Planning Techniques and the Action Language Golog

Gabriele Röger

Dissertation

Technische Fakultät
Albert-Ludwigs-Universität Freiburg
Germany

PhD advisor and first reviewer:
Prof. Dr. Bernhard Nebel, *University of Freiburg, Germany*

Second reviewer:
Prof. Dr. Carmel Domshlak, *Technion, Israel*

Examination committee:
Prof. Dr. Wolfram Burgard (chair), *University of Freiburg, Germany*
Prof. Dr. Carmel Domshlak, *Technion, Israel*
Prof. Dr. Georg Lausen (co-chair), *University of Freiburg, Germany*
Prof. Dr. Bernhard Nebel, *University of Freiburg, Germany*

Date of disputation:
June 18, 2014

To Niko

Abstract

The action language Golog allows specifying the behavior of autonomous systems with very flexible programs that leave certain aspects open to be resolved by the system. Such open aspects are often planning tasks, where the system needs to find a suitable course of actions to reach a given goal. The first part of this thesis aims to make highly efficient planning systems available to the Golog system as sub-solvers for such tasks. The main barrier is that both systems use different formalisms to represent their knowledge about the world and that the basic action theories underlying Golog are much more expressive than the PDDL fragment most commonly used by planning systems. We therefore identify a maximal fragment of basic action theories that can be translated to PDDL. An empirical evaluation shows that Golog systems can impressively benefit from the integration of a planning system.

The second part of the thesis concentrates on the internals of the planning systems. The dominant approach in automated planning is heuristic search.

For *optimal* planning, this usually means using the A* algorithm with some admissible heuristic. Well-known theoretical analyses suggest that such heuristic search algorithms can obtain better than exponential scaling behavior, provided that the heuristics are accurate enough. We show that for a number of common planning benchmark domains, including ones that admit optimal solution in polynomial time, general search algorithms such as A* must *necessarily* explore an exponential number of search nodes even under the optimistic assumption of *almost perfect* heuristic estimators, whose heuristic error is bounded by a small additive constant. We therefore argue that other enhancements are necessary to further improve the scaling behavior of optimal heuristic planners.

These results do not carry over to *satisficing* planning, where the system does not need to prove the optimality of the solution. One possibility to better guide the search is to develop new, stronger estimators. Alternatively, we can use multiple existing heuristics concurrently to exploit their complementary strengths. We empirically examine several ways of using multiple heuristics in a satisficing best-first search algorithm to compare their performance in terms of coverage, plan quality and runtime.

Zusammenfassung

Mit der Aktionssprache Golog kann man das Verhalten autonomer Systeme sehr flexibel spezifizieren, indem man in Programmen bestimmte Aspekte offen lässt, die dann durch das Golog-System eigenständig gelöst werden. Solche offenen Aspekte sind oft Planungsaufgaben, bei denen das System eine geeignete Aktionsfolge finden muss, um ein gegebenes Ziel zu erreichen. Der erste Teil dieser Arbeit hat zum Ziel, dem Golog-System für solche Aufgaben hochgradig effiziente Planungssysteme als Unterkomponenten verfügbar zu machen. Die Herausforderung liegt dabei darin, dass beide Systeme ihr Wissen über die Welt in unterschiedlichen Formalismen repräsentieren, wobei die Basic Action Theories, die Golog zu Grunde liegen, deutlich ausdrucksstärker sind als das PDDL-Fragment, das den meisten Planungssystemen als Eingabesprache dient. Wir identifizieren daher ein maximales Fragment der Basic Action Theories, das auch in PDDL formuliert werden kann. Eine empirische Evaluation bestätigt, dass Golog-Systeme beeindruckend stark von solch einer Integration eines Planungssystems profitieren.

Der zweite Teil dieser Arbeit konzentriert sich auf die Planungssysteme selbst, insbesondere auf heuristische Suche als vorherrschende Methode in diesem Bereich.

Im optimalen Planen findet üblicherweise der A*-Algorithmus mit einer zulässigen Heuristik Verwendung. Bekannte theoretische Analysen legen nahe, dass solche heuristischen Suchverfahren mit ausreichend präzisen Heuristiken ein besser als exponentielles Skalierungsverhalten erreichen können. Wir zeigen für eine Reihe üblicher Planungsbenchmarkdomänen – unter anderem solche, die in polynomieller Zeit optimal lösbar sind –, dass allgemeine Suchverfahren wie A* zwingend eine exponentielle Zahl von Suchknoten betrachten müssen, selbst unter der optimistischen Annahme fast optimaler Heuristiken, deren Heuristikfehler durch eine kleine additive Konstante begrenzt ist. Wir schließen daraus, dass für eine weitere Verbesserung des Skalierungsverhaltens optimaler heuristischer Planungssysteme zusätzliche, orthogonale Verfahren notwendig sind.

Dieses Ergebnis lässt sich nicht auf Planen ohne Optimalitätsgarantie übertragen, da solche Systeme nicht den aufwändigen Beweis der Optimalität erbringen müssen. In diesem Bereich stellt also die Entwicklung neuer, besserer Schätzfunktionen eine sinnvolle Möglichkeit dar, die Suche besser zu leiten. Alternativ kann man auch mehrere Heuristiken gleichzeitig verwenden, um ihre komplementären Stärken zu nutzen. Wir untersuchen daher empirisch verschiedene Möglichkeiten, mehrere Heuristikfunktionen in einer Bestensuche zu verwenden, um sie bezüglich der Anzahl der gelösten Instanzen, der Planqualität und der Laufzeit zu vergleichen.

Acknowledgments

This thesis was written during my time at the University of Freiburg and it would not have been possible without the broad support I experienced there.

I would like to thank my advisor Bernhard Nebel for giving me all the freedom I wanted but still pushing me into the right direction when necessary. He is a great person in more than one respect.

Special thanks go to Carmel Domshlak for serving as a second reviewer of this thesis and for agreeing to attend my defense remotely. He was also a wonderful host during my research visit in Israel.

My work greatly benefited from the collaboration with Jens Claßen, Gerhard Lakemeyer, and Malte Helmert within the DFG project PLATAS.

I do not want to miss the opportunity to thank my former colleagues Alexander Kleiner, Christian Dornhege, Jan-Georg Smaus, Malte Helmert, Michael Brenner, Patrick Eyerich, Robert Mattmüller, Sebastian Kupferschmid, and Thomas Keller for many scientific and non-scientific discussions that made the Foundations of Artificial Intelligence group in Freiburg such a great place to work. I also very much appreciate the support by the non-scientific members of the group who kept me free of most technical and administrative ties. Uli Jakob did not only keep our computer systems running smoothly, satisfying all special setup wishes. He also backed me up with the local arrangement of ICAPS 2011 and with running the ACAI summer school 2011 – rescuing me several times with technical as well as non-technical assistance. Our secretary Roswitha Hilden was perfect – highly supportive to the members of the group but if necessary resolutely representing our interests towards the rest of the world. I thank both of them for everything they did for me – including the maintenance of the coffee machine.

I also thank my friends Barbara Frank, Jochen Eisinger, Malte Helmert, Marei Hopert, Margret Keuper, and Sebastian Kupferschmid. I spend lots of time with them at the university and they gave me moral support and advice whenever urgently needed. (One of them deserves my special gratitude for his persistent nagging that it is time to submit my thesis.)

Despite all this support, this thesis would not have become real without my family, especially without my husband Niko. Thanks for the wonderful life that we share together.

Contribution and Publications

Most of the results in this thesis evolved in collaboration with my colleagues at the research group on Foundations of Artificial Intelligence at the University of Freiburg and from discussions with my advisor Prof. Dr. Bernhard Nebel. Some results also originated from the collaboration with the RWTH Aachen University, most importantly Jens Claßen and Prof. Dr. Gerhard Lakemeyer, within the joint DFG (German Research Foundation) project PLATAS. Almost all results have already been published before.

The first part of this thesis is based on three main publications. Two of them focus on a deeper theoretical understanding of the contribution of different aspects of planning formalism on their expressive power.

The paper *Expressiveness of ADL and Golog: Functions Make a Difference* (Röger and Nebel, 2007) examines the influence of the usage of situation-independent functions and functional fluents on the expressive power of the planning formalisms under consideration. This paper is mainly my own work with some helpful feedback from my advisor.

The paper *On the Relative Expressiveness of ADL and Golog: The Last Piece in the Puzzle* (Röger et al., 2008) considers the role of the domain closure axiom and the unique names axioms for constants, as well as the influence of a more compact or incomplete specification of the initial situation. It also shows that we can safely weaken a syntactic restriction on the axioms that specify how action applications affect the world state. Some of the proof ideas in this work originate from discussions with Malte Helmert, but all proofs were fully developed by me. I also drafted the paper and co-wrote the final version.

The last paper (Claßen et al., 2008) on the integration of planning and Golog incorporated in this thesis is an empirical study. It shows that the IndiGolog system impressively benefits from an integration with the FF planning system. The paper resulted from a collaboration with the RWTH Aachen University. The benchmark domains and the experiments were developed by Jens Claßen and me. We had an equal contribution in these parts as well as in the actual writing of the paper. For this thesis, I refined some of the benchmark domains and improved the code quality. In addition, I used the Fast Downward planner instead of FF to also allow for a more fair comparison with an optimality guarantee from the planner's side.

The second part of this thesis is on the potential and limitations of heuristic search for planning.

It covers the paper *How Good is Almost Perfect?* (Helmert and Röger, 2008), in which we consider heuristics with a constant absolute error in the context of A* search. Pohl (1977) showed that with such heuristics A* requires only a linear number of node expansions if the search space satisfies certain assumptions. We argue that these assumptions are commonly violated in planning tasks and show theoretically and empirically that A* indeed requires an exponential number of node expansions on many common benchmark domains, even with a very small heuristic error. The algorithm and the implementation for the empirical study were mainly my work, and I contributed half of the theoretical results and the presentation. The paper was awarded with an outstanding paper award at AAAI 2008.

The paper *The More, the Merrier: Combining Heuristic Estimators for Satisficing Planning* (Röger and Helmert, 2010) studies different ways of ex-

ploiting the information of several heuristics within greedy best-first search. It covers not only methods that were already known from existing optimal and satisficing planning systems, but also introduces a new method based on Pareto-optimality. While the combination of heuristic information is a common topic in optimal planning, the insights from there cannot be carried over to satisficing planning. This was the first study comparing the performance of such methods in the satisficing context. I implemented the approaches that were not already available in the Fast Downward Planning system and took care of the experimental evaluation. I also drafted the paper and co-wrote the final version.

Contents

1 Introduction

We live in a dynamic world that can be altered and affected by actions. Consequently, it is a natural question which actions we need to perform to reach our aims. This is the central subject of this thesis:

> *Given a description of the current world situation, a set of actions and a desired world situation, what course of actions do we need to execute to reach a desired situation?*

Such a course of actions is called a *plan* and the general question is the *planning problem*. For a concrete input, i.e., a combination of an initial world situation, a set of actions and a goal situation, we call it a *planning task*. We are mostly interested in the *classical* planning setting, where

- the plan is made by a single central instance, the *planning system*,
- the world is only altered by actions under the control of the central planning system,
- the outcome of the actions is deterministic,
- plans are sequential, i.e., actions cannot be executed concurrently, and
- the initial world situation is fully observable, i.e., there is no uncertainty about the current state.

The agents performing the actions do not need to be human or artificial embodied agents (robots), but they can also be abstract systems (e. g., a computer program booking flights and hotels for a journey), or the plan can be executed by several such agents together.

A simple example for a (toy) planning task is the Rubik's cube: The initial world situation describes the current configuration of the cube, each action corresponds to turning a face, and the goal is that every face contains only one color.

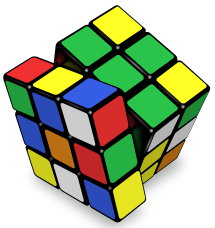


Figure 1.1: Rubik's cube

However, we are not interested in domain-specific planners such as one solving only the Rubik's cube. We rather consider general problem solvers that allow to formalize the dynamics of the domain as part of the input and use only domain-independent techniques for generating a plan.

The hardness of the planning problem depends on the expressivity of the planning formalism, e. g., what input tasks can be expressed at all, how compact their representation is, how concisely the plan can be represented, etc.

The *Planning Domain Definition Language* (PDDL) is the predominant language for specifying planning tasks in the research area of *Automated Planning*, which focuses on the actual development of planning systems. For this reason, PDDL strives for a compromise between the expressive power required for practical applications and limitations that keep a solution of the planning problem still achievable. It defines several levels and fragments that represent different degrees in this field of conflicting priorities. To give an idea of the hardness of the classical planning problem: in the most-widely used PDDL fragment it is EXPSPACE-complete.

In contrast, researchers in the area of *Knowledge Representation and Reasoning* consider a much wider range of *action languages* that are used to describe dynamic worlds. The central question is not to find a plan but research rather concentrates on problems like answering questions about the future evolution of the world (*projection problem*) or updating the representation of a world situation after an action application (*progression problem*). If it comes to actual planning, the predominant approach in this research area is to rely on general solvers for even more powerful formalisms (usually based on logic programming). Since these cannot easily exploit the characteristics that are specific to the planning problem, they usually do not scale very well.

Golog is one specific attempt to make such a highly expressive formalism still usable in practice: it augments the description of a dynamic world (specified in the situation calculus) with a rough sketch of a plan rather than describing only the goal situations. This way, it can significantly prune the search space which must be explored to find a plan. However, Golog systems do not incorporate sophisticated techniques (such as planning-specific heuristics) as they are standard in PDDL planning systems.

In the first part of this thesis we would like to make these advanced techniques available to the Golog systems. Instead of re-implementing the approaches there, we pursue the more sustainable approach that allows the Golog system to translate certain subproblems to PDDL. This way it can "outsource" classical planning problems and always make use of state-of-the-art planning approaches without additional work. The emphasis will lie on the *theoretical study* of the boundaries of this approach, i.e., what fragment of the situation calculus can maximally be compiled to PDDL.

The second part of this thesis concentrates on the actual planning problem and its predominant approach – *heuristic search*. We will first examine the limitations of pure heuristic search in *optimal* planning where a solution must be guaranteed to be a shortest possible plan. Then we study the potential of using multiple heuristics in the *satisficing* setting, where we are still interested in cheap – but not necessarily optimal – plans.

Part I

Integrating Golog and Planning

2
Motivation

Intelligent agents act in dynamic environments with the aim of reaching or maintaining a certain world situation. For planning their actions they crucially require information about the current situation and the dynamics of the world, e. g., how the actions of the agent alter the world state.

This information is typically represented within logic-based formalisms because they provide an unambiguous semantics and one can build on a broad theoretical foundation and on powerful reasoning mechanisms.

In most practically realistic scenarios, the agent does not have full information about the world state but can successively gather more information with its sensors. It is also common that other agents – collaborative, adversarial or neither – act in the same world and change it exogenously from the perspective of the planning agent. In addition, the agent often faces uncertainty about the outcome of the actions, e. g., the movements of a robot can be affected unpredictably by a deep-pile carpet.

All these aspects make the planning problem significantly harder and so far there exist no general problem solvers that address all these challenges and are still sufficiently efficient for practical applications.

This general problem also can be observed for logic-based action formalism as a conflict of expressivity and practical feasibility of reasoning. Such action languages are an important topic in the research areas of planning and knowledge representation. While action languages in both fields share the same origin, they developed in different directions: the planning community focused on an efficient plan generation and therefore was very careful with extending the expressive power of the formalism in which the planning tasks are given. Research in knowledge representation concentrated on expressive languages that can describe a wide range of scenarios but are hard to handle practically.

The situation calculus (McCarthy, 1963; Reiter, 2001) is one of the best-established action formalisms in knowledge representation, where *basic action theories* describe the initial situation and dynamics of the world. Since planning for general basic action theories is prohibitively expensive, the Golog (Levesque et al., 1997; de Giacomo et al., 2000; de Giacomo and Levesque, 1999) family of action languages allows to define a *skeleton* for a plan in a procedural manner. This way, Golog stands in the middle ground of general planning, where the system autonomously decides about the entire course of actions via some kind of

automated reasoning, and deterministically programmed agents that follow an entirely predefined program. The advantage of Golog over fully deterministic programs is that not all possible situations of the world must be covered in detail, so that the system is more adaptive and easier to deploy in altering environments. Such Golog-based systems have successfully been applied for the interactive museum tour-guide robot RHINO (Burgard et al., 1998, 1999) and for the domestic service robot Caesar (Schiffer et al., 2012), but also for travel planning in the area of web service composition (Sohrabi et al., 2009).

However, also in such general scenarios many subproblems arise that satisfy the restrictions of the languages used by classical planning systems, which exploit them for an efficient plan generation. Therefore Golog systems could benefit significantly from the possibility to hand over such subproblems to state-of-the-art planning systems. This is exactly the aim of the first part of this thesis, which can build on first attempts in this direction: Claßen and Lakemeyer (2006) already showed that the semantics of PDDL corresponds to progression in the situation calculus. Eyerich et al. (2006) defined restrictions on basic action theories that mimic the requirements of planning systems on the task specification and showed how such restricted basic action theories can be transformed into the planning language PDDL. A prototype by Claßen et al. (2007) demonstrates how a planner can be invoked by the Golog system, but this is merely a proof of concept where the PDDL domain was hand-crafted.

Our work will focus on the theoretical part of the integration, examining which restrictions defined by Eyerich et al. are really necessary to stay within the expressive power supported by most planning systems. This is not only interesting for the integration of Golog and planning but also improves the understanding which features contribute to the expressivity of a language and which are only syntactic sugar.

We also hope that our results will form a first step towards a deeper understanding of the similarities and differences of the action languages used in planning and in knowledge representation and reasoning. In the last decade we could observe a significant improvement of planning systems that came along with many extensions of PDDL, so that the expressivity of the agent formalisms from both fields appears to slowly converge. Instead of maintaining these almost parallel worlds, a stronger integration could be beneficial for both sides. The planning community could profit from the theoretical foundations on reasoning in expressive formalisms and the knowledge representation and reasoning community would benefit from the experience in practically exploiting specific characteristics of formalisms.

The rest of this first part of the thesis is structured as follows: We will begin with the presentation of related work on action formalisms and frameworks for the comparison of the expressivity of different formalisms. Then we introduce the formalisms under consideration, namely PDDL and the situation calculus with basic action theories and Golog. Afterwards we adapt compilation schemes (Nebel, 2000a), the framework that we use for the comparison of the formalisms, to the demands of these formalisms. An in-depth examination of the requirements of Eyerich et al. (2006) will form the core of this part of the thesis. To demonstrate the benefit of the overall integration, we will conclude with experimental results for a fully automated, integrated system.

3
Related Work

3.1 Action Languages in the Field of Knowledge Representation and Reasoning

The idea of designing intelligent agents whose model of the world and goal specification are given as logic formulas goes back to McCarthy's *advice taker* (McCarthy, 1959). Earlier systems were programmed imperatively and followed a predetermined program instead of inferring good actions on their own. Since the underlying formalism of the advice taker was not very elaborated, four years later McCarthy (1963, reprinted in 1968) introduced the *Situation Calculus* to address the specific needs of logical formalisms representing dynamically changing worlds.

However, his version did not establish itself as a major formalism, possibly because it was lacking a solution to the frame problem (succinctly expressing that everything not changed by an operator[1] application persists), which was only observed some years later (McCarthy and Hayes, 1969). In this thesis we use the version of the situation calculus by Reiter (2001), which he developed in cooperation with Fiora Pirri and Hector Levesque (Levesque et al., 1998; Pirri and Reiter, 1999) and which is predominant nowadays. Reiter's solution of the frame problem for deterministic actions (1991; 2001) is based on two different earlier proposals, one by Pednault (1989), the other proposed independently by Haas (1987) and by Davis (1990). It solves the frame problem by formalizing the dynamics of the domain as a set of *successor state axioms*, which express that a proposition is true if the last action application made it true or if it has been true before this action application and has not been made false.

Another important aspect is the *projection* problem, which asks whether a given formula holds after applying a given sequence of actions: Reiter describes a *regression* method that allows to convert a formula about a future situation (after applying a certain sequence of actions) into a logically equivalent formula about the current situation. Another solution to the projection problem is *progression*, which updates the underlying model over the course of actions, therefore "forgetting" irrelevant aspects from the past and keeping the internal representation of the agent small. Lin and Reiter (1997) showed that

[1] We use the terms *action* and *operator* synonymously in this work.

not all situation calculus theories can be progressed within the formalism and identified subclasses for which such a progression always exists.

Since its introduction, the situation calculus has been extended with numerous features such as continuous processes (Pinto, 1994; Reiter, 1996), indirect effects (Lin, 1995), probabilistic uncertainty (Bacchus et al., 1995), or epistemic features (Moore, 1979; Scherl and Levesque, 1993).

Based on Reiter's version of the situation calculus, Levesque et al. (1997) introduced the language Golog as a possibility to annotate situation calculus theories with a procedural sketch of a program that leaves certain aspects open. The idea is that a Golog interpreter follows the program and makes suitable choices for the open aspects via some kind of reasoning. The semantics of Golog programs is defined via macro expansion to situation calculus formulas.

There are also many different Golog versions that extend the basic language with different practically relevant features: One step towards a practical applicability was *ConGolog* (de Giacomo et al., 2000), which added (interleaved) concurrency, interrupts, and exogenous events. However, as with all earlier variants, a ConGolog interpreter must compute an entire concrete action sequence for the program offline before it starts to perform any action. In a dynamic environment, this is impractical if the precomputation takes very long or if the agent must collect additional information about the state of the world during runtime. These issues gave rise to *IndiGolog* (de Giacomo and Levesque, 1999) which allows an incremental program execution and sensing. IndiGolog is particularly interesting for this thesis because it integrates a special search operator, where a planner can easily be integrated to solve a particular subproblem. Other Golog variants support additional features that are beyond the scope of this thesis, such as sGolog (Lakemeyer, 1999) with sensing, *CCGolog* (Grosskreutz, 2002) with continuous change, and other variants that integrate time (Reiter, 1998), or concurrency (Finzi and Pirri, 2004).

While the situation calculus is well-established, it is by far not the only action language in the field of knowledge representation and reasoning. The most similar ones are probably the *fluent calculus* (Thielscher, 1999) and the $\mathcal{ES}$ formalism (Lakemeyer, 2010).

The fluent calculus uses a different solution for the frame problem than the situation calculus, where there is a *state update axiom* for each action that describes how the action application changes the state. *FLUX* (Thielscher, 2005) is a programming language for the fluent calculus, similar to Golog for the situation calculus. A study of the relationships between the fluent calculus with FLUX on the one side and the situation calculus with Golog on the other side can be found in the work by Schiffel and Thielscher (2006).

The logic $\mathcal{ES}$ is a modal variant of the situation calculus, motivated by disadvantages of its extensions with knowledge and time.

The action languages $\mathcal{A}$ (Gelfond and Lifschitz, 1993), $\mathcal{B}$ and $\mathcal{C}$ (Gelfond and Lifschitz, 1998) describe how actions change the world by means of causal laws. $\mathcal{A}$ is closely related (Gelfond and Lifschitz, 1998) to the propositional fragment of Pednault's ADL (Pednault, 1989), described in the next section on action languages in the field of planning.

The *event calculus* (Kowalski and Sergot, 1986; Shanahan, 1995; Miller and Shanahan, 1999) was originally designed for database updates and narrative understanding. Therefore it is not based on global states like the situation calculus but rather on local events that are associated with time points or time

periods. Another narrative-based language, which also has been used in the field of planning (Doherty and Kvarnström, 2001), is the family of temporal action logics *TAL* (Doherty et al., 1998), inspired by the formalisms of Sandewall (1994).

3.2 Action Languages in Classical Planning

Planning and knowledge representation for dynamic worlds share the same origin. Both fields started to diverge with the robotic project Shakey at the Stanford Research institute. While the original proposal used a situation calculus formalization and theorem-proving to plan the actions (Green, 1969), the inefficiency of this approach led to the introduction of the STRIPS system (Stanford Research Institute Problem Solver) by Fikes and Nilsson (1971). This system uses a rather different language, where the frame problem is not solved within the logical formalism but by an external assumption that everything not affected by an operator stays unchanged. It also makes a shift to an action-centric specification of the dynamics, where an action is given by three components: The *precondition* of an action is a formula that defines when it is applicable, the *add effect* is a list of formulas that must be true after the action application, and the *delete effect* is an analogous list of formulas that must be false after the action application. Fikes and Nilsson (1971) used only a fragment of their entire language specification, which resulted in a much less expressive language but allowed for more efficient planning, based on means-ends analysis. However, the semantics of the full language was given only informally and it was easily possible to specify unsound operators. Only in 1987, Lifschitz (1987) defined a formal semantics and specified when an operator specification is sound. His notion of soundness is always relative to a given set of sentences that are considered to be *essential*. In the simplest case, where the essential sentences are exactly the ground atoms, the definition boils down to a very restrictive formalism, where action effects are restricted to conjunctions of literals. While the planning algorithm of the STRIPS system is now obsolete, this simple action formalism still persists (with the additional requirement that action preconditions are conjunctions of atoms) and is nowadays simply referred to as STRIPS. To allow a more compact representation of planning tasks without going back to the full expressive power of the situation calculus, Pednault (1989) introduced the action description language ADL with conditional effects, quantification, nonlinear functions and open worlds.

However, neither STRIPS nor ADL established themselves as a standard input language, so that almost every planning systems from that time developed their own fragment or variant as task description language (Wilkins, 1988; Carbonell et al., 1992; Erol et al., 1994; McDermott, 1996; Penberthy and Weld, 1992). As a result, these systems were hardly comparable to each other and there were no common benchmark suites. This lack of comparison motivated the introduction of the International Planning Competition (McDermott, 2000) in 1998, for which McDermott et al. (1998) defined PDDL as a new common input language. This first PDDL version subsumed many of the features of the specialized languages, such as conditional and quantified effects, axioms, safety constraints, or hierarchical actions. These were grouped with so-called requirements so that the planners could indicate which fragment they support.

PDDL is nowadays the predominant language for formalizing planning instances. Since its introduction for the first International Planning Competition in 1998 it has undergone many revisions (Fox and Long, 2003; Edelkamp and Hoffmann, 2004; Gerevini and Long, 2005; Helmert et al., 2008) and the language can now be divided into different levels (Fox and Long, 2003; Gerevini and Long, 2005): The classical fragment is referred to as level 1, higher levels add numeric extensions, discretised and continuous durative actions, spontaneous events, physical processes, preferences and plan trajectory constraints. Since we consider classical planning, we use level 1 which can be subdivided further into different fragments. These allow a fine-grained support of certain language features, starting from simple STRIPS up to the ADL fragment, possibly including action costs and derived predicates.

The SAS$^+$ formalism (Bäckström and Nebel, 1995; Jonsson and Bäckström, 1998) is a generalization of grounded STRIPS that uses finite-domain variables instead of only binary variables. It is not commonly used as an input language in planning and an attempt to integrate its flavor in PDDL for the IPC 2008, based on the more general functional STRIPS formalism (Geffner, 2000), petered out due to insufficient interest from the community. However, it is highly relevant for the internal representation of planning systems. For example, the Fast Downward planning system (Helmert, 2006) first translates the PDDL input to a finite-domain representation (Helmert, 2009), which is an extension of SAS$^+$ with conditional effects and axioms.

While PDDL is the standard input language for planning systems, it did not establish itself in the related field of heuristic state-space search. In this area most systems are highly domain-specific so that the dynamics of the domain are hard-coded in the solvers. However, a first attempt for a generic input language has been made with the PSVN formalism (Hernádvölgyi and Holte, 1999), which is conceptually similar to functional STRIPS.

3.3 Comparison of Planning Formalisms

In the literature, we can find several transformations that aim to compare the complexity or expressivity of formalism.

Presumably the best-known transformation for decision problems is *Karp reduction* (Karp, 1972). For two problems $\mathcal{X}$ and $\mathcal{Y}$, problem $\mathcal{X}$ is polynomial time reducible to $\mathcal{Y}$ if every instance x of $\mathcal{X}$ can be transformed to an instance y of $\mathcal{Y}$ in polynomial time (in the size of x), so that $x \in \mathcal{X}$ iff $y \in \mathcal{Y}$. Transferred to planning formalisms and the problem whether a plan exists for a given task, this would mean that formalism $\mathcal{X}$ is reducible to $\mathcal{Y}$ if we could translate each planning task $\Pi_\mathcal{X}$ of $\mathcal{X}$ to a task $\Pi_\mathcal{Y}$ in formalism $\mathcal{Y}$ such that $\Pi_\mathcal{X}$ has a plan iff $\Pi_\mathcal{Y}$ has a plan.

In contrast to Karp-reduction, *exact structure-preserving reduction* (or ESP reduction, Bäckström, 1995) aims to also capture the structure of the plan space. As before, one has to compute a task of the target formalism in polynomial time but in addition it must hold that the number of plans of length at most k must be equal for the original instance and the resulting instance for all $k \geq 0$. Bäckström (1995) applied ESP reductions to show that two variants of STRIPS, propositional TWEAK (Chapman, 1987) and the SAS$^+$ formalism all have equivalent expressivity.

The *compilation scheme* approach by Nebel (2000a) does not postulate a strict structural similarity of the plan space but still requires that shortest plans of the transformed tasks do not grow unreasonably. It distinguishes different levels of compilability, relative to the necessary blow-up of the plan size. In addition, compilation schemes measure how *concise* planning *domains* can be represented in each formalism, independent of the computational effort required to translate the domain specification. Originally, Nebel (2000a) introduced compilation schemes to compare only the expressive power of *propositional* formalisms. However, to examine the influence of axioms on the expressivity of PDDL, Thiébaux et al. (2005) presented a variant that also can be applied to *non-propositional* (schematic) PDDL.

