ |
| open stacks | 2.86% |
| pathways | 48.10% |
| rovers | 9.29% |
| storage | 10.81% |
| trucks | 13.21% |

Table 5.3: General action compilation with metric preserving transition costs results for optimal planning

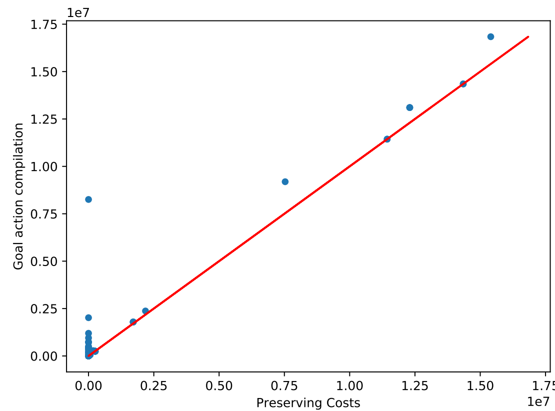



Figure 5.4: Node expansions for goal action compilation and general action compilation with preserving costs

are not metric preserving, analyzing them for optimality must be taken with caution. Optimality, in this case is only optimal in regards to the new task, *NOT* the original task. However, the results in Tables 5.4 and 5.5 illustrate, that with improved heuristic friendliness, the coverage can be increased. This emphasizes the fact that heuristics supporting negative action costs would also improve the results of the metric preserving penalty function.

| | Fast Downward | | | | Symple |
|---|---|---|---|---|---|
| | $h^{\text{blind}}$ | $h^{\text{cpdb}}$ | $h^{\text{max}}$ | $h^{\text{ms}}$ | |
| open stacks | 2.86% | 14.29% | 2.86% | 13.96% | 25.56% |
| pathways | 48.10% | 45.71% | 48.10% | 20.00% | 71.90% |
| rovers | 6.43% | 6.43% | 6.43% | 6.43% | 33.09% |
| storage | 10.81% | 9.07% | 10.81% | 7.72% | 59.26% |
| trucks | 12.50% | 0.00% | 11.43% | 0.00% | 0.00% |

Table 5.4: General action compilation with positively shifted transition costs results for optimal planning

| | Fast Downward | | | | Symple |
|---|---|---|---|---|---|
| | $h^{\text{blind}}$ | $h^{\text{cpdb}}$ | $h^{\max}$ | $h^{\text{ms}}$ | |
| open stacks | 2.86% | 14.29% | 2.86% | 13.96% | 25.28% |
| pathways | 48.10% | 45.71% | 48.10% | 20.00% | 74.29% |
| rovers | 10.00% | 10.00% | 10.00% | 7.86% | 40.71% |
| storage | 10.81% | 9.07% | 10.81% | 7.53% | 56.09% |
| trucks | 13.21% | 0.00% | 12.86% | 0.00% | 0.00% |

Table 5.5: General action compilation with adapted transition costs results for optimal planning

**Satisficing solutions**

When running the planning tasks on a satisficing setting (finding nonoptimal plans using the $h^{\text{ff}}$ heuristic), obviously the coverage increases. One drawback, here is that only the goal action compilation, and the general action compilation with the shifted cost and adapted costs are supported, as negative action costs are not supported by non of the non-admissible heuristics implemented in the Fast Downward planning system. Additionally, the Symple planning system does not support negative action costs nor satisfying planning. Therefore, Symple was not used for this setting. Even though the shifted and adapted penalty functions are not metric preserving, the quality of the resulting plans can be compared, by ignoring the plan length or action costs, focusing purely on the soft constraint penalties. Note that for the pathways domain no hard goals where specified, thus the goal was to maximize the achieved soft trajectory constraints. As the general action compilation with shifted or adapted penalty adds penalties for remaining in non-accepting states of the constraints, the empty plan is sometimes *optimal* in the sense that doing nothing becomes the viable option. This obviously does not reflect the original intent of the task, and is a fragment of the non metric preserving compilations. In Figure 5.5 the results for the three compilations are presented, showing that in the cases where a plan was found, the plan quality is very high, as the real penalty (collected for not achieving the constraints) is way lower than the total possible penalty (Maximal penalty if no constraint where fulfilled).

Analyzing the results for the rovers domain (Figure 5.6), it becomes apparent, the the different compilations sometimes result in a high variance, concerning the coverage.Although all compilations provide results with relative low penalties, the number of solved tasks is highest for the goal action compilation (50 in total), whereas the general action compilations where only able to solve 26 (shifted transition costs) and 9 (adapted transition costs) instances. This is the result of the necessity of compiling away conditional effects together with state-dependent action costs (Section 4.2), adding a

large overhead to the planning task (First adding conditional effects and state dependent action costs, and then compiling them away again). A massive improvement here would be to natively implement the changeset semantics (Definition 42) in to the planning system, removing the necessity of compiling away the state dependent action costs and conditional effects.

For the storage domain (Figure 5.7), it can be seen, that the goal action compilation again produced the best results, not only for coverage, but also in quality, with high penalties in both general action compilations. Similar results are observed for the trucks domain (Figure 5.8).

| | Fast Downward | | | |
|---|---|---|---|---|
| | preserving | shifted | adapted | goal action |
| open stacks | 0 % | 14.29% | 14.29% | 86.79% |
| pathways | 0 % | 48.10% | 48.10% | 63.33% |
| rovers | 0 % | 19.29% | 22.14% | 51.43% |
| storage | 0 % | 10.81% | 10.81% | 28.19% |
| trucks | 0 % | 39.29% | 45.00% | 86.07% |

Table 5.6: Coverage of Fast Downward with satisficing planning

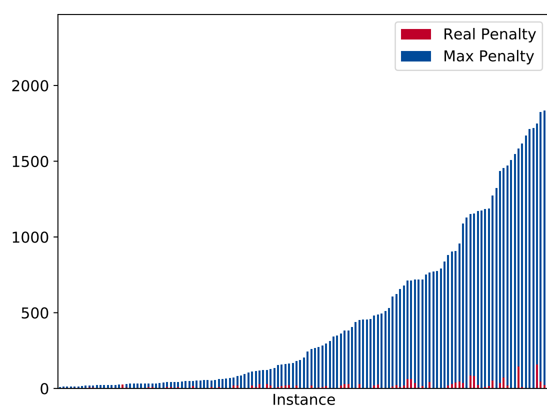
| | Symple | | | |
|---|---|---|---|---|
| | preserving | shifted | adapted | goal action |
| open stacks | 0 % | 24.64% | 24.29% | 0 % |
| pathways | 0 % | 71.90% | 74.29% | 0 % |
| rovers | 0 % | 32.86% | 40.71% | 0 % |
| storage | 0 % | 30.89% | 29.34% | 0 % |
| trucks | 0 % | 0 % | 0 % | 0 % |

Table 5.7: Coverage of Symple with satisficing planning

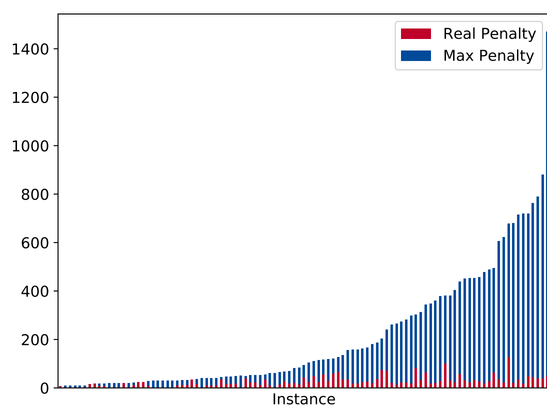
Comparing the results for the goal action compilation to the results of the original IPC-5 competition, Table 5.8 shows the times each configuration produced a plan with the lowest penalty. The two planners considered for comparison where HPlan-P (J. Baier et al., 2006) and MIPS-XXL (Edelkamp et al., 2006). These two planners where awarded with the *Distinguished Performance in Satisficing Planning (SimplePreferences Domains)* and *Distinguished Performance in Satisficing Planning (QualitativePreferences Domains)* awards, and therefore provide a reasonable basis for comparison. Simple preferences hereby refer to soft goal constraints (*at-end* constraints), and qualitative preferences are any other kind of soft trajectory constraint. Even though, modern hardware provides much better performance, the aim was to find plans with lower penalty, disregarding planning time. As can be seen in Table 5.8 the goal action compilation provides slightly better results compared to the others.

| | Goal Action C. | HPlan-P | MIPS-XXL |
|---|---|---|---|
| wins | 45 | 38 | 35 |

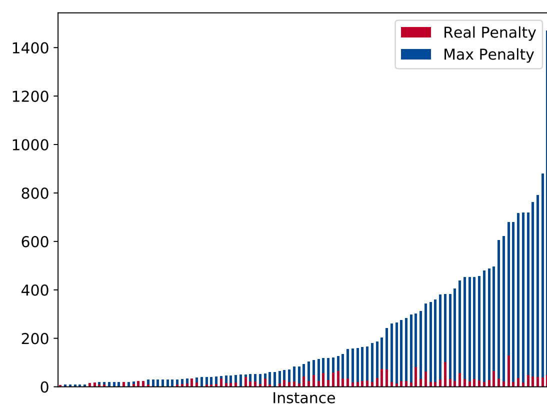
Table 5.8: Comparison to original IPC-5




(a) Goal action compilation

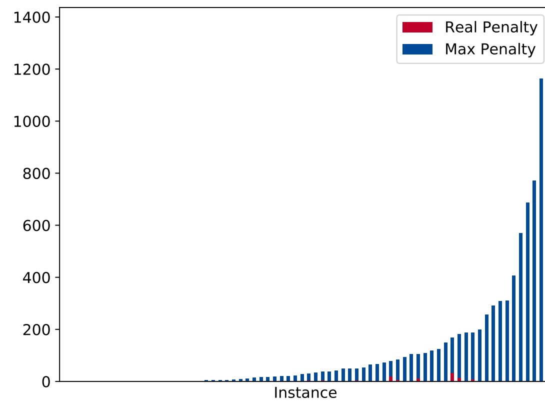



(b) General action compilation with shifted transition costs

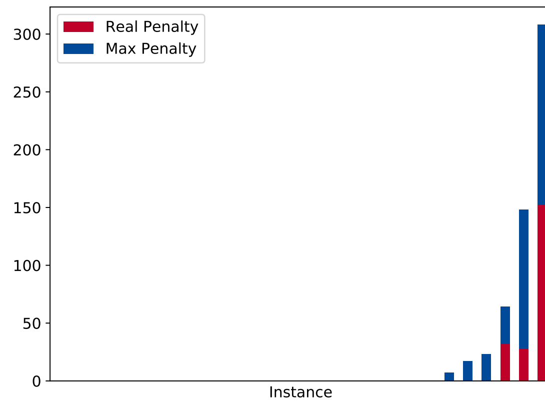



(c) General action compilation with adapted transition costs

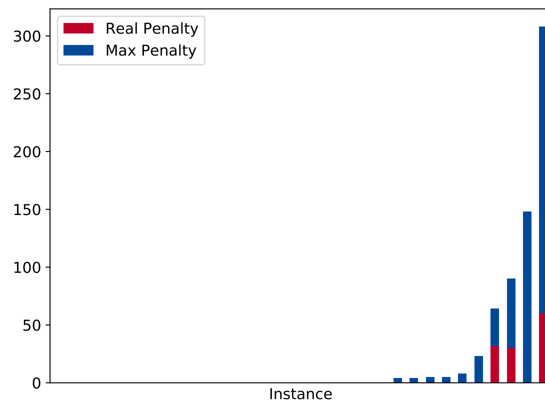
Figure 5.5: Results for the pathways domain with satisficing planing




(a) Goal action compilation

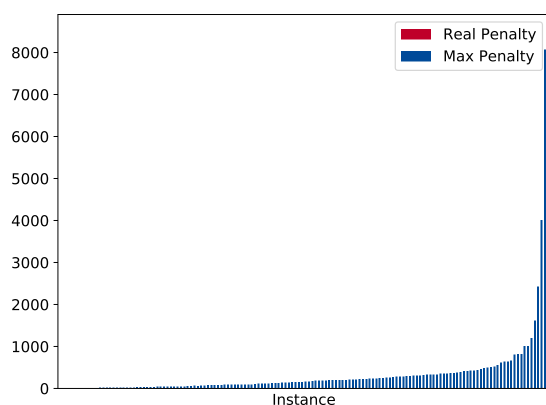



(b) General action compilation with shifted transition costs

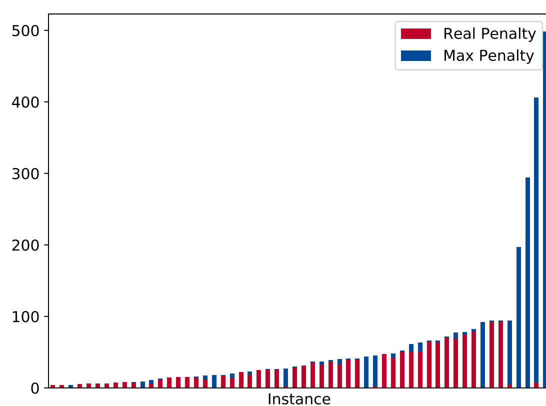



(c) General action compilation with adapted transition costs

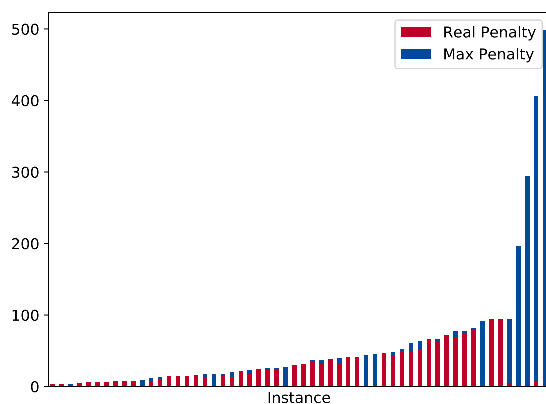
Figure 5.6: Results for the rovers domain with satisficing planing