3.4 Integrating Golog and Planning

There are three lines of research in the literature that aim to integrate planning and Golog-style procedural programming.

The first line of research tries to transfer the general idea of Golog programs to planning: Baier et al. (2007, 2008) propose to annotate PDDL planning tasks with procedural domain control knowledge that restricts the search space by specifying a skeleton of the desired plan. The PDDL task together with the domain control knowledge program is then translated to a new PDDL tasks where all plans are also plans for the original task but in addition follow the given program.

A second line of research (Blom and Pearce, 2010; Blom, 2011) directly transfers planning techniques to Golog by re-implementing and extending them for the required wider class of problems. The main disadvantage of this approach is that this implementation incorporates a significant engineering effort and that it needs to be repeated every time a new planning technique should be integrated into the Golog system.

Our work contributes to the third line of research that tries to make planning systems available for Golog interpreters. The IndiGolog implementation already provides a mechanism to call an iterative deepening search for determining a sequence of actions that achieves a given subgoal. Claßen and Lakemeyer (2006) suggested to call an efficient PDDL planner instead. For this purpose, it is necessary to relate the situation calculus with PDDL and to automate the translation from situation calculus theories to PDDL tasks and the re-translation of the resulting plans. To provide a theoretical basis for the translation, Claßen et al. (2006; 2007) showed that the semantics of the ADL fragment of PDDL corresponds to progression in the situation calculus (resp. $\mathcal{ES}$). Claßen et al. (2007) also presented a situation-calculus semantics for the more expressive temporal level of PDDL. The work on the semantics led by Claßen was complemented by the work by Eyerich et al. (2006), who defined restrictions on basic action theories that lead to the same expressivity as the ADL fragment of PDDL. The compilation schemes they used for the comparison define an efficient translation from the defined subset to PDDL. However, there was no implementation of this translation but only a very small proof of concept with a manually translated domain (Claßen et al., 2007).

4

Two Formalisms for Dynamically Changing Worlds

In this work we use the notion *planning formalism* in a rather general meaning, covering many different logic-based formalism for dynamically changing worlds, which all share certain characteristics:

An instance of a planning task always comprises a *domain definition*, an *initial world specification*, and a *goal specification*. The domain definition fixes the vocabulary and the dynamics of the domain. These dynamics specify what actions are applicable in a certain world state and how they affect the world. The initial world specification gives (not necessarily complete) information about the initial situation of the world. It can also introduce additional constants describing objects in the world and specify certain restrictions on the objects, e. g., restrain the objects in the world to the given constants (domain closure). The goal specification describes the desired situations of the world. The aim is to find a course of actions (a *plan*) that transforms the initial situation to the desired world situation.

While our notion of planning formalisms is quite general it is by far not the most general possible definition:

- We only consider *non-adversarial* settings with a single agent choosing the actions.
- Our setting is *static*, i.e., there are no exogenous events and all changes of the world are the result of action applications[1].
- We assume that actions are deterministic, i.e., their is no uncertainty about how their application affects the world.
- We will only consider *sequential* action applications, i.e., actions cannot be executed in parallel as for example in partial-order (graphplan-style) planning (Sacerdoti, 1975; Blum and Furst, 1997) or temporal planning (Fox and Long, 2003).

[1] In our experimental evaluation we will show that the approach is also useful for non-static scenarios with sensing.

- We do not consider *sensing*, so the agent cannot gain additional information about the current world state[1].
- Due to the previous points we do not need to consider *conditional plans* where the actually executed actions depend on information that is obtained during the plan execution. Therefore we can restrict ourselves to *sequential plans*, which consist of a sequence of actions.
- We do not consider action costs or advanced plan metrics. If we want to compare the quality of plans we will always do this based on their length (= number of actions). The main reason for this decision is that action costs are not present in Golog and when this research was done in 2007 and 2008, action costs also were not widely supported by PDDL planning systems. Back then, they were a special case of a more expressive PDDL level, requiring general support of numeric fluents. This changed only with the International Planning Competition in 2011 when action costs were re-introduced as a restricted fragment of numeric fluents and were used in the competition benchmark tasks.

In the following we will introduce the two formalisms that we need to consider for integrating Golog and planning. The first is a standard formalism used in the planning community, the second the situation calculus, which is the basis of Golog.

4.1 Classical Planning and PDDL

Classical sequential planning fits our setting for general planning formalisms as described above. In addition, we have full information about all objects in the world and the initial situation, which therefore defines a single *initial state*. Since action application is deterministic, this implies that the classical planning setting provides full observability.

Before we go into the details of the corresponding PDDL planning formalism, we first give some intuition for classical planning with the help of an illustrative example: In the *elevator domain* (inspired by the Miconic-10 domain of the 2nd International Planning Competition in 2000 and not to be confused with the elevators domain of the International Planning Competition 2008) there are persons on several floors of a building who want to use the elevator to go to other floors.

Figure 4.1 shows the initial state and the goal of a simple elevators task with four floors and four passengers initially located at three different floors. At the beginning, the elevator is at the lowermost floor. The aim is to transport the passengers using the elevator, which may *move* between adjacent floors and *stop* to board all waiting passengers and unboard all passengers that have the current floor as destination. Hence, the dynamics of the domain are described by the *move* and the *stop* actions.

A plan for this task is a sequence of such actions so that at the end the passengers are at their goal destinations. The example illustrates that there is not necessarily a single goal state that perfectly describes the goal but that the goal is rather specified by a *goal condition* that can capture several states: all situations with the passengers being at their destinations are goal situations, no matter where the elevator is located.

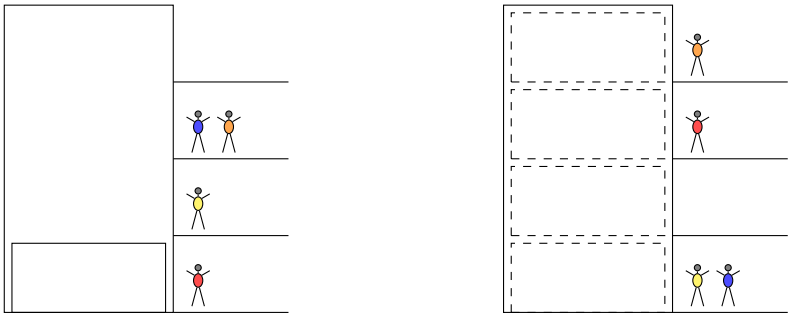


Figure 4.1: An elevators task

We roughly consider PDDL level 1 for our integration of planning and Golog. However, we do not consider action costs, which were introduced with the International Planning Competition in 2008, because they only provide a metric for the plan quality different from minimizing the number of actions in a plan and there is no analogous concept in Golog. We also do not consider derived predicates, because this feature is not widely supported by planning systems.[2]

Before we go into the details of the PDDL fragment under consideration we first want to give a first impression of PDDL by means of the example elevators task from Figure 4.1. PDDL separates the *domain*, describing the dynamics of the world, from the actual problem definition, which defines the concrete initial state and goal. Figures 4.2 and 4.3 show these two parts for our example, which together form the complete specification of the task.

The domain description mainly defines the language and the operators. In the example, the language consists only of predicates but in general it could also introduce constants that exist in all tasks of the domain. The operators are specified in a schematic manner with parameters that need to be replaced with constants to get the actual actions that alter the world state. Each operator has a precondition defining when the operator can be applied and an effect describing how the operator changes the world state. The problem description can extend the domain language by (additional) constants and specifies the initial state and the goal. In PDDL there is always a single initial state, given as the list of ground atoms, which are initially true. By the semantics of PDDL all other ground atoms are initially false. The goal is given by a logical formula, which can be true in several world states.

In this thesis, we will not work with the LISP-style syntax of PDDL (which has been designed to be easily parsable) but use an (equivalent) mathematical notation. Our definition is based on the one by Helmert (2008) but separates the domain from the rest of the task definition and does not include derived predicates.

Definition 1 (PDDL domain)**.** *A* PDDL domain *is given by a pair* $\mathcal{D} = \langle \mathcal{L}, \mathcal{O} \rangle$

[2]Of the 16 heuristics currently implemented in the Fast Downward planning system (Helmert, 2006), none properly considers derived predicates in the heuristic computation and seven cannot be used at all in the presence of such predicates.

```
(define (domain elevators)
  (:requirements :adl)

  (:predicates
    (origin ?person ?floor)
    (destination ?person ?floor)
    (adjacent ?floor1 ?floor2)
    (boarded ?person)
    (served ?person)
    (lift-at ?floor)
    (floor ?floor)
  )

  (:action stop
   :parameters (?floor)
   :precondition (lift-at ?floor)
   :effect (and
               (forall (?person)
                  (when (and (boarded ?person)
                             (destination ?person ?floor))
                        (and (not (boarded ?person))
                             (served  ?person))))
               (forall (?person)
                   (when (and (origin ?person ?floor)
                              (not (served ?person)))
                         (boarded ?person)))))

  (:action move
    :parameters (?f1 ?f2)
    :precondition (and (lift-at ?f1) (adjacent ?f1 ?f2))
    :effect (and (lift-at ?f2) (not (lift-at ?f1))))
)
```

Figure 4.2: PDDL domain specification of the elevators task from Figure 4.1. The definition of the action `stop` is the same as in the simple-ADL version of the IPC 2000 Miconic domain, but we restricted the movements of the elevator to adjacent floors to keep the problem specification in Figure 4.3 short.

with the following components:

- $\mathcal{L}$ *is a first-order language, whose alphabet consists of*
  - *finitely many constant symbols and relation symbols,*
  - *infinitely many variable symbols,*
  - *logical symbols* $\exists, \wedge, \neg, \rightarrow$ *and parentheses, brackets, and punctuation symbols, and*
  - *the equality symbol* $=$*.*
- $\mathcal{O}$ *is a set of* schematic operators *over* $\mathcal{L}$*. A schematic operator* $\langle n, \chi, e \rangle$ *consists of an operator name* n*, a first-order formula* χ *over* $\mathcal{L}$ *called*

```
(define (problem elevator-example)
 (:domain elevators)
 (:objects p0 p1 p2 p3 f0 f1 f2 f3)
 (:init
   (lift-at f0)
   (floor f0) (floor f1) (floor f2) (floor f3)
   (origin p0 f0) (destination p0 f2)
   (origin p1 f1) (destination p1 f0)
   (origin p2 f2) (destination p2 f0)
   (origin p3 f2) (destination p3 f3)
   (adjacent f0 f1) (adjacent f1 f0)
   (adjacent f1 f2) (adjacent f2 f1)
   (adjacent f2 f3) (adjacent f3 f2)
 )
 (:goal
   (forall (?x) (or (floor ?x) (served ?x)))
 )
)
```

Figure 4.3: PDDL problem specification of the elevators task from Figure 4.1

its precondition *and its* effect e. *Effects are recursively defined by finite application of the following rules:*

- *A literal over* $\mathcal{L}$ not *mentioning the equality symbol* $=$ *is an effect called* simple effect.
- *If* $e_1, \ldots, e_n$ *are effects, then* $e_1 \wedge \cdots \wedge e_n$ *is an effect, called* conjunctive effect.
- *If* φ *is a first-order formula over* $\mathcal{L}$ *and* e *is an effect, then* $\varphi \triangleright e$ *is an effect, called a* conditional effect.
- *If* $v_1, \ldots, v_k$ *are different variable symbols in* $\mathcal{L}$ *and* e *is an effect, then* $\forall v_1 \ldots v_k e$ *is an effect called* universal effect.

Free variables of an effect are defined recursively as in first-order logic, where the set of free variables of a conditional effect is defined as $\mathit{free}(\varphi \triangleright e) = \mathit{free}(\varphi) \cup \mathit{free}(e)$.

Also analogous to first-order logic, we define $e[x_1/t_1, \ldots, x_n/t_n]$ *as the result of substituting term* t_i *for each free occurrence of variable* x_i *in* e.

The free variables of a schematic operator are defined as $\mathit{free}(\langle n, \chi, e\rangle) = \mathit{free}(\chi) \cup \mathit{free}(e)$. *Free variables are also referred to as the* parameters *of the schematic operator.*

Such a PDDL domain describes the general setting of a world but leaves the concrete task open. For the actual task definition we need in addition the initial state and goal specification and possibly a set of additional task-specific constants:

Definition 2 (PDDL task)**.** *A* PDDL task *is given by a tuple* $\Pi = \langle \mathcal{D}, C, I, \gamma\rangle$ *where*

- $\mathcal{D} = \langle \mathcal{L}, \mathcal{O} \rangle$ *is a PDDL domain.*
- C *is a finite set of* constants. *We define the* language $\mathcal{L}_\Pi$ of task Π *as the language we receive by extending the set of constants in the alphabet of the domain language* $\mathcal{L}$ *with the constants in* C.
- I *is a set of ground atoms of* $\mathcal{L}_\Pi$ *(not mentioning the equality symbol* $=$*) called the* initial state.
- γ *is a first-order sentence over* $\mathcal{L}_\Pi$ *called the* goal description.

Apart from the mathematical notation (instead of the LISP style), our definition (almost) conforms to the ADL fragment of PDDL as defined in the specification of PDDL 3.1. The only two differences are that we omitted *types* from our specification and that we removed certain restrictions on the form of conditional effects.

Types are syntactic sugar and allow to restrict action and predicate parameters and quantified variables to certain objects of the universe. The semantics of types is defined in terms of unary predicates and they can easily be transformed to such. Since this is also true within the compilation framework which we will use to compare the expressive power of planning formalisms, there is no need to treat them specially in this work.

The original definition of PDDL 3.1 does not allow for nested conditional effects and for universal effects in conditional effects. This is not a fundamental restriction because more complicated conditional effects can easily be replaced with PDDL 3.2 effects in polynomial time[3].

The semantics of PDDL planning can be defined based on the semantics of predicate logic. We begin with the formal notion of a *state* of a PDDL task, which is a Herbrand interpretation for the language of the task:

Definition 3 (state and induced structure)**.** *Let* $\Pi = \langle \langle \mathcal{L}, \mathcal{O} \rangle, C, I, \gamma \rangle$ *be a PDDL task and let* A *be the set of all ground atoms of* $\mathcal{L}_\Pi$.

- A state s *of* Π *is a valuation* $s : A \to \{0, 1\}$.
- A *state* s induces *a structure* $\mathcal{A}_s = (\mathcal{U}, \mathcal{I})$ *for* Π *as follows:*
  - $\mathcal{U} = \{\mathbf{c} \mid c$ *is a constant symbol of* $\mathcal{L}_\Pi\}$ *(the* universe*)*
  - *For each constant symbol* c *of* $\mathcal{L}_\Pi$, $\mathcal{I}(c) = \mathbf{c}$.
  - *For each predicate symbol* P *of* $\mathcal{L}_\Pi$, $\mathcal{I}(P) = \{(\mathbf{c_1}, \ldots, \mathbf{c_n}) \mid s(P(c_1, \ldots, c_n)) = 1\}$.
- *We say that a state* s satisfies *a closed formula* χ *(and write* $s \models \chi$*) if* $\mathcal{A}_s \models \chi$ *according to the semantics of first-order logic.*

Before we introduce the semantics of the application of operators, we first need to define the analog of *closed* formulas for operators:

[3]This can be seen from the equivalences $\varphi \rhd (e_1 \wedge \cdots \wedge e_n) \equiv (\varphi \rhd e_1) \wedge \cdots \wedge (\varphi \rhd e_n)$, $\varphi \rhd (\psi \rhd e) \equiv (\varphi \wedge \psi) \rhd e$ by Rintanen (2005, Table 2.1) and the equivalence $\varphi \rhd \forall x_1 \ldots x_n e \equiv \forall x_1 \ldots x_n (\varphi \rhd e)$ holding if no variable $x_1, \ldots, x_n$ occurs in φ, which can be ensured by a suitable renaming.

Definition 4 (ground operator)**.** *Let* Π *be a PDDL task and let* $\mathcal{C}$ *denote the set of all constant symbols of* $\mathcal{L}_\Pi$*. Let further* $o = \langle n, \chi, e\rangle$ *be a schematic operator of* Π*.*

Each possible parameter mapping $p : free(o) \to \mathcal{C}$ *defines a* ground operator $inst(o, p) = \langle n, \chi', e'\rangle$ *of* o*, where* χ' *and* e' *are the result of substituting* $p(x)$ *for each free occurrence of a variable* x *in* χ *and* e *respectively.*

We now can define what it means to *apply* a ground operator in a state:

Definition 5 (operator applicability, change set, successor state)**.** *Let* Π *be a PDDL task. Let* $\mathcal{C}$ *denote the set of all constant symbols and* A *be the set of all ground atoms of* $\mathcal{L}_\Pi$*.*

- *A ground operator* $\langle n, \chi, e\rangle$ *is* applicable *in state* s *if* $s \models \chi$*.*
- *We define the* change set[4] $[e]_s$ *of an effect* e *without free variables in state* s *inductively as follows:*
  - *For a simple effect* e*,* $[e]_s = \{e\}$
  - $[e_1 \wedge \cdots \wedge e_n]_s = \bigcup_{i\in\{1,\ldots,n\}} [e_i]_s \setminus \{\neg a \mid a \in A \text{ and } \exists i : a \in [e_i]_s\}$
  - $[\chi \triangleright e]_s = \begin{cases} [e]_s & \text{if } s \models \chi \\ \{\} & \text{otherwise} \end{cases}$
  - $[\forall v_1 \ldots v_n e]_s = [\bigwedge_{(c_1,\ldots,c_n)\in\mathcal{C}^n} e[v_1/c_1, \ldots, v_n/c_n]]_s$

  By construction, the change set can only contain ground literals of $\mathcal{L}_\Pi$*.*
- *For a state* s *and a ground operator* $o = \langle n, \chi, e\rangle$ *with* $s \models \chi$*, the successor state* $app_o(s)$ *of* s *with respect to* o *is the state* s' *with* $s' \models [e]_s$ *and* $s'(v) = s(v)$ *for all ground atoms* v *that do not occur as literal in* $[e]_s$*.*

A *plan* in classical planning is a sequence of applicable operators that leads from the initial state to a goal state.

Definition 6 (Plan of a PDDL task)**.** *Let* $\Pi = \langle\langle \mathcal{L}, \mathcal{O}\rangle, C, I, \gamma\rangle$ *be a PDDL task. A sequence* $\langle o_0, \ldots, o_{n-1}\rangle$ *of ground operators is a* plan *for* Π *if there are states* $s_0, \ldots, s_n$ *such that*

- *for all ground atoms* a *of* Π*,* $s_0(a) = \begin{cases} 1 & \text{if } a \in I \\ 0 & \text{otherwise} \end{cases}$
- o_i *is applicable in* s_i
- $s_{i+1} = app_{o_i}(s_i)$ *for* $i \in \{0, \ldots, n-1\}$
- $s_n \models \gamma$

[4] The general character of the definition is taken from Rintanen (2005) but we extended it to non-propositional operators and adapted it to fit the add-after-delete semantics of PDDL.

For a sequence of operators $\langle o_0, \ldots, o_n \rangle$ applicable in a state s we also use the short notation $app_{\langle o_0,\ldots,o_n \rangle}(s)$ to denote the state resulting from the sequential application of these operators: $app_{\langle o_0,\ldots,o_n \rangle}(s) = app_{o_n}(app_{\langle o_0,\ldots,o_{n-1} \rangle}(s))$, where $app_{\langle\rangle}(s) = s$.

Given a PDDL task, we are primarily interested in the *planning problem* of finding a plan or proving that no plan exists. In *satisficing* planning, any plan is a solution but we prefer shorter plans over longer ones, whereas in *optimal* planning a solution must be a shortest possible plan.

Since we will need several complexity results on classical planning in this thesis, we will in the following introduce some related decision problems and discuss their complexity.

Computational Complexity of Classical Planning

The following decision problem relates to satisficing planning, where we are only interested in finding some plan.

Definition 7 (Plan existence problem)**.** *The* plan existence problem *for PDDL is the following decision problem:*

INPUT: *A PDDL task* Π
QUESTION: *Is there a plan for* Π?

Theorem 1. *The plan existence problem for PDDL is* EXPSPACE*-complete.*

Proof. Erol et al. (1995) have shown that the plan existence problem for a more restricted formalism (without universal effects and quantified conditions) is EXPSPACE-complete, so the hardness result directly translates to PDDL.

For the membership in EXPSPACE, consider the non-deterministic procedure shown in Algorithm 1.

In line 1, $initialState(\mathcal{L}, C, I)$ creates the initial state by assigning each ground atom its value according to I. As long as the current state does not satisfy the goal, the algorithm guesses the next action to apply. If it is applicable, it updates the current state with the respective successor state. The procedure accepts if the current state is a goal state.

The algorithm is correct: If the input is a solvable PDDL task, the procedure can obviously guess actions corresponding to a plan and accept the input. Otherwise, it will not terminate, because in an accepting run the applied action sequence would correspond to a plan.

It remains to show that the algorithm needs only exponential space:

Let c be the number of constant symbols in $\mathcal{L}$ and C, let p be the number of predicate symbols in L and a be the maximal arity of these predicates. Then there are at most pc^a ground atoms and any state can be represented in exponential space.

The successor state after applying an action (line 5) can obviously be computed in exponential time (Definition 5) and therefore also in exponential space (if there are no universal effects the update can be done in polynomial time).

In lines 2 and 4, we need to check whether the current state satisfies a first-order sentence. For a variable-free formula this is possible in polynomial time. For an arbitrary sentence we can iterate over the instantiations with all possible variable assignments for the bound variables. Although the number

Algorithm 1 Nondeterministic decision procedure for PDDL plan existence

```
   Input: PDDL task Π = ⟨⟨ℒ, 𝒪⟩, C, I, γ⟩
1  s ← initialState(ℒ, C, I)
2  while s ⊭ γ do
3  |  a ← guess next action
4  |  if a is applicable in s then
5  |  |  s ← app_o(s)
6  accept
```

of instantiations can be exponential, each single one requires only linear space (in the size of the formula).

Since the non-deterministic procedure shown in Algorithm 1 decides plan existence in exponential space and EXPSPACE = NEXPSPACE by Savitch's theorem (Savitch, 1970), we conclude that the plan existence problem for PDDL is in EXPSPACE. □

If the domain is fixed in advance, the plan existence problem is easier:

Theorem 2. *The plan existence problem for PDDL with a fixed domain is in* PSPACE.

Proof. Since PSPACE = NPSPACE, it is sufficient to show that the problem is in NPSPACE. For this purpose consider a variation of the non-deterministic procedure shown in Algorithm 1, where the domain $\mathcal{D} = \langle \mathcal{L}, \mathcal{O} \rangle$ is fixed in advance and the input consists only of a set of task-specific constants C, an initial state specification I and a goal specification γ for this domain.

The arguments in the proof of Theorem 1 for the correctness of the procedure still apply. We will in the following explain how the argumentation for the space requirement must be altered to show that the procedure only requires polynomial space in the input.

In the setting with a fixed domain, the number of predicate symbols p and the maximal predicate arity a are fixed. So the number pc^a of ground atoms is polynomial in the input. The computation of the successor state is possible in polynomial time because the maximal quantifier rank in any universal effect is fixed. The test whether a state satisfies a first-order sentence is already possible in polynomial space if the domain is part of the input.

Since the non-deterministic procedure decides plan existence in polynomial space and PSPACE = NPSPACE by Savitch's theorem, we conclude that the plan existence problem for PDDL with a fixed domain is in PSPACE. □

In *optimal* planning we are interested in finding a plan of minimal length. This corresponds to the following decision problem:

Definition 8 (Bounded plan existence problem)**.** *The* bounded plan existence problem *for PDDL is the following decision problem:*

INPUT: *A PDDL task* Π *and a bound* $k \in \mathbb{N}_0$
QUESTION: *Is there a plan for* Π *of length at most* k*?*

For this thesis, we only require a very specific result on the bounded plan existence problem with a fixed domain and goal, i.e., the input consists only of the set of constants C and the initial state specification I.

Theorem 3. *The bounded plan existence problem for PDDL with a fixed domain and a fixed goal is solvable in polynomial time.*

Proof. Let l be the given bound on the plan length. Let o denote the number of schematic operators in the domain, and a denote the maximal number of parameters of the operators. Then a task with c constants has at most oc^a ground operators and there are at most $\sum_{0 \leq i \leq l} (oc^a)^i$ operator sequences of length at most l. Since l, o and a are constant in this setting, this is a polynomial number, so we can enumerate all plan candidate sequences in polynomial time (in the number of constants).

It remains to show that we can also verify a plan candidate in P: The successor state after applying an operator can obviously be computed in polynomial time (Definition 5) because the maximal quantifier rank of the universal effects is fixed. We can evaluate operator preconditions and the goal formula for a given state by trying all possible bindings of the quantified variables. Testing a binding is linear in the length f of the formula and for a formula with quantifier rank r there are c^r bindings. So we can test the applicability of an operator or the truth of the goal in a state in time $O(fc^r)$. Since the operator preconditions and goal are fixed in this setting, this provides us with a polynomial bound (again in the number of constants) for a single test. As the length of the candidate sequences is bound by l, we can therefore decide in polynomial time whether a candidate actually is a plan.

So we can decide the bounded plan existence problem in polynomial time by enumerating all action sequences of length at most l and testing each sequence whether it is a plan. □

We have seen that though the semantics of PDDL are *based* on predicate logic, changes in the world are not described within the logic itself. We had to define their semantics separately, using individual interpretations for each state.

This is different for the situation calculus, where everything is captured within a logic formalism and each model covers all changes to the world and all situations.

4.2 Situation Calculus and Golog

The *situation calculus* is a logic formalism which is specially designed for representing dynamically changing worlds. In this thesis we use the version of the situation calculus by Reiter (2001). It solves the frame problem by formalizing the dynamics of the domain as a set of *successor state axioms*, *action precondition axioms*, and *unique names axioms*. A theory consisting of such axioms and a description of the initial situation is called a *basic action theory*. The compilability from such basic action theories to PDDL will be the main subject of the theoretical results in this part of the thesis.

While basic action theories describe the dynamics and the initial situation of the task, they do not indicate at all how a plan would possibly look like.

This can be done by annotating the basic action theory with a Golog program that procedurally describes a plan but leaves some aspects open. Such open aspects can for example be a choice of actions arguments like to which floor the elevator should move next or a choice between two different courses of actions.

In the following, we will first introduce the situation calculus in general, before we formally define the notion of basic action theories. Based on this, we then introduce the action language Golog.

Situation Calculus

The situation calculus is designed for representing dynamic systems with a three-sorted second-order language. All changes to the world are the result of *actions* and each action leads to a new *situation*, which can be identified with a sequence of actions. Besides *actions* and *situations* there is a third sort *object* that is used for everything else. Following Reiter's style, we will use variables a for actions, s for situations and $x, y, \ldots$ for objects (each with subscripts and superscripts). There is a special predicate $poss(a, s)$ meaning that it is possible to perform action a in situation s. All situations except the initial situation are formed with a function $do(a, s)$ meaning that action a is applied in situation s. Functions and predicates whose values vary from one situation to the next are called *fluents* and take a situation term as their last argument. There are also some foundational axioms for situations, which ensure that dynamic worlds in the situation calculus behave like one assumes intuitively. To make all this precise, we briefly state formal definitions (Reiter 2001, pp. 47–48; slightly reformulated):

Definition 9 (Language $\mathcal{L}_{sitcalc}$ of the situation calculus)**.** *The* language $\mathcal{L}_{sitcalc}$ of the situation calculus *is the three-sorted second-order language with disjoint sorts situation, action, and object and the following alphabet:*

- *Countably infinitely many individual variable symbols of each of the three sorts.*
- *Countably infinitely many predicate variables of all arities.*
- *Logical symbols* $\exists, \wedge, \neg$ *and parentheses, brackets, and punctuation symbols (we use the usual definitions of a full set of connectives and quantifiers).*
- *The equality symbol* $=$*.*
- *Two function symbols of sort* situation*:*
  - *A constant symbol* $\mathbf{s_0}$*, denoting the initial situation.*
  - *A binary function symbol* $do : action \times situation \to situation$*.*
- *A binary predicate symbol* $\sqsubset$*:* $situation \times situation$*, that will be used to define an ordering relation on situations.*
- *A binary predicate symbol* $poss : action \times situation$*.*
- *A finite or countable set* $\mathcal{R}^c$ *of predicate symbols. These are used for* situation-independent relations*. Each* $r \in \mathcal{R}^c$ *has an associated arity* $ar(r) \in \mathbb{N}_0$ *and is of sort* $(action \cup object)^{ar(r)}$*.*

- *A finite or countable set* $\mathcal{F}^c$ *of function symbols. These are used for* situation-independent object functions. *Each* $f \in \mathcal{F}^c$ *has an associated arity* $ar(f) \in \mathbb{N}_0$ *and is of sort* $(action \cup object)^{ar(r)} \rightarrow object$.
- *A finite or countable set* $\mathcal{F}^a$ *of function symbols. Each* $f \in \mathcal{F}^a$ *has an associated arity* $ar(f) \in \mathbb{N}_0$ *and is of sort* $(action \cup object)^{ar(r)} \rightarrow action$. *These are called* action functions.
- *A finite or countable set* $\mathcal{R}^f$ *of predicate symbols. Each* $r \in \mathcal{R}^f$ *has an associated arity* $ar(r) \in \mathbb{N}_1$ *and is of sort* $(action \cup object)^{ar(r)-1} \times situation$. *These are called* relational fluents.
- *A finite or countable set* $\mathcal{F}^f$ *of function symbols. These are the* functional fluents. *Each* $f \in \mathcal{F}^f$ *has an associated arity* $ar(f) \in \mathbb{N}_0$ *and is of sort* $(action \cup object)^{ar(f)-1} \times situation \rightarrow (action \cup object)$.