(a) Goal action compilation

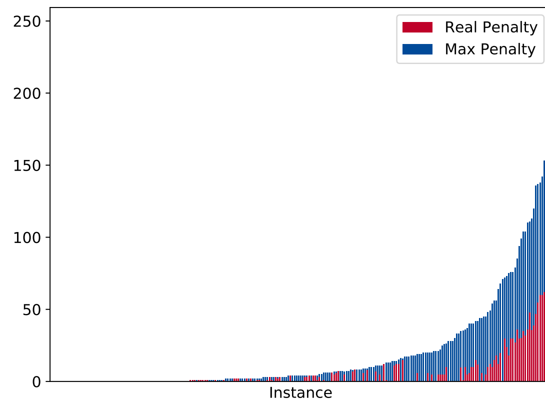



(b) General action compilation with shifted transition costs

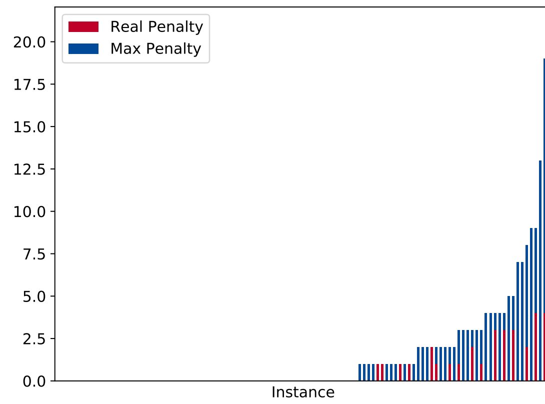



(c) General action compilation with adapted transition costs

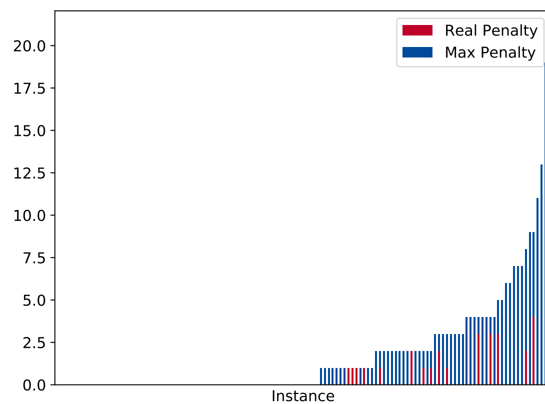
Figure 5.7: Results for the storage domain with satisficing planing




(a) Goal action compilation




(b) General action compilation with shifted transition costs




(c) General action compilation with adapted transition costs

Figure 5.8: Results for the trucks domain with satisficing planing

# Chapter 6

# Soft trajectory constraints in FOND planning

As with classical planning, one can also formulate trajectory constraints towards the resulting policy. Here, the same PDDL 3.0 constraints are used, but as policies are not a linear sequence of actions the semantics of the trajectory constraints is slightly modified. Two possible interpretations are that all execution paths from the initial state $s_0$ to the goal state $s_\star$ following the policy $\pi$ must fulfill the trajectory constraint $\varphi$ (*universal* interpretation $\mathcal{A}.\varphi$), or that it is sufficient to have at least one execution path fulfilling the constraint (*existential* interpretation $\mathcal{E}.\varphi$) (Pistore and Vardi, 2007). In some cases these $\mathcal{A}.\varphi$ and $\mathcal{E}.\varphi$ however, are to strict and more flexible interpretations are required, such as the two length combination $\mathcal{AE}.\varphi$ and $\mathcal{EA}.\varphi$. These state that for every prefix there exists a suffix fulfilling $\varphi$ or that there exists a prefix such that every suffix fulfills $\varphi$ respectively (Pistore and Vardi, 2007).

In FOND planning, one could use the quality of a policy determined by the number state action pairs. However, when dealing with soft trajectory constraints, the smallest policy is not always the best policy (in regards to the constraints) as they may ignore constraints all together. It is therefore often beneficial to take larger policies into account that fulfill the constraints. Therefore, an alternative quality measure is used, based on the Bellman optimality equation (Bellman and Dreyfus, 2015; Bertsekas et al., 2005) used in MDP planning, which basically calculates the expected total cost of executing an action in a given state.

**Definition 50 (Policy trace).** A policy trace

$$\mu = (\langle s_0, a_0 \rangle), \ldots, \langle s_{n-1}, a_{n-1} \rangle, \langle s_n, \emptyset \rangle)$$

for policy $\pi$ in planning task $\Pi$ is a sequence of state action pairs with:

- $s_0$ is the initial state of the planning task $\Pi$
- $s_n$ accepts the goal condition ($s_n \models s_\star$)
- $s_{i+1} \in s_i[a_i]$ for all $i = 0, \ldots, n$

Let $M_\pi$ be the set of all possible traces $\mu$ consistent with $\pi$,

**Definition 51 (Policy quality in FOND planning).** Given a policy $\pi$, a goal condition $s_\star$ and a state $s$. The quality of $s$ in $\pi$ is then calculated by the equations:

$$V^\pi(s) = \begin{cases} 0 & \text{if} s \models s_\star \\ Q^\pi(s, \pi(s)) & \text{otherwise} \end{cases} \tag{6.1}$$

$$Q^\pi(s, a) = c_a + \sum_{s' \in \mathcal{S}} \mathbb{P}[s'|s, a] \cdot V^\pi(s') \tag{6.2}$$

where the probability of reaching state $s'$ after applying action $a$ in state $s$, $\mathbb{P}[s'|s[a]]$ is assumed to be uniformly distributed over all possible outcomes of $a$. The quality of a policy $\pi$ is then $Q^\pi = V^\pi(s_0)$, and an optimal policy is then a policy with the lowest expected cost $\pi^\star = \mathit{argmin}_{\pi \in \Pi} V^\pi(s_0)$.

This definition is equivalent to the expected total cost $c(\mu)$ over all possible traces $\mu \in M_\pi$ from the initial state $s_0$ to the goal state $s_\star$.

$$Q^\pi = \mathbb{E}[c(\mu)|\mu \in \pi] \tag{6.3}$$

$$c(\mu) = \sum_{i \in \mu} c_{a_i} \tag{6.4}$$

Equation 6.1 can be rewritten iteratively:

$$Q^\pi = \begin{cases} 0 & \text{if} s_0 \models s_\star \\ \sum_{\mu \in M_\pi} \mathbb{P}(\mu|\pi) * c(\mu) & \text{otherwise} \end{cases} \tag{6.5}$$

$$\mathbb{P}(\mu|\pi) = \prod_{i \in \mu} \mathbb{P}[s_{i+1}|s_i, a_i] \tag{6.6}$$

with $\prod_{i \in \mu} \mathbb{P}[s_{i+1}|s_i, a_i] = 0$ for infinite traces $\mu = (s_0, s_1, \ldots)$. This definition can then be easily adapted to incorporate soft trajectory constraints by adding the penalties to the total cost $c(\mu)$ of the given trace. Given a set of soft trajectory constraints $\Phi$ and their associated value $\omega_\varphi$ the cost of a trace is defined as:

$$c_\gamma(\mu) = \sum_{i \in \mu} c_{a_i} + \sum_{\varphi \in \Phi | \mu \not\models \varphi} \omega_\varphi \tag{6.7}$$

Minimizing $Q^{\pi}$ will then result in the planning system trying to achieve *universal* achievement of the soft trajectory constraints ($\mathcal{A}.\varphi$). As soft trajectory constraints are not required to be fulfilled in the goal state, some soft constraints might not be achieved at all and others only on some traces, resulting in *existential* achievement($\mathcal{E}.\varphi$) of the constraints. An optimal plan, fulfilling all constraints however will achieve *universal* achievement($\mathcal{A}.\varphi$).

The following section describes a method of planning in FOND with state trajectory constraints using the heuristic search algorithm $LAO*$, which given a consistent heuristic generates the optimal policy.

**Guiding the search towards fulfilling the soft trajectory constraints.**

For a FOND planning task $\Pi$ with soft trajectory constraints $\Phi$, the same approach as described in Chapter 5 can be applied. However, to be able to apply the reward shaping described in the general action compilation (Section 5.1) an equivalent definition of state-dependent action costs needs to be added to the FOND setting. As the effect of an action is not only dependent on the current state but also on a non-deterministic choice, the action cost function would have to be state and possibly effect dependent. This can be modeled by adding a state-dependent action cost function to every possible outcome of the action:

**Definition 52 (State- and effect-dependent action costs).** Given an action $a = \langle pre, eff \rangle$ with non-deterministic effects $eff = \langle eff_1, \dots, eff_n \rangle$ then a cost function $c_a(s)_{eff_i} : \mathcal{S} \rightarrow \mathbb{N}$ for every $eff_i \in eff$ is defined assigning a cost of application to every possible outcome of the action $a$.

As the compilation introduced in Section 5 increases the search space significantly (worst case double exponential), an alternative approach was chosen for the FOND setting.

Given a FOND planning task with soft trajectory constraints $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, \Phi \rangle$ the first step is to track the state of each soft trajectory constraint $\varphi \in \Phi$, and secondly adapt the heuristic to guide the search in to fulfilling them.

**Tracking soft trajectory constraints in FOND planning.** Given a FOND planning task $\Pi = \langle \mathcal{V}, \mathcal{A}, s_0, s_\star, \Phi \rangle$ with the set of variables $\mathcal{V}$, a set of non-deterministic actions $\mathcal{A}$, an initial state $s_0$, a goal definition $s_\star$ and a set of soft trajectory constraints $\Phi$. First a deterministic finite automaton $\mathcal{A}_\varphi$ is created for every soft trajectory constraint $\varphi \in \Phi$. Then, for every $\mathcal{A}_\varphi$ a tracking variable $\tau_\varphi$ with domain $\mathcal{D}_{\tau_\varphi} = size(\mathcal{A}_\varphi)$ tracking state of $\mathcal{A}_\varphi$ along the current trace is added to $\mathcal{V}$ and initialized in $s_0$ to the initial state of

$\mathcal{A}_{\varphi}$. Then for updating the tracking variables $\tau$ the definition of the action application (Definition 37) is altered:

**Definition 53 (Action application with trajectory constraint tracking).** Given an action $a = \langle \mathit{pre}, \mathit{eff} \rangle$ with precondition *pre* and a non deterministic effect $\mathit{eff} = \{\mathit{eff}_1, \ldots, \mathit{eff}_n\}$ the result of applying action $a$ in state $s$ is defined as the new set of states

$$\mathcal{S}' = \{s'_1, \ldots, s'_n\}$$

with $s_i = \mathit{update}(\mathit{eff}_1, s)$

The $\mathit{update}(\mathit{eff}_i, s)$ function takes the effect $\mathit{eff}_i$ and applies it to state $s$ such that $s'_i = [\mathit{eff}_i]_s$. Additionally, for every soft trajectory constraint $\varphi \in \Phi$ the corresponding tracking variable $\tau_\varphi$ is updated. This is done by checking every outgoing transition $\langle q, \delta, q' \rangle$ for $q = s(\tau_\varphi)$. If $s'_i \models \delta$ then $s'_i(\tau_\varphi) = q'$.

**Guiding the search.** Now that the status of the soft trajectory constraints can be tracked during the search process, it needs to be guided towards fulfilling them. In contrast to the previous work described in Section 5, the action costs are not altered, as to keep the planning task compact and prevent state-dependent action costs. Therefore, an alternative way of informing the heuristic is required. For this the transition cost table (Table 5.1) is revisited, and the heuristic value $h(s)$ augmented by the penalty/reward value. A slight generalization of heuristics is used, defining heuristics over state traces $\mu = (s_0, \ldots, s_n)$, instead of states, where $l(\mu)$ denotes the last state of the trace $\mu$. This is done, as the heuristic value taking state trajectory constraints into account, depends on the trace $\mu$ for reaching the current state, and the trace $\mu'$ of reaching a goal state.

The optimal soft trajectory constraint aware heuristic $h^*(\mu)$ minimizing the expected cost of reaching the goal state $s_\star$ from any current state $\mathit{last}(\mu)$ is given by

$$h^*(\mu) = \min_{\pi_{l(\mu)} \in \Pi} \mathbb{E}\left[c_h(\mu, \mu') | \mu' \in M_{\pi_{l(u)}}\right] \tag{6.8}$$

$$c_h(\mu, \mu') = \sum_{i=0}^{|\mu|} c_{a_i} + \sum_{\varphi \in \Phi | (\mu, \mu') \not\models \varphi} \omega_\varphi \tag{6.9}$$

where $\pi_{l(\mu)}$ is a strong cyclic policy starting at $l(\mu)$, and $M_{\pi_{l(u)}}$ the set of all traces consistent with $\pi_{l(u)}$. This can be rewritten as the sum over expected

values:

$$
\begin{aligned}
h^*(\mu) = \min_{\pi_{l(\mu)} \in \Pi} ( \\
& \mathbb{E}\left[c(\mu')|\mu' \in M_{\pi_{l(u)}}\right] + \\
& \mathbb{E}\left[\left(\sum_{\varphi \in \Phi|(\mu,\mu') \not\models \varphi} \omega_\varphi\right) |\mu' \in M_{\pi_{l(u)}}\right] \\
& )
\end{aligned}
\tag{6.10}
$$

Calculating $h^*$ is however equivalent of finding an optimal plan. Therefore, a new heuristic function based on classical heuristics is needed. Given a trace $\mu$, and a set of soft trajectory constraints $\Phi$ with their associated values $\omega_\varphi$, the new heuristic function is given by

$$
h'(\mu) = h_{\mathrm{c}}(l(\mu)) + \min_{\pi_{l(\mu)} \in \Pi} \left(\mathbb{E}\left[\left(\sum_{\varphi \in \Phi|(\mu,\mu') \not\models \varphi} \omega_\varphi\right) |\mu' \in M_\pi\right]\right) \tag{6.11}
$$

where $h_{\mathrm{c}}$ is a classical heuristic using the all outcome determinization [1] ignoring the soft trajectory constraints to calculate the heuristic of a state in a FOND task. As

$$
p(\mu) = \min_{\pi_{l(\mu)} \in \Pi} \left(\mathbb{E}\left[\left(\sum_{\varphi \in \Phi|(\mu,\mu') \not\models \varphi} \omega_\varphi\right) |\mu' \in M_\pi\right]\right)
$$

in general, is still hard to compute, three possible approximations are presented.

**Zero penalty approximation.** The trivial way of approximating $p$ is to simply ignore it by replacing $p$ by 0, resulting in $h'(\mu) = h_{\mathrm{c}}(l(\mu))$. However, this is uninformative regarding the soft trajectory constraints.

**Relaxed reachability approximation.** The idea here is, that given the tracking variable $\tau_\varphi$ for a soft trajectory constraint $\varphi$, storing the status of the underlying DFA $\mathcal{A} = \langle \Sigma, Q, q_0, Q_a \rangle$. If $\tau_\varphi$ can not assume the value of an accepting state $\tau_\varphi \in Q_a$ in the relaxed planning task starting at $l(\mu)$, add the

[1] Every possible outcome of an action is encoded in to a new action, and the planner selects the most suitable outcome.

penalty $\omega_\varphi$ for $\varphi$ to the heuristic value. Formally $p(\mu)$ is replaced by $p^+(\mu)$ with

$$p^+(\mu) = \sum_{\varphi \in \Phi} \omega * [(\tau_\varphi = v \text{ for some } v \notin Q_a) \text{ is relaxed reachable from } l(\mu)] \tag{6.12}$$

For this, it is necessary to calculate the relaxed plan such as in the $h^{\text{max}}$ or $h^{\text{add}}$ heuristic. For better performance, a second approximation is provided, not relying on any additional calculations during search.