Following Reiter's convention, we will in the following often omit leading universal quantifiers in expressions of the situation calculus. If not stated otherwise, these expressions are sentences where any free variable is implicitly universally quantified.

Foundational axioms for situations

We already have adumbrated that a situation should correspond to a history or finite sequence of actions and that we want to use the relation $\sqsubset$ to define an ordering on the situations. In the situation calculus one achieves this by the following four foundational axioms for situations (Reiter, 2001, p. 50):

Definition 10 (Foundational axioms Σ for situations)**.**

$$\forall P(P(\mathbf{s_0}) \wedge \forall a, s(P(s) \rightarrow P(do(a, s))) \rightarrow \forall s P(s)) \tag{4.1}$$

$$do(a_1, s_1) = do(a_2, s_2) \rightarrow a_1 = a_2 \wedge s_1 = s_2 \tag{4.2}$$

$$\neg s \sqsubset \mathbf{s_0} \tag{4.3}$$

$$s \sqsubset do(a, s') \leftrightarrow s \sqsubset s' \vee s = s' \tag{4.4}$$

The first axiom limits the sort *situation* to the smallest set that contains $\mathbf{s_0}$ and is closed under the application of the function *do* to an action and a situation. The second axiom together with axiom 4.1 implies that two situations are equal iff they comply to the same sequence of actions applied to the initial situation $\mathbf{s_0}$.

The other two axioms capture the intended meaning of the predicate $\sqsubset$: expression $s \sqsubset s'$ denotes that s is a prefix of s'.

Basic action theories

The description of a dynamical world in the situation calculus is principally encoded in a particular syntactic form, the so-called *basic action theories*, providing a solution to the frame problem. Before we properly can define the notion of basic action theories, we need to introduce some further concepts (Reiter, 2001, pp. 31, 58–60).

Unique names axioms for actions state whether two actions are equal. Distinct action names A and B define distinct actions and identical actions have identical arguments:

Definition 11 (Unique names axioms for actions)**.** *The set of unique names axioms for a situation calculus signature $\mathcal{S}$ with action functions $\mathcal{F}^a$ consists of the following axioms:*

- *For each pair $A, B \in \mathcal{F}^a$ of distinct action function symbols there is an axiom*
$$A(\bar{x}) \neq B(\bar{y}).$$
- *For each action function symbol $A \in \mathcal{F}^a$ there is an axiom*
$$A(x_1, \ldots, x_n) = A(y_1, \ldots, y_n) \rightarrow x_1 = y_1 \wedge \cdots \wedge x_n = y_n$$

A formula is called *uniform* in situation s if it does not mention the predicates *poss* or $\sqsubset$ and the only permitted occurrence of a situation term is the occurrence of situation s in the situation argument position of a fluent.

Definition 12 (Uniform formula)**.** *Let s be a term of sort situation. A* term that is uniform in s *is inductively constructed according to the following rules:*

1. *Any term that does not mention a term of sort situation is uniform in s.*
2. *If g is an n-ary non-fluent function symbol, and $t_1, \ldots, t_n$ are terms that are uniform in s and whose sorts are appropriate for g, then $g(t_1, \ldots, t_n)$ is uniform in s.*
3. *If f is an $(n+1)$-ary functional fluent symbol, and $t_1, \ldots, t_n$ are terms that are uniform in s and whose sorts are appropriate for f, then the term $f(t_1, \ldots, t_n, s)$ is uniform in s.*

The formulas that are uniform in s *are inductively defined by:*

1. *Any formula that does not mention a term of sort situation is uniform in s.*
2. *When F is an $(n+1)$-ary relational fluent and $t_1, \ldots, t_n$ are terms uniform in s whose sorts are appropriate for F, then $F(t_1, \ldots, t_n, s)$ is a formula uniform in s.*
3. *If φ_1 and φ_2 are formulas uniform in s, so are $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$ and $\exists v(\varphi_1)$ provided v is a variable not of sort situation.*

Whether it is possible to perform an action is stated by so-called *action precondition axioms*:

Definition 13 (Action precondition axiom)**.** Action precondition axioms *are of the form*
$$poss(A(x_1, \ldots, x_n), s) \leftrightarrow \Pi_A(x_1, \ldots, x_n, s),$$
where A is an action function symbol with arity n and $\Pi_A(x_1, \ldots, x_n, s)$ is a formula that is uniform in s and whose free variables are among $x_1, \ldots, x_n, s$.

The value of a fluent after performing an action is specified by a *successor state axiom*:

Definition 14 (Successor state axiom)**.** *A* successor state axiom *for a relational fluent F is of the form*

$$F(x_1, \ldots, x_n, do(a, s)) \leftrightarrow \Phi_F(x_1, \ldots, x_n, a, s), \tag{4.5}$$

where $\Phi_F(x_1, \ldots, x_n, a, s)$ is a formula uniform in s whose free variables are among $x_1, \ldots, x_n, a, s$. Similarly, a successor state axiom for a functional fluent f is of the form

$$f(x_1, \ldots, x_n, do(a, s)) = y \leftrightarrow \Phi_f(x_1, \ldots, x_n, y, a, s)$$

with analogous restrictions on $\Phi_f(x_1, \ldots, x_n, y, a, s)$.

After the introduction of these concepts we now can state the definition of basic action theories from Reiter (2001, p. 60):

Definition 15 (Basic action theory)**.** *A basic action theory (BAT) T is a situation calculus theory of the form*

$$T = \Sigma \cup T_{SSA} \cup T_{APA} \cup \ T_{UNA} \cup T_{s_0},$$

where

- *Σ are the foundational axioms for situations,*
- *T_{SSA} is a set of successor state axioms for functional and relational fluents, one for each fluent occurring in T.*[5]
- *T_{APA} is a set of action precondition axioms, one for each action function symbol occurring in T.*[5]
- *T_{UNA} is the set of unique names axioms for all action function symbols occurring in T.*[5]
- *T_{s_0} is the* initial database*, a set of first-order sentences that are uniform in $\mathbf{s_0}$.*

*The successor state axiom for a functional fluent f must actually define a value for f in the next situation, and this value must be unique (*functional fluent consistency property*).*

In order to distinguish the components of a basic action theory, we sometimes write it as a tuple $\langle \Sigma, T_{SSA}, T_{APA}, T_{UNA}, T_{s_0} \rangle$.

An example for such a basic action theory for the elevator domain is shown in Figure 4.4.

Reiter (2001, p. 38) also already defined a notion of a plan for the situation calculus:

[5] Reiter (2001) does not explicitly require that there is one axiom for each fluent (resp. a full set of unique names axioms for actions). We nevertheless postulate this, because it is the more common definition (Levesque et al., 1998; Pirri and Reiter, 1999).

$T = \Sigma \cup T_{SSA} \cup T_{APA} \cup T_{UNA} \cup T_{s_0}$ is a BAT with the following components:

- Σ consists of the foundational axioms for situations as specified in Definition 10.
- T_{SSA} consists of the successor state axioms

$$\begin{aligned}
\mathit{boarded}(p, do(a, s)) \leftrightarrow\ & \exists f(a = \mathit{stop}(f) \wedge \mathit{origin}(p, f) \wedge \neg \mathit{served}(p, s)) \\
& \vee \mathit{boarded}(p, s) \wedge \\
& \quad \neg \exists f(a = \mathit{stop}(f) \wedge \mathit{destination}(p, f)) \\
\mathit{served}(p, do(a, s)) \leftrightarrow\ & \exists f(a = \mathit{stop}(f) \wedge \mathit{destination}(p, f) \wedge \mathit{boarded}(p, s)) \\
& \vee \mathit{served}(p, s) \\
\mathit{lift\text{-}at}(f, do(a, s)) \leftrightarrow\ & \exists f'(a = \mathit{move}(f', f)) \\
& \vee \mathit{lift\text{-}at}(f, s) \wedge \neg \exists f'(a = \mathit{move}(f, f') \wedge f \neq f')
\end{aligned}$$

- T_{APA} consists of the action precondition axioms

$$\begin{aligned}
\mathit{poss}(\mathit{stop}(f), s) &\leftrightarrow \mathit{lift\text{-}at}(f, s) \\
\mathit{poss}(\mathit{move}(f, f'), s) &\leftrightarrow \mathit{lift\text{-}at}(f, s) \wedge \mathit{adjacent}(f, f')
\end{aligned}$$

- T_{UNA} consists of the unique names axioms for actions

$$\begin{aligned}
\mathit{stop}(f) &\neq \mathit{move}(f', f'') \\
\mathit{stop}(f) = \mathit{stop}(f') &\rightarrow f = f' \\
\mathit{move}(f_1, f_2) = \mathit{move}(f_1', f_2') &\rightarrow f_1 = f_1' \wedge f_2 = f_2'
\end{aligned}$$

- T_{s_0} is the initial database

$$\begin{aligned}
& \neg \mathit{boarded}(p, \mathbf{s_0}) \\
& \neg \mathit{served}(p, \mathbf{s_0}) \\
\mathit{lift\text{-}at}(f, \mathbf{s_0}) \leftrightarrow\ & f = \mathbf{f_0} \\
\mathit{floor}(f) \leftrightarrow\ & f = \mathbf{f_0} \vee f = \mathbf{f_1} \vee f = \mathbf{f_2} \vee f = \mathbf{f_3} \\
\mathit{origin}(p, f) \leftrightarrow\ & (p = \mathbf{p_0} \wedge f = \mathbf{f_0}) \vee (p = \mathbf{p_1} \wedge f = \mathbf{f_1}) \vee \\
& (p = \mathbf{p_2} \wedge f = \mathbf{f_2}) \vee (p = \mathbf{p_3} \wedge f = \mathbf{f_2}) \\
\mathit{destination}(p, f) \leftrightarrow\ & (p = \mathbf{p_0} \wedge f = \mathbf{f_2}) \vee (p = \mathbf{p_1} \wedge f = \mathbf{f_0}) \vee \\
& (p = \mathbf{p_2} \wedge f = \mathbf{f_0}) \vee (p = \mathbf{p_3} \wedge f = \mathbf{f_3}) \\
\mathit{adjacent}(f, f') \leftrightarrow\ & (f = \mathbf{f_0} \wedge f' = \mathbf{f_1}) \vee (f = \mathbf{f_1} \wedge f' = \mathbf{f_0}) \vee \\
& (f = \mathbf{f_1} \wedge f' = \mathbf{f_2}) \vee (f = \mathbf{f_2} \wedge f' = \mathbf{f_1}) \vee \\
& (f = \mathbf{f_2} \wedge f' = \mathbf{f_3}) \vee (f = \mathbf{f_3} \wedge f' = \mathbf{f_2})
\end{aligned}$$

Figure 4.4: Basic action theory for the example elevator domain with the initial state as depicted in Figure 4.1

Definition 16 (Plan relative to a theory)**.** *For a theory T and a goal formula γ with a single free variable of sort situation, a* plan for γ relative to T *is a variable-free situation term s that satisfies*

$$T \models \mathit{executable}(s) \wedge \gamma(s),$$

where $\mathit{executable}(s)$ means that the action sequence s can be executed with respect to poss:

$$\mathit{executable}(s) \stackrel{def}{=} \forall a, s'((\mathit{do}(a, s') \sqsubset s \vee \mathit{do}(a, s') = s) \rightarrow \mathit{poss}(a, s')). \quad (4.6)$$

In this notion of planning for the situation calculus the full world description including the initial situation is fixed in the theory and the tasks only differ in the goal description. For practical applications, we need a wider notion of domain, which leaves the freedom to also change the the initial situation from task to task. For this purpose, we will define *BAT tasks* as a new concept analogously to PDDL planning tasks and show, how they relate to Reiter's definition of planning.

Definition 17 (BAT domain)**.** *A* BAT domain *is given by a tuple $\mathcal{D} = \langle \mathcal{S}, P, E \rangle$ with the following components:*

- *$\mathcal{S}$ is a signature for $\mathcal{L}_{sitcalc}$.*
- *P is a set of action precondition axioms (appropriate for $\mathcal{S}$), one for each action function symbol in $\mathcal{S}$.*
- *E is a set of successor state axioms (appropriate for $\mathcal{S}$), one for each fluent symbol in $\mathcal{S}$.*

A task specification adds a description of the initial situation and the goal, and can also contain a set of task-specific constants:

Definition 18 (BAT task)**.** *A* BAT task *is given by a tuple $\Pi = \langle \mathcal{D}, C, I, \gamma \rangle$ where*

- *$\mathcal{D} = \langle \mathcal{S}, P, E \rangle$ is a BAT domain.*
- *C is a finit set of task-specific constant symbols. We define the* signature $\mathcal{S}_\Pi$ of task Π *as the signature we receive by extending the set of constants in $\mathcal{S}$ by the constants in C.*
- *I is the* initial state specification*, a set of first-order sentences over $\mathcal{S}_\Pi$ that are uniform in $\mathbf{s_0}$.*
- *γ is the* goal description*, a first-order formula over $\mathcal{S}_\Pi$ with a single free variable of sort situation.*

Such a BAT task induces a basic action theory and a planning task in Reiter's notion as follows:

Definition 19 (Theory of a BAT task)**.** *A BAT task $\Pi = \langle \mathcal{D}, C, I, \gamma \rangle$ with $\mathcal{D} = \langle \mathcal{S}, P, E \rangle$ induces a basic action theory $\mathcal{T}(\Pi) = \langle \Sigma, T_{SSA}, T_{APA}, T_{UNA}, T_{s_0} \rangle$ as follows:*

- Σ *are the foundational axioms of the situation calculus.*
- $T_{SSA} = E$
- $T_{APA} = P$
- T_{UNA} *are the unique names axioms for all action function symbols in* Σ.
- $T_{s_0} = I$

Note that we did not include the unique names axioms for actions and the foundational axioms in our definition of a BAT task, but these are implicitly added by the semantics specified in the definition of the induced basic action theory. However, this is not critical to the expressive power: the foundational axioms are constant and can therefore be added in constant time (and space). The unique names axioms for actions are completely determined by the signature of the domain and could be added to the domain in polynomial time in the size of the domain description (because there must be an action precondition for every action function symbol in the domain).

Definition 20 (Plan for a BAT task)**.** *A* plan *for a BAT task* $\Pi = \langle \mathcal{D}, C_t, I, \gamma \rangle$ *is a plan for* γ *relative to* $\mathcal{T}(\Pi)$*. The* length *of the plan is the number of action applications in the situation* γ*, i.e., the number of occurrences of function symbol do.*

Definition 19 specifies a unique theory for a BAT task. It is obvious that we can vice versa map each basic action theory to a corresponding BAT task.

Golog

Golog extends a basic action theory with a procedural sketch of the desired action sequence. In the following we refer to the actions of the basic action theory as *primitive actions* in contrast to *complex actions* or *programs* of the Golog program.

Before we formally introduce the language, we will first have a look at a brief example program for the elevators basic action theory from Figure 4.4. It is shown in Figure 4.5. Informally, the program runs as long as there is still a passenger that has not been served. If there is already someone in the cabin, the program will first transport her to her destination (possibly with the side effect that other boarded passengers get served or new passengers enter the elevator). Otherwise, it collects at least one waiting passenger.

Note that in Golog all situation arguments are suppressed and are only reinserted by the semantics definition. Golog comprises structures that are well-known from other programming languages like the **while** loop or the **if** statement used in lines 1 and 2. However, it also provides the possibility to include nondeterminism in the program. In this example, we use the π operator that nondeterministically chooses an argument (consider for example the argument f' in line 5). While in general *all* objects could be assigned to the parameter, the interpreter must choose one with which it can execute the rest of the program. So in line 5, it must choose an object f' such that the primitive action $move(f, f')$ can be applied in this situation. In the special

```
1  while ∃p(¬served(p) ∧ ∃f(destination(p, f))) do
2      if ∃p(boarded(p)) then
3          (πf.
4              ∃p(boarded(p) ∧ destination(p, f))?;
5              (πf′.move(f′, f));
6              stop(f))
7      else
8          (πf.
9              ∃p(¬served(p) ∧ origin(p, f))?;
10             (πf′.move(f′, f));
11             stop(f))
12     endIf
13 endWhile
```

Figure 4.5: Example Golog program for the elevators basic action theory from Figure 4.4.

case of this example, f' must always be chosen to be the current location of the elevator cabin. However, we can additionally constraint the applicability of a (sub-)program with so-called *test actions*. We use such a test action in line 4 (recognizable by the question mark) to ensure that the nondeterministic choice in line 3 selects a floor that is the destination of a boarded passenger.

The choice of a parameter is not the only possibility to include nondeterminism in a Golog program. In addition, it also provides a nondeterministic choice $(\delta|\delta')$ of two subprograms δ and δ' and a nondeterministic iteration $(\delta)^*$ that executes δ zero ore more times.

Golog also allows to define named procedures that can be called as subroutines in the program. Overall, Golog programs can be constructed from the following components:

α	primitive action
$\varphi?$	test
$\delta_1; \delta_2$	sequence
$(\delta_1 \mid \delta_2)$	nondeterministic choice
$(\pi x.\delta(x))$	nondeterministic choice of argument
$(\delta)^*$	nondeterministic iteration
if φ **then** δ_1 **else** δ_2 **endIf**	conditional
while φ **do** δ **endWhile**	loop
proc $P(\bar{x})$ δ **endProc**	procedure definition
$P(\bar{x})$	procedure call

The **if** statement and the **while** loop are actually only syntactic sugar and can be expressed in terms of the other constructs:

$$\textbf{if } \varphi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endIf} \equiv ((\varphi?; \delta_1)|(\neg\varphi?; \delta_2)), \text{ and}$$
$$\textbf{while } \varphi \textbf{ do } \delta \textbf{ endWhile} \equiv (\varphi?; \delta)^*; \neg\varphi?$$

The semantics of the constructs is defined in terms of macro expansion to situation calculus formulas. For most statements it is very simple and only the expansion for the procedures is somewhat involved because they can be called recursively. Since for this thesis it is sufficient to understand the meaning of the statements intuitively, we refer the interested reader to the definition by Levesque et al. (1997).

ConGolog (de Giacomo et al., 2000) extends the basic version with exogenous actions that are not under the control of the agent and the following program constructs:

$\delta_1 \| \delta_2$	concurrent execution
$\delta_1 \rangle\!\rangle \delta_2$	prioritized concurrency
$\delta^{\|}$	concurrent iteration
$\langle \varphi \longrightarrow \delta \rangle$	interrupt

The concurrent execution $\delta_1 \| \delta_2$ executes the programs δ_1 and δ_2 in parallel, interleaving the primitive actions of both programs. The prioritized concurrency $\delta_1 \rangle\!\rangle \delta_2$ similarly executes the programs in parallel but δ_1 has a higher priority, so that primitive actions from δ_2 are only executed if δ_1 cannot proceed or has terminated. The concurrent iteration $\delta^{\|}$ is the concurrent version of the nondeterministic iteration where δ is executed zero or more times concurrently. Interrupts $\langle \varphi \longrightarrow \delta \rangle$ executes δ if the trigger condition φ becomes true but are considered to be stuck if φ is false. In combination with prioritized concurrency this allows to describe systems that are reactive to certain situations like for example changes caused by exogenous actions.

One major practical disadvantage of the basic version as well as ConGolog is that the entire sequence of primitive actions must be precomputed by look ahead search to ensure that the nondeterministic choices allow a successful program execution. It also does not integrate sensing actions that allow to gather additional information about the current state of the environment.

IndiGolog (de Giacomo and Levesque, 1999) is an extension of ConGolog that addresses these limitations by allowing such sensing actions and executing the program online. If not specified otherwise, the interpreter resolves all nondeterministic choices with an arbitrary[6] decision, not ensuring that the program can be continued successfully. Since this way programs would regularly get stuck, IndiGolog allows to switch on the original look ahead search behavior for parts of the program. This is done by means of the search operator Σ. A statement $\Sigma(\delta)$ causes the interpreter to search for a executable sequence of primitive operators for δ under the assumption that all required information is available and the environment does not change. If the latter happens as a result of an exogenous action, the interpreter replans the non-executed rest of δ. The current IndiGolog implementation does not allow sensing within a search statement.

For our experiments in Section 7 we want to simulate dynamic environments that cannot be handled by classical planning alone due to exogenous events like newly arriving passengers in the elevators domain. In addition, the search operator of IndiGolog makes it very simple to integrate a planning system. Therefore, we chose this Golog version as basis for this thesis. Since our

[6] This choice is deterministic but intransparent to the user.

theoretical study only considers the basic action theory, this choice only affects the experimental evaluation.

5 Compilation Schemes

In Chapter 3 on related work we already briefly introduced several approaches for comparing the complexity of problems or expressivity of planning formalism.

One obvious problem with Karp reductions (Karp, 1972) is that the domain and the task specification are translated together, so all planning formalisms for which plan existence is complete for a given complexity class would have the same expressiveness under this transformation. Another problem is that Karp reduction is only defined for decision problems, which are in our context the plan existence and the bounded plan existence problem. For the plan *generation* problem we also would like to consider the resulting plans.

ESP reduction (Bäckström, 1995) considers the structure of the plan space, but it still does not translate the domain separately, therefore not measuring how *concise* planning *domains* can be represented in each formalism.

To examine such a notion of expressivity for propositional planning formalisms, Nebel (2000a) introduced so-called *compilation schemes*, which do not only separate the components of the planning task but also distinguish different levels of compilability, relative to the required blow-up of the plan size.

Compilation schemes were originally introduced for comparing different *propositional* planning formalisms. Since we do not only consider propositional formalisms but would like to build on the general approach, we need to extend the original definition to first-order formalisms. To provide a better understanding of our definition, we first briefly present Nebel's original definition.

5.1 Compilation Schemes for Propositional Planning Formalisms

There is no singular definition of compilation schemes for propositional planning but the framework has undergone several revisions and adaptions to specific comparisons (Nebel et al., 1997; Nebel, 1999, 2000a). We will introduce the most recent (and most involved version) from 2000.

In the planning formalisms considered by Nebel (2000a) a planning instance is defined as follows.

Definition 21 (Propositional planning instance, Nebel (2000a))**.** *A [propositional] planning instance is a tuple*

$$\Pi = \langle \Xi, \mathbf{I}, \mathbf{G} \rangle \,, \tag{5.1}$$

where

- $\Xi = \langle \Sigma, \mathbf{O} \rangle$ *is the* domain structure *consisting of a finite set of propositional atoms* Σ *and a finite set of operators* $\mathbf{O}$*,*
- $\mathbf{I} \subseteq \hat{\Sigma}$ *is the* initial state specification*,*
- $\mathbf{G} \subseteq \hat{\Sigma}$ *is the* goal specification*.*

In Nebel's definition the set $\hat{\Sigma}$ is defined as the set of literals over Σ plus the constants $\top$ and $\bot$ denoting truth and falsity, respectively. For us, it is sufficient to define $\hat{\Sigma}$ as the set of literals over Σ and to represent sets including $\bot$ as sets that contain at least one atom positively and negatively. Nebel's definition of operators is in the most restrictive case equivalent to STRIPS operators, i.e., preconditions are conjunctions of atoms and effects conjunctions of literals. In the less restrictive formalisms he considers conditional effects, literals in preconditions and effect conditions, and arbitrary Boolean formulas in these conditions. A plan is a sequence of operators whose application leads from every state satisfying all elements of $\mathbf{I}$ to a state satisfying all elements of $\mathbf{G}$.

Compilation schemes compare how concisely planning *domains* and *plans* can be expressed in different formalisms, not how concisely an individual planning *instance* can be expressed. The intuition behind compilation schemes is that it is justifiable to perform significant work on translating a domain description from one formalism to another, as long as this remains a one-time effort, and individual *instances* of the domain can subsequently be transformed efficiently.

As compilation schemes should measure the *expressivity* of a formalism, the mapping may use arbitrary computational resources; it does not even need to be computable. However, the result must be of polynomial size (in the size of the input), and the transformation of the domain description must not depend on the initial state and the goal. The translation of the initial state and the goal is done by so-called *state translation functions* which are very limited: they should be efficiently computable and not depend on the whole specification.

To compare the expressive power of two planning formalisms, compilation schemes moreover have to consider the size of the generated plans. Before we can state concretely which demands they make on the plan length, we first need to define the notion of compilation schemes formally.

A compilation scheme maps each planning instance Π of the source formalism $\mathcal{X}$ to an instance $F(\Pi)$ of the target formalism $\mathcal{Y}$.

Definition 22 (Compilation scheme for propositional planning formalisms[1], Nebel (2000a))**.** *Assume a tuple of functions* $\mathbf{f} = \langle f_\xi, f_i, f_g, t_i, t_g \rangle$ *that induce a function* F *from [propositional]* $\mathcal{X}$*-instances* $\Pi = \langle \Xi, \mathbf{I}, \mathbf{G} \rangle$ *to [propositional]* $\mathcal{Y}$*-instances* $F(\Pi)$ *as follows:*

$$F(\Pi) = \langle f_\xi(\Xi), f_i(\Xi) \cup t_i(\Sigma, \mathbf{I}), f_g(\Xi) \cup t_g(\Sigma, \mathbf{G}) \rangle.$$

[1] Earlier versions (Nebel et al., 1997; Nebel, 1999) of this definition required t_i and t_g to be the identity function and therefore omitted condition 2.

If the following three conditions are satisfied, we call **f** *a* compilation scheme from $\mathcal{X}$ to $\mathcal{Y}$*:*

1. *there exists a plan for* Π *iff there exists a plan for* $F(\Pi)$*,*
2. *the* state translation functions t_i *and* t_g *are* modular*, i.e., for* $\Sigma = \Sigma_1 \cup \Sigma_2$*, and* $S \subseteq \hat{\Sigma}$*, the functions* t_x *(for* $x = i, g$*) satisfy*

$$t_x(\Sigma, S) = t_x(\Sigma_1, S \cap \hat{\Sigma}_1) \cup t_x(\Sigma_2, S \cap \hat{\Sigma}_2),$$

   and they are polynomial-time computable*,*
3. *and the size of the results of* f_ξ, f_i *and* f_g *is* polynomial in the size of the arguments*.*

If with a compilation scheme **f** there is a guarantee that for each plan π of instance Π there is a plan π' solving $F(\Pi)$ such that $|\pi'| \leq |\pi| + k$ for some positive integer k, **f** is called a *compilation scheme that preserves plan size exactly* (accepting a constant additive increase). Nebel (2000a) also considers weaker restrictions on the required growth of the plan length, where plans are allowed to grow linearly, or even polynomially in the size of π and Π.

If there exists a compilation scheme from a propositional planning formalism $\mathcal{X}$ to a propositional planning formalism $\mathcal{Y}$, $\mathcal{X}$ is *compilable* to $\mathcal{Y}$, written $\mathcal{X} \preceq^x \mathcal{Y}$, where the superscript x is u, e, l, or p, indicating that the compilation scheme is unrestricted or preserves plan size exactly, linearly or polynomially, respectively.

If $\mathcal{X} \preceq^e Y$ or $\mathcal{X} \preceq^l Y$, the target formalism $\mathcal{Y}$ is considered at least as expressive as the source formalism $\mathcal{X}$. If plans are required to grow polynomially and there is no other compilation scheme preserving plan size linearly, this indicates that the source formalism is more expressive than the target formalism, but the compilation might still be of practical interest (Nebel, 2000b). If there is even a super-polynomial blow-up required, there must be a considerable difference in expressive power.

For the practical application of a positive result ($\mathcal{X} \preceq^x \mathcal{Y}$), we also are interested in an efficient transformation of the domain. If for a compilation scheme $\mathbf{f} = \langle f_\xi, f_i, f_g, t_i, t_g \rangle$ the function f_ξ is polynomial-time computable then **f** is called a *polynomial-time compilation scheme*. If such a compilation scheme exists from $\mathcal{X}$ to $\mathcal{Y}$, this polynomial-time compilability is denoted by $\mathcal{X} \preceq_p^x \mathcal{Y}$, where x is used as before.

One property that is especially relevant for the practical application of compilation schemes (but also theoretically interesting) is whether the relations $\preceq^x$ are transitive and reflexive. *Reflexivity* is obvious, setting f_ξ, t_i and t_g to the identity function and defining the result of f_i and f_g as the empty set. Nebel also claims *transitivity* to be obvious (Nebel, 2000a, Proposition 4) but agreed in personal communication that for the proof he had in mind, we require additional conditions for transitivity.

The main issue is that the compilation of the domain can contribute to the initial state and the goal specification. Therefore, a simple concatenation of two compilation schemes would break the condition that the individual parts of a task must be translated independently. So, for transitivity, we have to resolve this dependency. It is still an open question whether this is always possible

for general compilation schemes, so we accomplish transitivity by introducing additional restrictions on the compilation. There are several possibilities to define such restrictions and a suitable definition also depends on the expressivity of the formalisms under consideration (e. g., if the formalisms allow conditional effects, we can combine two almost arbitrary compilation schemes by handling the interactions in the compiled domain description).