**Automaton reachability approximation.** Alternative to the checking if the abstract planning task may assume an accepting value of $\tau_\varphi$ it is also possible to simply check if reaching an accepting state in the DFA $\mathcal{A} = \langle \Sigma, Q, q_0, Q_a \rangle$, ignoring edge labels, is possible from the current value of $\tau_\varphi$ in state $l(\mu)$. This can be done by a simple automaton reachability check from the current value of $\tau_\varphi$, to any accepting state $q \in Q_a$. This can be effectively computed by simply precomputing for every a state $q \in Q$ if there exists an accepting state $q_\star \in Q_a$ such that $q_\star$ is reachable from $q$ ignoring edge labels. Formally $p(\mu)$ is replaced by $p^{\mathcal{A}}(\mu)$ with

$$p^{\mathcal{A}}(\mu) = \sum_{\varphi \in \Phi} \omega_\varphi * [\mathcal{A}_\varphi(l(\mu)(\tau)) \cap Q_a = \emptyset] \tag{6.13}$$

$$\tag{6.14}$$

where $\mathcal{A}_\varphi(v)$ is the set of states in $\mathcal{A}_\varphi$ reachable from state $v$ ignoring edge labels.

**Proposition 12.** If $h_{\text{c}}$ is admissible then the heuristic function $h'$ with relaxed reachability approximation is admissible.

*Proof.* Following the inequality

$$\min(f(x)) + \min(g(x)) \leq \min(f(x) + g(x)) \text{ for all } x \in \mathbb{R} \tag{6.15}$$

$h^*(\mu)$ can be approximated by its lower bound $h^*(\mu) = a(\mu) + p(\mu)$, where

$$a(\mu) = \min_{\pi_{l(\mu)}} \left( \mathbb{E} \left[ c(\mu') | \mu' \in M_{\pi_{l(\mu)}} \right] \right) + \tag{6.16}$$

$$p(\mu) = \min_{\pi_{l(\mu)}} \left( \mathbb{E} \left[ \left( \sum_{\varphi \in \Phi | (\mu, \mu') \not\models \varphi} \omega_\varphi \right) | \mu' \in M_{\pi_{l(\mu)}} \right] \right) \tag{6.17}$$

Let $h^*_{\text{c}}$ be the classical perfect soft trajectory constraint agnostic heuristic function over the all outcome determinization. Then

$$h^*_{\text{c}}(\mu) \leq a(\mu) \tag{6.18}$$

as the all outcome determinization chooses the cheapest possible outcome for an action, and the minimum over the possible outcomes is strictly less or equal to the expected value:

$$\begin{aligned} h_{\mathrm{c}}^{*}(\mu) = \min_{\pi_{l(\mu)}} \left(\min\left[c(\mu')|\mu' \in M_{\pi_{l(\mu)}}\right]\right) \leq \\ \min_{\pi_{l(\mu)}} \left(\mathbb{E}\left[c(\mu')|\mu' \in M_{\pi_{l(\mu)}}\right]\right) = a(\mu) \end{aligned} \tag{6.19}$$

Thus, it is sufficient to show that the penalties $p^{+}$ gathered by the relaxed reachability approximation is less or equal to $p$ ($p^{+} \leq p$). This is now shown for a single soft trajectory constraint $\varphi$, as the generalization for all soft trajectory constraints follows by simply summing up all constraint penalties. This is now shown on a case-by-case basis:

- If there exists a trace $\mu' \in M_{\pi}$ so that $(\mu, \mu') \models \varphi$ holds, then the fact $\tau_{\varphi} = q$ for some $q \in Q_a$ is reachable from $l(\mu)$, and therefore also relaxed reachable from $l(\mu)$. Thus, $p^{+}(\mu) = 0$ but $p \geq 0$ as the expected value in $p$ may contain traces for which $\varphi$ is not fulfilled ($p^{+} \leq p$).
- If for all traces $\mu \in M_{\pi}$ it holds that $(\mu, \mu') \not\models \varphi$, then $p(\mu) = \omega_{\varphi}$ and as $p^{+}(\mu) \leq \omega_{\varphi}$, it follows that $p^{+} \leq p$.

From this follows that $p^{+} \leq p$, and because $h_{\mathrm{c}} \leq h_{\mathrm{c}}^{*}$ (for admissible heuristic functions $h_{\mathrm{c}}$)

$$h'(\mu) = h_{\mathrm{c}}(l(\mu)) + p(\mu) \leq h_{\mathrm{c}}^{*}(l(\mu)) + p(\mu) \leq a(\mu) + p(\mu) \leq h^{*}(\mu) \tag{6.20}$$

holds by definition of $h'$, admissibility of $h_{\mathrm{c}}$, Equation 6.18 and $p^{+} \leq p$, and Equation 6.15. □

**Corollary 2.** If $h_{\mathrm{c}}$ is admissible then the heuristic function $h'$ with automaton reachability approximation is admissible.

*Proof.* As every accepting state $q \in Q_a$ reachable in the relaxed task is also reachable in the automaton reachability. □

Additionally there may exist accepting states $q \in Q_a$ that are reachable in the automaton, but not reachable in the relaxed task. This follows, that some automaton transition requirements might never become true (unreachable in the planning task).

As a final remark to soft trajectory constraint aware heuristics concerns the goal awareness.

**Proposition 13 (Goal awareness).** As soft trajectory constraints add positive penalties to the original heuristic $h_\mathrm{c}$ if the constraint is not fulfilled. This leads to a heuristic value (as calculated in Equation 6.11) $h' > 0$ in states $s$ that fulfill the goal condition $s \models s_\star$ but do not fulfill the constraints. However, if all traces fulfill the soft constraints, there will be no penalty added to the goal state heuristic values. Thus, only in this case goal awareness is preserved.

One drawback of this approach is that it is planner specific and needs to be implemented for every planning system that requires soft trajectory constraints. On the other hand, this approach keeps planning task compacter, as no additional actions, conditional effects or state-dependent action costs are required.

Note that this approach can be extended to hard trajectory constraints by adding the fact that each $\tau_\varphi$ must represent an accepting state to the goal condition, similarly to trajectory constraints defined in Section 5. For the case of hard constraints Camacho et al. (2017) provides a more general method not restricting the LTL formula to the ones provided by PDDL 3.0, compiling the FOND task with hard LTL constraints (or temporally extended goals) to a classical FOND task. In general, these FOND tasks with temporally extended goals are *2EXPTIME*-complete (De Giacomo and Rubin, 2018).

#### Empirical Evaluation

For the empirical evaluation, the above concept was implemented into the MyND (Mattmüller, 2013) planner. The benchmark set consisted of the same domains and instances as the ones used in Section 5.2. Additionally, the existing blocksworld domain with nondeterministic actions (Example 9) was augmented to include soft trajectory constraints (*blocksworld-stc* in Table 6.1). The experiments where then executed with the $h^{\max}$ and $h^{\mathrm{ff}}$ heuristic with the *automaton reachability approximation*. As can be seen in Table 6.1 in most cases the coverage of the MyND implementation is comparable to the coverage by the compilation approach executed by *Fast Downward* and *Sympl* (Compared with their best results in optimal and satisficing configuration). In the case of the augmented blocksworld domain *blocksworld-stc* the coverage was almost 100% as the generated instances where smaller than the deterministic domains or the unaltered blocksworld domain, and the number of soft trajectory constraints was also lower. As already discussed previously, a more informed heuristic such as in the *relaxed reachability approximation* would further improve the overall coverage of the approach, especially when using the $h^{\max}$ heuristic. Additionally, a more

informed heuristic, could also improve on the overall quality of the found policies, as with the *automaton reachability approximation* the policies often do not fulfill many soft trajectory constraints, as depicted in Figure 6.1, Figure 6.4, and Figure 6.5.

| | Fast Downward | | Symple | MyND | |
|---|---|---|---|---|---|
| | $h^{\mathrm{ff}}$ | $h^{\max}$ | | $h^{\mathrm{ff}}$ | $h^{\max}$ |
| open stacks | 86.79% | 7.14% | 32.14% | 68.57% | 0.00% |
| pathways | 63.33% | 19.52% | 40.95% | 46.67% | 40.95% |
| rovers | 51.43% | 22.86% | 34.29% | 67.86% | 19.29% |
| storage | 28.19% | 14.09% | 21.81% | 51.42% | 49.41% |
| trucks | 86.07% | 22.14% | 0.00% | 36.07% | 13.93% |
| blocksworld | – | – | – | 70.00% | 30.00% |
| blocksworld-stc | – | – | – | 93.88% | 93.88% |

Table 6.1: Comparing goal action compilation with MyND implementation on deterministic instances

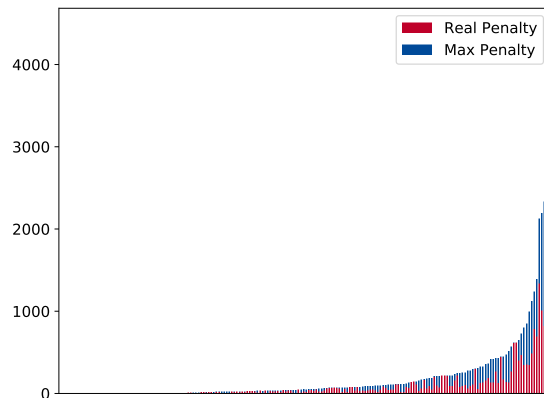



Figure 6.1: Open stacks domain maximal penalty vs. gathered penalty

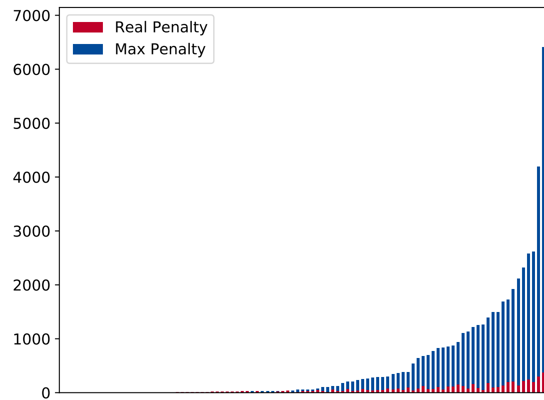



Figure 6.2: Pathways domain maximal penalty vs. gathered penalty

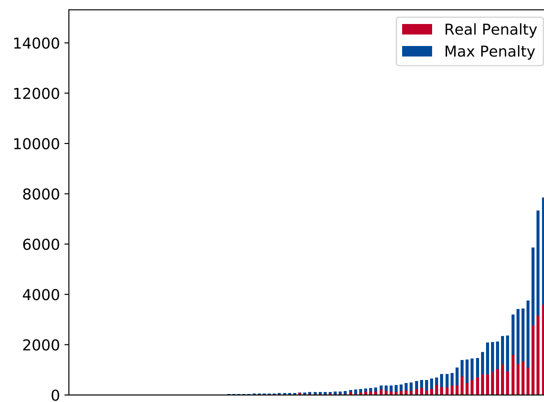



Figure 6.3: Rovers domain maximal penalty vs. gathered penalty

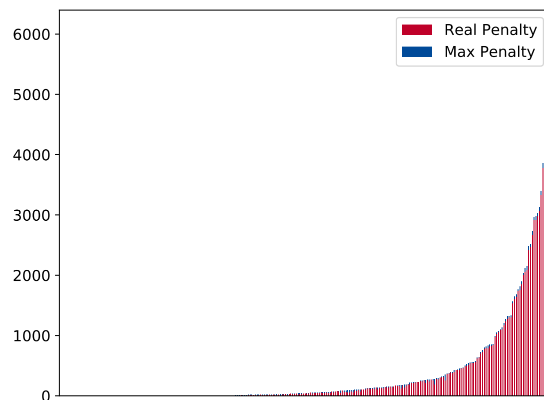



Figure 6.4: Storage domain maximal penalty vs. gathered penalty

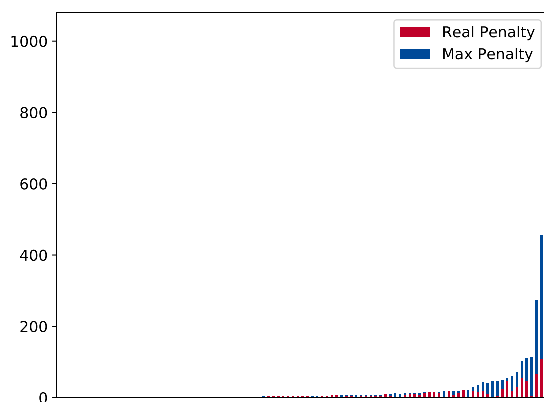



Figure 6.5: Trucks domain maximal penalty vs. gathered penalty

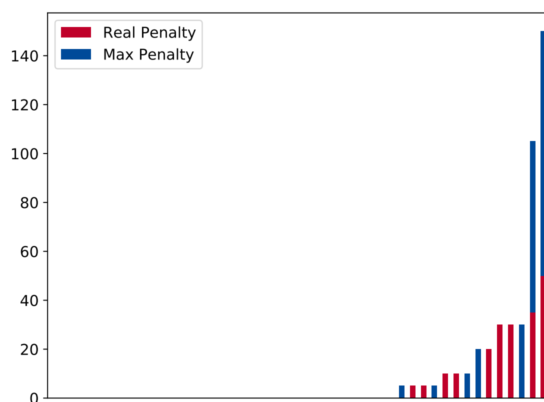



Figure 6.6: Blocksworld domain maximal penalty vs. gathered penalty

# Chapter 7

# Application in Digital Preservation

A major issue, when dealing with digital content is the loss of data. This can be the result of missing or faulty data management. In the case of university research, the additional issue of frequent staff change is added to the problem. Data that is stored on laptops, tablets, USB sticks and other individual devices may be lost due to hardware failure or the fact that the employee responsible for the data leaves the university. Many institutions provide some kind of central storage system, where employees may store their data. However, in most cases, no regulation on what data is required to be stored on the central server are in place, nor any guidelines on how the data is organized exist. This leads to the case, that most users develop their own unique way of storing and organizing data on laptops, private drives, USB sticks and the central server. This leads to a decentralized and, for an outsider, unorganized storage of data. This leads to irretrievable data, in the sense that it is known to exist, but it is unknown where or in what format. This results in a massive workload on the side of researchers trying to recall and built upon existing results. Additionally, from a systems administrators perspective, it is unknown which data may be deleted freeing resources for other users, resulting in data from users long gone still occupying resources. One solution to these issues is digital preservation. It provides guidelines, software, and hardware to store data for a long time in an organized and retrievable way. The main topics of digital preservation include among other things:

1. **Data appraisal** provides policies describing which data needs to be archived, and which data is explicitly not archived. For instance publications are usually archived along with software used to generate the underlying data. On the other hand the generated data itself must not

be archived, if it can be regenerated from the software.

2. **Data identification** deals with the problem of uniquely identifying data by assigning (system)unique resource identifiers, and descriptive meta data. In the case of a publication, such meta data may consist of the title and where it was published.

3. **Data integrity** ensures the integrity of the digital files by providing means to protect the files from intentional or unintentional alterations, as well as means of identifying these changes. A classical way of identifying alterations is to store a hash value alongside the files.

4. **Data characterization** deals with identifying the kind of data which is archived, and in which format it is stored. Additionally, it also provides means of identifying how a given file may be read or interpreted at a later stage. In the case of a publication, this will usually identify the file as a PDF or WORD file, and associate the respective readers, or format specifications for the file type.

5. **Data sustainability** deals with the issues concerning longevity of a file. It may enact processes for file conversion from older obsolete formats to newer open standards.

6. **Data authenticity** provides means of ensuring that the files stored in the archive are the files they claim to be.

7. **Data access** provides measures to ensure regulated file access such as access control and security.

To support the BrainLinks-BrainTools cluster of excellence at the Albert-Ludwigs University of Freiburg in their goal of applying digital preservation to their research process, a software called OntoRAIS was developed. This software deals with the issues of integrity, characterization, authenticity and access as defined in the above list. As research data, in contrast to data from cultural heritage, is usually more short lived, with data usually obsolete within a couple of years (with exception of publications), due to research advances, data sustainability was not a priority in this project.

The rest of this chapter introduces OntoRAIS (**Onto**logy based **R**esearch **A**rchival **I**nformation **S**ystem) and its components. One major issue when dealing with such information systems is the constant evolution of the software and its context. Thus, processes that are executed using the software need to be specified in a flexible way, which can adapt to changing requirements. For this a light weight workflow description language is presented in Section 6, followed by a method for generating such workflows applying FOND planning.

## 7.1 OntoRAIS

The OntoRAIS system consist of four major modules (Figure 7.1). The client applications providing user friendly interaction with the services provided by the server application. The server provides services for ingesting and annotating files, user management, process management, and access control. The storage is a simple redundant storage drive an is not closer described in this work. Finally, OntoRAIS facilitates ontology based data access (consisting of the library QUEST, an Ontology, a set of mappings , and a database, described in more detail in Section 7.1.2) to retrieve data from the database in a meaningful way .

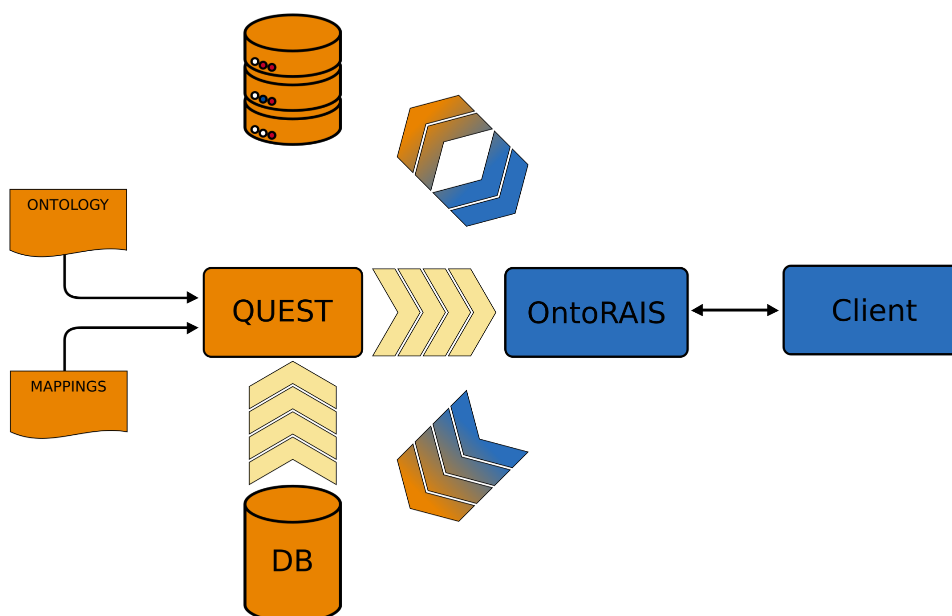



Figure 7.1: OntoRAIS system architecture

### 7.1.1 Clients

Two clients were implemented, providing interaction with the back end server via a traditional WEB API based on HTTP(S). The web client hereby provides an interface for the core functionality of adding new files, and annotating and linking them to each other. Additional to adding new files to the archive, a major use case is the access of existing files. Therefore the client provides interfaces for searching, accessing, and downloading files. Furthermore a simple project management system was implemented such that people and files can be assigned to projects reflecting the research projects

carried out by the individual research groups within the cluster. Figure 7.2 shows the view of an archived object, and Figure 7.3 shows the view of a research project in the archive. Figure 7.4 shows the meta data entry step of the process ingesting an article to the archive. Additionally, the web

Figure 7.2: Object view in web client

client provides the interface for managing the archive itself such as user and process management. One shortcoming of the web client is the fact that the HTTP protocol is not suitable for large file uploads. Additionally, the web client does not integrate in to existing research workflows too well, as it requires the user to fully ingest all files together with all the meta data at once, and the fact that it needs to be executed in the browser. Therefore, a light weight desktop client was developed, providing only the basic upload functionality together with a minimum of meta data required for later identification. The missing meta data, links, and project assignments can then be later added using the web client.

### 7.1.2 Server

This is the main component of the archiving system, it provides a web API for interacting with the systems components, and manages the stored data. It is responsible for performing integrity checks, and provides user access

Figure 7.3: Project view in web client

Figure 7.4: View of ingest process in web client

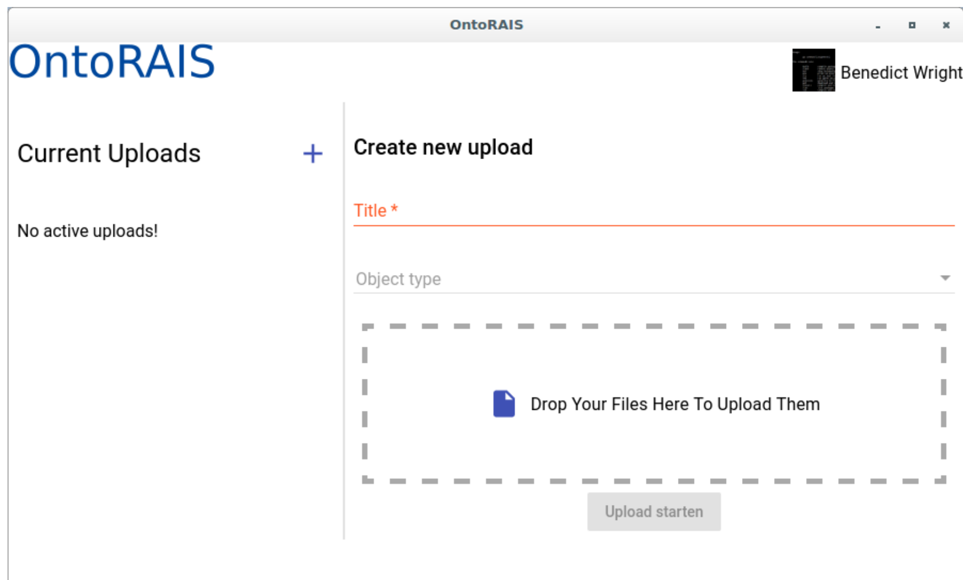



Figure 7.5: View of the desktop client

control to the individual resources. One outstanding feature of the architecture is that it applies ontology based data access to manage the underlying data. Hereby, business objects are modeled using an Ontology, and the data is stored in a database. This adds a semantic layer on top of the data layer, adding flexibility to defining data and relations. In contrast to classical database driven applications, the data is not queried using SQL but rather the SPARQL query language. This approach combines the expressiveness of ontologies with the storage capabilities of databases. In the following this method is briefly introduced.

### Ontology based data access

**Definition 54 (Ontologies).** A ontology signature is a tuple

$$\langle \mathcal{N}_C, \mathcal{N}_R, \mathcal{N}_O \rangle$$

where $\mathcal{N}_C$ is a set of concept names, $\mathcal{N}_R$ a set of role names, and $\mathcal{N}_O$ a set of individuals. The ontology $\mathcal{O}$ itself is a tuple

$$\langle \mathcal{T}\text{-Box}, \mathcal{A}\text{-Box} \rangle$$

where the $\mathcal{T}$-Box consists of concept assertions, stating how concepts and roles are related to each other, and the $\mathcal{A}$-Box consists of concept and role assertions.

Depending on the underlying description logic (DL) the available concept construction rules vary. In the rest of the thesis the $\mathcal{ALCQ}$ fragment (Baader et al., 2010) is used.

**Definition 55 ($\mathcal{ALCQ}$).** The $\mathcal{ALCQ}$ language is constructed by the following rules:

$$\mathcal{ALCQ} := \bot | \top | A | \neg C | C \sqcap D | C \sqcup D | \exists R.C | \forall R.C | \leq nR.C | \geq nR.C$$

where $\top, \bot$ are the universal an empty concepts respectively. $A$ is an atomic concept and $\neg C$ the negation. $C \sqcap D$ and $C \sqcup D$ are the intersection and union of two concepts. Finally, $\exists R.C, \forall R.C, \leq nR.C, \geq nR.C$ are quantified concept constructors.

When dealing with modern ontology languages such as OWL (W3C, 2012) roles are often defined as relations between domain types and range types. Formally, this is defined as

$$\begin{aligned} \text{domain}(R) = C \quad &:= \quad \exists R.\top \sqsubseteq C \\ \text{range}(R) = C \quad &:= \quad \top \sqsubseteq \forall R.C \end{aligned}$$

with $R \in \mathcal{N}_R$ and $C$ a concept, stating that the domain (left hand side) or range (right hand side) of the role $r$ may only be of type $C$.

From above definitions, the possible $\mathcal{T}$-Box and $\mathcal{A}$-Box assertions can be defined:

**Definition 56 ($\mathcal{T}$-Box assertions).** The $\mathcal{T}$-Box consist of the following type assertions:

1. $C \sqsubseteq D$ (general inclusion axioms) with $C, D$ a concept, stating that the set of individuals of concept $C$ is a subset of the set of individuals of concept $D$.

Note that the domain domain$(R) = C$ and range range$(R) = C$ assertions, are a simple abbreviation for such an assertion.

**Definition 57 ($\mathcal{A}$-Box assertions).** The $\mathcal{A}$-Box consists of two possible types of assertions:

1. $i\ R\ j$ (role assertion) with $i, j \in \mathcal{N}_O$ and $R \in \mathcal{N}_R$ stating that $i$ is in relation $R$ with $j$.

2. $i : C$ (concept assertion) with $i \in \mathcal{N}_O$ and $C \in \mathcal{N}_C$ stating that individual $i$ is of concept $C$.

Ontology based data access is then the process of accessing data stored in a database through a semantic layer provided by an ontology (Bagosi et al., 2014).

**Definition 58 (Ontology based data access system).** An ontology based data access system is a tuple $\mathcal{D} = \langle \mathcal{O}, \mathcal{B}, \mathcal{M} \rangle$, where $\mathcal{O}$ is an ontology signature together with a $\mathcal{T}$-Box. $\mathcal{B}$ is a relational database storing the $\mathcal{A}$-Box assertions, and $\mathcal{M}$ is a set of mappings $\psi(C) \rightarrow \varphi(C)$ from concepts and roles to database entries, where $\psi$ is a SPARQL (W3C, 2013) query over $\mathcal{O}$ with free variables $X$, and $\varphi$ is a SQL query over $\mathcal{B}$ returning values for $X$ (Bagosi et al., 2014).

**Example 22.** Using the *ontop* (Bagosi et al., 2014) framework and an ontology $\mathcal{O}$ with the signature:

$$\begin{aligned} \mathcal{N}_C &= \{\text{BOOK}, \text{TITLE}\} \\ \mathcal{N}_R &= \{\text{hasTitle}\} \\ \mathcal{N}_O &= \{1, The\ Description\ Logic\ Handbook\} \end{aligned}$$

and an empty $\mathcal{T}$-Box.

The mapping for BOOK in $\mathcal{M}$ could be defined as:

$$\begin{aligned} \psi(\text{BOOK}) &:= \text{?id : BOOK} \\ \varphi(\text{BOOK}) &:= \text{SELECT id FROM books_table} \end{aligned}$$

and the mapping for hasTitle could be defined as:

$$\begin{aligned} \psi(\text{hasTitle}) &:= \text{?id hasTitle ?title} \\ \varphi(\text{hasTitle}) &:= \text{SELECT id, title FROM title_table} \end{aligned}$$

Note, that the columns from the SQL select queries correspond to the free variables (marked by ?) from the SPARQL query. Given the database tables in Table 7.1 the induced $\mathcal{A}$-Box is then

$$\begin{aligned} &1 : \text{BOOK} \\ &1 \text{ hasTitle "The Description Logic Handbook"} \end{aligned}$$

stating that the individual 1 is a book with the title “The Description Logic Handbook”.

| books_table |
|---|
| id |
| 1 |

| title_table | |
|---|---|
| id | title |
| 1 | The Description Logic Handbook |

Table 7.1: Database tables for BOOK and hasTitle

While ontology based data access provides ways of retrieving an $\mathcal{A}$-Box from a database, it doe not provide ways of storing new assertions. This

is due to the view update problem (Tomasic, 1988), that, given a projection from multiple databases/tables to a singe view, find the minimum necessary changes to the underlying databases/tables, such that the wanted update to the view is realized. In this work the QUEST library (Bagosi et al., 2014) was used which provides all required OBDA functionality for the OntoRAIS application.

Objects in the server's data layer are modeled using such an ontology. Hereby, every class in the data layer represents a concept in the ontology, and members of the classes represent roles. Additionally every class has a member for storing the unique identifier of the object. Instantiating an object then consist of a SPARQL query to the ontology based data access system. For adding new instances, corresponding to an $\mathcal{A}$-$\mathrm{Box}$ assertion, the underlying database needs to be updated. In general this is nontrivial and is an instance of the view update problem. However, by applying rigid restrictions to how the mappings in $\mathcal{M}$ are defined, this can be circumvented. Therefore, the requirements towards the mappings $\mathcal{M}$ is such that each SQL query in $\varphi$ consists of a single SELECT statement over a single table only. This SELECT query can then easily be used to generate update, delete, and insert statements using simple query rewriting:

**Example 23 (Transforming simple SELECT queries).** From a given simple SELECT query of the form

$$\text{SELECT } id, col_1, \ldots, col_n \text{ FROM } table$$

the insert, delete and update queries can be rewritten as:

$$\begin{aligned}
&\text{UPDATE } table \text{ SET } col_1 = v_1, \ldots, col_n = v_n \text{ WHERE } id = id \\
&\text{INSERT INTO } table(id, col_1, \ldots, col_n) \text{ VALUES } (id, v_1, \ldots, v_n) \\
&\text{DELETE FROM } table \text{ WHERE } id = id
\end{aligned}$$

Once the data layer of the application is modeled, processes for interacting and modifying the data is required. Thus, a way of modeling the processes or workflows for the individual business cases is needed. The main business case for a digital preservation system is the ingestion of new data to the archive. In general this process will consist of uploading files, entering meta data, and linking to projects, authors, and related work. Most existing preservation systems are aimed at archival experts such as librarians or custodians. Therefore, one aim of this project was to develop user interfaces focused on research employees, with limited knowledge of digital preservation. Additionally, to the optimized user interfaced, the workflows also needed optimization in regards to usability and effectiveness, reducing the workload and mental burden of the users. Therefore, a individual workflow for every data type is required. As creating these workflows manually is

tedious, error prone, and need to be updated whenever the requirements towards the system changes, an automated approach is desirable. In the following a method for generating such workflows is presented.