Here we present a set of requirements that covers all compilation schemes in Nebel's original work: Most important, the contributions of the state translation functions and the domain translation functions to the compiled initial state and the compiled goal description, respectively, may not interfere with each other by adding literals for the same atoms.

Definition 23 (Modularity-preserving propositional compilation schemes)**.** *Let* $\mathbf{f} = \langle f_\xi, f_i, f_g, t_i, t_g \rangle$ *be a compilation scheme from a propositional formalism* $\mathcal{X}$ *to a propositional formalism* $\mathcal{Y}$*.*

We call **f** modularity-preserving *iff there are polynomial-time computable functions* l_i *and* l_g *from sets of propositional atoms to sets of propositional atoms such that (for* $x = i, g$*)*

1. $t_x(\Sigma, S) \subseteq \widehat{l_x(\Sigma)}$ *for* $S \subseteq \hat{\Sigma}$*,*
2. $f_x(\langle \Sigma, \mathbf{O} \rangle) \cap \widehat{l_x(\Sigma)} = \emptyset$*,*
3. $l_x(\Sigma) = l_x(\Sigma_1) \cup l_x(\Sigma_2)$ *for* $\Sigma = \Sigma_1 \cup \Sigma_2$*,*
4. $l_x(\Sigma_1) \cap l_x(\Sigma_2) = l_x(\{\})$ *if* $\Sigma_1 \cap \Sigma_2 = \emptyset$*, and*
5. $t_x(\Sigma', S \cap \hat{\Sigma}') = t_x(\Sigma, S) \cap \widehat{l_x(\Sigma')}$ *for* $\Sigma' \subseteq \Sigma$*.*

We call l_i *and* l_g *the* modularity-preserving functions *of* **f***.*

With this additional requirement, the relation $\preceq^x$ is transitive.

Theorem 4. *If* $\mathcal{X} \preceq^x \mathcal{Y}$ *with a modularity-preserving compilation and* $\mathcal{Y} \preceq^x \mathcal{Z}$ *with a modularity-preserving compilation then* $\mathcal{X} \preceq^x \mathcal{Z}$ *with a modularity-preserving compilation.*

The proof for this theorem can be found in Appendix A.1. Since we want to compare non-propositional planning formalisms, we need to carry over Nebel's concepts to more general planning formalisms.

5.2 Compilation Schemes for Non-Propositional Planning Formalisms

Compilation schemes have been applied to non-propositional planning formalisms before, however, these extensions are not perfectly suitable for our scenario.

Thiébaux et al. (2005) use compilation schemes to show that the extension of PDDL with axioms (derived predicates) increases the expressive power. In an earlier version (Thiébaux et al., 2003), they define compilation schemes based on an early version of the propositional compilation schemes (Nebel, 2000b). However, this definition does not allow task-specific constants and is

therefore not applicable to our scenario. In the journal article (Thiébaux et al., 2005) they revised the definition accordingly but still do not allow to modify the original initial state and goal specification (only extending them depending on the domain description and the constants). Since we want to compare formalisms that use a very different syntax for the initial state specification, this is too inflexible for our setting.

We build in our thesis on the previous work by Eyerich et al. (2006) (and Eyerich, 2006) who also used compilation schemes to compare the expressive power of basic action theories and PDDL. However, they did not attempt to specify a unifying definition but instead gave two different compilation scheme definitions, one for each direction. Due to this and because their specified compilation schemes do not perfectly match their formal definition, we want to use a new general definition that covers all compilations in this work.

In our more general formalisms, a planning instance $\Pi = \langle \mathbf{D}, \mathbf{C}, \mathbf{I}, \mathbf{G} \rangle$ also comprises a *domain structure* $\mathbf{D}$, an *initial state specification* $\mathbf{I}$, and a *goal specification* $\mathbf{G}$, but in addition a constant specification $\mathbf{C}$ because the components of the planning instance are no longer propositional but can contain first-order elements. A *plan* is again some sequence of actions that leads from the initial states to a goal state.

It certainly makes sense to retain the first and third requirement of Nebel's compilation schemes, which stipulate that the compilation must preserve plan existence and that the results of all individual transformation functions must be polynomial in the input size.

The remaining, second requirement is twofold, postulating that the state translation functions are modular and polynomial-time computable. There is no obvious way to carry over modularity from sets of atoms to general formula and we also do not see a reason to do so: first, it has only been required for one of Nebel's compilability results; second, it is unclear whether modularity is already sufficient for the transitivity of the compilability relation; and third, there is no large practical advantage if we anyway require that the initial state and goal specification should be compilable in polynomial time, which is the second aspect of this requirement and which is indeed relevant for practical applications.

Since the initial state should be able to initialize predicates for the constants, the respective compilation function needs to access them. The same is true for the signature defined as part of the domain. The latter is also relevant for the translation of the domain-specific constants to allow the addition of new constants relative to the entire set of constants in the task.

We go even further: instead of limiting the access of the state-translation functions for the constants and the initial state to only the *signature* of the domain, we also give it access to the *dynamics* of the domain. In the positive case, i.e., when we really want to apply the compilation, it appears natural to also consider the domain of the application when translating the initial state; but also in the negative case there is no compelling reason to be as restrictive as the original definition when adapting the framework to non-propositional formalisms. However, we retain the requirement that the initial state and the goal specification are translated separately.

Definition 24 (Compilation scheme)**.** *Let* $\mathbf{f}$ *be a tuple* $\langle f_d, t_c, t_i, t_g \rangle$ *of functions that induces a function* F *from* $\mathcal{X}$*-instances* $\Pi = \langle \mathbf{D}, \mathbf{C}, \mathbf{I}, \mathbf{G} \rangle$ *to* $\mathcal{Y}$*-instances*

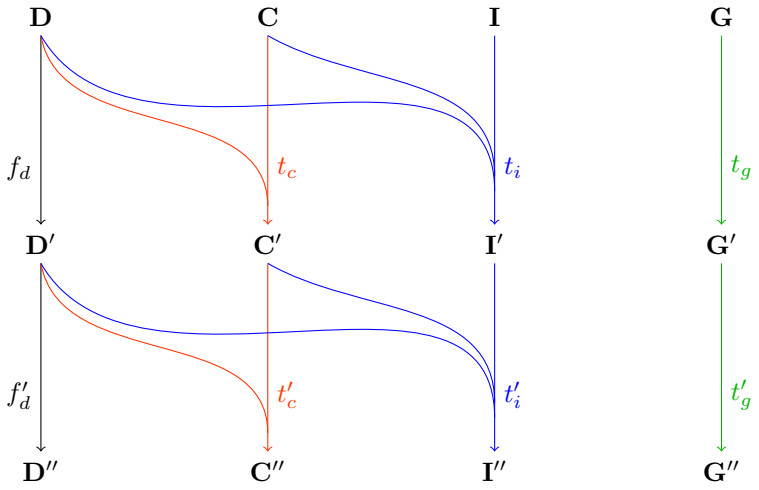


Figure 5.1: Dependencies of consecutive compilation schemes.

$F(\Pi)$*:*

$$F(\Pi) = \langle f_d(\mathbf{D}), t_c(\mathbf{D}, \mathbf{C}), t_i(\mathbf{D}, \mathbf{C}, \mathbf{I}), t_g(\mathbf{G})\rangle.$$

We call $\mathbf{f}$ *a compilation scheme from* $\mathcal{X}$ *to* $\mathcal{Y}$ *iff*

1. *there exists a plan for* Π *iff there exists a plan for* $F(\Pi)$*,*
2. *the translation functions* t_c, t_i *and* t_g *are* polynomial-time computable*,*
3. *and the size of the result of the domain translation function* f_d *is* polynomial in the size of the argument $\mathbf{D}$*.*

$\mathbf{f}$ *is a* polynomial-time *compilation scheme if* f_d *is also polynomial-time computable.*

We use the notations $\mathcal{X} \preccurlyeq^x \mathcal{Y}$ and $\mathcal{X} \preccurlyeq^x_p \mathcal{Y}$ analogously to Nebel's $\preceq^x$ and $\preceq^x_p$. For the plan length restriction x, we distinguish u for a unrestricted plan length, r for a plan length restricted by any function only depending on $|\pi|$, and e for compilability where plan length is preserved exactly (up to a constant additive increase). Note that $\mathcal{X} \preccurlyeq^{\mathrm{e}} \mathcal{Y}$ implies $\mathcal{X} \preccurlyeq^{\mathrm{r}} \mathcal{Y}$ which implies $\mathcal{X} \preccurlyeq^{\mathrm{u}} \mathcal{Y}$.

Theorem 5. *Let* $\mathcal{X} \preccurlyeq^a_p \mathcal{Y}$*.*
If $\mathcal{Y} \preccurlyeq^b \mathcal{Z}$ *then* $\mathcal{X} \preccurlyeq^c \mathcal{Z}$*, where* $c = \mathrm{e}$ *if* $a = b = \mathrm{e}$ *and* $c = \mathrm{u}$ *otherwise*[2]*.*
If $\mathcal{Y} \preccurlyeq^b_p \mathcal{Z}$ *then* $\mathcal{X} \preccurlyeq^c_p \mathcal{Z}$ *with* c *as before.*

Proof. Let $\mathcal{X}, \mathcal{Y}$, and $\mathcal{Z}$ be planning formalisms and let $\mathbf{f} = \langle f_d, t_c, t_i, t_g\rangle$ be a polynomial-time compilation scheme from $\mathcal{X}$ to $\mathcal{Y}$ and $\mathbf{f}' = \langle f'_d, t'_c, t'_i, t'_g\rangle$ be a compilation scheme from $\mathcal{Y}$ to $\mathcal{Z}$.

Consider an $\mathcal{X}$-instance $\Pi = \langle \mathbf{D}, \mathbf{C}, \mathbf{I}, \mathbf{G}\rangle$. We can depict the dependencies of the compilation schemes as shown in Figure 5.1.

[2] A more fine-grained specification of the the bound c on the increase in plan size would be possible but is not necessary for the results in this thesis.

From the picture it becomes obvious that we can define a compilation scheme $\mathbf{f}'' = \langle f''_d, t''_c, t''_i, t''_g \rangle$ from $\mathcal{X}$ to $\mathcal{Z}$ that compiles an $\mathcal{X}$-instance $\Pi = \langle \mathbf{D}, \mathbf{C}, \mathbf{I}, \mathbf{G} \rangle$ as follows:

$$\begin{aligned} f''_d(\mathbf{D}) &= f'_d(f_d(\mathbf{D})) \\ t''_c(\mathbf{D}, \mathbf{C}) &= t'_c(f_d(\mathbf{D}), t_c(\mathbf{D}, \mathbf{C})) \\ t''_i(\mathbf{D}, \mathbf{C}, \mathbf{I}) &= t'_i(f_d(\mathbf{D}), t_c(\mathbf{D}, \mathbf{C}), t_i(\mathbf{D}, \mathbf{C}, \mathbf{I})) \\ t''_g(\mathbf{G}) &= t'_g(t_g(\mathbf{G})) \end{aligned}$$

By definition, the result is equivalent to the task $\Pi'' = \langle \mathbf{D}'', \mathbf{C}'', \mathbf{I}'', \mathbf{G}'' \rangle$ which we receive when concatenating the compilations $\mathbf{f}$ and $\mathbf{f}'$. Therefore, $\mathbf{f}''$ preserves plan existence.

Since $\mathbf{f}$ is a polynomial-time compilation scheme, the translation function f_d can be efficiently computed. Moreover, all state-translation functions $t_c, t_i, t_g, t'_c, t'_i, t'_g$ are polynomial-time computable by the requirements of compilation schemes. Therefore the functions t''_c, t''_i, and t''_g can be efficiently computed.

The size of the result of $f''_d(\mathbf{D})$ is polynomial in the size of the input because both subfunction f_d and f'_d maintain this property.

Therefore, $\mathbf{f}''$ is a compilation scheme from $\mathcal{X}$ to $\mathcal{Z}$. If $\mathbf{f}'$ is a polynomial-time compilation scheme then the function f''_d can be computed in polynomial time and also $\mathbf{f}''$ is a polynomial-time compilation scheme.

It remains to show that if both $\mathbf{f}$ and $\mathbf{f}'$ preserve plan size exactly then also $\mathbf{f}''$ preserves plan size exactly. Let k and k' be the additive constants allowed for a plan length increase from $\mathbf{f}$ and $\mathbf{f}'$, respectively. Then for every plan π of Π there is a plan π' for $\Pi' = \langle \mathbf{D}', \mathbf{C}', \mathbf{I}', \mathbf{G}' \rangle$ with $|\pi'| \leq |\pi| + k$ and for every plan π' of Π' there is a plan π'' of Π'' with $|\pi''| \leq |\pi'| + k'$. Overall, for every plan π of Π there is a plan π'' of Π'' with $|\pi''| \leq |\pi| + k + k'$. Therefore we can fix the maximal additive plan increase incurred by $\mathbf{f}''$ as $k + k'$. □

This theorem is not only useful as a transitivity argument to show that a formalism *is* compilable to another one but also for showing that a formalism *is not* compilable to another formalism.

Corollary 1. *Let $\mathcal{X}, \mathcal{Y}$ and $\mathcal{Z}$ be planning formalism such that $\mathcal{X} \preccurlyeq^{\mathrm{u}}_{p} \mathcal{Y}$ and $\mathcal{X} \not\preccurlyeq^{\mathrm{u}} \mathcal{Z}$. Then $\mathcal{Y} \not\preccurlyeq^{\mathrm{u}} \mathcal{Z}$.*

We conclude the introduction of compilation schemes with a formal definition under which circumstances we consider two formalisms to be equally expressive.

Definition 25 (Formalisms with same expressive power)**.** *We say that two planning formalisms $\mathcal{X}$ and $\mathcal{Y}$ have the* same expressive power *if $\mathcal{X} \preccurlyeq^{\mathrm{e}} \mathcal{Y}$ and $\mathcal{Y} \preccurlyeq^{\mathrm{e}} \mathcal{X}$. We write this as $\mathcal{X} \approx^{\mathrm{e}} \mathcal{Y}$.*

6

Relative Expressiveness of PDDL and Basic Action Theories

6.1 Restricted Basic Action Theories

Starting from a similar definition of BAT tasks, Eyerich et al. (2006) added several restrictions to achieve the same expressivity as PDDL. We formulate their restrictions more precisely and adapt them to our notion of BAT tasks.[1] In addition, we list another restriction (R4) that makes an implicit assumption of Eyerich et al. explicit.

Definition 26 (Restricted BAT task)**.** *A* restricted BAT task *(RBAT task)* $\Pi = \langle \langle \mathcal{S}, P, E \rangle, C, I, \gamma \rangle$ *is a BAT task that satisfies the following restrictions:*

R1 *The* usage of functions *is restricted as follows:*

1. *All situation-independent object functions in $\mathcal{S}$ must be constants, i.e., functions of sort $\epsilon \to$ object.*
2. *There are no functional fluents in $\mathcal{S}$.*

R2 *The* successor state axioms *in E are of a certain form. The successor state axiom of a relational fluent F fits the schema*

$$F(x_1, \ldots, x_n, do(a, s)) \leftrightarrow \left(\bigvee_{l=1}^{p} \alpha_l\right) \vee \left(F(x_1, \ldots, x_n, s) \wedge \neg \bigvee_{l=1}^{q} \delta_l\right) \tag{6.1}$$

for some $p, q \geq 0$, where the α_l and δ_l are of the form

$$\exists y_1, \ldots, y_m \left(a = A\left(y_1, \ldots, y_m\right) \wedge \varphi\right). \tag{6.2}$$

Each action function may appear as $a = A(y_1, \ldots, y_m)$ in at most one subformula α_l and in at most one subformula δ_l.

[1] The main difference is that we make the signature explicit.

R3 *The initial state specification I consists of exactly the following sentences:*

1. *For each relational fluent symbol F in $\mathcal{S}$ there is either an expression*

$$\neg F(x_1, \ldots, x_n, \mathbf{s_0}) \tag{6.3}$$

or an expression

$$F(x_1, \ldots, x_n, \mathbf{s_0}) \leftrightarrow (x_1 = \mathbf{c}_{11} \wedge \cdots \wedge x_n = \mathbf{c}_{1n}) \vee \cdots \vee (x_1 = \mathbf{c}_{m1} \wedge \cdots \wedge x_n = \mathbf{c}_{mn}), \tag{6.4}$$

where the $\mathbf{c}_{ij}$ are constants.

2. *There are analogous expressions for all situation-independent predicates in $\mathcal{S}$.*
3. *There are unique names axioms $\mathbf{c}_i \neq \mathbf{c}_j$ for each pair $\mathbf{c}_i$, $\mathbf{c}_j$ of different constant symbols of $\mathcal{S}_\Pi$ (the extension of $\mathcal{S}$ with the constant symbols C).*
4. *There is a* domain closure axiom

$$\bigvee_{\mathbf{c} \in C_\Pi} x = \mathbf{c}$$

for constants, where C_Π is the set of all constant symbols in $\mathcal{S}_\Pi$.

R4 *The usage of sort action is further restricted as follows:*

1. *Arguments of situation-independent relations or relational fluents must not be of sort action, i.e., each such relation R is of sort object$^{ar(r)}$ or object$^{ar(r)-1} \times$ situation, respectively.*
2. *The right-hand side of the action precondition axioms, the goal formula γ and the sub-expressions φ in restriction R2 must not contain any action symbols.*
3. *Arguments of action functions in $\mathcal{S}$ cannot be of sort action.*

Eyerich et al. have shown that their restrictions lead to the same expressive power as PDDL by giving compilation schemes that preserve plan size exactly in both directions (Eyerich et al., 2006; Eyerich, 2006). However, they did not show that these restrictions define a *maximal* fragment of the situation calculus with this expressive power.

The aim of this chapter is to identify such a maximal fragment in order to make planning systems maximally available to the Golog interpreter. Since we will only soften the given restrictions, the compilation from PDDL to RBAT by Eyerich et al. will still be a valid proof that this new fragment is at least as expressive as PDDL. However, for showing that it is not more expressive, we will need to extend the compilation from RBAT to PDDL.

Compiling RBAT to PDDL

To build a stable basis for this work, first we will introduce the compilation scheme by Eyerich et al. and formally prove its correctness.

This is particularly important since the full description of the compilation scheme from RBAT to PDDL is only available in German (Eyerich, 2006) and the original proof is very sketchy and a bit imprecise.

We will use that a uniform formula of the situation calculus that does not contain any function symbols except constants of sort *object* and (maybe) one situation, can be transformed to a first-order formula that fits the restrictions of PDDL by suppressing all situation terms. We write this transformation as $\theta(\varphi)$, e. g., $\theta(package(x) \wedge location(y) \wedge at(x, y, s)) = package(x) \wedge location(y) \wedge at(x, y)$.

Definition 27 (Compilation scheme from RBAT to PDDL)**.**
Let $\Pi = \langle \mathcal{D}, C, I, \gamma \rangle$ *be a RBAT task with* $\mathcal{D} = \langle \mathcal{S}, P, E \rangle$.

The compilation scheme $\mathbf{f}_{RBAT \to PDDL} = \langle f_d, t_c, t_i, t_g \rangle$ *from RBAT to PDDL is defined as follows:*

- *The function* f_d *maps* $\mathcal{D} = \langle \mathcal{S}, P, E \rangle$ *to the domain* $\langle \mathcal{L}, \mathcal{O} \rangle$ *of the PDDL task.*

  The language $\mathcal{L}$ *contains for each predicate symbol* R *of sort* $object^{ar(R)}$ *in* $\mathcal{S}$ *(for situation-independent relations) a predicate symbol* R *of the same arity. For each predicate symbol* F *of sort* $object^{ar(R)-1} \times situation$ *in* $\mathcal{S}$ *(a* relational fluent*),* $\mathcal{L}$ *contains a predicate symbol* F *of arity* $ar(R) - 1$*. The constant symbols of* $\mathcal{L}$ *are exactly the constant symbols of sort object in* $\mathcal{S}$*. In addition,* $\mathcal{L}$ *contains the standard logical connectives and the equation symbol* $=$*.*

  For the compilation of the successor state axioms and action precondition axioms to the actions $\mathcal{O}$*, the precondition of the actions are taken from the action precondition axioms whereas the effect of each action on the fluents is collected from the successor state axioms.*

  More precisely, the set of schematic operators $\mathcal{O}$ *contains exactly one operator* $\langle A, \theta(\Pi_A(\bar{z}, s)), e \rangle$ *for each action precondition axiom*

  $$poss(A(\bar{z}), s) \leftrightarrow \Pi_A(\bar{z}, s)$$

  in P*, where* e *is a conjunctive effect whose subeffects are extracted from the successor state axioms in* E*. For each successor state axiom*

  $$F(\bar{x}, do(a, s)) \leftrightarrow \big(\bigvee_{l=1}^{p} \alpha_l \big) \vee \big(F(\bar{x}, s) \wedge \neg \bigvee_{l=1}^{q} \delta_l \big)$$

  there are at most two subeffects:

  - *if there is a subformula* α_l *of the form* $\exists \bar{y} \, (a = A \, (\bar{y}) \wedge \varphi)$ *we assume w.l.o.g. that all bound variables are distinct from each other and the free variables and that no variable symbols from* $\bar{z}$ *occur in* α_l*. Otherwise we rename variables accordingly.*
    For α_l *effect* e *contains a subeffect*

    $$\forall \bar{x} (\theta(\varphi[\bar{y}/\bar{z}]) \triangleright F(\bar{x})),$$

where $[\bar{y}/\bar{z}]$ *for* $\bar{y} = y_0, \ldots, y_n$ *and* $\bar{z} = z_0, \ldots, z_n$ *means that each occurrence of variable* y_i *is replaced with variable* z_i $(0 \leq i \leq n)$*. If there is no* φ*, the effect condition is not included.*

– *if there is an analogous subformula* δ_l*, effect* e *contains an analogous subeffect*

$$\forall \bar{x}(\theta(\varphi[\bar{y}/\bar{z}]) \triangleright \neg F(\bar{x})).$$

- *The set of constants of the PDDL task is exactly the set of task-specific constants* C *of the RBAT task:* $t_c(\mathcal{D}, C) = C$*.*

- *The initial state of the PDDL task is the result of function* $t_i(\mathcal{D}, C, I)$ *which maps to the smallest set that contains for each expression*

$$R(x_1, \ldots, x_n, \mathbf{s_0}) \leftrightarrow (x_1 = \mathbf{c}_{11} \wedge \cdots \wedge x_n = \mathbf{c}_{1n}) \vee \cdots \vee (x_1 = \mathbf{c}_{m1} \wedge \cdots \wedge x_n = \mathbf{c}_{mn})$$

or

$$R(x_1, \ldots, x_n) \leftrightarrow (x_1 = \mathbf{c}_{11} \wedge \cdots \wedge x_n = \mathbf{c}_{1n}) \vee \cdots \vee (x_1 = \mathbf{c}_{m1} \wedge \cdots \wedge x_n = \mathbf{c}_{mn})$$

in I *the atoms* $R(\mathbf{c}_{11}, \ldots, \mathbf{c}_{1n}), \ldots, R(\mathbf{c}_{m1}, \ldots, \mathbf{c}_{mn})$*.*

- *The goal of the PDDL task is the goal of the BAT task with all situation arguments suppressed:* $t_g(\gamma) = \theta(\gamma)$*.*

Before we show the correctness of this compilation scheme, we prove four lemmas on the underlying structures:

Lemma 1. *Let* Π *be a RBAT and let* C_{all} *denote the set of all constant symbols in* $\mathcal{L}_\Pi$*. For every model of* $\mathcal{T}(\Pi)$*, i.e., every* domain of discourse $\mathcal{U}$ *and interpretation* $\mathcal{I}$ *for* $\mathcal{S}$ *such that* $\mathcal{U}, \mathcal{I} \models \mathcal{T}(\Pi)$*, the domain of discourse* $\mathcal{U}$ *contains exactly one element of sort object for each constant symbol in* C_{all}*. (It usually contains additional objects of sort action and situation.)*

Furthermore, $\mathcal{I}$ *restricted to the constant symbols of sort object is a bijection to the set* $\mathcal{U}_{object} \subset \mathcal{U}$ *that contains exactly the elements of sort object.*

Proof. From the domain closure axiom (restriction $R3.4$) we conclude that $\mathcal{U}$ contains at most one object for each constant in C_{all}. The unique names axioms for constants (restriction $R3.3$) ensure that all constants in C_{all} denote different objects, so the universe of the RBAT task contains *exactly* one object for each constant in C_{all}.

The unique names axioms for constants also imply that $\mathcal{I}|_{C_{all}}$ is injective. Since $|C_{all}| = |\mathcal{U}_{object}|$, it is also surjective to $\mathcal{U}_{object}$. □

Based on this lemma, we can relate the domain of discourse of the RBAT task to the domain of discourse of the PDDL task:

Lemma 2. *Let* Π *be a RBAT task and* $F(\Pi)$ *be the PDDL task resulting from the compilation defined in Definition 27.*

Let $\mathcal{U},\mathcal{I}$ *be a model of* $\mathcal{T}$ *and let* $\mathcal{U}_{object} \subset \mathcal{U}$ *be the subset of the domain of discourse that contains exactly the elements of sort object.*

All structures $\mathcal{A}_s = (\mathcal{U}',\mathcal{I}')$ *for a state* s *of PDDL task* $F(\Pi)$ *share the same domain of discourse* $\mathcal{U}'$ *and we can define a bijection* $o : \mathcal{U}_{object} \to \mathcal{U}'$.

Proof. Let C_{all} be the set of all constant symbols of sort *object* occurring in Π. From Lemma 1 we know that $\mathcal{I}|_{C_{all}} : C_{all} \to \mathcal{U}_{object}$ is bijective.

From the compilation functions f_d and t_c it is obvious that C_{all} consists exactly of the constant symbols in $\mathcal{L}_{F(\Pi)}$.

Let $\mathcal{A}_{s'} = (\mathcal{U}',\mathcal{I}')$ and $\mathcal{A}_{s''} = (\mathcal{U}'',\mathcal{I}'')$ be structures for states s' and s'' of $F(\Pi)$. By definition, $\mathcal{U}'$ and $\mathcal{U}''$ contain exactly one element $\mathbf{c}$ for each constant symbol c of $\mathcal{L}_{F(\Pi)}$ and, hence, $\mathcal{U}' = \mathcal{U}''$. Since $\mathcal{I}'(c) = \mathcal{I}''(c) = \mathbf{c}$ for all constant symbols c in $\mathcal{L}_{F(\Pi)}$, the interpretation $\mathcal{I}'|_{C_{all}} : C_{all} \to \mathcal{U}'$ is bijective and $\mathcal{I}'|_{C_{all}} = \mathcal{I}''|_{C_{all}}$.

As both $\mathcal{I}|_{C_{all}}$ and $\mathcal{I}'|_{C_{all}}$ are bijective, we can define the bijection o along the interpretation of the constant symbols in C_{all} as $o(x) = \mathcal{I}'(\mathcal{I}^{-1}(x))$. □

To prove the correctness of the compilation scheme, we want to base the truth evaluation of formulas only on the truth of ground atoms of the PDDL task and on its objects. For a PDDL task this information is obviously sufficient: the latter fixes the range of quantifiers, the former is exactly the information captured by a state. However, we can show that under certain restrictions (which the crucial formulas of RBAT task satisfy) the analogous information is sufficient to evaluate the truth of formulas in the RBAT task.

Lemma 3. *Let* Π *be a RBAT task,* $\mathcal{U},\mathcal{I},\alpha$ *be a model of* $\mathcal{T}(\Pi)$ *with variable assignment* α*, and let* φ *be a situation calculus formula for the same signature where*

- φ *is uniform in* s*,*
- s *is the only free variable of* φ *(or* φ *is closed),*
- φ *does not mention any function symbols apart from constants of sort object, and*
- φ *does not mention any symbols of sort action.*

To decide whether $\mathcal{U},\mathcal{I},\alpha \models \varphi$*, it is sufficient to know the set* C_{all} *of constant symbols of sort object in* Π *and for all parameter vectors* $\bar{c}$ *of such constant symbols and predicate symbols* $R \neq \sqsubset$ *whether*

- $\mathcal{U},\mathcal{I},\alpha|_{\{s\}} \models R(\bar{c},s)$ *if* R *is a relational fluent, or*
- $\mathcal{U},\mathcal{I},\alpha|_{\{s\}} \models R(\bar{c})$ *if* R *is a situation-independent predicate.*

Proof. We denote the set of constant symbols of sort *object* in the signature of Π with C_{all} and the subset of $\mathcal{U}$ that contains all elements of sort *object* with $\mathcal{U}_{object}$.

The truth value of φ can be determined from the truth values of all atoms $R(t_1, \ldots, t_n)$ and all identities $(t_1 = t_2)$ occurring in φ, where t_j are terms (for $j \in \{1, \ldots, n\}$).

The only terms that can occur in φ are constant symbols of sort *object*, bound occurrences of variables x of sort *object* and the variable s of sort *situation*.

In the semantics definition, bound variables are replaced with all possible elements of the universe of the same sort. For sort *object* these are exactly the elements of $\mathcal{U}_{object}$. By Lemma 1, $\mathcal{I}|_{C_{all}} : C_{all} \to \mathcal{U}_{object}$ is a bijection, so we alternatively can determine the truth value of quantified subfomulas of φ with

$$\mathcal{U}, \mathcal{I}, \alpha \models \exists x(\psi) \text{ iff exists } c \in C_{all} \text{ such that } \mathcal{U}, \mathcal{I}, \alpha \models \psi[x/c], \text{ and}$$
$$\mathcal{U}, \mathcal{I}, \alpha \models \forall x(\psi) \text{ iff for all } c \in C_{all} \text{ such that } \mathcal{U}, \mathcal{I}, \alpha \models \psi[x/c].$$

As a result, we alternatively can consider several formulas where all variable symbols of sort *object* have been replaced with constant symbols. Therefore we assume in the following that the only terms in φ are constant symbols of sort *object* or variable s.