**Automatic workflow generation**

The workflows considered here are for adding new documents to the archive. However, this can be generalized to any kind of data driven workflow, where the target is the entry of some information to a knowledge base (e.g. user creation).

Such a workflow consist of two types of actions, one for collecting information such as user interaction or web service calls, and the second for actually creating the $\mathcal{A}$-Box assertions and adding new individuals to $\mathcal{N}_O$. In this setting it is assumed that the $\mathcal{T}$-Box stays unaltered.

Additionally, one must distinguish two types of concepts. Primitive concepts, which are always atomic concepts, and correspond to primitive data types such as strings. In the example above this would be TITLE, which corresponds to a simple string. And on the other hand complex concepts, such as ARTICLE from above example, which do not correspond to a single primitive value. Workflows may only be generated for a single complex concept.

Formally: Given a concept $C_T$, generate a workflow (consisting of above mentioned action types) such that after executing the workflow, a set of new individuals has been added to $\mathcal{N}_O$ and a set of $\mathcal{A}$-Box assertions has been created, ensuring that all relevant data has been added. Relevant data is considered to be any related concept $C_T{}'$ and role $R$ such that domain$(R) = C_T{}' \in \mathcal{T}$-Box and range$(R) = C_T \in \mathcal{T}$-Box for some $R \in \mathcal{N}_R$ or vice versa. Thus, the set of individuals added to $\mathcal{N}_O$ consists of an individual of type $C_T$ and an individual for all possible primitive concepts $C_T{}'$. If the concept $C_T{}'$ is complex, it is assumed, that a separate workflow is required for creating such an individual, and therefore already exists at execution time of the workflow. The new $\mathcal{A}$-Box assertions are then all required concept assertions for the new individuals, together with their associated role assertions. Note that for existing individuals the role assertions connecting to new individuals are still asserted to the $\mathcal{A}$-Box. Note that there could exist cyclic relevances between two concepts $A$ and $B$, such that both workflows for $A$ and $B$ require the presence of an individual of the concept. However, as the individuals of complex concepts are assumed to already exist during workflow generation, this is not an issue that needs to be dealt with here.

**Example 24.** Recalling Example 22 and adding the $\mathcal{T}$-Box assertions

$$\text{domain(hasTitle)} = \text{BOOK}$$
$$\text{range(hasTitle)} = \text{TITLE}$$

Creating the workflow for the concept BOOK, the relevant data is then hasTitle and TITLE. Given that TITLE is a primitive concept of type string, the workflow would consist of two activities:

1. User input for the title string
2. Add two individuals $b$ of concept BOOK and $t$ of concept TITLE to $\mathcal{N}_O$, and add the $\mathcal{A}$-Box assertion $b$ hasTitle $t$.

The rest of this section describes how planning can be used to generate such a workflow. The approach uses FOND planning, thus the resulting workflows do not consider concurrency. However, branching dependent on the actions outcomes is supported. As the workflows in the OntoRAIS application are single user and sequential, this restriction is sufficient. Extending this to workflows, executed in parallel by multiple users, would require changing to a different planning setting such as temporal planning (planning with action durations and concurrency), or the use of action debinding and deordering (Waters et al., 2006).

First a planning task is generated using information from the ontology $\mathcal{O}$, a target concept $C_T$, and a schematic planning description $\widehat{\Pi}$. In schematic planning descriptions, actions and predicates may consist of free or typed variables, which are instantiated before the actual planning task is executed. This provides a more abstract syntax for describing the predicates and actions. The language used for the schematic planning description is PDDL (McDermott et al., 1998; Fox and Long, 2003; Edelkamp and Hoffmann, 2004; Gerevini and Long, 2005). A schematic planning description consist of the triple $\widehat{\Pi} = \langle T, P, A \rangle$, where $T$ is a typing system consisting of types and derived types, $P$ is a set of n-ary predicates consisting of a name and a list of typed arguments, and $A$ is a set of lifted actions each consisting of a set of typed input parameters, a set of preconditions, and a set of effects. The planner does not consider cardinality, as it is assumed that creating multiple entities of the same type is merely a repetition of the same actions. Executing the planner on the grounded version of the planning task results in a policy $\pi$, stating which action needs to be executed in which state. This policy $\pi$ is then converted into a workflow, by converting planning actions in to workflow activities. At this point the cardinality restrictions from the ontology are added to the activities. Finally, similar activities can be merged into a single workflow step, resulting in an executable workflow (Figure 7.6).

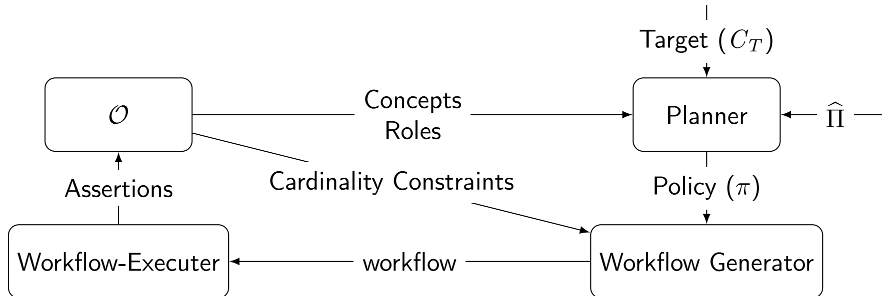



Figure 7.6: Overview of the workflow generation architecture

**Creating the schematic planning task.** First the schematic planning description is created. In this case the typing system $T$ consists of two top level types `concept-type` and `role-type`. For each atomic concept $C \in \mathcal{N}_C$ a corresponding type $t_C$ is added to $T$ as subtype of `concept-type` . For each role $R \in \mathcal{N}_R$ a corresponding type $t_R$ is added to $T$ as subtype of `role-type`.

The objects in the concrete planning task represent either individuals already in $\mathcal{N}_O$ or new individuals that are added to the set $\mathcal{N}_O$ during the execution of the final workflow. Additionally, each required role assertion $(o\ R\ o')$ is represented by an object $o_R$ in the planning task with the type $t_R$. This is required by the planning task, as to be able to assign individuals to the corresponding roles.

**Example 25.** The description of the schematic planning task will be accompanied by examples using the ontology $\mathcal{O} = \langle \mathcal{T}\text{-Box}, \mathcal{A}\text{-Box}\rangle$ and the

signature $\langle \mathcal{N}_C, \mathcal{N}_R, \mathcal{N}_O \rangle$, with:

$$\mathcal{N}_C = \{Article, Publication, Textual, DigitalArchiveObject, ArchiveObject, Person, File, Keyword, Title\}$$

$$\mathcal{N}_R = \{isAuthorOf, hasFile, hasKeywork, hasTitle\}$$

$$\mathcal{T}\text{-Box} = Article \sqsubset Publication \sqsubset Textual \sqsubset DigitalArchiveObject \sqsubset ArchiveObject \sqsubset \top,$$
$$Keyword \sqsubset \text{String}, Title \sqsubset \text{String},$$
$$domain(isAuthor) = Person, range(isAuthor) = ArchiveObject,$$
$$domain(hasFile) = DigitalArchiveObject, range(hasFile) = File,$$
$$domain(hasKeyword) = ArchiveObject,$$
$$range(hasKeyword) = Keyword,$$
$$domain(hasTitle) = ArchiveObject, range(hasTitle) = Title,$$

The set of individuals $\mathcal{N}_O$ and the $\mathcal{A}$-Box are empty.

The following predicates are defined in $P$. The first a predicate identifies if an object is to be added to $\mathcal{N}_O$ or if it can be selected from the existing set of individuals in $\mathcal{N}_O$. This corresponds to objects of primitive concept types or complex concept types. Additionally, this predicate is static, as it can not be altered during either the planning phase nor the workflow execution. For each concept $C \in \mathcal{N}_C$, $P$ contains the predicate

$$selectable(o \ - \ t_C).$$

**Example 26.** $selectable(o - Article), selectable(o - Publication), selectable(o - Textual), \ldots$

The workflows considered here are concerned with creating new individuals, thus information about these individuals must be collected. The next four predicates state how the information was gathered. These predicates are mutually exclusive, as each data may only be acquired once. For each primitive concept $C \in \mathcal{N}_C$, $P$ contains the predicates

$$gathered(o \ - \ t_C) \text{ and}$$
$$generated(o \ - \ t_C).$$

The predicate *gathered* states that *o* has been gathered from the user via some input method, whereas *generated* states that *o* has been generated by an external application.

**Example 27.** *gathered*$(o - $ *Keyword*$)$, *generated*$(o - $ *Keyword*$)$, *gathered*$(o - $ *Title*$)$, *generated*$(o - $ *Title*$)$

For each complex concept $C \in \mathcal{N}_C$, $P$ contains the predicates

$$\textit{user-selected}(o - t_C) \text{ and}$$
$$\textit{server-selected}(o - t_C).$$

The predicate *user-selected* states that the user has selected *o* from the existing set of individuals $\mathcal{N}_O$. Whereas, *server-selected* states that an external application has preselected the individual from the set of existing individuals $\mathcal{N}_O$.

**Example 28.** *user-selected(o - Article), server-selected(o - Article), user-selected(o - Publication), server-selected(o - Publication), user-selected(o - Textual), server-selected(o - Textual),...*

When executing a workflow that gathers information, it is usually required that the user review the entered information to ensure correctness. For this $P$ contains a predicate for each concept $C \in \mathcal{N}_C$, marking any object as reviewed.

$$\textit{reviewed}(o - t_C).$$

**Example 29.** *reviewed(o - Article), reviewed(o - Publication), reviewed(o - Textual),...*

The following predicate is required as to ensure that during planning, roles are associated with the correct domain and range. This predicate is static, as the configuration of a role can not be altered during planning or workflow execution. Thus, for each $R \in \mathcal{N}_R$ with domain$(R) = C$ and range$(R) = C'$, $P$ contains the predicate

$$\textit{role}(o - t_C, r - t_R, o' - t_{C'}).$$

**Example 30.** *role*$(o - $ *Person*$, r - $ *isAuthorOf*$, o' - $ *ArchiveObject*$)$,*role*$(o - $ *DigitalArchiveObject*$, r - $ *hasFile*$, o' - $ *File*$)$,...
where a concrete instance could be:

*role(Franz Baader, BaaderisAuthorOfTheDLHandbook, The DL Handbook).*

Finally, the last two predicates are required to state if an individual of the target concept $C_T$ and all the individuals of the relevant concepts and roles have been added to $\mathcal{N}_O$ and a corresponding concept or role assertion has been made to the $\mathcal{A}$-Box. Thus, for each $R \in \mathcal{N}_R$, $P$ contains a predicate

$$\textit{asserted-role}(r - t_R)$$

stating the role assertion has been added to the $\mathcal{A}$-Box.

**Example 31.** *asserted-role*$(r\ -\ $*isAuthorOf*$)$,*asserted-role*$(r\ -\ $*hasFile*$)$,...

Note, that a single predicate *asserted-concept* is sufficient to mark new individuals as added to $\mathcal{N}_O$ and to mark that a corresponding $\mathcal{A}$-Box assertion has been made, as no new individual can exist without a corresponding concept assertion. For each concept $C \in \mathcal{N}_C$ $P$ contains the predicate

$$\textit{asserted-concept}(o\ -\ t_C)$$

stating that the individual *o* has been added to $\mathcal{N}_O$, and that a corresponding concept assertion has been made to $\mathcal{A}$-Box.

**Example 32.** *asserted-concept*$(o -$ *Article*$)$, *asserted-concept*$(o -$ *Publication*$)$, *asserted-concept*$(o\ -\ $*Textual*$)$, ...

The set $A$ of action schemas consists of three types of actions: user interactions gathering, selecting, and reviewing data; web service calls for file uploads or the automatic extraction of information from a document; and application internal actions representing concept and role assertions to the $\mathcal{A}$-Box and the adding of a new individual to $\mathcal{N}_O$. Adding new individuals to $\mathcal{N}_O$ is not explicitly handled by an extra action, as new concept assertions always correspond to new individuals.

1. **User interaction**

$$\begin{aligned}\textit{gather}(o\ -\ \texttt{concept-type}) :=\ & \\ & \langle \neg \textit{gathered}(o) \wedge \neg \textit{generated}(o) \wedge \neg \textit{selectable}(o), \\ & \textit{gathered}(o)\rangle\end{aligned}$$

This action corresponds to a simple user input in the final workflow, gathering a value for the stated property. The gathered property hereby refers to an individual for a primitive concept. It takes an object of a certain concept type as parameter, requiring it to not yet be gathered, or generated. The result of this action is then that the object is marked as gathered, indicating that the user has entered the data.

$$\begin{aligned}\textit{user-select}(o\ -\ \texttt{concept-type}) :=\ & \\ & \langle \neg \textit{user-selected}(o) \wedge \neg \textit{server-selected}(o) \wedge \\ & \textit{selectable}(o),\ \textit{user-selected}(o)\rangle\end{aligned}$$

Similar to *gather* this action represents user interaction, selecting an individual from a complex concept already in $\mathcal{N}_O$. The precondition ¬ *server-selected(o)* ensures that objects already preselected by a web service are not re-selected. The selected object is then marked as *user-selected(o)*.