Since $\mathcal{I}|_{C_{all}}$ is bijective it holds for identities $(c = c')$, that $\mathcal{U}, \mathcal{I} \models (c = c')$ iff c and c' are the same symbol.

For a situation-independent predicate $R(c_1, \ldots, c_n)$ it holds that $\mathcal{U}, \mathcal{I} \models R(c_1, \ldots, c_n)$ iff $(\mathcal{I}(c_1), \ldots, \mathcal{I}(c_n)) \in \mathcal{I}(R)$. For a fluent $R(c_1, \ldots, c_n, s)$ it holds that $\mathcal{U}, \mathcal{I}, \alpha \models R(c_1, \ldots, c_n, s)$ iff $(\mathcal{I}(c_1), \ldots, \mathcal{I}(c_n), \alpha(s)) \in \mathcal{I}(R)$.

Overall, the specified information is sufficient to decide whether a model and a variable assignment satisfy a formula with the given restrictions. □

Based on the previous lemmas we can show that for applicable action sequences these relevant truth values match for the situation of the RBAT task and the corresponding reached state of the PDDL task.

Lemma 4. *Let $\mathbf{f}_{RBAT \to PDDL} = \langle f_d, t_c, t_i, t_g \rangle$ be the compilation scheme from definition 27. Let Π be a RBAT task and $F(\Pi)$ be the result of the compilation.*

In the following, we denote the (ground) action of the PDDL task $F(\Pi)$ that has been introduced for action a of the RBAT task by a'. For an executable situation $s = do(a_n, \ldots do(a_2, do(a_1, \mathbf{s_0})) \ldots)$, we denote the corresponding PDDL state that is reached after applying action sequence $\langle a'_1, \ldots, a'_n \rangle$ with s'.

For all executable situations s it holds for all situation-independent ground predicates $R(\bar{c})$ that

$$\mathcal{T}(\Pi) \models R(\bar{c}) \text{ iff } s'(R(\bar{c})) = 1,$$

and for all relational ground fluents $R(\bar{c}, s)$ that

$$\mathcal{T}(\Pi) \models R(\bar{c}, s) \text{ iff } s'(R(\bar{c})) = 1.$$

In addition, the set of possible actions (w.r.t. the action precondition axioms) in an executable situation s corresponds to the set of applicable actions in the corresponding state s'.

Proof. Let $\Pi = \langle \mathcal{D}, C, I, \gamma \rangle$ be a RBAT task with $\mathcal{D} = \langle \mathcal{S}, P, E \rangle$ and let $F(\Pi) = \langle \mathcal{D}', C', I', \gamma' \rangle$ be the result of the compilation. We show the claim on the

predicate values by induction over the executable situations. The proof of the correspondence of the applicable actions is part of the induction step.

Base case: The initial database I contains for each relational fluent symbol R an expression:

$$R(x_1, \ldots, x_n, \mathbf{s_0}) \leftrightarrow (x_1 = \mathbf{c}_{11} \wedge \cdots \wedge x_n = \mathbf{c}_{1n}) \vee \cdots \vee (x_1 = \mathbf{c}_{m1} \wedge \cdots \wedge x_n = \mathbf{c}_{mn}) .$$

Hence, $\mathcal{T}(\Pi) \models R(c_1, \ldots, c_n, \mathbf{s_0})$ if and only if $(c_1, \ldots, c_n) \in \{(\mathbf{c}_{i1}, \ldots, \mathbf{c}_{in}) \mid i \in \{1, \ldots, m\}\}$. By the definition of t_i, these are exactly the tuples for which $R(c_1, \ldots, c_n) \in I'$, so we can conclude for the initial state s'_0 of $F(\Pi)$ that $s'_0(R(c_1, \ldots, c_n)) = 1$ iff $\mathcal{T}(\Pi) \models R(c_1, \ldots, c_n, \mathbf{s_0})$. We can proof the base case for the situation-independent predicates analogously.

Induction hypothesis: For the executable situation s, it holds for all situation-independent ground predicates $R(\bar{c})$ that $\mathcal{T}(\Pi) \models R(\bar{c})$ iff $s'(R(\bar{c})) = 1$, and for all relational ground fluents $R(\bar{c}, s)$ that $\mathcal{T}(\Pi) \models R(\bar{c}, s)$ iff $s'(R(\bar{c})) = 1$.

Induction step: $s \mapsto do(a, s)$

We first show that the set of actions that are applicable in s (according to the action precondition axioms) corresponds to the set of actions applicable in s': We have previously (Lemma 1) shown that every element of the domain of discourse can be denoted by a constant symbol of sort *object*, so let w.l.o.g. $a = A(\mathbf{c}_1, \ldots, \mathbf{c}_n)$ with arbitrary constants $\mathbf{c}_i$. Consider the action precondition axiom $poss(A(x_1, \ldots, x_n), s) \leftrightarrow \Pi_A(x_1, \ldots, x_n, s)$ for action a. In the following, we denote the precondition $\Pi_A(x_1, \ldots, x_n, s)[x_1/\mathbf{c}_1, \ldots, x_n/\mathbf{c}_n]$ by χ. From the definition of f_d we know that the precondition of a' is $\chi' := \theta(\Pi_A(x_1, \ldots, x_n, s))[x_1/\mathbf{c}_1, \ldots, x_n/\mathbf{c}_n]$. Since χ does not contain any action symbols (by restriction $R4.2$), is uniform in s and has s as its only free variable, it follows from Lemma 3 and the induction hypothesis that $\mathcal{T}(\Pi) \models \chi$ iff $s' \models \chi'$. Hence, an arbitrary action a is applicable in an executable situation s iff a' is applicable in s'.

Since this induction ranges only over executable situations $do(a, s)$, we can conclude that a' is applicable in s'. We now show that the induction hypothesis also holds for $do(a, s)$ and $app_{a'}(s')$.

The proof for the situation-independent predicates is trivial: Let $R(\bar{c})$ be such a predicate. Since no action of the PDDL task changes the value of $R(\bar{c})$ (by the definition of f_d), we know that $app_{a'}(s')(R(\bar{c})) = 1$ iff $s'(R(\bar{c})) = 1$. So we conclude with the induction hypothesis that $\mathcal{T}(\Pi) \models R(\bar{c})$ if and only if $app_{a'}(s')(R(\bar{c})) = 1$.

For the proof for the relational fluents, let $R(\bar{c}, s)$ be such a fluent and let $a = A(\bar{\mathbf{c}})$. Consider the successor state axiom

$$R(\bar{x}, do(a, s)) \leftrightarrow \big(\bigvee_{l=1}^{p} \alpha_l\big) \vee \big(R(\bar{x}, s) \wedge \neg \bigvee_{l=1}^{q} \delta_l\big).$$

According to restriction $R2$, there is at most one α_l for action symbol A of the form $\exists \bar{y}\,(a = A\,(\bar{y}) \wedge \varphi_\alpha)$ and at most one δ_l of the form $\exists \bar{y}\,(a = A\,(\bar{y}) \wedge \varphi_\delta)$. We denote these with α_{l^*} and δ_{l^*}, respectively. For the rest of this proof, we assume w.l.o.g. that there actually are such α_{l^*} and β_{l^*}. (If there are not, we could add them with a trivially false φ_α or φ_δ respectively, only increasing the size of the domain description polynomially.)

Since for all other α_l and β_l it holds that $\mathcal{T}(\Pi) \models \neg\alpha_l[a/A(\bar{\mathbf{c}})]$ and $\mathcal{T}(\Pi) \models \neg\beta_l[a/A(\bar{\mathbf{c}})]$, we conclude that

$$\begin{aligned}\mathcal{T}(\Pi) \models R(\bar{x}, do(A(\bar{\mathbf{c}}), s)) \\ \text{iff } \mathcal{T}(\Pi) \models \alpha_{l^*} \vee (R(\bar{x}, s) \wedge \neg\delta_{l^*}) \\ \text{iff } \mathcal{T}(\Pi) \models \exists\bar{y}(A(\bar{\mathbf{c}}) = A(\bar{y}) \wedge \varphi_\alpha) \vee \\ (R(\bar{x}, s) \wedge \neg\exists\bar{y}\,(A(\bar{\mathbf{c}}) = A\,(\bar{y}) \wedge \varphi_\delta)).\end{aligned} \tag{6.5}$$

We can conclude from the unique names axiom

$$A(x_1, \ldots, x_n) = A(y_1, \ldots, y_n) \rightarrow x_1 = y_1 \wedge \cdots \wedge x_n = y_n$$

that $\mathcal{T}(\Pi) \models \exists\bar{y}(A(\bar{\mathbf{c}}) = A(\bar{y}) \wedge \varphi)$ iff $\mathcal{T}(\Pi) \models \varphi[\bar{y}/\bar{\mathbf{c}}]$. With equation (6.5) this results in

$$\mathcal{T}(\Pi) \models R(\bar{x}, do(A(\bar{\mathbf{c}}), s)) \text{ iff } \mathcal{T}(\Pi) \models \varphi_\alpha[\bar{y}/\bar{\mathbf{c}}] \vee (R(\bar{x}, s) \wedge \neg\varphi_\delta[\bar{y}/\bar{\mathbf{c}}]). \tag{6.6}$$

For the predicate $R(\bar{c}, do(a, s))$ this leads to

$$\begin{aligned}\mathcal{T}(\Pi) \models R(\bar{c}, do(A(\bar{\mathbf{c}}), s)) \text{ iff } \mathcal{T}(\Pi) \models \varphi_\alpha[\bar{y}/\bar{\mathbf{c}}, \bar{x}/\bar{c}] \vee \\ (R(\bar{c}, s) \wedge \neg\varphi_\delta[\bar{y}/\bar{\mathbf{c}}, \bar{x}/\bar{c}]).\end{aligned} \tag{6.7}$$

On the PDDL side, action a' is an instantiation of a schematic operator $\langle A, \chi, e\rangle$ with parameters $\bar{z}$. By the definition of f_d, e is a conjunction of universal effects of which two subeffects can affect predicate $R(\bar{x})$:

$$\begin{aligned}e_\alpha &:= \forall\bar{x}(\theta(\varphi_\alpha[\bar{y}/\bar{z}]) \triangleright R(\bar{x})) \text{ and} \\ e_\delta &:= \forall\bar{x}(\theta(\varphi_\delta[\bar{y}/\bar{z}]) \triangleright \neg R(\bar{x})).\end{aligned}$$

For the instantiated action $A(\bar{\mathbf{c}})$, we use for the effect conditions of the subeffects affecting $R(\bar{c})$ the notation

$$\begin{aligned}\alpha' &:= \theta(\varphi_\alpha[\bar{y}/\bar{z}])[\bar{z}/\bar{\mathbf{c}}, \bar{x}/\bar{c}] \text{ and} \\ \delta' &:= \theta(\varphi_\delta[\bar{y}/\bar{z}])[\bar{z}/\bar{\mathbf{c}}, \bar{x}/\bar{c}].\end{aligned}$$

So action a' makes predicate $R(\bar{c})$ true when applied in state s' iff $s' \models \alpha'$. It makes the predicate false iff $s' \models \delta' \wedge \neg\alpha'$ (add-after-delete semantics). In all other cases applying action a' leaves $R(\bar{c})$ unchanged: $app_{a'}(s')(R(\bar{c})) = s'(R(\bar{c}))$ iff $s' \not\models \alpha' \vee \delta'$. Put differently,

$$app_{a'}(s')(R(\bar{c})) = 1 \text{ iff } s' \models \alpha' \text{ or } (s'(R(\bar{c})) = 1 \text{ and } s' \not\models \delta'). \tag{6.8}$$

Since $\theta(\cdot)$ only suppresses the situation term s, we conclude with the bijection from Lemma 2 and the induction hypothesis that $s' \models \alpha'$ iff $\mathcal{T}(\Pi) \models \varphi_\alpha[\bar{y}/\bar{\mathbf{c}}, \bar{x}/\bar{c}]$ and that $s' \models \neg\delta'$ iff $\mathcal{T}(\Pi) \models \neg\varphi_\delta[\bar{y}/\bar{\mathbf{c}}, \bar{x}/\bar{c}]$. From the induction hypothesis we also know that $s'(R(\bar{c})) = 1$ iff $\mathcal{T}(\Pi) \models R(\bar{c}, s)$, so we conclude with equation 6.8 that

$$\begin{aligned}app_{a'}(s')(R(\bar{c})) = 1 \text{ iff } \mathcal{T}(\Pi) \models \varphi_\alpha[\bar{y}/\bar{\mathbf{c}}, \bar{x}/\bar{c}] \text{ or} \\ (\mathcal{T}(\Pi) \models R(\bar{c}, s) \text{ and } \mathcal{T}(\Pi) \models \neg\varphi_\delta[\bar{y}/\bar{\mathbf{c}}, \bar{x}/\bar{c}]) \\ \text{iff } \mathcal{T}(\Pi) \models \varphi_\alpha[\bar{y}/\bar{\mathbf{c}}, \bar{x}/\bar{c}] \vee \\ (R(\bar{c}, s) \wedge \neg\varphi_\delta[\bar{y}/\bar{\mathbf{c}}, \bar{x}/\bar{c}])\end{aligned} \tag{6.9}$$

From equations 6.7 and 6.9 follows directly that

$$\mathcal{T}(\Pi) \models R(\bar{c}, do(a, s)) \text{ iff } app_{a'}(s')(R(\bar{c})) = 1,$$

which finishes the proof for the relational fluents. □

After the somewhat tedious proofs of these lemmas, we can now show that the compilation scheme from Definition 27 is actually correct.

Theorem 6 (Correctness of $\mathbf{f}_{\text{RBAT}\to\text{PDDL}}$)**.** *$\mathbf{f}_{RBAT\to PDDL}$ is a compilation scheme from restricted basic action theories to PDDL preserving plan size exactly.*

Proof. Let $\Pi = \langle \mathcal{D}, C, I, \gamma \rangle$ be a RBAT task with $\mathcal{D} = \langle \mathcal{S}, P, E \rangle$ and let $F(\Pi) = \langle \mathcal{D}', C', I', \gamma' \rangle$ be the result of the compilation. We show that each plan of Π can directly be transformed in a plan of $F(\Pi)$ and vice versa. We use the same notation as in Lemma 4.

"⇒": Let $s = do(a_n, \dots, do(a_1, do(a_0, \mathbf{s_0}))\dots)$ be a plan for Π. Then, it holds by definition that $\mathcal{T}(\Pi) \models \mathit{executable}(s) \wedge \gamma(s)$. From Lemma 4 we know that when s is executable then state $s' := app_{\langle a'_0,\dots,a'_n\rangle}(s_I)$ is reachable from I. In addition, the truth values of the predicates of the situation and the state comply. Since γ satisfies the requirements of Lemma 3 (due to restriction $R4.2$), we can conclude that $s' \models \theta(\gamma)$. Since, by the compilation scheme, the goal of $F(\Pi)$ is $\theta(\gamma)$, the action sequence $a'_0, \dots, a'_n$ is a plan for $F(\Pi)$.

"⇐": Let $a'_0, \dots, a'_n$ be a plan for $F(\Pi)$ that reaches a goal state s'. Let a_i denote the RBAT action for which each action a'_i has been introduced by f_d. From Lemma 4 we know that $s := do(a_n, \dots, do(a_1, do(a_0, \mathbf{s_0}))\dots)$ is executable. We further know that the truth values of the predicates for s and s' comply in the sense of Lemma 4. By the same argumentation as in the other direction this implies that $\mathcal{T}(\Pi) \models \gamma(s)$, so s is a plan for Π.

Hence, $\mathbf{f}_{\text{RBAT}\to\text{PDDL}}$ is solution-preserving and preserves plan size exactly. Further, it is trivial to check that all functions of the compilation scheme are polynomial-time computable, which also implies that the size of their result is polynomial in the size of the input. □

In the rest of the chapter, we will inspect the restrictions by Eyerich et al. (2006) whether they are really necessary for the same expressivity as PDDL or whether they can be softened. We begin with the restrictions on the predicate specifications in the initial database.

6.2 Predicate Specifications in the Initial Database

Restrictions R3.1 and R3.2 require that the initial database enumerates all initially true ground atoms for the relational fluents and the situation-independent predicates (no space-saving representation):

$$\begin{aligned} F(x_1, \dots, x_n, \mathbf{s_0}) \leftrightarrow (x_1 = \mathbf{c}_{11} \wedge \dots \wedge x_n = \mathbf{c}_{1n}) \vee \dots \vee \\ (x_1 = \mathbf{c}_{m1} \wedge \dots \wedge x_n = \mathbf{c}_{mn}), \end{aligned} \tag{6.4}$$

There are two ways of loosening these restrictions: firstly, there could be no such expression for some of the predicates (leaving their interpretation unspecified) and secondly, less restricted formulas which still guarantee a unique model could be allowed in the predicate specifications.

Less Restricted Formulas in the Specification of the Predicates

We begin with the examination of less restricted formulas on the right-hand side of equivalence (6.4). One possibility is to allow arbitrary first-order formulas that are uniform in $\mathbf{s_0}$ on the right-hand side of equivalence (6.4). This would also permit "recursive" definitions which are known to be non-compilable (Thiébaux et al., 2005). However, we can actually prove a stronger result which holds for *acyclic* specifications, i.e., for the case where there is a strict order $<$ on the set of predicate symbols such that each specification of a predicate P depends only on atoms of predicates P' with $P' < P$.

Consider a requirement that replaces restriction R3.2 and allows acyclic specifications:

R3.2' For each n-ary situation-independent predicate P there is an expression

$$P(x_1, \ldots, x_n) \leftrightarrow \varphi_P(x_1, \ldots, x_n), \tag{6.10}$$

where φ_P is a formula uniform in $\mathbf{s_0}$ whose free variables are among $x_1, \ldots, x_n$. Further, there must exist a strict order $<$ on the set of predicates such that in each formula φ_P only occur atoms of predicates P' with $P' < P$.

We denote the resulting formalism by $\mathrm{RBAT}_{compact}$ and show that there is probably no compilation scheme from RBAT to PDDL restricting the plan size.

Theorem 7 (R3.2 is necessary)**.** *Let restriction R3.2 of RBAT be replaced by restriction R3.2'. Unless* PSPACE = P *there is no compilation scheme from the resulting formalism* $RBAT_{compact}$ *to PDDL that restricts the increase of the plan size:* $RBAT_{compact} \not\preceq^{\mathrm{r}} PDDL$.

Proof. Consider the following class of planning tasks (example shown in Figure 6.1): There is one unary situation-independent predicate *isTrue* and a 0-ary predicate *goal*, but no fluent and no action. We use two constants **T** and **F** for which *isTrue* is initialized as

$$isTrue(x) \leftrightarrow x = \mathbf{T}. \tag{6.11}$$

Further, the initial database contains the domain closure axiom and the unique names axioms for constants. The goal formula requires predicate *goal*() to be true. Thus, the only aspect that differs from task to task is the specification of predicate *goal* in the initial database.

Let $\mathbf{f} = \langle f_d, t_c, t_i, t_g \rangle$ be a compilation scheme to PDDL that restricts plan size. As the domain and the goal description are fixed, the results of f_d and t_g are fixed and provide us a fixed PDDL domain and a fixed goal. As there are no actions in the original instance, plans of the original instance have a length bound of 0. Since $\mathbf{f}$ restricts the plan size by some function, this implies a constant bound k on shortest plan lengths for tasks of the PDDL domain. Note that in this setting (fixed domain, fixed goal, constant length bound), PDDL plan existence can be decided in P (Theorem 3).

With this compilation scheme $\mathbf{f}$ we can decide the quantified Boolean formula problem, which is PSPACE-complete (Stockmeyer and Meyer, 1973), in

polynomial time: Consider a quantified Boolean formula φ. We convert φ to a first-order formula φ' by substituting each occurrence of a variable x by $\mathit{IsTrue}(x)$. Predicate *goal* is then initialized as

$$\mathit{goal}() \leftrightarrow \varphi'. \tag{6.12}$$

Formula φ is obviously satisfied iff *goal*() (which is also the goal of the BAT task) is satisfied. We can now use the compilation scheme **f** to translate the initial database to an initial state (and a set of constants) for the PDDL task and test (k-bounded) plan existence in polynomial time. With the quantified Boolean formula problem being PSPACE-complete, this implies PSPACE = P. □

$\Pi = \langle \langle \mathcal{S}, P, E \rangle, C, I, \gamma \rangle$ is a BAT task with the following components:

- $\mathcal{S}$ defines a 0-ary predicate symbol *goal* and
  a unary predicate symbol *isTrue* with an argument of sort *object*.
- $P = \{\}$
- $E = \{\}$
- $C = \{\mathbf{T}, \mathbf{F}\}$
- I consists of the sentences:

$$\mathit{isTrue}(x) \leftrightarrow x = \mathbf{T},$$
$$x = \mathbf{T} \vee x = \mathbf{F},$$
$$\mathbf{T} \neq \mathbf{F},$$
$$\mathit{goal}() \leftrightarrow \forall x \exists y \forall z \left((\mathit{isTrue}(x) \wedge \mathit{isTrue}(y) \wedge \mathit{isTrue}(z)) \vee \neg \mathit{isTrue}(z) \right)$$

- $\gamma = \mathit{goal}()$

Figure 6.1: Example task from the family mentioned in the proof of Theorem 7 for the quantified Boolean formula $\forall x \exists y \forall z \left((x \wedge y \wedge z) \vee \neg z \right)$. All tasks of the family differ only in the last sentence of the initial database.

There are surely other possibilities of weakening the restrictions on the right-hand side of equivalence (6.4). Every specification which defines a unique model whose true ground atoms can be enumerated in polynomial time can clearly be compiled. Even in cases with more than polynomially many true ground atoms, it is often possible to compile a space-saving representation by means of initializing actions. However, examining this in more detail does not seem worthwhile to us.

Incomplete Information about the Truth Values of the Predicates

A second possibility of weakening restriction R3.2 is to allow to omit a specification for a given predicate P altogether, so that the plan must work for all

interpretations of P. This is both interesting in its own right and useful for the discussion of unique names axioms in the following section.

We can show that we lose compilability to PDDL even if there may only be a single unary situation-independent predicate whose truth values are not specified in the initial database. In the following, we denote the resulting formalism by RBAT_{incomp}.

We begin our argumentation with a lemma on the complexity of the plan existence problem for this formalism. The proof is a variation of Rintanen's (2004) proof on the complexity of propositional planning without observability.

Lemma 5. *The plan existence problem for $RBAT_{incomp}$ is* EXPSPACE*-hard.*

Proof. Let $M = \langle \Sigma_M, \Gamma_M, Q, \delta, \square, q_0, Q_{acc} \rangle$ be a deterministic Turing machine with space bound $2^{n^k} - 1$ for input strings σ of length n. We denote the i-th symbol of σ by σ_i.

We use constants $\mathbf{c} \in \Gamma_M$ for the tape alphabet of the Turing machine, $\mathbf{q} \in Q$ for the states, and $\blacktriangleleft$, $\blacktriangledown$, and $\blacktriangleright$ for the movements of the R/W head.

The key idea is to keep track of only one randomly chosen tape cell (the *watched* tape cell) and to ensure that the Turing machine is reliably simulated relative to this tape cell. The cell is picked by means of the unspecified unary predicate. As a plan must work for all models, the Turing machine must consequently be reliably simulated for the entire tape.

For the selection of the watched tape cell we use n^k constants $\mathbf{p}_0, \ldots, \mathbf{p}_{n^k-1}$ and an (unspecified) unary predicate *watched*. We use the fact that we can encode numbers from 0 to $2^{n^k} - 1$ with n^k bits and interpret the truth value of $watched(\mathbf{p}_i)$ as the value of the i-th bit of the number of the watched tape cell. We will later use the constants $\mathbf{p}_i$ analogously for the position of the R/W head. To distinguish these constants from other ones we introduce a situation-independent predicate *index* with the following expression in the initial database:

$$index(p) \leftrightarrow \bigvee_{0 \leq i < n^k} p = \mathbf{p}_i$$

During the simulation of the Turing machine a relational fluent $contains(c, s)$ will keep track of the content of the watched tape cell. As the watched tape cell differs from model to model, we cannot specify the value of this predicate in the initial database but must rather use an additional initializing action *initialize*. To ensure the execution of this action we introduce an auxiliary relational fluent $initialized(s)$ which must be true before executing any other action. In the initial situation this predicate is naturally false:

$$\neg initialized(\mathbf{s_0}) \tag{6.13}$$

After the execution of action *initialize* it will stay true forever:

$$initialized(do(a, s)) \leftrightarrow initialized(s) \vee a = initialize() \tag{6.14}$$

The initializing action should not be used twice which leads to the following action precondition axiom:

$$Poss(initialize(), s) \leftrightarrow \neg initialized(s) \tag{6.15}$$

For the actual initialization of *contains* we need n more constants $\mathbf{d}_1, \ldots, \mathbf{d}_n$ which can be understood as numbers 1 to n. A situation-independent predicate *binDec* is used to encode the relationship between the decimal and binary representation of the numbers $1, \ldots, n$. For this purpose we define an auxiliary set *BD*, which contains exactly the pairs for which *binDec* will be true.

$$BD = \{(\mathbf{p}_j, \mathbf{d}_i) \mid 1 \leq i \leq n, 0 \leq j < \lceil \log n \rceil, \text{ the } j\text{-th symbol in the binary representation of } i \text{ is } 1\} \tag{6.16}$$

This results in the following expression for *binDec* in the initial database:

$$\mathit{binDec}(p, d) \leftrightarrow \bigvee_{(\mathbf{p}_j, \mathbf{d}_i) \in BD} (p = \mathbf{p}_j \wedge d = \mathbf{d}_i) \tag{6.17}$$

We record the input of the Turing machine via a situation-independent predicate *input*:

$$\mathit{input}(d, c) \leftrightarrow \bigvee_{1 \leq i \leq n} (d = \mathbf{d}_i \wedge c = \sigma_i) \tag{6.18}$$

Now we can have a closer look at the fluent $\mathit{contains}(c, s)$ that denotes the content of the watched tape cell in each situation. As mentioned above, we cannot determine the correct value in the initial situation and, thus, assume it to be false for all objects and therefore also for all members of Σ_M:

$$\neg \mathit{contains}(c, \mathbf{s_0}) \tag{6.19}$$

There are two actions affecting this predicate: *initialize* and an action *step* which simulates a step of the Turing machine and on which we will comment below. For the moment, we denote the condition under which *step* makes *contains* true by $\gamma^{\mathit{contains}}_{\mathit{step}^+}$ and the one making it false by $\gamma^{\mathit{contains}}_{\mathit{step}^-}$. Below we will explain these expressions in detail, now we concentrate on the effect of action *initialize*: after the initialization, predicate *contains* is true for a symbol c if one of the first n cells is watched and c corresponds to the input of this cell, or if another cell is watched and $c = \square$. For better readability we use an abbreviatory predicate $\mathit{watches}(d)$ which is not part of the basic action theory but must be substituted by the expression given in equation (6.21). This predicate is true for constant $\mathbf{d}_i$ if the i-th tape cell is watched.

$$\begin{aligned} \mathit{contains}(c, \mathit{do}(a, s)) \leftrightarrow\ & a = \mathit{initialize}() \wedge \\ & \big(\exists d\,(\mathit{watches}(d) \wedge \mathit{input}(d, c)) \vee \\ & \quad \forall d\,(\neg \mathit{watches}(d) \wedge c = \square)\big) \vee \\ & \gamma^{\mathit{contains}}_{\mathit{step}^+} \vee \\ & \mathit{contains}(c, s) \wedge \neg \gamma^{\mathit{contains}}_{\mathit{step}^-} \end{aligned} \tag{6.20}$$

$$\mathit{watches}(d) \leftrightarrow \forall p(\mathit{index}(p) \rightarrow (\mathit{watched}(p) \leftrightarrow \mathit{binDec}(p, d))) \tag{6.21}$$

Each step of the Turing machine is simulated by an action $\mathit{step}(c, q, c', q', d)$ which corresponds to a transition $\delta(c, q) = (c', q', d)$. Before we can state the

precondition axiom for this action we first have to introduce another relational fluent $state(q, s)$ denoting the state of the Turing machine in each situation. In the initial situation the Turing machine is in state $\mathbf{q_0}$, which leads to the following expression in the initial database:

$$state(q, \mathbf{s_0}) \leftrightarrow q = \mathbf{q_0} \tag{6.22}$$

A step of the Turing machine cannot be performed if the initialization step has not yet been done. In addition, each step must conform to the transition function δ which implies that the Turing machine must be in the correct state. The content of the tape cell is only verified for the single watched cell. In the following we will repeatedly need an expression stating that the R/W head stands on the watched cell in situation s. Thus, for better readability we use an abbreviatory predicate $readsWatched(s)$ which is not part of the basic action theory but must be substituted by the expression given in equation (6.30).

$$\begin{aligned} Poss(step(c, q, c', q', d), s) \leftrightarrow\ & initialized(s) \wedge state(q, s) \wedge \\ & \bigvee\nolimits_{\delta(\mathbf{c},\mathbf{q})=(\mathbf{c'},\mathbf{q'},\mathbf{d})} (c = \mathbf{c} \wedge q = \mathbf{q} \wedge c' = \mathbf{c'} \wedge q' = \mathbf{q'} \wedge d = \mathbf{d}) \wedge \\ & (readsWatched(s) \rightarrow contains(c, s)) \end{aligned} \tag{6.23}$$

The change of the state is then sufficiently described by the penultimate parameter of *step*:

$$\begin{aligned} state(q, do(a, s)) \leftrightarrow\ & \exists c, q', c', d(a = step(c, q', c', q, d)) \vee \\ & state(q, s) \wedge \neg\exists c, c', q', d(a = step(c, q, c', q', d) \wedge \neg(q' = q)) \end{aligned} \tag{6.24}$$

Above the influence of *step* on *contains* has been abbreviated as $\gamma^{contains}_{step^-}$ and $\gamma^{contains}_{step^+}$. Now we can describe them in detail. Expression $\gamma^{contains}_{step^-}$ covers the case where the watched tape cell contains the symbol c in situation s but no longer does after applying action *step*. This is the case if the R/W head stands on the watched cell and the action writes a symbol different from c onto the tape.

$$\gamma^{contains}_{step^-} \leftrightarrow \exists q, c', q', d(a = step(c, q, c', q', d) \wedge readsWatched(s) \wedge \neg(c = c')) \tag{6.25}$$

Conversely, the watched cell surely contains symbol c if the R/W head is positioned there and action *step* writes symbol c.

$$\gamma^{contains}_{step^+} \leftrightarrow \exists c', q, q', d(a = step(c', q, c, q', d) \wedge readsWatched(s)) \tag{6.26}$$

A remaining open aspect is the movement of the R/W head. As indicated above, we use the same encoding for the watched tape cell and the position of the R/W head but in the latter case the value can change. Thus, we use a fluent $pos(p, s)$ which denotes this position. The Turing machine starts on the first tape cell, hence, in the initial situation the predicate is false for all constants:

$$\neg pos(p, \mathbf{s_0}) \tag{6.27}$$

Before we can define the successor state axiom for *pos* we first require an auxiliary predicate $less(p, p')$ denoting that p represents a less significant bit than p'.

$$less(p, p') \leftrightarrow \bigvee_{0<i<n^k, 0\leq j<i} (p = \mathbf{p}_j \wedge p' = \mathbf{p}_i) \tag{6.28}$$

When incrementing a binary number, one has to flip exactly the bits for which all less significant bits have been 1. When decrementing, the less significant bits must have been 0. If a step moves the R/W head to the right, we have to increment the position. If it moves it to the left, we decrement it.

$$\begin{aligned} pos(p, do(a, s)) \leftrightarrow \exists c, q, c', q', d(a = step(c, q, c', q', d) \wedge index(p) \wedge \\ ((d = \blacktriangleright \wedge \forall p'(less(p', p) \rightarrow pos(p', s))) \vee \\ (d = \blacktriangleleft \wedge \forall p'(less(p', p) \rightarrow \neg pos(p', s))))) \vee \\ pos(p, s) \wedge \neg\exists c, q, c', q', d(a = step(c, q, c', q', d) \wedge \\ ((d = \blacktriangleright \wedge \forall p'(less(p', p) \rightarrow pos(p', s))) \vee \\ (d = \blacktriangleleft \wedge \forall p'(less(p', p) \rightarrow \neg pos(p', s))))) \end{aligned} \tag{6.29}$$

Having introduced predicate *pos* we now can define the expression substituting predicate *readsWatched*:

$$readsWatched(s) \leftrightarrow \forall p(index(p) \rightarrow (pos(p, s) \leftrightarrow watched(p))) \tag{6.30}$$

In addition to these expressions we also add the domain closure axiom, the unique names axioms, and the specification for the special predicates to the initial database.

The goal description must be formulated such that there is a plan iff the Turing machine accepts the input. Thus, we have to test whether it reaches an accepting state:

$$\bigvee_{\mathbf{q} \in Q_{acc}} state(\mathbf{q}, s) \tag{6.31}$$

Note that we do not check *initialized* in the goal: if the goal is true in a situation s but $initialized(s)$ is not, this implies that $\mathbf{q}_0 \in Q_{acc}$ (and s must be the initial situation). As such a Turing machine accepts every input, this is a reliable simulation.

In every other case each plan must start with action *initialize*. The explanations above should have made it clear that after this action predicate *contains* conforms with the content of the watched tape cell. Obviously, each action $step(c, q, c', q', d)$ corresponds to a transition $\delta(c, q) = (c', q', d)$ and the Turing machine is reliably simulated relative to the watched tape cell. Note that the transitions are exactly defined by the parameters of *step*, thus, a plan describes the same behavior of the Turing machine for all models. As a plan must hold for all models (regardless of the watched tape cell), the Turing machine must consequently be reliably simulated for the entire tape. Hence, there is a plan iff the execution of the Turing machine leads to an accepting state. □

Lemma 5 directly leads us to the theorem we actually want to show:

Theorem 8 (R3.2 is necessary)**.**
There is no compilation scheme from $RBAT_{incomp}$ to PDDL: $RBAT_{incomp} \not\preceq^{\mathrm{u}}$ PDDL.

Proof. Let M be a Turing machine with space bound 2^{n^k} that accepts an EXPSPACE-hard language. Let $\mathcal{D} = \langle \mathcal{S}, P, E \rangle$ be the domain description of the basic action theory for M according to the proof of Lemma 5. Note that in this formulation the actual input of the Turing machine (and also its length)

only influences the set of constants and the initial database while the domain description is not affected.

Assume there is a compilation scheme $\mathbf{f} = \langle f_d, t_c, t_i, t_g \rangle$ to PDDL. Then there exists a PDDL domain description $f_d(\mathcal{D})$ corresponding to M. We could, in polynomial time, translate each input of the Turing machine to a set of constants and an initial state such that there is a plan iff M halts. As the plan existence problem for PDDL with a fixed domain is in PSPACE (cf. Theorem 2) this implies that EXPSPACE $\subseteq$ PSPACE, which is known to be false. □

The results in this section apply to the specification of situation-independent predicates. Obviously they directly carry over to the specification of relational fluents: we can adapt the proofs by transforming the situation-independent predicates to relational fluents that do not change their truth values.

Corollary 1 (R3.1 is necessary)**.** *Omitting restriction R3.1 adds expressive power to RBAT.*

There are two remaining restrictions concerning the initial database: the domain closure axiom and the unique names axioms for constants. As the latter restriction is closely related to the previous theorem, we will examine it first.

6.3 Unique Names Axioms for Constants

Restriction R3.3 requires that the initial database contains a unique names axiom $c_i \neq c_j$ for each pair of different constant symbols c_i, c_j.

We will show that omitting this restriction leads to the same expressivity as permitting a single unary unspecified predicate in the initial database. In the previous section we have seen that the resulting formalism cannot be compiled to PDDL.

We denote the formalism resulting from RBAT by omitting restriction R3.3 by RBAT_{noUNA}. The restriction is weakened in such a way that there can be unique names axioms for some pairs of constants.

Lemma 6 ($\text{RBAT}_{incomp} \preccurlyeq_p^{\text{e}} \text{RBAT}_{noUNA}$)**.** *There is a polynomial-time compilation scheme from $RBAT_{incomp}$ to $RBAT_{noUNA}$ preserving plan size exactly.*

Proof. Let $\Pi = \langle \mathcal{D}, C, I, \gamma \rangle$ be a RBAT_{incomp} task with $\mathcal{D} = \langle \mathcal{S}, P, E \rangle$ and let R be its unspecified unary predicate. The key idea is that the compilation scheme $\mathbf{f} = \langle f_d, t_c, t_i, t_g \rangle$ replaces atoms $R(x)$ where x denotes the same object as constant $\mathbf{c}$ by an equality test for two new constants $\mathbf{c}'$ and $\mathbf{c}''$. In addition, we have to prevent the usage of the newly introduced objects everywhere else. We do this by means of a new unary predicate *original* identifying the original objects.

Let C_{dom} be the set of constant symbols of sort *object* occurring in $\mathcal{S}$. By the definition of the basic action tasks, $C_{dom} \dot{\cup} C$ form the set C_{all} of all constant symbols of the task. We will describe below how the compilation transforms the action precondition axioms P and the successor state axioms E.

The translation function $f_d(\mathcal{D})$ extends the original $\mathcal{S}$ with two new auxiliary constants $\mathbf{c}'$ and $\mathbf{c}''$ for each constant $\mathbf{c} \in C_{dom}$. It also adds a new binary

situation-independent predicate *partners* with parameters of sort *object*^{2} and a unary situation-independent predicate *original* with a parameter of sort *object* to the signature.

Function $t_c(\mathcal{D}, C)$ adds analogously two new auxiliary constants $\mathbf{c}'$ and $\mathbf{c}''$ for each constant $\mathbf{c} \in C$.

The translation function $t_i(\mathcal{D}, C, I)$ initializes predicate *partners* as

$$\mathit{partners}(x, x', x'') \leftrightarrow \bigvee_{\mathbf{c} \in C_{all}} (x = \mathbf{c} \wedge x' = \mathbf{c}' \wedge x'' = \mathbf{c}''), \tag{6.32}$$

where $\mathbf{c}'$ and $\mathbf{c}''$ are the constants that have been introduced for $\mathbf{c}$, and marks the original constants with

$$\mathit{original}(x) \leftrightarrow \bigvee_{\mathbf{c} \in C_{all}} (x = \mathbf{c}). \tag{6.33}$$

It also introduces unique names axioms in such a way that for all $\mathbf{c} \in C_{all}$, the equality of $\mathbf{c}'$ and $\mathbf{c}''$ remains unspecified. All other pairs of constants are different from each other: Let w.l.o.g. $C_{all} = \{\mathbf{c_1}, \ldots, \mathbf{c_n}\}$ and define

$$X = \{(\mathbf{c_i}, \mathbf{c_j}) \mid 1 \leq i < j \leq n\} \cup \tag{6.34}$$
$$\{(\mathbf{c_i}, \mathbf{c'_j}) \mid 1 \leq i, j \leq n\} \cup \tag{6.35}$$
$$\{(\mathbf{c_i}, \mathbf{c''_j}) \mid 1 \leq i, j \leq n\} \cup \tag{6.36}$$
$$\{(\mathbf{c'_i}, \mathbf{c''_j}) \mid 1 \leq i, j \leq n, i \neq j\}. \tag{6.37}$$

Then there is a unique names axiom $c_i \neq c_j$ for each pair $(c_i, c_j) \in X$. Since $|X| = (n-1)n/2 + n^2 + n^2 + (n^2 - n) = 3.5n^2 - 1.5n$, these unique names axioms can be generated in polynomial time.

The domain closure axiom for constants also ranges over the new constants:

$$\bigvee_{\mathbf{c} \in C_{all}} x = \mathbf{c} \vee x = \mathbf{c}' \vee x = \mathbf{c}''.$$

The translation function t_g transforms γ as follows: to prevent the usage of the new objects, all quantifications in the formula are modified to only range over objects that satisfy *original*. This can be done by replacing subformulas $\exists x(\varphi)$ with $\exists x(\mathit{original}(x) \wedge \varphi)$ and subformulas $\forall x(\varphi)$ with $\forall x(\mathit{original}(x) \rightarrow \varphi)$. Then it removes each occurrence of the predicate $R(x)$, whose truth values have not been specified in I, by substituting it by

$$\exists x', x''(\mathit{partners}(x, x', x'') \wedge x' = x'').$$

The function f_d transforms the action precondition axioms and the successor state axioms analogously.

Clearly, each plan of the original task is a plan for the resulting task and vice versa. Moreover, the modified basic action theory can be computed in polynomial time, and the independence requirements for compilation schemes are satisfied. Hence, we have presented a polynomial-time compilation scheme from RBAT_{incomp} to RBAT_{noUNA} preserving plan size exactly. □

The lemma shows that RBAT_{noUNA} is at least as expressive as RBAT_{incomp}, which is more expressive than PDDL. This leads directly to the following theorem.

Theorem 9 (R3.4 is necessary)**.**
There is no compilation scheme from $RBAT_{noUNA}$ to PDDL.

Proof. In Lemma 6 we established that $\mathrm{RBAT}_{incomp} \preccurlyeq^{\mathrm{e}}_{p} \mathrm{RBAT}_{noUNA}$ and in Theorem 8 that $\mathrm{RBAT}_{incomp} \not\preccurlyeq^{\mathrm{u}} \mathrm{PDDL}$. By Corollary 1 this implies that $\mathrm{RBAT}_{noUNA} \not\preccurlyeq^{\mathrm{u}} \mathrm{PDDL}$. □

With this theorem it is clear that RBAT_{noUNA} is more expressive than PDDL. To specify its expressivity more precisely we will also show that it is not only at least as expressive as RBAT_{incomp} but has exactly the same expressive power.

Lemma 7 ($\mathrm{RBAT}_{noUNA} \preccurlyeq^{\mathrm{e}}_{p} \mathrm{RBAT}_{incomp}$)**.** *There is a polynomial-time compilation scheme from $RBAT_{noUNA}$ to $RBAT_{incomp}$ preserving plan size exactly:*

Proof. We will prove this theorem in two steps: first we will introduce a new predicate $\mathit{equal}(x, y)$ taking over the role of the equality $=$. In a second step we will replace this new predicate *equal* by a unary predicate.

Let $\Pi_{noUNA} = \langle \langle \mathcal{S}, P, E \rangle, C, I, \gamma \rangle$ be the source task. Since we allow unique names axioms for some constant symbols, we define the set $D = \{(x, y) \mid x \text{ and } y \text{ must be different according to the unique names axioms in } I\}$.

Then we add the missing unique names axioms for constants to the source task. We introduce a new predicate $\mathit{equal}(x, y)$ whose values are not specified in the initial database. This predicate should take over the role of the equivalence $=$ for the comparison of terms of sort *object*. Thus, we replace each comparison $t_1 = t_2$ in a successor state axiom, action precondition axiom or the goal where t_1 and t_2 are of sort *object* with the atom $\mathit{equal}(t_1, t_2)$.

Due to the additional unique names axioms the situation described by the initial database does no longer conform to the original one. We solve this by an initializing action *initialize*(). This action will be enforced to be executed once at the beginning of each plan by an additional predicate *initialized* and modifications analogous to those in the proof of Lemma 5. Action *initialize* should make all ground atoms true that were true in the original theory T because their subterms were equal to those for which the atom was declared to be true. Thus, on the one hand we transform the situation-independent predicates to relational fluents and on the other hand we extend each successor state axiom $P(x_1, \ldots, x_n, \mathit{do}(a, s)) \leftrightarrow \varphi$ to

$$P(x_1, \ldots, x_n, \mathit{do}(a, s)) \leftrightarrow \varphi \vee (a = \mathit{initialize}() \wedge \exists y_1, \ldots, y_n (\mathit{equal}(x_1, y_1) \wedge \cdots \wedge \mathit{equal}(x_n, y_n) \wedge P(y_1, \ldots, y_n, s))) \quad (6.38)$$

In the following we have to substitute variables x_i occurring freely in a formula φ by variables y_i. Instead of writing $\varphi[x_1/y_1] \ldots [x_n/y_n]$ we use the abbreviation $\varphi[x_i/y_i]$.

All atoms that change their value in the original basic action theory because of the equality of some constants should change their value in the new theory, too. For this, we further modify the successor state axioms by substituting each expression corresponding to a φ in the schema in restriction R2 by an expression

$$\exists y_1, \ldots, y_n (\mathit{equal}(x_1, y_1) \wedge \cdots \wedge \mathit{equal}(x_n, y_n) \wedge \varphi[x_i/y_i]), \quad (6.39)$$

where the y_i are new variables.

If *equal* describes an equivalence relation, these changes obviously lead to a proper substitution of the built-in equality $=$. However, there are also models where the unspecified predicate *equal* does not represent an equivalence relation. We handle this by introducing an additional relational fluent *noEquivalenceRelation*(s) which is false in the initial situation and is set to its real value by action *initialize*. This is done by the successor state axiom

$$\begin{aligned} \mathit{noEquivalenceRelation}(\mathit{do}(a,s)) \leftrightarrow\ & \mathit{noEquivalenceRelation}(s) \vee \\ & (a = \mathit{initialize}() \wedge \neg(\forall x(\mathit{equal}(x,x)) \wedge \\ & \quad \forall x,y(\mathit{equal}(x,y) \rightarrow \mathit{equal}(y,x)) \wedge \\ & \quad \forall x,y,z(\mathit{equal}(x,y) \wedge \mathit{equal}(y,z) \rightarrow \mathit{equal}(x,z)))). \end{aligned} \tag{6.40}$$

In addition, we have to ensure that *equal* conforms to the set D of pairs of originally different constant symbols. Hence, we introduce a new situation-independent predicate *different*(x,y) that is initialized as

$$\mathit{different}(x,y) \leftrightarrow \bigvee_{(\mathbf{c},\mathbf{c}') \in D} (x = \mathbf{c} \wedge y = \mathbf{c}').$$

To identify the models where *equal* does not conform to the set D we add a relational fluent *incorrectEqualities* that is initially false. We set its intended value by means of the initializing action:

$$\begin{aligned} \mathit{incorrectEqualities}(\mathit{do}(a,s)) \leftrightarrow\ & \mathit{incorrectEqualities}(s) \vee \\ & a = \mathit{initialize}() \wedge \exists x,y(\mathit{equal}(x,y) \wedge \mathit{different}(x,y)) \end{aligned} \tag{6.41}$$

The action precondition axioms and the goal are altered in such a way that each action can be executed and the goal is true when *noEquivalenceRelation* or *incorrectEqualities* is true.

For those models where *equal* describes no equivalence relation or does not conform to D all situations starting with action *initialize* satisfy the goal. For all other models, a situation is a plan iff it is a plan for the original task Π_{noUNA} (minus the initializing action). Hence, the presented transformations define a polynomial-time compilation scheme (preserving plan size exactly) from RBAT$_{noUNA}$ to RBAT plus a single unspecified predicate of arity 2.

In the second step of our proof we will present a polynomial-time compilation scheme from the latter formalism to RBAT$_{incomp}$.

We start with a restricted basic action theory T' with one binary predicate R whose values are not specified in the initial database. We will replace this predicate by a new unspecified unary predicate R'.

For this, we introduce for each two constants $\mathbf{c_i}$ and $\mathbf{c_j}$ in the original set of all constants C_{all} a new constant $\mathbf{c_{i,j}}$ and add a situation-independent predicate *pair*(x,y,z) which is initialized in the initial database as

$$\mathit{pair}(x,y,z) \leftrightarrow \bigvee_{\mathbf{c_i},\mathbf{c_j} \in C_{all}} (x = \mathbf{c_i} \wedge y = \mathbf{c_j} \wedge z = \mathbf{c_{i,j}}). \tag{6.42}$$

We replace each occurrence of an atom $R(x,y)$ by $\exists z(\mathit{pair}(x,y,z) \wedge R'(z))$. Everywhere else we have to prevent the usage of the additional constants by

means of an additional unary predicate *original* that identifies the original constants and by modifying all quantifications of the original theory to range only over these constants. This can be done as shown in the proof of Lemma 6.

Obviously, these modifications define a polynomial-time compilation scheme from the intermediate formalism to RBAT_{incomp} preserving plan size exactly. Thus, by Theorem 5 overall there is a polynomial-time compilation scheme from RBAT_{noUNA} to RBAT_{incomp} preserving plan size exactly. □

Lemmas 6 and 7 result in the following corollary.

Corollary 2 ($\text{RBAT}_{incomp} \approx_p^{\text{e}} \text{RBAT}_{noUNA}$)**.**
$RBAT_{incomp}$ and $RBAT_{noUNA}$ have the same expressive power.

The only restriction on the initial database that we have not examined so far is the domain closure axiom for objects.

6.4 Domain Closure Axiom

The domain closure axiom states that there are no objects except for the named constants. At first glance, one would assume that this restriction is necessary to stay within the same expressive power as PDDL because this formalism has a built-in domain closure assumption. If we abandon the domain closure axiom, there are infinitely many models of the initial database and a plan must work for all models.

Consider as an example the task shown in Figure 6.2: We want to have a party to make our friends Jim and Luke happy. The problem is that our neighbors will feel disturbed by the party and will get angry. We can avoid this by inviting them as well. The task is formulated so that in the presence of the domain closure axiom there cannot be anybody who lives nearby and is not a friend, so $s^* = do(\textit{have-party}(), do(\textit{invite-friends}(), \mathbf{s_0}))$ is a plan. If we abandon the domain closure axiom, there can be an arbitrary number of neighbors who would become angry and s^* would no longer work for all these models. However, there would still be a plan for this instance: $do(\textit{have-party}(), do(\textit{invite-neighbors}(), do(\textit{invite-friends}(), \mathbf{s_0})))$. If we consider the same task but without action *invite-neighbors*, s^* would still be a plan in the presence of the domain closure axiom, but the task would be unsolvable without it.

The example shows that leaving out the domain closure axiom can reduce the set of possible plans, exploiting that *lives-far-away* is false for all additional objects. However, actions can also behave qualitatively differently for different models without involving predicates; for example, a conditional effect might only trigger if there is at least one object that is different from all named constants.

As we have seen, the domain closure axiom has significant impact on the meaning of the rest of the task specification. Since the other restrictions on RBAT tasks are very restrictive it is still far from obvious that this axiom is necessary for compilability. Indeed, it is *not* necessary.

Since the overall compilation scheme is very involved, we develop it in several steps, each treating different aspects.

$\Pi_{\text{party}} = \langle \langle \mathcal{S}, P, E \rangle, C, I, \gamma \rangle$ is a BAT task with the following components:

- $\mathcal{S}$ consists of predicate symbols

$$\begin{aligned} \mathit{friend} &: \mathit{object} \\ \mathit{lives\text{-}far\text{-}away} &: \mathit{object} \\ \mathit{happy} &: \mathit{object} \times \mathit{situation} \\ \mathit{invited} &: \mathit{object} \times \mathit{situation} \\ \mathit{anyone\text{-}angry} &: \mathit{object} \times \mathit{situation} \end{aligned}$$

  and action functions

$$\begin{aligned} \mathit{have\text{-}party} &: \rightarrow \mathit{action} \\ \mathit{invite\text{-}friends} &: \rightarrow \mathit{action} \\ \mathit{invite\text{-}neighbors} &: \rightarrow \mathit{action} \end{aligned}$$

- E consists of the sentences

$$\begin{aligned} \mathit{happy}(p, \mathit{do}(a, s)) \leftrightarrow\ & a = \mathit{have\text{-}party}() \wedge \mathit{invited}(p, s) \\ & \vee\ \mathit{happy}(p, s) \\ \mathit{anyone\text{-}angry}(\mathit{do}(a, s)) \leftrightarrow\ & a = \mathit{have\text{-}party}() \wedge \\ & \exists p(\neg \mathit{lives\text{-}far\text{-}away}(p) \wedge \neg \mathit{invited}(p, s)) \vee \\ & \mathit{anyone\text{-}angry}(s) \\ \mathit{invited}(p, \mathit{do}(a, s)) \leftrightarrow\ & a = \mathit{invite\text{-}friends}() \wedge \mathit{friend}(p) \vee \\ & a = \mathit{invite\text{-}neighbors}() \wedge \neg \mathit{lives\text{-}far\text{-}away}(p) \\ & \vee\ \mathit{invited}(p, s) \end{aligned}$$

- $P = \{\mathit{poss}(\mathit{have\text{-}party}(), s), \mathit{poss}(\mathit{invite\text{-}friends}(), s),$
  $\mathit{poss}(\mathit{invite\text{-}neighbors}(), s)\}$
- $C = \{\mathbf{Jim}, \mathbf{Luke}\}$
- I consists of the sentences:

$$\begin{aligned} \mathit{friend}(p) &\leftrightarrow p = \mathbf{Jim} \vee p = \mathbf{Luke} \\ \mathit{lives\text{-}far\text{-}away}(p) &\leftrightarrow p = \mathbf{Jim} \\ \neg \mathit{invited}(p, \mathbf{s_0}) & \\ \neg \mathit{happy}(p, \mathbf{s_0}) & \\ \neg \mathit{anyone\text{-}angry}(\mathbf{s_0}) & \\ \mathbf{Jim} &\neq \mathbf{Luke} \\ x &= \mathbf{Jim} \vee x = \mathbf{Luke} \end{aligned}$$

- $\gamma = \neg \mathit{anyone\text{-}angry}(s) \wedge \forall p(\mathit{friend}(p) \rightarrow \mathit{happy}(p, s))$

Figure 6.2: Example task for the impact of the domain closure axiom. The rest of the task specification is shown in Figure 6.4.

We first show the following partial result: for tasks with an upper bound on the quantifier rank (the maximum number of nested quantifications in a formula) of the goal formula, the requirements on the initial database are so restrictive that we can omit the domain closure axiom. We denote the family of these formalisms that specify a bound r on the quantifier rank of the goal formula and have the same requirements as RBAT except that there is no domain closure axioms in the initial database by RBAT^r_{nodca}. The proof is based on a compilation scheme from RBAT^r_{nodca} to RBAT. To illustrate this computation, Figures 6.3 and 6.4 show the result for the party example.

Theorem 10 ($\text{RBAT}^r_{nodca} \preccurlyeq^{\text{e}}_p \text{RBAT}$)**.**
In the presence of a fixed bound r on the quantifier rank of the goal description, omitting the domain closure axiom does not increase the expressive power of restricted basic action theories: there is a polynomial-time compilation scheme from $RBAT^r_{nodca}$ to RBAT preserving plan size exactly.

Proof. We refer to objects of a model which are different from all named constants as "unnamed objects". We begin our proof with two observations:

- In all situations that occur in a plan, all unnamed objects are interchangeable. For example, if $P(o_1, o_2, o_3, s)$ is true, o_1 and o_3 are different unnamed objects, and o_2 is a named object, then $P(o'_1, o_2, o'_3, s)$, where again o'_1 and o'_3 are different unnamed objects, is also true.
- For determining whether there is a plan, it is sufficient to consider only the models with up to k unnamed objects, where k is the maximum quantifier rank of all formulas of the theory (including implicit universal quantifiers).

The first observation can easily be shown by induction over the situations: In the initial situation (and for situation-independent predicates) all atoms containing unnamed objects are false due to R3.1 and R3.2.

For the inductive case, consider situation $s' = do(A, s)$. For each relational fluent P, the truth value of $P(\bar{x}, s')$ is determined by a formula $\Phi(\bar{x}, s)$ which is uniform in s (by the definition of successor state axioms). If we evaluate this formula for different tuples $\bar{x}$ which only differ by permuting unnamed objects, the same truth values are obtained: by the induction hypothesis, unnamed objects are interchangeable in situation s.

If all unnamed objects are interchangeable, then only the *number* of unnamed objects can affect the validity of a plan. For statements like "There are m distinct unnamed objects" it is necessary to bind at least m variables. With the quantifier rank bounded by k, it is not possible to make statements about more than k objects, and hence all models with more than k unnamed objects behave the same as the model with exactly k such objects.

These observations lead quite directly to a compilation scheme from RBAT without a domain closure axiom to RBAT. We explicitly represent the discriminable models and modify the basic action theory in such a way that each action is executed in all models in parallel. The goal formulation is then modified such that the original goal must be true in all models.