The differentiation between data gathered or selected by the user and data generated or selected by an external application is done, as to be able

to review these at different stages of the workflow, and to be able to distinguish them later on in the workflow generation. Usually data generated or selected by an external application is reviewed directly after the action has finished, whereas user data is usually reviewed in a summary at a later stage. This is also the reason for having two distinct review actions, one for objects gathered or selected by the user, and one for objects generated or selected by an external application:

$$\begin{aligned}&\textit{review-all}() := \\ &\quad \langle \emptyset, \\ &\quad \forall o \; - \; \texttt{concept-type}: \; (\textit{gathered}(o) \vee \textit{user-selected}(o)) \rhd \textit{reviewed}(o)\rangle\end{aligned}$$

This action does not have any parameters or preconditions and simply sets all objects already gathered or selected as reviewed. This is a shortcut to reviewing all objects individually.

$$\begin{aligned}\textit{review-generated}(o \; - \; \texttt{concept-type}) := \\ \langle \textit{generated}(o) \vee \textit{server-selected}(o), \\ \textit{reviewed}(o)\rangle\end{aligned}$$

Data generated or selected by an external application needs to be reviewed at a different stage as data entered by the user, as they may contain errors created by the application. Therefore above action reviews this data, and set is as reviewed.

2. **Web service calls**

The following actions describe interactions between the application and external services. The definition of the corresponding schematic actions depend on the individual applications. Hereby, the schematic action must reflect the data flow performed by the application (data generated, selected,or asserted). In the following this is described using two examples such as a file upload action *upload* and an automated meta data extraction action *extract-data*.

$$\begin{aligned}\textit{upload}(f \; - \; \textit{File}) := \\ \langle \neg \textit{asserted-concept}(f), \\ \textit{asserted-concept}(f)\rangle\end{aligned}$$

The *upload* action takes a file object $f$ as parameter, and requires it to not yet be asserted (the file does not yet exist in the archive). The effect then sets the predicate *asserted-concept(f)* to true, indicating that the individual representing the file has been added to $\mathcal{N}_O$, the concept assertion $f : \textit{File}$ has been made to $\mathcal{A}$-Box, and the file has been uploaded. As the internal proceedings of the external application are of no interest to the current

workflow, the upload is not modeled by the action. It is assumed, that the web application providing the upload service implements its own process for uploading and storing the file.

$$\begin{aligned}
&\textit{extract-data}(f - \textit{File}, k - \textit{Keyword}, t - \textit{Title}) := \\
&\quad \langle \textit{asserted-concept}(f) \wedge \\
&\quad \neg \textit{generated}(k) \wedge \neg \textit{gathered}(k) \wedge \\
&\quad \neg \textit{generated}(t) \wedge \neg \textit{gathered}(t) \wedge \\
&\quad \exists o - \texttt{concept-type} : ( \\
&\qquad \exists r - \textit{hasFile} : \textit{role}(o, r, f) \wedge \\
&\qquad \exists r' - \textit{hasKeyword} : \textit{role}(o, r', k) \wedge \\
&\qquad \exists r''' - \textit{hasTitle} : \textit{role}(o, r''', t)), \\
&\quad (\textit{generated}(k) \wedge \textit{generated}(t)) \vee \top \rangle
\end{aligned}$$

This action requires a file object *f*, a keyword object *k*, and a title object *t*. The file parameter *f* indicates which file the extractor will be working with, and the rest of the parameters are the values that are then set to either selected or generated. The preconditions are, that no data has been acquired yet, and that there exists a corresponding role for setting the new individuals in to relation with another individual. This indicates that the external application has generated a keyword and the title. Note that this action has two possible outcomes, as extracting meta data from files not always succeeds. Also, only one keyword and author is considered in this action, as cardinality is handled by the workflow generator later on in the process.

3. **Application internal actions**

Finally two actions for making roles assertions and concept assertions to the $\mathcal{A}$-Box are required. These correspond to save procedures, storing the gathered data in the underlying database.

$$\begin{aligned}
&\textit{assert-role}(d - \texttt{concept-type}, p - \texttt{role-type}, r - \texttt{concept-type}) := \\
&\quad \langle \neg \textit{asserted-role}(p) \wedge \textit{asserted-concept}(d) \wedge \\
&\quad \textit{asserted-concept}(r) \wedge \textit{role}(d, p, r), \\
&\quad \textit{asserted-role}(p) \rangle
\end{aligned}$$

This action takes a role identified by the triple $(d, p, r)$. The preconditions state that domain and range are reviewed before they can be put into relation to each other. Additionally the *role(p, d, r)* predicate must exist, stating that it is a legal role assertion. As a result the role assertion represented by the triple $\langle d, p, r \rangle$ is made to the $\mathcal{A}$-Box.

$$\begin{aligned}
&\textit{assert-concept}(o - \texttt{concept-type}) := \\
&\quad \langle \neg \textit{asserted-concept}(o) \wedge \textit{reviewed}(o), \textit{asserted-concept}(o) \rangle
\end{aligned}$$

Stating that the individual $o$ has been added to $\mathcal{N}_O$, and that a concept assertion to the $\mathcal{A}$-Box has been made.

**Creating the concrete planning task** After the schematic planning problem has been defied, the planning instance is created, by defining the required objects $O$, the initial state $s_0$, and the goal condition $s_\star$. As mentioned in the beginning of this section, the goal of the resulting workflow is to add a new individual to $\mathcal{N}_O$, along side all "relevant" data. The target concept for which the workflow is generated is referred to as $C_T$ and represents a complex concept in $\mathcal{N}_C$. The concrete planning task is then created as follows:

- A object $o_{C_T}$ of type $t_{C_T}$ is added to the set of planning object $O$. The predicate *asserted-concept*$(o_{C_T})$ is set to $false$ in the initial state $s_0$ and is required to be $true$ in the goal condition $s_\star$.
- For each relevant primitive concept $C \in \mathcal{N}_C$ an object $o$ of type $t_C$ representing this new individual is added to $O$. Additionally, the predicate *asserted-concept*$(o)$ is set to $false$ in $s_0$ and to $true$ in the goal condition $s_\star$.
- For each relevant complex concept $C$ an object $o$ of type $t_C$ representing this individual is added to $O$, and the predicates *asserted-concept*$(o)$ and *selectable*$(o)$ are set to true in $s_0$.
- For each relevant role $R \in \mathcal{N}_R$ an object $p$ of type $t_R$ representing the role assertion is added to $O$. Additionally, for each relevant role $R \in \mathcal{N}_R$ with *domain(R)=C* and *range(R)* $= C'$ and the objects $o$ of type $t_C$ and $o'$ of type $t_{C'}$ (already in $O$) the predicate *role*$(d, p, r)$ is set to $true$ and *asserted-role*$(o)$ is set to $false$ in $s_0$ and *asserted-role*$(o)$ is required to be $true$ in $s_\star$.

**From policy to workflow.** For the above planning instance, a policy $\pi$ is generated. This policy describes when which action is executed. From this policy $\pi$ a workflow is generated, adding cardinality restrictions from the ontology to the role assertions. Further more, the concrete objects introduced by the planning task are removed and replaced by free variables, which are then concretized during workflow execution.

**Definition 59 (Workflow).** A workflow is a policy $\pi : 2^{\mathcal{V}} \to \mathfrak{A}$ from state over variables $\mathcal{V}$ to activities $\mathfrak{A}$. These activities are functions $\mathfrak{a} : 2^{\mathcal{V}} \to 2^{\mathcal{V}}$ modifying the states over $\mathcal{V}$.

To create the workflow, first a directed acyclic graph $\mathcal{G}$ representation of the policy $\pi$ is created. In this graph each node $n$ corresponds to a planning state

and the outgoing edges correspond to the possible outcomes of the action $a = \pi(s)$ applied in state $s$ according to $\pi$. From this graph, a workflow is generated by traversing the graph, and creating an activity $\mathfrak{a}$ for every node $n$, and an entry in the transition system $\Delta$ for every outgoing edge from $n$.

Depending on the action $a = \pi(s(n))$ a new activity is created:

- $a =$ *gather*$(o - t_C)$, $a =$ *select*$(o - t_C)$: A new activity is created, that implements gathering information from the user (distinguishing between selection from the set of existing individuals and the input of new values). In the planning task $o$ is a placeholder for an individual of type $t_c$ stemming from a role $r$ where the target concept $C_T$ was either in the domain or range. From this role, the cardinality determines the required amount of $o$. Hereby, the new values are associated with the object $o$ from the planning task, as to be able to identify them in later steps of the workflow. This activity is therefore responsible for gathering the appropriate amount of objects of type $t_c$ from the user.

- $a =$ *review*$(o - t_C)$, $a =$ *review-all*$()$: A new activity is created reviewing either all already entered information, or the information associated with the planning object $o$, acquired in a previous activity.

- $a =$ *upload*$(f -$*File*$)$ or other web service calls: A new activity is created, which is able to call external services and stores the results. Similar to the activities corresponding to *gather* or select actions, the cardinality of the underlying role $R$ is used to state how many objects the service call provides. Note that there currently exists no means of knowing how many individuals a certain service all will produce. Therefore, the activity executes the action (e.G. file upload). If the result fulfills the cardinality restriction, the next activity can be performed, otherwise the same action is repeated (second file upload).

- $a =$ *assert-concept*$(o - \texttt{concept-type})$, *assert-role*$(o - \texttt{concept-type}, r - \texttt{role-type}, o' - \texttt{concept-type})$: A new activity is created, which performs the actual $\mathcal{A}$-Box assertion and inserts the new individual to $\mathcal{N}_O$. In the case of *assert-concept*$(o - \texttt{concept-type})$, the individuals created and associated with $o$ in the previous activities are added to $\mathcal{N}_O$ and corresponding concept assertion are made to $\mathcal{A}$-Box. Similar in the case of *assert-role*$(o - \texttt{concept-type}, r - \texttt{role-type}, o' - \texttt{concept-type})$ an $\mathcal{A}$-Box assertion is created for all individuals from previous activities, associated with $o$ and $o'$.

The workflow then maps states to activities, where the activities modify the state variables, and the workflow policy is taken from the original policy $\pi$ from the planning task. However, the original policy $\pi$ does not consider cardinality restrictions. Therefore, the workflow policy is adapted as to be

able to distinguish states that fulfill the cardinality restrictions, and those that do not.

For example, if a state $s(n)$ has two successor states $s(n_1)$ and $s(n_2)$ and these are distinguishable only by the facts *gathered(o)*$=true$ and *gathered(o)*$=false$, and the ontology requires $N$ individuals of $o$, the policy would state:

$$\langle s(n) \wedge \text{amount}(o) = N \rightarrow \mathfrak{a}(n_1)$$
$$\langle s(n) \wedge \text{amount}(o) \neq N\rangle \rightarrow \mathfrak{a}(n_2)$$

where *gathered*$(o) = true$ is translated in to the right amount of objects $o$ be gathered.

This process results in a set of activities $A$ connected to their successor activities. To streamline the resulting workflow, activities with the same types of actions can be merged in to one activity. Hereby, all activities that aggregate information such as *gather* and *select* may be merged. Also, all *assert* activities may be combined into a single activity. Note that every activity may only consist of a single action that has multiple possible outcomes, to ensure correct branching.

The resulting workflow collects all relevant data for a given concept $C_T$ in the amount required by the ontology, and then creates all necessary entries in $\mathcal{N}_O$ and makes all the required $\mathcal{A}$-Box assertions.

**Example 33.** Returning to the motivation of this work, generating a workflow for adding a new *Article* to a digital preservation system, is now described using the definitions from Example 25. From this the planning task is generated as described above. The resulting policy is shown in Figure 7.7 and in Figure7.8 where states where replaced by the actions to be executed in those states, together with the streamlined workflow, with dashed lines indicating how the original actions where merged in to workflow activities.

**Usability Improvements using Soft Trajectory Constraints** As can be seen in Figure 7.8 the resulting workflow, being correct, does not necessarily reflect a user expectation towards the order of steps. It may be more intuitive, if the process starts by selecting the file *gather(filename)* followed by the *upload(filename1,file1)*. Improvements, concerning usability can be achieved by adding soft trajectory constraint to the planning task such as *within 3 asserted(file)* stating that the file $file$ is asserted within the first three states ($s_1$: filename is not selected, $a_1$: select filename, $s_2$: filename is selected, $a_2$: upload file, $s_3$: file is asserted). A second major usability issue concerns user feedback. Whenever a system takes automated actions, the user should be provided with some kind of visual feedback. In the case of the automatic meta data extraction action *extract-data*, this is realized by

reviewing the generated data. In Example 33 this happens directly after the extraction action is performed. However, in more complex settings, this is not the case. Especially, when there is meta data that is not provided by the extraction process. The constraint

*(sometime-within 2 generated(obj) reviewed(obj))*

would ensure, that within 2 states of the object being generated, it is also reviewed, ensuring that the user is given feedback and the chance to edit the input as soon as possible. Depending on the actual workflow, the actions taken, and the environment, different constraint may be defined, addressing different usability issues.

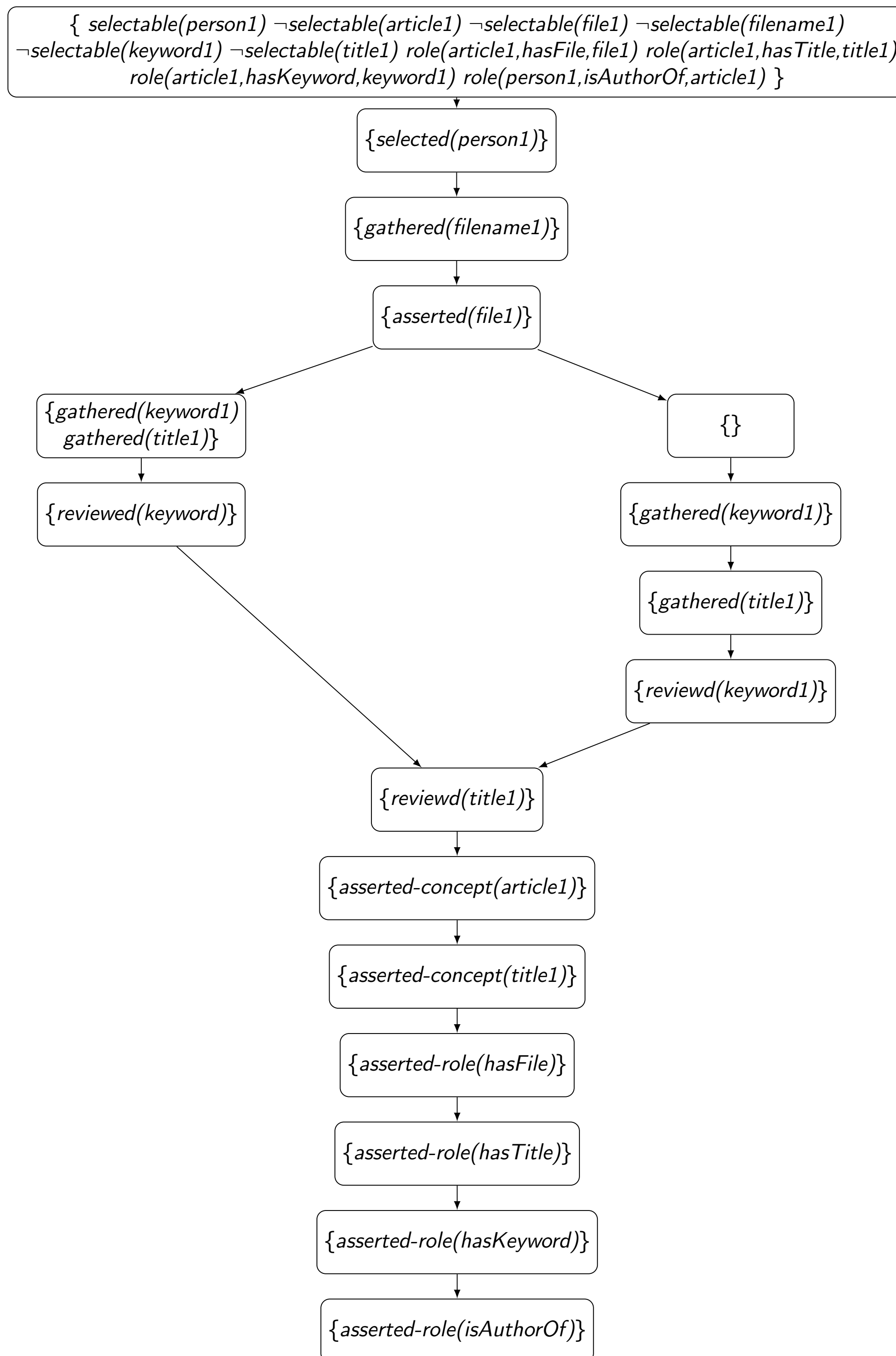



Figure 7.7: Policy of the planning task from Example 33 with only the differences in the states represented.