Let r be the bound on the quantifier rank of the goal (which is fixed for all tasks of the source formalism) and let d be the maximum quantifier rank of the

$\mathcal{D} = \langle \mathcal{S}, P, E \rangle$ is a BAT domain with the following components:

- $\mathcal{S}$ consists of predicate symbols

$$\begin{aligned}
\mathit{friend} &: \mathit{object} \\
\mathit{lives\text{-}far\text{-}away} &: \mathit{object} \\
\mathit{world} &: \mathit{object} \\
\mathit{in\text{-}world} &: \mathit{object}^2 \\
\mathit{happy\text{-}in\text{-}world} &: \mathit{object}^2 \times \mathit{situation} \\
\mathit{invited\text{-}in\text{-}world} &: \mathit{object}^2 \times \mathit{situation} \\
\mathit{anyone\text{-}angry\text{-}in\text{-}world} &: \mathit{object}^2 \times \mathit{situation}
\end{aligned}$$

  and action functions

$$\begin{aligned}
\mathit{have\text{-}party} &: \rightarrow \mathit{action} \\
\mathit{invite\text{-}friends} &: \rightarrow \mathit{action} \\
\mathit{invite\text{-}neighbors} &: \rightarrow \mathit{action}
\end{aligned}$$

- E consists of the sentences

$$\begin{aligned}
\mathit{happy\text{-}in\text{-}world}(p, w, \mathit{do}(a, s)) \leftrightarrow\ & a = \mathit{have\text{-}party}() \wedge \\
& \quad \mathit{invited\text{-}in\text{-}world}(p, w, s) \\
& \vee \mathit{happy\text{-}in\text{-}world}(p, w, s) \\
\mathit{anyone\text{-}angry\text{-}in\text{-}world}(w, \mathit{do}(a, s)) \leftrightarrow\ & a = \mathit{have\text{-}party}() \wedge \\
& \quad \exists p(\mathit{in\text{-}world}(p, w) \wedge \\
& \qquad \neg \mathit{lives\text{-}far\text{-}away}(p) \wedge \\
& \qquad \neg \mathit{invited\text{-}in\text{-}world}(p, w, s)) \\
& \vee \mathit{anyone\text{-}angry\text{-}in\text{-}world}(w, s) \\
\mathit{invited\text{-}in\text{-}world}(p, w, \mathit{do}(a, s)) \leftrightarrow\ & a = \mathit{invite\text{-}friends}() \wedge \\
& \quad \mathit{friend}(p) \vee \\
& a = \mathit{invite\text{-}neighbors}() \wedge \\
& \quad \neg \mathit{lives\text{-}far\text{-}away}(p) \\
& \vee \mathit{invited\text{-}in\text{-}world}(p, w, s)
\end{aligned}$$

- $P = \{\mathit{poss}(\mathit{have\text{-}party}(), s), \mathit{poss}(\mathit{invite\text{-}friends}(), s),$
  $\mathit{poss}(\mathit{invite\text{-}neighbors}(), s)\}$

Figure 6.3: Compilation result for the domain of the task in Figure 6.2

$\Pi_{\text{party}} = \langle \mathcal{D}, C, I, \gamma \rangle$ is a BAT task with the following components:

- $\mathcal{D}$ is the domain specified in Figure 6.3
- $C = \{\mathbf{Jim}, \mathbf{Luke}\} \cup \{\mathbf{w_0}, \mathbf{w_1}\} \cup \{\mathbf{c_1^1}\}$
- I consists of unique names axioms for all constants in C and the sentences:

$$
\begin{aligned}
\mathit{friend}(p) &\leftrightarrow p = \mathbf{Jim} \vee p = \mathbf{Luke} \\
\mathit{lives\text{-}far\text{-}away}(p) &\leftrightarrow p = \mathbf{Jim} \\
\neg \mathit{invited\text{-}in\text{-}world}(p, w, \mathbf{s_0}) & \\
\neg \mathit{happy\text{-}in\text{-}world}(p, w, \mathbf{s_0}) & \\
\neg \mathit{anyone\text{-}angry\text{-}in\text{-}world}(w, \mathbf{s_0}) & \\
\mathit{world}(w) &\leftrightarrow w = \mathbf{w_0} \vee w = \mathbf{w_1} \\
\mathit{in\text{-}world}(x, w) &\leftrightarrow (x = \mathbf{Jim} \wedge w = \mathbf{w_0}) \vee \\
&\quad\ (x = \mathbf{Jim} \wedge w = \mathbf{w_1}) \vee \\
&\quad\ (x = \mathbf{Luke} \wedge w = \mathbf{w_0}) \vee \\
&\quad\ (x = \mathbf{Luke} \wedge w = \mathbf{w_1}) \vee \\
&\quad\ (x = \mathbf{c_1^1} \wedge w = \mathbf{w_1}) \\
\bigvee\nolimits_{\mathbf{c} \in C} x &= \mathbf{c}
\end{aligned}
$$

- $\gamma = \forall w(\mathit{world}(w) \rightarrow (\neg \mathit{anyone\text{-}angry\text{-}in\text{-}world}(w, s) \wedge \forall p(\mathit{in\text{-}world}(p, w) \rightarrow (\mathit{friend}(p) \rightarrow \mathit{happy\text{-}in\text{-}world}(p, w, s)))$

Figure 6.4: Constants C, initial database I and goal formula γ of the compilation result for the example task in Figure 6.2

formulas in the domain description. Then $k := \max(r, d)$ is the maximum quantifier rank of the theory. We add $k + 1$ new domain constants $\mathbf{w_0}, \mathbf{w_1}, \ldots, \mathbf{w_k}$ and a unary situation-independent predicate *world* to distinguish these constants from the other constants. Further, we add $(k+1)k/2$ domain constants $\mathbf{c_1^1}, \mathbf{c_2^1}, \mathbf{c_2^2}, \ldots, \mathbf{c_k^1}, \ \ldots, \mathbf{c_k^k}$ and a predicate $\mathit{in\text{-}world}(x, w)$. This fluent predicate describes which constants exist in which world. Each world $\mathbf{w_i}$ (except $\mathbf{w_0}$) contains constants $\mathbf{c_i^1}, \ldots, \mathbf{c_i^i}$. In addition, the constants of the original basic action theory should exist in each world. Let for this purpose C_{all} denote the set of all constant symbols of sort *object* in the original tasks. We can specify *in-world* the initial database as

$$\begin{aligned} \textit{in-world}(x, w) \leftrightarrow\ & (x = \mathbf{c_1^1} \wedge w = \mathbf{w_1}) \vee \\ & (x = \mathbf{c_2^1} \wedge w = \mathbf{w_2}) \vee (x = \mathbf{c_2^2} \wedge w = \mathbf{w_2}) \vee \cdots \vee \\ & (x = \mathbf{c_k^1} \wedge w = \mathbf{w_k}) \vee \cdots \vee (x = \mathbf{c_k^k} \wedge w = \mathbf{w_k}) \vee \\ & \big(\bigvee\nolimits_{\mathbf{w} \in \{\mathbf{w_0}, \ldots, \mathbf{w_k}\}} \bigvee\nolimits_{\mathbf{c} \in C_{all}} (x = \mathbf{c} \wedge w = \mathbf{w})\big) \end{aligned} \tag{6.43}$$

We replace each relational fluent $F(\bar{x}, s)$ with a fluent *F-in-world*$(\bar{x}, w, s)$. These new relational fluents should be true in the initial database (for all worlds w) iff the original ones were true. Therefore, we replace each formula

$$\begin{aligned} F(x_1, \ldots, x_n, \mathbf{s_0}) \leftrightarrow\ & (x_1 = \mathbf{c}_{11} \wedge \cdots \wedge x_n = \mathbf{c}_{1n}) \\ & \vee \cdots \vee (x_1 = \mathbf{c}_{m1} \wedge \cdots \wedge x_n = \mathbf{c}_{mn}), \end{aligned} \tag{6.44}$$

in the initial database with

$$\begin{aligned} \textit{F-in-world}(x_1, \ldots, x_n, w, \mathbf{s_0}) \leftrightarrow\ & \\ \bigvee\nolimits_{\mathbf{w} \in \{\mathbf{w_0}, \ldots, \mathbf{w_k}\}} & \big((x_1 = \mathbf{c}_{11} \wedge \cdots \wedge x_n = \mathbf{c}_{1n} \wedge w = \mathbf{w}) \vee \cdots \vee \\ & (x_1 = \mathbf{c}_{m1} \wedge \cdots \wedge x_n = \mathbf{c}_{mn} \wedge w = \mathbf{w}) \big). \end{aligned} \tag{6.45}$$

An action should be executable if it exists in all models and if its precondition is satisfied in all worlds. Thus, we replace each action precondition axiom $\textit{Poss}(A(x_1, \ldots, x_n), s) \leftrightarrow \psi$ with a new expression

$$\begin{aligned} \textit{Poss}(A(x_1, \ldots, x_n), s) \leftrightarrow\ & \textit{in-world}(x_1, \mathbf{w}_0) \wedge \cdots \wedge \textit{in-world}(x_n, \mathbf{w}_0) \wedge \\ & \forall w(\textit{world}(w) \rightarrow (\psi_w \wedge \bigwedge\nolimits_{\mathbf{c} \in C_\psi} \textit{in-world}(\mathbf{c}, w))). \end{aligned} \tag{6.46}$$

Thereby, C_ψ is the set of all constant symbols of sort *object* occurring in ψ, and ψ_w is created from ψ by the following modifications:

- Each occurrence of a relational fluent $F(\bar{x}, s)$ is replaced with predicate *F-in-world*$(\bar{x}, w, s)$.
- Each existentially quantified subexpression $\exists y(\psi)$ is replaced with the expression $\exists y(\textit{in-world}(y, w) \wedge \psi_w)$
- Each universally quantified subexpression $\forall y(\psi)$ is replaced with the expression $\forall y(\textit{in-world}(y, w) \rightarrow \psi_w)$

The successor state axioms are transformed analogously: Each formula φ from the schema in formula 6.2 is transformed exactly as the formula ψ above. This is obviously possible in polynomial time and preserves the form of the successor state axioms required in restriction R2.

With these modifications a fluent *F-in-world*$(\bar{x}, \mathbf{w_i}, s)$ is true for a executable situation s iff $F(\bar{x}, s)$ is true in the model with i additional objects. Thus, if we change the goal description φ to

$$\forall w(\textit{world}(w) \rightarrow \varphi_w) \tag{6.47}$$

every plan of the new basic action theory is a plan for the new basic action theory and vice versa. Hence, the transformations form a polynomial-time compilation scheme from RBAT^r_{nodca} to RBAT preserving plan size exactly. □

This theorem together with Theorems 6 and 5 gives compilability to PDDL.

Corollary 2 ($\mathrm{RBAT}^{r}_{nodca} \preccurlyeq^{\mathrm{e}} \mathrm{PDDL}$)**.** *For every bound $r \in \mathbb{N}$ on the quantifier rank of the goal, there is a polynomial-time compilation scheme from $RBAT^{r}_{nodca}$ to PDDL preserving plan size exactly.*

We denote the analogous formalism without a restriction on the quantifier rank by $\mathrm{RBAT}^{\infty}_{nodca}$. As another step to the final result, we will present a compilation from a very restricted subset of all $\mathrm{RBAT}^{\infty}_{nodca}$ tasks to RBAT. In this subset, the goal formula may not mention any predicate symbols except $=$. We denote this sub-formalism with $\mathrm{RBAT}^{\infty,=}_{nodca}$.

Lemma 8 ($\mathrm{RBAT}^{\infty,=}_{nodca} \preccurlyeq^{\mathrm{e}}_{p} \mathrm{RBAT}$)**.**
If the goal may not mention predicate symbols apart from $=$ then omitting the domain closure axioms does not increase the expressive power of restricted basic action theories: there is a polynomial-time compilation scheme from $RBAT^{\infty,=}_{nodca}$ to RBAT preserving plan size exactly.

Proof. Let $\Pi = \langle \mathcal{D}, C, I, \gamma \rangle$ be a $\mathrm{RBAT}^{\infty,=}_{nodca}$ task. Given a model of $\mathcal{U}, \mathcal{I}$ of $\mathcal{T}(\Pi)$, the model satisfies the goal formula γ either in all situations or in none: the interpretation of all predicates except $=$ is irrelevant for the goal and the interpretation of $=$ and the constant symbols is situation-independent. W.l.o.g., we assume in the following that γ is given in prenex normal form.

Let C_{all} denote the set of all constant symbols of Π. Then $k := |\mathcal{U}| - |C_{all}|$ is the number of "unnamed" objects in the universe. We use in the following an arbitrary but fixed numbering $1, \ldots, k$ of these unnamed objects.

Assume for a moment that we know the number k when transforming the goal (only for the specific universe $\mathcal{U}$). We will change the representation of objects in the goal formula: instead of referring to them by a single variable x of sort *object* (or a constant symbol), we will use $k+2$ variables x_{type}, x_{const}, and $x^1, \ldots, x^k$. The intended representation of an object denoted by constant $\mathbf{c}$ is $x_{\mathrm{type}} = \mathbf{C}$, $x_{\mathrm{const}} = \mathbf{c}$, and $x^i = \mathbf{0}$ for $i \in \{1, \ldots, k\}$. The intended representation of the j-th unnamed object is $x_{\mathrm{type}} = \mathbf{U}$, $x_{\mathrm{const}} = \mathbf{None}$, and $x^i = \mathbf{1}$ for $i \in \{1, \ldots, j\}$ and $x^i = \mathbf{0}$ for $i \in \{j+1, \ldots, k\}$. The variables x^i can be seen as a unary representation of the number j of the unnamed object.

For this representation, the compilation adds new constant symbols **None**, $\mathbf{C}, \mathbf{U}, \mathbf{0}$, and $\mathbf{1}$. In addition, the transformation of the initial situation marks the original constants with a new situation-independent predicate *constant* and sentence

$$\mathit{constant}(x, \mathbf{s_0}) \leftrightarrow \bigvee_{\mathbf{c} \in C_{all}} x = \mathbf{c}$$

and adds a domain closure axiom

$$x = \mathbf{None} \vee x = \mathbf{C} \vee x = \mathbf{U} \vee x = \mathbf{0} \vee x = \mathbf{1} \vee \bigvee_{\mathbf{c} \in C_{all}} x = \mathbf{c}$$

and the required unique names axioms for constants.

For the translation of the goal formula γ we use the following transformation t_k (for $k > 0$):

$$\begin{aligned}
t_k(\forall x(\psi)) = \forall x_{\text{type}}, x_{\text{const}}, x^1, \ldots, x^k \big(& \\
\big((x_{\text{type}} = \mathbf{C} \wedge \mathit{constant}(x_{\text{const}}) \wedge \bigwedge_{i \in \{1,\ldots,k\}} x^i = \mathbf{0}) \vee & \\
(x_{\text{type}} = \mathbf{U} \wedge x_{\text{const}} = \mathbf{None} \wedge & \\
x^1 = \mathbf{1} \wedge \bigwedge_{i \in \{2,\ldots,k\}} (x^i = \mathbf{1} \vee \bigwedge_{j \in \{i,\ldots,k\}} x^j = \mathbf{0}))) & \\
\rightarrow t_k(\psi) \big) &
\end{aligned}$$

$$\begin{aligned}
t_k(\exists x(\psi)) = \exists x_{\text{type}}, x_{\text{const}}, x^1, \ldots, x^k \big(& \\
\big((x_{\text{type}} = \mathbf{C} \wedge \mathit{constant}(x_{\text{const}}) \wedge \bigwedge_{i \in \{1,\ldots,k\}} x^i = \mathbf{0}) \vee & \\
(x_{\text{type}} = \mathbf{U} \wedge x_{\text{const}} = \mathbf{None} \wedge & \\
x^1 = \mathbf{1} \wedge \bigwedge_{i \in \{2,\ldots,k\}} (x^i = \mathbf{1} \vee \bigwedge_{j \in \{i,\ldots,k\}} x^j = \mathbf{0}))) & \\
\wedge t_k(\psi) \big) &
\end{aligned}$$

$$t_k(x = \mathbf{c}) = (x_{\text{type}} = \mathbf{C} \wedge x_{\text{const}} = \mathbf{c})$$

$$t_k(x = y) = (x_{\text{type}} = x_{\text{type}} \wedge x_{\text{const}} = y_{\text{const}} \wedge \bigwedge_{i \in \{1,\ldots,k\}} x^i = y^i)$$

$$t_k(\varphi \circ \psi) = (t_k(\varphi) \circ t_k(\psi)) \text{ for } \circ \in \{\wedge, \vee, \rightarrow\}$$

$$t_k(\neg\varphi) = \neg t_k(\varphi)$$

For the special case $k = 0$, we define

$$\begin{aligned}
t_0(\forall x(\psi)) &= \forall x(\mathit{constant}(x) \rightarrow t_0(\psi)) \\
t_0(\exists x(\psi)) &= \exists x(\mathit{constant}(x) \wedge t_0(\psi)) \\
t_0(\varphi \circ \psi) &= (t_0(\varphi) \circ t_0(\psi)) \text{ for } \circ \in \{\wedge, \vee, \rightarrow\} \\
t_0(\neg\varphi) &= \neg t_0(\varphi) \\
t_0(\varphi) &= \varphi \text{ otherwise}
\end{aligned}$$

For a formula φ, this transformation is polynomial-time computable in $k + |\varphi|$.

If we compiled γ to $t_k(\gamma)$, it would hold for the transformed task $F(\Pi)$ that interpretation $\mathcal{U}, \mathcal{I} \models \gamma(\mathbf{s_0})$ iff any applicable action sequence is a plan for $F(\Pi)$. However, a compilation scheme must be plan-preserving for *all* models and the goal translation function knows neither $|C_{all}|$ nor $|\mathcal{U}|$. Since there is no domain closure axiom in Π, there can also be models with an infinite universe. We can resolve this by a slightly more complicated transformation of γ.

Let r be the quantifier rank of γ. We define the goal transformation function t_g of the compilation scheme as

$$t_g(\gamma) = \bigwedge_{j \in \{0,\ldots,r\}} t_j(\gamma).$$

We need to show that task Π has a plan iff the compiled task $F(\Pi) = \langle \mathcal{D}', C', I', \gamma' \rangle$ has a plan. Since in both tasks, the goal formula has no free variables (in particular, it does not depend on a situation s), it is sufficient to consider only the initial situations $\mathbf{s_0}$ of Π and $\mathbf{s_0}'$ of $F(\Pi)$, respectively. In the following, we denote the set of all constant symbols of sort *object* occurring in Π by C_{all}. From the compilation, it follows for the corresponding set C'_{all} for $F(\Pi)$ that $C'_{all} = C_{all} \cup \{\mathbf{None}, \mathbf{C}, \mathbf{U}, \mathbf{0}, \mathbf{1}\}$.

We first show that if $\mathbf{s_0}$ is a plan for Π then $\mathbf{s_0}'$ is a plan of $F(\Pi)$:

Let $\mathcal{U}', \mathcal{I}'$ be a model of $\mathcal{T}(F(\Pi))$. We need to show that $\mathcal{U}', \mathcal{I}' \models \gamma'$. Due to the unique names axioms for constants and the domain closure axiom in I', $\mathcal{I}'$ maps each constant symbol to a distinct object and $\mathcal{U}' = \{\mathcal{I}'(\mathbf{c}) \mid \mathbf{c} \in C'_{all}\}$. Due to the sentence for predicate *constant* in I', it is interpreted as $\mathcal{I}(constants) = \{\mathcal{I}'(\mathbf{c}) \mid \mathbf{c} \in C_{all}\}$.

For $k \in \{0, \ldots, r\}$, let $\mathcal{U}_k, \mathcal{I}_k$ be an arbitrary model of $\mathcal{T}(\Pi)$ with $\mathcal{U}_k = \{\mathcal{I}'(\mathbf{c}) \mid \mathbf{c} \in C_{all}\} \cup \{u_i \mid i \in \{1, \ldots, k\}\}$ and $\mathcal{I}_k|_{C_{all}} = \mathcal{I}'|_{C_{all}}$. Such a model exists because $\mathcal{T}(\Pi)$ cannot be inconsistent (due to the syntactic restrictions) and the specification is consistent with the initial database I. Since $\mathbf{s_0}$ is a plan for Π, it also works for $\mathcal{U}_k, \mathcal{I}_k$ and therefore $\mathcal{U}_k, \mathcal{I}_k \models \gamma$. This implies that $\mathcal{U}', \mathcal{I}' \models t_k(\gamma)$, using the new representation for the objects from $\mathcal{U}_k$. Overall, $\mathcal{U}', \mathcal{I}' \models t_k(\gamma)$ for all $k \in \{0, \ldots, r\}$ and we conclude that $\mathcal{U}', \mathcal{I}' \models \gamma'$.

For the other direction, we need to show that if $\mathbf{s_0}'$ is a plan for $F(\Pi)$ then $\mathbf{s_0}$ is a plan of Π.

Let $\mathcal{U}, \mathcal{I}$ be a model of $\mathcal{T}(\Pi)$. We need to show that $\mathcal{U}, \mathcal{I} \models \gamma$. Due to the unique names axioms for constants in I, it holds that $\{\mathcal{I}(\mathbf{c}) \mid \mathbf{c} \in C_{all}\} \subseteq \mathcal{U}$.

We first consider the case that $|\mathcal{U}| \leq |C_{all}| + r$, i.e., $\mathcal{U}$ contains at most k objects that cannot be denoted by constants of Π. Define $k := |\mathcal{U}| - |C_{all}|$. Let $\mathcal{U}', \mathcal{I}'$ be an arbitrary model of $\mathcal{T}(F(\Pi))$ with $\mathcal{U}' = \{\mathcal{I}(\mathbf{c}) \mid \mathbf{c} \in C_{all}\} \cup \{o_{\mathbf{None}}, o_{\mathbf{C}}, o_{\mathbf{U}}, o_{\mathbf{0}}, o_{\mathbf{1}}\}$. As $\mathcal{T}(\Pi)$ cannot be inconsistent and this specification is consistent with the initial database I', such a model must exist. Since already the initial situation $\mathbf{s_0}'$ is a plan for $F(\Pi)$, it follows that $\mathcal{U}', \mathcal{I}' \models \gamma'$ and therefore $\mathcal{U}', \mathcal{I}' \models t_k(\gamma)$. Reverting the changed representation of objects in $\mathcal{U}$, this gives $\mathcal{U}, \mathcal{I} \models \gamma$.

We now consider the case, where $l := |\mathcal{U}| - |C_{all}| > r$. Let w.l.o.g. $Q = \{x_1, \ldots, x_r\}$ be the set of variables occurring in γ, and denote the all-quantified variables in γ by A and the existentially quantified variables by E. We can show that $\mathcal{U}, \mathcal{I} \models \gamma$ by specifying for each variable mapping $\alpha_A : A \to \mathcal{U}$ an extended variable mapping $\alpha_Q : Q \to \mathcal{U}$ such that $\alpha_Q|_A = \alpha_A$ and $\alpha_Q \models \varphi$ where φ is the matrix of γ. Let α_A be such a variable mapping. Consider a model $\mathcal{U}_\alpha, \mathcal{I}_\alpha$ of $\mathcal{T}(\Pi)$ with $|\mathcal{U}_\alpha| = |C_{all}| + r$, $\{\mathcal{I}(\mathbf{c}) \mid \mathbf{c} \in C_{all}\} \cup \{\alpha_A(x) \mid x \in A\} \subseteq \mathcal{U}_\alpha \subseteq \mathcal{U}$ and $\mathcal{I}_\alpha|_{C_{all}} = \mathcal{I}|_{C_{all}}$. By the argumentation for the previous case it holds that $\mathcal{U}_\alpha, \mathcal{I}_\alpha \models \gamma$, so there is a variable mapping $\beta : Q \to \mathcal{U}_\alpha$ with $\beta|_A = \alpha|_A$ and $\beta \models \varphi$ for the matrix φ of γ. Therefore, this variable mapping β satisfies all requirements of the desired mapping α_Q and we conclude overall that $\mathcal{U}, \mathcal{I} \models \gamma$.

All transformations satisfy the requirements of a polynomial-time compilation scheme. Moreover, if there is a plan for source task Π then the empty action sequence is a plan for the translated task. Therefore, the compilation scheme preserves plan size exactly. □

To show that we can omit the domain closure axiom even with arbitrary goal formulas, we need to combine the two previous compilation schemes and

extend them for models with arbitrarily many unnamed objects in the presence of predicates in the goal formula.

Theorem 11 ($\mathrm{RBAT}^{\infty}_{nodca} \preccurlyeq^{\mathrm{e}}_{p} \mathrm{RBAT}$)**.**
Omitting the domain closure axiom does not increase the expressive power of restricted basic action theories: there is a polynomial-time compilation scheme from $RBAT^{\infty}_{nodca}$ to RBAT preserving plan size exactly.

Proof. Let $\Pi = \langle \mathcal{D}, C, I, \gamma \rangle$ be a $\mathrm{RBAT}^{\infty}_{nodca}$ task. We will build the overall compilation scheme on those from the proofs of Theorem 10 and Lemma 8.

Let a_i be the maximum arity of any situation-independent predicate and a_f be the maximum arity of any fluent predicate of the signature, and let d denote the maximum quantifier rank of the formulas in the domain description.

The domain description is compiled as in the compilation scheme of Theorem 10 but simulating up to $k = \max(a_i, a_f - 1, d)$ unnamed objects (i.e., there are worlds $\mathbf{w_0}, \ldots, \mathbf{w_k}$). In addition, we extend the signature with a unary predicate *constant* as in the compilation scheme of Lemma 8 and a unary predicate *maxworld*.

For the treatment of worlds with more than k unnamed objects, we introduce additional constant symbols $\mathbf{None}, \mathbf{C}, \mathbf{U}, \mathbf{0}$, and $\mathbf{1}$, as in the proof of Lemma 8, plus $a := \max(a_i, a_f - 1)$ constant symbols $\mathbf{u_1}, \ldots, \mathbf{u_a}$, which we will motivate later.

The initial database is essentially translated as in the compilation scheme from Theorem 10, except that we mark $\mathbf{w_k}$ with sentence $\mathit{maxworld}(w) \leftrightarrow w = \mathbf{w_k}$ and add the same sentence for predicate *object* as in the proof of Lemma 8. In addition, we extend the domain closure axiom and the unique names axioms for actions suitably for the entire set of constant symbols.

We assume w.l.o.g. that the goal formula is given in prenex normal form and that the arguments of predicates are pairwise different variable symbols. It is translated to a conjunction of subformulas for different scenarios.

The first scenario is that the quantifier rank r of the goal is less or equal to k (which we do not know when transforming the goal formula). In this case, the same goal transformation as for Theorem 10 suffices. Therefore, we define the formula

$$\gamma_0 := \forall w(\mathit{world}(w) \rightarrow \gamma_w),$$

with γ_w denoting the same transformation as in the compilation scheme of Theorem 10.

The second scenario should cover all models of $\mathcal{T}(\Pi)$ that have more than k unnamed objects but not more than the quantifier rank r of the goal.

For a given number j of unnamed object, we can test whether $j > k$ with the condition

$$\chi_j := \neg\exists w\big(\mathit{world}(w) \wedge \exists x_1, \ldots, x_j\big(\bigwedge\nolimits_{1 \le i \le j}(\neg \mathit{constant}(x_i) \wedge \mathit{inworld}(x_i, w) \wedge \bigwedge\nolimits_{i < l \le j} x_i \neq x_l)\big)\big).$$

In the proof of Lemma 8 we have already seen how we can treat such models w.r.t. the equality predicate. For all other predicates, we exploit the observation from the proof of Theorem 10 that all worlds with more than k unnamed objects develop essentially the same way as the world with exactly k

unnamed objects and that unnamed objects are fully interchangeable: If for an executable situation s and atoms $P(x_1, \ldots, x_n, s)$ and $P(y_1, \ldots, y_n, s)$ it holds for an interpretation that

- if x_i is interpreted to the same object as a constant symbol $\mathbf{c}$ then also y_i is interpreted to this object, and vice versa, and
- x_i and x_j are interpreted to the same object iff y_i and y_j are interpreted to the same object

then $P(x_1, \ldots, x_n, s)$ is true under the interpretation iff $P(y_1, \ldots, y_n, s)$ is true under the interpretation.

We will use the newly introduced constants $\mathbf{u_1}, \ldots, \mathbf{u_a}$ as proxy objects to test the second condition for the unnamed objects. The idea is that for models with i unnamed objects we replace an atom $P(x_1, \ldots, x_n, s)$, where all x_i are variables, with a formula

$$\exists y_1, \ldots, y_n \big(P(y_1, \ldots, y_n) \wedge \bigwedge\nolimits_{1 \leq i \leq n} \big(\mathit{constant}(y_i) \wedge y_i = x_i \vee \bigvee\nolimits_{1 \leq l < i} y_i = y_l \vee y_i = \mathbf{u_i} \big) \big).$$

This formula specifies that each parameter y_i is equal to x_i if it denotes a constant and otherwise it is either equal to an earlier parameter or denoted by a new proxy object $\mathbf{u}_i$.

However, if we apply the transformation from Lemma 8 for the predicate $=$, we remove the original variables $x_1, \ldots, x_n$. Let for each replaced variable x_i the variables $x_{i,\text{type}}$, $x_{i,\text{const}}$, and $x_i^1, \ldots, x_i^j$ be the ones for the alternative representation. To extend the definition of t_j from the proof of Lemma 8 (there called t_k) to relational fluents we use in addition a variable w for the largest fully simulated world (suitably bound later).

$$\begin{aligned} t_j(P(x_1, \ldots, x_n, s), w) = \exists y_1, \ldots, y_n \big(& \\ & \bigwedge\nolimits_{1 \leq i < l \leq n} \big(y_i = y_l \leftrightarrow (x_{i,\text{const}} = x_{l,\text{const}} \wedge \bigwedge\nolimits_{1 \leq m \leq j} x_i^m = x_l^m) \big) \wedge \\ & \mathit{P\text{-}in\text{-}world}(y_1, \ldots, y_n, w, s) \wedge \\ & \bigwedge\nolimits_{1 \leq i \leq n} \big(\mathit{constant}(y_i) \wedge y_i = x_{i,\text{const}} \vee \bigvee\nolimits_{1 \leq l < i} y_i = y_l \vee y_i = \mathbf{u_i} \big) \big). \end{aligned}$$

The second line ensures that the (in-) equalities of the new variables $\bar{y}$ are consistent with all (in-)equalities of the representatives of the original parameters $\bar{x}$. The third line ensures that we evaluate P with the truth values from the simulated world w.

If a situation-independent predicate has an unnamed object as parameter, it is false in all models of $\mathcal{T}(\Pi)$. This simplifies the translation to

$$\begin{aligned} t_j(P(x_1, \ldots, x_n), w) = \exists y_1, \ldots, y_n \big(& \\ P(y_1, \ldots, y_n) \wedge \bigwedge\nolimits_{1 \leq i \leq n} \big(\mathit{constant}(y_i) \wedge y_i = x_{i,\text{const}} \big) \big). & \end{aligned}$$

For all other formulas φ, transformation $t_j(\varphi, w)$ is defined analogously to the transformation $t_j(\varphi)$ in the proof of Lemma 8 except that it passes the additional parameter w through.

Overall, we define for each $1 \leq j \leq r$ a formula

$$\gamma_j := \chi_j \rightarrow \exists w(\mathit{maxworld}(w) \wedge t_j(\gamma, w))$$

and define the entire goal transformation function as

$$t_g(\gamma) = \bigwedge_{0 \leq j \leq r} \gamma_j.$$

Since $\mathcal{T}(\Pi)$ cannot distinguish universes with $\max(r, k) + |C_{all}|$ and more objects, this goal formula works also for models with more than $\max(r, k) + |C_{all}|$ objects in the universe.