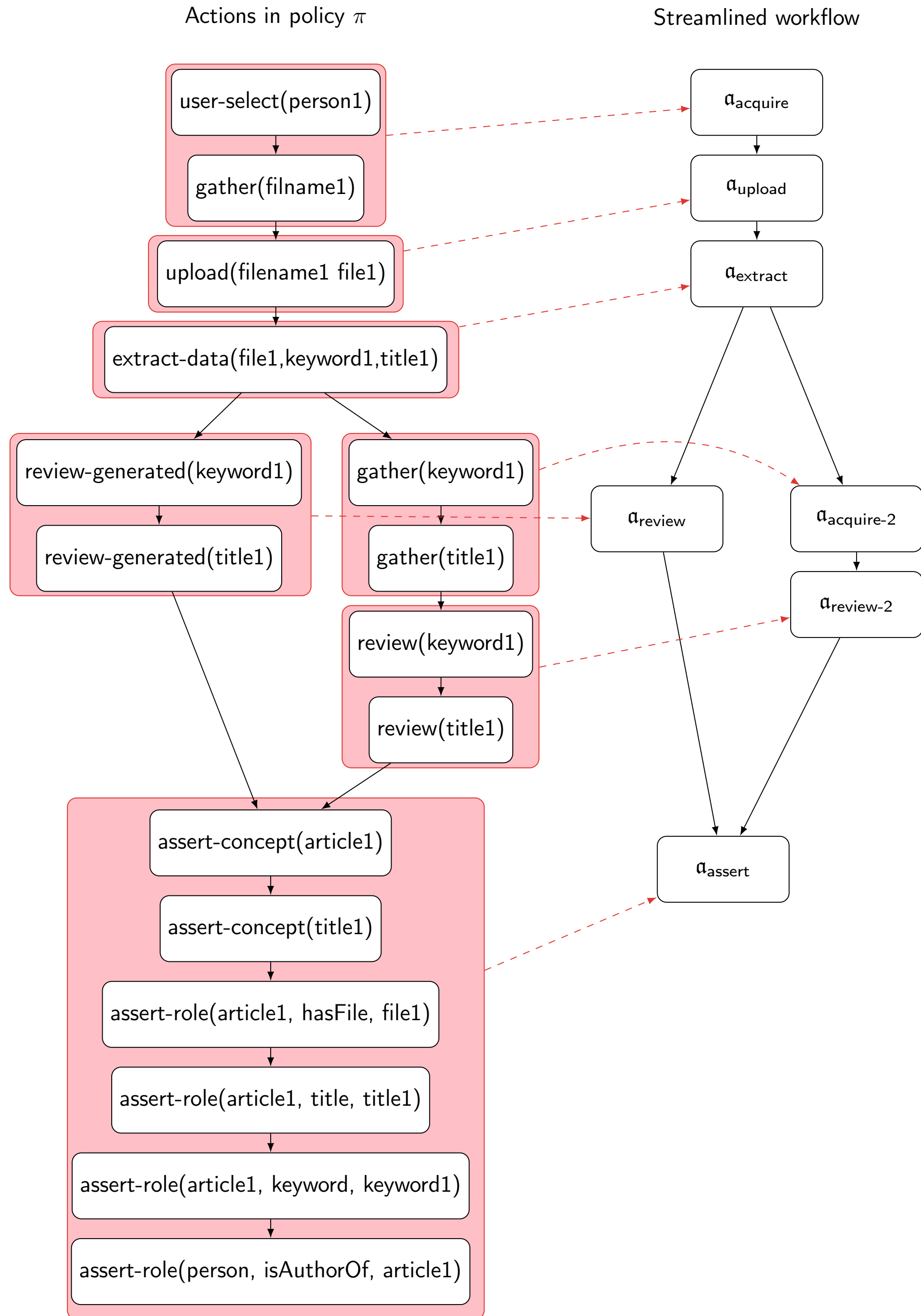



Figure 7.8: Converting a policy $\pi$ to a workflow

# Chapter 8

# Future Work

Here, now a short introduction to open questions, that arose during the work on the presented topics, is given.

**Variable interdependence:** As shown in Chapter 4.2 and by Frances and H. Geffner (2015), it is not uncommon to have variables that depend on each others values. some of these dependencies could be indirect, such as a constraint $a > b$ stating that the value of variable $a$ must always be strictly larger than that of $b$, or more direct such as $a = b + 1$. Currently work in the field variable interdependence is limited. However, a deeper understanding of how variables interact with each others could dramatically improve the efficiency of not only the search but also the grounding process, as unreachable facts and states could be pruned before the actual search is started.

**Trajectory constraints:** Soft and hard trajectory constraints have been analyzed in the past and are well understood (Wright et al., 2018b; Edelkamp, 2006; Keyder and H. Geffner, 2009; J. A. Baier et al., 2009; Torres and J. A. Baier, 2015; to name only a few). However, from a modelers perspective, it is sometimes easier to define such constraints in the form of action trajectory constraints (e.g. stating that an action $a$ is to be performed before action $b$). This has already been mentioned in the PDDL 3.0 definition (Gerevini and Long, 2005). However, as of the authors knowledge, no further work has been done on this topic. One rather simple way of dealing with these action trajectory constraints is to compile them to state trajectory constraints. This could be achieved by adding a predicate *executed-action-a* to the planning task, which is only set to true by action $a$.

Alternatively to compiling soft trajectory constraints in to the planning task

as described in Chapter 5, it is often beneficial to the support constraints directly in the planner. This reduces the overhead introduced by conditional effects, state-dependent action costs or other means of tracking the constraints and guiding the search. For this however, soft trajectory constraint aware heuristics are required. Such heuristics would have to add penalties to not only heuristic values of states that already violate constraints (without the possibility of recovering) but also such states that lead towards non accepting states. For this, similar to the FOND setting, heuristic functions need to be defined not only over states, but over the whole state trajectory.

**Fully observable non-deterministic planning** As discussed in Section 3.3 heuristics guiding the search towards fulfilling soft trajectory constraints are essential for solving tasks with these constraints. A initial analysis was provided. However, the two heuristic approximations (relaxed reachability and automata reachability) are relatively uniformed, as they only provide penalty, if there is no way for the evaluated state to fulfill a given constraint. A more informed heuristic would be based on the expected penalty for a given node. Thus, also adding a penalty to the heuristic value of a node if there exists a path that does not fulfill the constraint. Additionally, nodes that already fulfill soft trajectory constraints such that they can not be violated later should be favored in the search.

# Chapter 9

# Conclusion

In this theses a generalized theory of EVMDDs on monoids was introduced, providing flexible ways of modeling decision diagrams over different types. This extends the previous work on EVMDDs where they were only defined over arithmetic expressions (Ciardo and Siminiceanu, 2002). The correctness of the construction and the theory was then proven, when the underlying monoids $M = \langle T, \bullet, \mathrm{e} \rangle$ are meet-semilattice, ordered, commutative, and there exists a monus operator $\dot{-}$on the underlying set $T$. These EVMDDs where then utilized, representing conditional effects, and state-dependent action costs in planning tasks. For both, conditional effects, and state-dependent action costs, a compilation scheme was presented, based on the EVMDD representation. Following an introductory example, illustrating the problems when dealing with conditional effects and state-dependent action costs separately, a theory was presented, dealing with these conjointly. Also, a combined compilation based on the EVMDD representation was presented. The planning formalism was then extended to soft trajectory constraints, providing two compilations, for incorporating these constraints into a classical planning task with conditional effects and state-dependent action costs. Then soft trajectory constraints are introduced in the dully observable non-deterministic planning setting. A method of tracking these constraints within the planning task is introduced, followed by an analysis of possible heuristic functions, guiding the search towards fulfilling these constraints.

Finally, a digital preservation application is presented. This application uses ontology based data access to manage the data in a meaningful way. Additionally, a method based on a formal data definition, using ontologies, and FOND planning is introduced, generating workflows for the different data types supported by the archive. As these workflows are on the one hand correct and lead to correct ingestion of new files, they also break the users expectation of how files should be added. Therefore, soft trajectory

constraints are added to improve usability of the workflows.

# List of Figures


2.1 Visual representation of the quasi-reduced EVMDDs over the monoid $M = \langle \mathbb{N}, +, 0 \rangle$ representing the arithmetic function $a^2 + 2bc + 8$, and variable orderings (top to bottom) $\langle a, b, c \rangle$, and $\langle c, b, a \rangle$. . . . . . 9
2.2 The AST over the expression $a^2 + 2bc + 8$ . . . . . 14
2.3 Visual representation of the EVMDDs created by Algorithm 3 and Algorithm 4. . . . . . 14
2.4 $\mathcal{E}_b$ and $\mathcal{E}_c$ aligned where the inputs are $\mathcal{E}_f = \mathcal{E}_b$ and $\mathcal{E}_g = \mathcal{E}_c$ . 15
2.5 Alignment for the last recursive call of *APPLY* with $\mathcal{E}_b$ and $\mathcal{E}_c$ and the operator $\times$ on the $0$ edge of $\mathcal{E}_b$ . . . . . 15
2.7 Evaluation of the EVMDD over state $s = \{a = 1, b = 1, c = 1\}$ 16

3.1 Logistics domain example with three cities one truck and two packages. . . . . . 28
3.2 Search graph: Irrelevant paths disregarded . . . . . 28
3.3 Relaxed planning task. After applying *move(Freiburg, Vienna)* the *Truck* now is in two locations simultaneous. . . . . . 31
3.4 Different types of task abstractions . . . . . 32
3.5 The EVMDD representing $x^2 + 2yz + 8$ with domains ($\mathcal{D}_x = 2, \mathcal{D}_y = 2, \mathcal{D}_z = 3$) . . . . . 39
3.6 SDAC task transformation . . . . . 41
3.7 Blocksworld example . . . . . 45

4.1 Agent moving on a grid layout trying to reach the goal destination. The agents location is indicated in red, whereas the goal location is indicated in green. . . . . . 49
4.2 The EVMDD representing the conditional effects from Example 4.1 . . . . . 50
4.3 The abstract syntax tree for conditional effects. . . . . . 50
4.4 Transition diagram of first applying action and transforming the results to the compiled state space, or first transforming the state in to the compiled state space and applying the compiled action sequence. . . . . . 57

4.5 Climbing example. Initial position bottom left, goal position top right. Darker shades indicate higher costs to move. . . . 61
4.6 Shannon reduced conditional effect EVMDD for the *move-right* action. . . . . . . . . . . . . . . . . . . . . . . . . . . 63
4.7 Cost function EVMDD for the *move-right* action. . . . . . . . . 64
4.8 Combination of cost and effect EVMDD for the *move-right* action. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64
4.9 Results for SDAC only compilation vs SDAC with CE compilation and $h^{\max}$ heuristic . . . . . . . . . . . . . . . . . . . . . 71
4.10 Results for SDAC only compilation vs SDAC with CE compilation and $h^{\text{add}}$ heuristic . . . . . . . . . . . . . . . . . . . . . 72
4.11 Results for SDAC only compilation vs SDAC with CE compilation and $h^{\text{ff}}$ heuristic . . . . . . . . . . . . . . . . . . . . . . 73

5.1 Parametrized NFA . . . . . . . . . . . . . . . . . . . . . . . . . 79
5.2 DFA of the *sometime-before* constraint . . . . . . . . . . . . . . 81
5.3 EVMDD compilation of the *penalize* action and the Keyder and H. Geffner (2009) *collect*,*forgo* and *end* actions annotated in red . . . . . . . . . . . . . . . . . . . . . . . . . . 85
5.4 Node expansions for goal action compilation and general action compilation with preserving costs . . . . . . . . . . . . . 91
5.5 Results for the pathways domain with satisficing planing . . . 95
5.6 Results for the rovers domain with satisficing planing . . . . . 96
5.7 Results for the storage domain with satisficing planing . . . . 97
5.8 Results for the trucks domain with satisficing planing . . . . . 98

6.1 Open stacks domain maximal penalty vs. gathered penalty . . 107
6.2 Pathways domain maximal penalty vs. gathered penalty . . . 108
6.3 Rovers domain maximal penalty vs. gathered penalty . . . . . 108
6.4 Storage domain maximal penalty vs. gathered penalty . . . . 108
6.5 Trucks domain maximal penalty vs. gathered penalty . . . . . 109
6.6 Blocksworld domain maximal penalty vs. gathered penalty . . 109

7.1 OntoRAIS system architecture . . . . . . . . . . . . . . . . . . 112
7.2 Object view in web client . . . . . . . . . . . . . . . . . . . . . 113
7.3 Project view in web client . . . . . . . . . . . . . . . . . . . . . 114
7.4 View of ingest process in web client . . . . . . . . . . . . . . . 114
7.5 View of the desktop client . . . . . . . . . . . . . . . . . . . . . 115
7.6 Overview of the workflow generation architecture . . . . . . . 121
7.7 Policy from Example 33 . . . . . . . . . . . . . . . . . . . . . . 131
7.8 Converting a policy $\pi$ to a workflow . . . . . . . . . . . . . . 132

# List of Tables

5.1 State Transition Costs . . . . . . . . . . . . . . . . . . . . . . 88
5.2 Goal action compilation results for optimal planning . . . . . 90
5.3 General action compilation with metric preserving transition costs results for optimal planning . . . . . . . . . . . . . . . . . 91
5.4 General action compilation with positively shifted transition costs results for optimal planning . . . . . . . . . . . . . . . . . 91
5.5 General action compilation with adapted transition costs results for optimal planning . . . . . . . . . . . . . . . . . . . . . 92
5.6 Coverage of Fast Downward with satisficing planning . . . . . 93
5.7 Coverage of Symple with satisficing planning . . . . . . . . . 93
5.8 Comparison to original IPC-5 . . . . . . . . . . . . . . . . . . 94

6.1 Comparing goal action compilation with MyND implementation on deterministic instances . . . . . . . . . . . . . . . . . . 107

7.1 Database tables for BOOK and hasTitle . . . . . . . . . . . . . 117

# List of Algorithms


1 APPLY algorithm for two EVMDDs . . . . . . . . . . . . . . . . . 12
2 ALIGN algorithm for two EVMDDs with same variable ordering 13
3 Create an EVMDD for a constant . . . . . . . . . . . . . . . . . 13
4 Create an EVMDD for a variable . . . . . . . . . . . . . . . . . 13
5 Evaluate an EVMDD on a given state . . . . . . . . . . . . . . . 17

# Bibliography


Amer, K (1984). “Equationally complete classes of commutative monoids with monus”. In: *Algebra Universalis* 18.1, pp. 129–131.

Baader, Franz, Calvanese, Diego, McGuinness, Deborah, Nardi, Daniele, and Patel-Schneider, Peter F., eds. (2010). *The Description Logic Handbook: Theory, Implementation, and Applications*. Paperback edition. Cambridge University Press.

Bäckström, Christer and Nebel, Bernhard (1995). “Complexity results for SAS+ planning”. In: *Computational Intelligence* 11.4, pp. 625–655.

Bagosi, Timea, Calvanese, Diego, Hardi, Josef, Komla-Ebri, Sarah, Lanti, Davide, Rezk, Martin, Rodriguez-Muro, Mariano, Slusnys, Mindaugas, and Xiao, Guohui (2014). “The ontop framework for ontology based data access”. In: *Chinese Semantic Web and Web Science Conference*. Springer, pp. 67–77.

Baier, Jorge A., Bacchus, Fahiem, and McIlraith, Sheila (2009). “A heuristic search approach to planning with temporally extended preferences”. In: *Artificial Intelligence Journal (AIJ)* 173.5–6, pp. 593–618.

Baier, Jorge A. and McIlraith, Sheila (2006). “Planning with Temporally Extended Goals Using Heuristic Search”. In: *Proceedings of the 16th International Conference on Automated Planning and Scheduling (ICAPS06)*.

Baier, Jorge, Hussell, Jeremy, Bacchus, Fahiem, and McIlraith, Sheila (2006). “Planning with temporally extended preferences by heuristic search”. In: *ICAPS 2006*, p. 20.

Bellman, Richard E and Dreyfus, Stuart E (2015). *Applied dynamic programming*. Vol. 2050. Princeton university press.

Bertsekas, Dimitri P, Bertsekas, Dimitri P, Bertsekas, Dimitri P, and Bertsekas, Dimitri P (2005). *Dynamic programming and optimal control*. Vol. 1. 3. Athena scientific Belmont, MA.

Bonet, Blai and Geffner, Hector (2001). “Planning as heuristic search”. In: *Artificial Intelligence* 129.1-2, pp. 5–33.

Büchi, J Richard (1990). “On a decision method in restricted second order arithmetic”. In: *The Collected Works of J. Richard Büchi*. Springer, pp. 425–435.

Bylander, Tom (1994). "The computational complexity of propositional STRIPS planning". In: *Artificial Intelligence* 69.1-2, pp. 165–204.

Camacho, Alberto, Triantafillou, Eleni, Muise, Christian J., Baier, Jorge A., and McIlraith, Sheila (2017). "Non-Deterministic Planning with Temporally Extended Goals: LTL over Finite and Infinite Traces". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI)*, pp. 3716–3724.

Ciardo, Gianfranco and Siminiceanu, Radu (2002). "Using edge-valued decision diagrams for symbolic generation of shortest paths". In: *International Conference on Formal Methods in Computer-Aided Design*. Springer, pp. 256–273.

Cimatti, Alessandro, Pistore, Marco, Roveri, Marco, and Traverso, Paolo (2003). "Weak, strong, and strong cyclic planning via symbolic model checking". In: *Artificial Intelligence* 147.1-2, pp. 35–84.

De Giacomo, Giuseppe, De Masellis, Riccardo, and Montali, Marco (2014). "Reasoning on LTL on Finite Traces: Insensitivity to Infiniteness". In: *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, pp. 1027–1033.

De Giacomo, Giuseppe and Rubin, Sasha (2018). "Automata-Theoretic Foundations of FOND Planning for LTLf and LDLf Goals". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pp. 4729–4735.

De Giacomo, Giuseppe and Vardi, Moshe Y (2013). "Linear Temporal Logic and Linear Dynamic Logic on Finite Traces". In: *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 854–860.

Dijkstra, Edsger W (1959). "A note on two problems in connexion with graphs". In: *Numerische mathematik* 1.1, pp. 269–271.

Dimopoulos, Yannis, Gerevini, Alfonso, Haslum, Patrik, and Saetti, Alessandro (2006). "The benchmark domains of the deterministic part of IPC-5". In: *Abstract Booklet of the competing planners of (ICAPS 06)*, pp. 14–19.

Edelkamp, Stefan (2006). "On the Compilation of Plan Constraints and Preferences". In: *Proceedings of the Sixteenth International Conference on International Conference on Automated Planning and Scheduling*. ICAPS 06. AAAI Press, pp. 374–377.

Edelkamp, Stefan and Hoffmann, Jörg (2004). "PDDL2. 2: The Language for the Classical Part of IPC-4—extended abstract—". In: *International Planning Competition*, p. 2.

Edelkamp, Stefan, Jabbar, Shahid, and Naizih, M (2006). "Large-scale optimal PDDL3 planning with MIPS-XXL". In: *Proceedings of the Fifth International Planning Competition*, pp. 28–30.

Fox, Maria and Long, Derek (2003). "PDDL2. 1: An extension to PDDL for expressing temporal planning domains". In: *Journal of artificial intelligence research* 20, pp. 61–124.

Frances, Guillem and Geffner, Hector (2015). “Modeling and Computation in Planning: Better Heuristics from More Expressive Languages”. In: *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling (ICAPS 2015)*.

Geffner, Tomas and Geffner, Hector (2018). “Compact policies for non-deterministic fully observable planning as SAT”. In: *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*, pp. 88–96.

Geißer, Florian (2018a). “Cartesian Heuristics for Planning with State-dependent Action Costs”. PhD. University of Freiburg, Germany.

Geißer, Florian (2018b). “Cartesian Heuristics for Planning with State-dependent Action Costs”. PhD thesis. Albert-Ludwigs-Universität Freiburg.

Geißer, Florian, Keller, Thomas, and Mattmüller, Robert (2015). “Delete Relaxations for Planning with State-Dependent Action Costs”. In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*.

Gerevini, Alfonso and Long, Derek (2005). *Plan constraints and preferences in PDDL3*. Tech. rep. Technical Report 2005-08-07, Department of Electronics for Automation, University of Brescia, Brescia, Italy.

Gerth, Rob, Peled, Doron, Vardi, Moshe Y, and Wolper, Pierre (1995). “Simple on-the-fly automatic verification of linear temporal logic”. In: *Protocol Specification, Testing and Verification XV*. Springer, pp. 3–18.

Hart, Peter E, Nilsson, Nils J, and Raphael, Bertram (1968). “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.

Haslum, Patrik, Botea, Adi, Helmert, Malte, Bonet, Blai, and Koenig, Sven (2007). “Domain-independent construction of pattern database heuristics for cost-optimal planning”. In: *AAAI*. Vol. 7, pp. 1007–1012.

Helmert, Malte (2006). “The fast downward planning system”. In: *Journal of Artificial Intelligence Research* 26, pp. 191–246.

Helmert, Malte (2008). *Understanding planning tasks: domain complexity and heuristic decomposition*. Vol. 4929. Springer.

Helmert, Malte, Haslum, Patrik, Hoffmann, Jörg, and Nissim, Raz (2014). “Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces”. In: *Journal of the ACM (JACM)* 61.3, p. 16.

Hoffmann, Jörg (2005). “Where ’Ignoring Delete Lists’ Works: Local Search Topology in Planning Benchmarks”. In: *Journal of Artificial Intelligence Research* 24, pp. 685–758.

Hoffmann, Jörg and Nebel, Bernhard (2001). “The FF planning system: Fast plan generation through heuristic search”. In: *Journal of Artificial Intelligence Research* 14, pp. 253–302.

Ivankovic, Franc, Haslum, Patrik, Thiébaux, Sylvie, Shivashankar, Vikas, and Nau, Dana S (2014). “Optimal Planning with Global Numerical State Constraints.” In: *ICAPS*.

Keyder, Emil and Geffner, Hector (2009). “Softgoals Can Be Compiled Away”. In: *Journal of Artificial Intelligence Research (JAIR)*, pp. 547–556.

Kissmann, Peter and Edelkamp, Stefan (2009). “Solving Fully-Observable Non-deterministic Planning Problems via Translation into a General Game”. In: *KI 2009: Advances in Artificial Intelligence*. Ed. by Bärbel Mertsching, Marcus Hund, and Zaheer Aziz. Springer Berlin Heidelberg, pp. 1–8.

Kupferschmid, Sebastian, Hoffmann, Jörg, Dierks, Henning, and Behrmann, Gerd (2006). “Adapting an AI planning heuristic for directed model checking”. In: *International SPIN Workshop on Model Checking of Software*. Springer, pp. 35–52.

Lai, Yung-Te, Pedram, M., and Vrudhula, S. B. K. (1996). “Formal verification using edge-valued binary decision diagrams”. In: *IEEE Transactions on Computers* 45.2, pp. 247–255.

Mattmüller, Robert (2013). “Informed Progression Search for Fully Observable Nondeterministic Planning”. PhD thesis. Albert-Ludwigs-Universität Freiburg.

Mattmüller, Robert, Geißer, Florian, Wright, Benedict, and Nebel, Bernhard (2017). “On the Relationship Between State-Dependent Action Costs and Conditional Effects in Planning.” In: *Proceedings of the 9th Workshop on Heuristics and Search for Domain-Independent Planning (HSDIP 2017)*.

Mattmüller, Robert, Geißer, Florian, Wright, Benedict, and Nebel, Bernhard (2018). “On the Relationship Between State-Dependent Action Costs and Conditional Effects in Planning.” In: *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*.

McDermott, Drew, Ghallab, Malik, Howe, Adele, Knoblock, Craig, Ram, Ashwin, Veloso, Manuela, Weld, Daniel, and Wilkins, David (1998). “PDDL-the planning domain definition language”. In:

Muise, Christian J, McIlraith, Sheila, and Beck, J Christopher (2012). “Improved Non-Deterministic Planning by Exploiting State Relevance.” In: *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*.

Nebel, Bernhard (2000). “On the Compilability and Expressive Power of Propositional Planning Formalisms”. In: *Journal of Artificial Intelligence Research (JAIR)* 12, pp. 271–315.

Ng, Andrew Y., Harada, Daishi, and Russell, Stuart J. (1999). “Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping”. In: *Proceedings of the 16th International Conference on Machine Learning (ICML)*, pp. 278–287.

Pistore, Marco. and Vardi, Moshe Y (2007). “The Planning Spectrum - One, Two, Three, Infinity”. In: *Journal of Artificial Intelligence Research* 30, pp. 101–132.

Pnueli, Amir (1977). “The temporal logic of programs”. In: *Foundations of Computer Science, 1977., 18th Annual Symposium on*. IEEE, pp. 46–57.

Rabin, Michael O. and Scott, Dana (1959). “Finite automata and their decision problems”. In: *IBM Journal of Research and Development* 3.2, pp. 114–125.

Rintanen, Jussi (2003). “Expressive equivalence of formalisms for planning with sensing”. In: *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS 2003)*, pp. 185–194.

Roux, Pierre and Siminiceanu, Radu (2010). “Model-Checking with Edge-Valued Decision Diagrams”. In:

Russell, Stuart J and Norvig, Peter (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited.

Sievers, Silvan, Wehrle, Martin, and Helmert, Malte (2014). “Generalized Label Reduction for Merge-and-Shrink Heuristics.” In: *AAAI*, pp. 2358–2366.

Speck, David (2018). “Symbolic Planning with Edge-Valued Multi-Valued Decision Diagrams”. MA thesis. University of Freiburg.

Speck, David, Geißer, Florian, and Mattmüller, Robert (2018a). “Symbolic Planning with Edge-Valued Multi-Valued Decision Diagrams”. In: *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling (ICAPS 2018)*. AAAI Press, pp. 250–258.

Speck, David, Geißer, Florian, and Mattmüller, Robert (2018b). “SYMPLE: Symbolic Planning based on EVMDDs”. In: *Ninth International Planning Competition (IPC-9): planner abstracts*, pp. 82–85.

Tomasic, Anthony (1988). “View update translation via deduction and annotation”. In: *ICDT ’88*. Ed. by Marc Gyssens, Jan Paredaens, and Dirk Van Gucht. Springer Berlin Heidelberg.

Torres, Jorge and Baier, Jorge A. (2015). “Polynomial-Time Reformulations of LTL Temporally Extended Goals into Final-State Goals”. In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1696–1703.

W3C (2012). *OWL 2 Web Ontology Language Document Overview (Second Edition)*. Tech. rep. W3C.

W3C (2013). *SPARQL 1.1 Overview*. Tech. rep. W3C.

Waters, Max, Nebel, Bernhard, Padgham, Lin, and Sardina, Sebastian (2006). “Plan Relaxation via Action Debinding and Deordering”. In: *Proceedings of the Fifth International Planning Competition*, pp. 39–42.

Wright, Benedict, Brunner, Oliver, and Nebel, Bernhard (2018a). “On the Importance of a Research Data Archive”. In: *Eighth AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18)*.

Wright, Benedict and Mattmüller, Robert (2016). “Automated Data Management Workflow Generation with Ontologies and Planning.” In: *Proceedings of the 30th Workshop on Planen/Scheduling und Konfigurieren/Entwerfen (PUK 2016)*.

Wright, Benedict, Mattmüller, Robert, and Nebel, Bernhard (2018b). “Compiling Away Soft Trajectory Constraints in Planning”. In: *Proceedings of*

*the 2018 Conference on Principles of Knowledge Representation and Reasoing (KR 2018)*.

Wright, Benedict, Mattmüller, Robert, and Nebel, Bernhard (2018c). “Compiling Away Soft Trajectory Constraints in Planning”. In: *Proceedings of the 2018 Workshop on Knowledge Engineering for Planning and Scheduling (KEPS 2018)*.