Overall, we presented a compilation scheme preserving plan size exactly. From the same argumentation as in the proof of Theorem 10 it follows that an action sequence is applicable in Π iff it is applicable in the compiled task $F(\Pi)$ and the explicitly represented worlds develop analogous to the respective models of Π. The goal formula treats all possible models that $\mathcal{T}(\Pi)$ can distinguish: Those with at most k unnamed objects are covered by γ_0, those with $k < j \leq r$ unnamed objects are covered by γ_j and all those with at least r unnamed objects are covered by γ_r. Therefore a situation s is a plan for Π iff it is a plan for $F(\Pi)$.

Since the transformation of the domain is polynomial-time computable, we have shown that $\mathrm{RBAT}^{\infty}_{nodca} \preccurlyeq^{\mathrm{e}}_{p} \mathrm{RBAT}$. □

With Theorems 6 and 5 we receive compilability to PDDL.

Corollary 3 ($\mathrm{RBAT}^{\infty}_{nodca} \preccurlyeq^{\mathrm{e}} \mathrm{PDDL}$)**.** *There is a polynomial-time compilation scheme from $RBAT^{\infty}_{nodca}$ to PDDL preserving plan size exactly.*

6.5 Situation-independent Object Functions

One requirement of restriction R1 is that all situation-independent object functions must be constants, i.e., functions of sort $\epsilon \rightarrow \mathit{object}$. In this section, we will examine whether we can allow arbitrary situation-independent object functions of sort $\mathit{object}^n \rightarrow \mathit{object}$ for $n \in \mathbb{N}_0$.

The restriction to constants is intimately connected with the domain closure axiom for constants required by R3.4, which excludes the existence of objects that cannot be denoted by constants. Usually it should be possible that a function denotes also new objects which cannot be accessed by the constant symbols. If we want to permit such functions of sort $\mathit{object}^n \rightarrow \mathit{object}$, we also have to abandon this domain closure axiom. The following result is not due to this, but also keeps its validity if we suitably generalize the domain closure axiom.

Theorem 12 (Restriction R1.1 is necessary if the functions can refer to objects that cannot be denoted by constants)**.** *There is no compilation scheme from RBAT to PDDL if functions of sort $\mathit{object}^n \rightarrow \mathit{object}$ are permitted for $n > 0$, even if they are restricted to $n = 1$.*

Proof. Let $(\Sigma, \Gamma, Q, \delta, q_0, \Box, Q_{acc})$ be a deterministic Turing machine (TM) with $\Sigma = \emptyset$ and $\Gamma = \{*, \Box\}$ that starts on an empty tape.

We will formulate a BAT in such a way that there is a plan iff the TM halts, and all information concerning a certain TM is encoded in the initial database, the task-specific constants, and the goal. As the state-translation functions of a compilation scheme must be polynomial-time computable and the plan existence problem is decidable for PDDL tasks, we could decide the halting problem if there was a compilation scheme.

We will exploit the absence of the restriction to represent infinitely many tape cells. Therefore, we use a single constant $\mathbf{p_0}$ to denote the first tape cell and a function $succ : object \rightarrow object$ which yields the successor of a cell. For a reliable simulation of the TM it is required that $succ(x) = succ(y) \rightarrow x = y$. Due to the requirements for the initial database we cannot simply add this sentence there. Thus, we use an additional action $init : \epsilon \rightarrow action$ which is necessary in each plan and makes the goal formula true for all models where the desired condition is not satisfied and the successor function for the tape cells is flawed. As a plan must work for all models, the existence of a plan is finally determined only by the models where the structure interpreting $succ()$ contains an infinite, linear sequence starting at $\mathbf{p_0}$.

We require the following constants of sort *object*:

- $*$ and $\square$ encode the alphabet of the Turing machine,
- $\mathbf{q} \in Q$ to encode the states of the Turing machine,
- $\blacktriangleleft$,$\blacktriangledown$, and $\blacktriangleright$ to encode the movements of the Turing machine, and
- $\mathbf{p_0}$ to denote the first tape cell.

In the following we will denote the set of these constants by $\mathcal{C}$. We will use the following predicates:

- $transition(c, q, c', q', d)$ to encode transition function δ,
- $state(q, s)$ denotes the state of the TM in situation s,
- $scan(p, s)$ denotes the tape cell the R/W-head stands on in a situation s,
- $star(p, s)$ encodes whether cell p contains a $*$,
- $initialized(s)$ is used to enforce action *init*, and
- $flawed(s)$ means that the structure interpreting $succ()$ does not contain an infinite, linear sequence starting at $\mathbf{p_0}$.

In the initial database the values of these predicates represent the initial status of the TM:

Predicate *transition* encodes the transition function δ of the Turing machine:

$$transition(c, q, c', q', d) \leftrightarrow \bigvee\nolimits_{\delta(\mathbf{c},\mathbf{q})=(\mathbf{c'},\mathbf{q'},\mathbf{d})} (c = \mathbf{c} \wedge q = \mathbf{q} \wedge c' = \mathbf{c'} \wedge q' = \mathbf{q'} \wedge d = \mathbf{d}) \quad (6.48)$$

The R/W head stands on the first tape cell, the TM is in state $\mathbf{q_0}$, and all cells contain a blank.

$$scan(p, \mathbf{s_0}) \leftrightarrow (p = \mathbf{p_0}) \tag{6.49}$$
$$state(q, \mathbf{s_0}) \leftrightarrow (q = \mathbf{q_0}) \tag{6.50}$$
$$\neg star(p, \mathbf{s_0}) \tag{6.51}$$

At the beginning the task is not initialized and we assume that the tape is not flawed. The latter will be set timely to the correct value by the action $init()$.

$$\neg initialized(\mathbf{s_0}) \tag{6.52}$$
$$\neg flawed(\mathbf{s_0}) \tag{6.53}$$

There should be a plan iff the tape is not flawed and the TM halts or if the tape is flawed. This leads to the goal formula

$$\exists q \left(state(q, s) \wedge \bigvee\nolimits_{\mathbf{q} \in Q_{acc}} (q = \mathbf{q}) \right) \vee flawed(s). \tag{6.54}$$

An action $step(q, c, p, q', c', d, p')$ simulates one step of the TM: Previously the TM is in state q and reads character c on cell p. Then, it changes to state q', writes a c' and moves in direction d to cell p'. This results directly in the following action precondition axiom:

$$\begin{aligned} Poss(step(q, c, p, q', c', d, p'), s) \leftrightarrow\ & flawed(s) \vee \\ & \big(state(q, s) \wedge scan(p, s) \wedge transition(q, c, q', c', d) \wedge \\ & \quad \big((c = *) \wedge star(p, s) \vee (c = \square) \wedge \neg star(p, s)\big) \wedge \\ & \quad \big((d = \blacktriangleleft) \wedge p = succ(p') \vee (d = \blacktriangledown) \wedge p = p' \vee \\ & \qquad (d = \blacktriangleright) \wedge p' = succ(p)\big) \wedge initialized(s)\big) \end{aligned} \tag{6.55}$$

Action $init$ can only be used once.

$$Poss(init(), s) \leftrightarrow \neg initialized(s) \tag{6.56}$$
$$initialized(do(a, s)) \leftrightarrow a = init() \vee initialized(s) \tag{6.57}$$

This action diagnoses whether the tape is flawed:

$$\begin{aligned} flawed(do(a, s)) \leftrightarrow\ & flawed(s) \vee a = init() \wedge \\ & \neg\big(\forall x, y (succ(x) = succ(y) \rightarrow x = y) \wedge \\ & \qquad \forall x \big(\bigwedge\nolimits_{\mathbf{c} \in \mathcal{C}} \neg succ(x) = \mathbf{c}\big)\big) \end{aligned} \tag{6.58}$$

In the cases where the tape is not flawed, the following successor state axioms provide a reliable simulation:

$$\begin{aligned} state(q', do(a, s)) \leftrightarrow\ & state(q', s) \wedge \neg \exists q, c, p, c', d, p' \\ & (a = step(q', c, p, q, c', d, p') \wedge \neg q = q') \vee \\ & \qquad \exists q, c, p, c', d, p' \, (a = step(q, c, p, q', c', d, p')) \end{aligned} \tag{6.59}$$

$$scan(p', do(a, s)) \leftrightarrow scan(p', s) \wedge \neg\exists q, c, p, q', c', d \\ (a = step(q, c, p', q', c', d, p) \wedge \neg p = p') \vee \\ \exists q, c, p, q', c', d\,(a = step(q, c, p, q', c', d, p')) \quad (6.60)$$

$$star(p, do(a, s)) \leftrightarrow star(p, s) \wedge \\ \neg\exists q, q', d, p'\big(a = step(q, *, p, q', \square, d, p')\big) \vee \\ \exists q, q', d, p'\big(a = step(q, \square, p, q', *, d, p')\big) \quad (6.61)$$

As there are no axioms constraining *succ*() there can be arbitrary structures interpreting *succ*(). If the structure is "flawed" (not containing an infinite, linear structure starting at $\mathbf{p_0}$), the action *init*() will force *flawed*(s) to become true, which will make formula (6.54) true. Since there are *always* non-flawed structures, formula (6.54) can only become true in all models if s is a halting computation on the non-flawed structures and, thus, the simulated TM halts. Conversely, it is obvious that a halting computation translates into a sequence of actions making formula (6.54) true.

We have presented a reduction where all information concerning a certain TM is encoded in the initial database, the task-specific constants, and the goal. Thus, according to the argumentation above, there cannot be any compilation scheme at all – not even with an unrestricted blow-up of the plan size. □

We have seen that situation-independent functions which can denote other objects than the constants add additional expressive power to RBATs and, thus, lead to a different expressiveness than PDDL. If we had not abandoned the domain closure axiom, the functions had only been synonyms for the known constants. As this case is only a special case of the considerations in the next section, we do not discuss it further.

6.6 Functional Fluents

As opposed to the previous section we require for functional fluents, i.e. functions of sort $object^n \times situation \rightarrow object$ that they always have objects that can also be denoted by constant symbols. We have several reasons for this: First of all, if these functions could introduce new objects (this implies that we use the relaxed domain closure axiom from the previous section) and we pass on successor state axiom for these functions, we would face the same difficulties as before. If they, in principle, can introduce new objects, but there must be a successor state axiom for each fluent, this would lead to the following situation: Assume a successor state axiom which in a certain situation $do(a, s)$ assigns an object that cannot be denoted by a constant of a function for situation s. As in each situation the value of a functional fluent must be unique and such objects cannot be distinguished by a uniform expression, there can be at most one additional object. In fact, this is not a matter of functions, but rather a matter of the initial database. If we would drop restriction R3.4, we could easily compile these tasks by means of an additional constant. Thus, it makes sense to restrict the values of the functional fluents to constants and situation-independent actions. Furthermore, in the book of Reiter (2001) we have found only examples that corroborate this assumption.

Why do we accept such a restriction for functional fluents but not for situation-independent functions? Situation-independent functions that are restricted to such an extent are in practice only synonyms for constants or other known actions, and, thus, add only marginal benefit. By contrast, functional fluents, which can change their value from situation to situation, are more like pointers in programming languages and can provide the possibility of a more brief and elegant formulation of a planning task.

Before we consider functional fluents, we have to determine what a reasonable extrapolation of the restrictions of Eyerich et al. (2006) to functional fluents looks like. If we omit such restrictions, the basic action theories are undecidable with similar proofs as above.

Obviously the following is a reasonable extension of restriction R2: There is a successor state axiom for each functional fluent. The successor state axiom of a functional fluent f has to fit the schema

$$f(x_1,\ldots,x_n,do(a,s)) = y \leftrightarrow \Big(\bigvee_{l=1}^{p} \alpha_l\Big) \vee \Big(f(x_1,\ldots,x_n,s) = y \wedge \neg \bigvee_{l=1}^{q} \delta_l\Big) \tag{6.62}$$

for some $p, q \geq 0$, where the α_l and δ_l are of the form

$$\exists y_1,\ldots,y_m\,(a = A\,(y_1,\ldots,y_m) \wedge \varphi) \tag{6.63}$$

Each action function may appear as $a = A\,(y_1,\ldots,y_m)$ in at most one subformula α_l and in at most one subformula δ_l.

Restriction R3.1 leads to a unique model in the initial database, in which the truth-values of the predicates are explicitly specified (i.e. not described by compact formulas). Analogously we require that the value of each functional fluent in $\mathbf{s_0}$ is explicitly specified:

For each $(n+1)$-ary functional fluent f there is an expression

$$\begin{aligned} f(x_1,\ldots,x_n,\mathbf{s_0}) = y \leftrightarrow\ & \\ & ((x_1 = \mathbf{c}_{11}) \wedge \ldots \wedge (x_n = \mathbf{c}_{1n}) \wedge (y = \mathbf{c}_1)) \vee \ldots \vee \\ & ((x_1 = \mathbf{c}_{m1}) \wedge \ldots \wedge (x_n = \mathbf{c}_{mn}) \wedge (y = \mathbf{c}_m))\,. \end{aligned} \tag{6.64}$$

We call the formalism defined by this extension of the restrictions on basic action theories to object fluents $\mathrm{RBAT}_{functions}$. This formalism can be compiled to PDDL:

Theorem 13 ($\mathrm{RBAT}_{functions} \preccurlyeq_p^{\mathrm{e}}$ PDDL)**.** *There is a polynomial-time compilation scheme from $RBAT_{functions}$ to PDDL preserving plan size exactly.*

Proof. We will specify a polynomial-time compilation scheme to show that $\mathrm{RBAT}_{functions} \preccurlyeq_p^{\mathrm{e}}$ RBAT. The claim follows then with Theorem 6 from Theorem 5.

For each function $f : object^n \times situation \to object$ we introduce a predicate

$$P_f : object^n \times object \times situation \tag{6.65}$$

and substitute the sentence $f(x_1, \ldots, x_n, \mathbf{s_0}) = y \leftrightarrow \varphi_f$ in the initial database by an expression

$$P(x_1, \ldots, x_n, y, \mathbf{s_0}) \leftrightarrow \varphi_f. \tag{6.66}$$

The successor state axiom $f(x_1, \ldots, x_n, do(a, s)) = y \leftrightarrow \psi_f$ analogously becomes $P_f(x_1, \ldots, x_n, y, do(a, s)) \leftrightarrow \psi_f$.

To eliminate all remaining occurrences of function f in a formula, we proceed as follows:

- If f occurs as $f(x_1, \ldots, x_n, s) = y$, where y is a constant or a variable, we substitute it by $P_f(x_1, \ldots, x_n, y, s)$.
- If f occurs as $f(x_1, \ldots, x_n, s) = g(y_1, \ldots, y_m, s)$, we introduce a new variable y and substitute it by $\exists y(P_f(x_1, \ldots, x_n, y, s) \wedge P_g(y_1, \ldots, y_m, y, s))$.
- If f occurs as a term (or subterm) in a predicate, we can substitute it introducing a new variable. For example, $P(f(y_1, \ldots, y_n, s), x_2, \ldots, x_m, s)$ becomes $\exists y(P_f(y_1, \ldots, y_n, y, s) \wedge P(y, x_2, \ldots, x_m, s))$.

Then $P_f(x_1, \ldots, x_n, y, s)$ is true in the compiled task iff $f(x_1, \ldots, x_n, s) = y$ is true in the original task, and the evaluation of the action precondition axioms and the goal for a situation s always yields the same result in both tasks. Therefore, the transformation defines a compilation scheme preserving plan size exactly, which is obviously computable in polynomial time. □

6.7 Restrictions on Successor State Axioms

In this section we examine the restriction on successor state axioms. The required schema in R2 is based on the idea that a fluent is true in a situation iff it has been made true by the last action or if it has been true in the previous situation and has not been made false. Reiter (2001) allows the syntactically more general form

$$F(x_1, \ldots, x_n, do(a, s)) \leftrightarrow \Phi_F(x_1, \ldots, x_n, a, s)$$

for successor state axioms, where $\Phi_F(x_1, \ldots, x_n, a, s)$ is a formula uniform in s whose free variables are among $x_1, \ldots, x_n, a, s$ (cf. Definition 14.

In the following we will show how the compilation scheme of Eyerich et al. can be altered to transform the general form of successor state axioms, too. However, for the moment we would like to retain some of the flavor of requirement R4.2 that restricts the usage of sort *action* in successor state axioms. We require in the following, that the formula $\Phi_F(\bar{x}, a, s)$ does not contain any variables of sort *action* besides a[2]. We denote the resulting formalism with RBAT_{ssa}.

We will first define how we alter the compilation scheme $\mathbf{f}_{\mathrm{RBAT}\to\mathrm{PDDL}}$ from Definition 27. Then we will show how the proof for its correctness (cf. Theorem 6) needs to be adapted for the new compilation scheme.

[2]Loosening this requirement will be a topic of Section 6.8 on restrictions on the usage of sort *action*.

Definition 28 (Compilation scheme from RBAT_{ssa} to PDDL)**.**
Let $\Pi = \langle \mathcal{D}, C, I, \gamma \rangle$ be a $RBAT_{ssa}$ task with $\mathcal{D} = \langle \mathcal{S}, P, E \rangle$. The compilation scheme $\mathbf{f}_{RBAT_{ssa} \to PDDL} = \langle f_d, t_c, t_i, t_g \rangle$ from $RBAT_{ssa}$ to PDDL differs from the compilation scheme $\mathbf{f}_{RBAT \to PDDL}$ from Definition 27 only in the computation of the action effects, which is done by the domain transformation function f_d.

Let $A(\bar{z})$ be the PDDL action introduced for the action function symbol A in $\mathcal{S}$. The PDDL effect e for $A(\bar{z})$ is a conjunction over formulas φ_F^A for each relational fluent F in $\mathcal{S}$.

Each φ_F^A is constructed from the respective successor state axiom

$$F(\bar{x}, do(a, s)) \leftrightarrow \Phi_F(\bar{x}, a, s)$$

as follows.

Consider $\Phi_F^A(\bar{x}, \bar{z}, s) := \Phi_F(\bar{x}, a, s)[a/A(\bar{z})]$. Since $\Phi_F(\bar{x}, a, s)$ may not contain any action variables besides a, sort action can only occur in $\Phi_F^A(\bar{x}, \bar{z}, s)$ as function terms. These can only appear with predicate symbol $=$ (by R4.1 and R4.3).

We eliminate all action symbols from $\Phi_F^A(\bar{x}, \bar{z}, s)$ by replacing subformulas $A'(y_1, \ldots, y_m) = A'(y'_1, \ldots, y'_m)$ with $(y_1 = y'_1) \wedge \cdots \wedge (y_m = y'_m)$ and replacing $A'(\bar{y}) = A''(\bar{y}')$ with $\bot$ (logical falsity) (for all action function symbols A' and A''). We then simplify the resulting formula with the well-known logical equivalences for $\bot$ and $\top$ (logical truth). Let $\Phi'^A_F(\bar{x}, \bar{z}, s)$ denote the resulting formula.

Since this formula is uniform in s (because $\Phi_F(\bar{x}, a, s)[a/A(\bar{z})]$) was already uniform in s) and does not contain any action symbols, we can transform it to a PDDL formula $\varphi_{F+}^A(\bar{x}, \bar{z}) := \theta(\Phi'^A_F(\bar{x}, \bar{z}, s))$. The desired equation φ_F^A is[3]

$$\varphi_F^A := \forall \bar{x}((\varphi_{F+}^A(\bar{x}, \bar{z}) \rhd F(\bar{x})) \wedge (\neg\varphi_{F+}^A(\bar{x}, \bar{z}) \rhd \neg F(\bar{x}))).$$

This compilation scheme solves the frame problem twice because it is already solved by the PDDL semantics but the compilation always includes an effect for each fluent, even if the action does not affect the fluent at all. Therefore, we can obviously omit subeffects φ_F^A of the form $\forall \bar{x}((F(\bar{x}) \rhd F(\bar{x})) \wedge (\neg F(\bar{x}) \rhd \neg F(\bar{x})))$.

The specified compilation scheme is correct, which we will show by adapting the original proof for the correctness of $\mathbf{f}_{\mathrm{RBAT} \to \mathrm{PDDL}}$.

Theorem 14 (Correctness of $\mathbf{f}_{\mathrm{RBAT}_{ssa} \to \mathrm{PDDL}}$)**.**
$\mathbf{f}_{RBAT_{ssa} \to PDDL}$ is a compilation scheme from $RBAT_{ssa}$ to PDDL preserving plan size exactly.

Proof. The proof of Theorem 6 depends on Lemmas 1–4 for $\mathbf{f}_{\mathrm{RBAT} \to \mathrm{PDDL}}$. Since the parts of the compilation that depend on the successor state axioms do not affect the proofs of Lemmas 1–3, these lemmas are equivalently valid for $\mathbf{f}_{\mathrm{RBAT}_{ssa} \to \mathrm{PDDL}}$.

[3] In the special cases where $\varphi_{F+}^A(\bar{x}, \bar{z})$ is $\top$ or $\bot$, $\varphi_F^A = \forall \bar{x} F(\bar{x})$ and $\varphi_F^A = \forall \bar{x} \neg F(\bar{x})$, respectively.

To show the analogous validity of Lemma 4, we need to adapt parts of the inductive step of its proof. More precisely, we need to show that for a RBAT$_{ssa}$ task Π and the compiled task PDDL $F(\Pi)$ it holds that

$$\mathcal{T}(\Pi) \models R(\bar{c}, do(A(\bar{\mathbf{c}}), s)) \text{ iff } app_{A'(\bar{\mathbf{c}})}(s')(R(\bar{c})) = 1,$$

where R is a relational fluent, $\bar{c}$ is a vector of constant symbols, s is an executable situation, s' is the respective state of the PDDL task, $A(\bar{\mathbf{c}})$ is a ground action applicable in s, and $A'(\bar{\mathbf{c}})$ is the corresponding PDDL action.

We may use that for situation s, it holds for all situation-independent ground predicates $P(\bar{d})$ that $\mathcal{T}(\Pi) \models P(\bar{d})$ iff $s'(P(\bar{d})) = 1$, and for all relational ground fluents $P(\bar{d}, s)$ that $\mathcal{T}(\Pi) \models P(\bar{d}, s)$ iff $s'(P(\bar{d})) = 1$ (induction hypothesis).

We will in the following use the same notation for the intermediate formulas of the compilation as in Definition 28. Let $R(\bar{x}, do(a, s)) \leftrightarrow \Phi_R(\bar{x}, a, s)$ be the successor state axiom for R. Then $\mathcal{T}(\Pi) \models R(\bar{c}, do(A(\bar{\mathbf{c}}), s))$ holds iff $\mathcal{T}(\Pi) \models \Phi_R(\bar{x}, a, s)[\bar{x}/\bar{c}, a/A(\bar{\mathbf{c}})]$ is true. Using the notation from Definition 28, this holds iff $\mathcal{T}(\Pi) \models \Phi^A_R(\bar{x}, \bar{z}, s)[\bar{x}/\bar{c}, \bar{z}/\bar{\mathbf{c}}]$ iff $\mathcal{T}(\Pi) \models \Phi'^A_R(\bar{x}, \bar{z}, s)[\bar{x}/\bar{c}, \bar{z}/\bar{\mathbf{c}}]$ (the latter due to the unique names axioms of $\mathcal{T}$).

Formula $\Phi'^A_R(\bar{x}, \bar{z}, s)[\bar{x}/\bar{c}, \bar{z}/\bar{\mathbf{c}}]$ satisfies the requirements of the analogon of Lemma 3 for the new compilation scheme. Therefore we can conclude with the induction hypothesis that $\mathcal{T}(\Pi) \models \Phi'^A_R(\bar{x}, \bar{z}, s)[\bar{x}/\bar{c}, \bar{z}/\bar{\mathbf{c}}]$ iff $s' \models \theta(\Phi'^A_R(\bar{x}, \bar{z}, s)[\bar{x}/\bar{c}, \bar{z}/\bar{\mathbf{c}}])$ iff $s' \models \theta(\Phi'^A_R(\bar{x}, \bar{z}, s))[\bar{x}/\bar{c}, \bar{z}/\bar{\mathbf{c}}]$, which, using the notation from the definition, holds iff $s' \models \varphi^A_{R^+}(\bar{x}, \bar{z})[\bar{x}/\bar{c}, \bar{z}/\bar{\mathbf{c}}]$.

So far, we have established that

$$\mathcal{T}(\Pi) \models R(\bar{c}, do(A(\bar{\mathbf{c}}), s)) \text{ iff } s' \models \varphi^A_{R^+}(\bar{x}, \bar{z})[\bar{x}/\bar{c}, \bar{z}/\bar{\mathbf{c}}].$$

Consider now the only subeffect of PDDL action $A'(\bar{\mathbf{c}})$ that affects predicate symbol R:

$$\forall \bar{x}((\varphi^A_{R^+}(\bar{x}, \bar{z})[\bar{z}/\bar{\mathbf{c}}] \triangleright R(\bar{x})) \wedge (\neg\varphi^A_{R^+}(\bar{x}, \bar{z})[\bar{z}/\bar{\mathbf{c}}] \triangleright \neg R(\bar{x}))).$$

Consider the application of $A'(\bar{\mathbf{c}})$ in state s'.

If $s' \models \varphi^A_{R^+}(\bar{x}, \bar{z})[\bar{x}/\bar{c}, \bar{z}/\bar{\mathbf{c}}]$ then the positive effect $\varphi^A_{R^+}(\bar{x}, \bar{z})[\bar{z}/\bar{\mathbf{c}}] \triangleright R(\bar{x})$ triggers for $\bar{x} = \bar{c}$ and $app_{A'(\bar{\mathbf{c}})}(s')(R(\bar{c})) = 1$.

If $s' \not\models \varphi^A_{R^+}(\bar{x}, \bar{z})[\bar{x}/\bar{c}, \bar{z}/\bar{\mathbf{c}}]$ then the negative effect $\neg\varphi^A_{R^+}(\bar{x}, \bar{z})[\bar{z}/\bar{\mathbf{c}}] \triangleright \neg R(\bar{x})$ triggers for $\bar{x} = \bar{c}$. Since the respective positive is the only other effect affecting $R(\bar{c})$ and it does not trigger, we conclude that in this case $app_{A'(\bar{\mathbf{c}})}(s')(R(\bar{c})) = 0$.

Since the rest of the proof of Lemma 4 is independent from the modification of the compilation scheme for general successor state axioms, the Lemma hold analogously for the new compilation scheme.

It is easy to check that after we have established the analogous validity of Lemmas 1–4, the proof of Theorem 6 directly carries over to the new compilation scheme $\mathbf{f}_{\text{RBAT}_{ssa} \to \text{PDDL}}$. □

We have seen that we can easily get rid of the restriction by Eyerich et al. (2006) on the form of the successor state axioms.

Corollary 4 (R2 is not necessary)**.** *Restriction R2 is not necessary for the compilability of RBAT to PDDL.*

However, we still did not consider fully general successor state axioms because we did not allow arbitrary action variables. Such restrictions on the usage of sort *action* will be the topic of the next section.

6.8 Restrictions on the Usage of Sort *action*

According to restriction R4, the usage of sort *action* is restricted as follows:

1. Arguments of situation-independent predicates or relational fluents must not be of sort *action*, i.e., each such relation R is of sort $\mathit{object}^{\mathrm{ar}(r)}$ or $\mathit{object}^{\mathrm{ar}(r)-1} \times \mathit{situation}$, respectively.

2. The right-hand side of the action precondition axioms, the goal formula γ and the sub-expressions φ in restriction R2 must not contain any action symbols.

3. Arguments of action functions in $\mathcal{S}$ cannot be of sort *action*.

Although these restrictions are formulated as individual points, they are so interdependent that it does not make much sense to consider them separately. For example, if we only loosen restriction R4.3 but keep R4.2 then the action parameters of sort *action* are irrelevant because they can neither be accessed in the action precondition axioms nor in the successor state axioms. Similarly, if giving up R4.1 but keeping R4.2, the predicate parameters of sort *action* cannot influence the dynamics of the domain or the truth of the goal formula and therefore do not influence whether an action sequence is a plan or not.

Therefore, we concentrate on the formalism $\mathrm{RBAT}_{\mathit{actions}}$ that is defined as formalism RBAT but without restriction R4. For predicates with action parameters we retain the spirit of R3.1 and R3.2 and require that their initial truth values are specified by explicitly listing all true ground atoms.

Theorem 15 ($\mathrm{RBAT}_{\mathit{actions}} \not\preceq^{\mathrm{u}}$ PDDL)**.** *There is no compilation scheme from $\mathrm{RBAT}_{\mathit{actions}}$ to PDDL.*

Proof. If we give up the restrictions on the usage of sort *action*, we face a similar situation as with the formalism $\mathrm{RBAT}_{\mathit{sit\text{-}independent\text{-}functions}}$, where we used the functions to denote infinitely many tape cells of a Turing machine. We can adapt the proof of Theorem 12 to show the analogous result for formalism $\mathrm{RBAT}_{\mathit{actions}}$:

Instead of using a constant $\mathbf{p_0}$ of sort *object* to denote the first tape cell, we use a new action $A_0 : \epsilon \to \mathit{action}$. The function $\mathit{succ} : \mathit{object} \to \mathit{object}$ is replaced with an action function $\mathit{succ} : \mathit{action} \to \mathit{action}$. Also the sort of the first argument of predicates *scan* and *star* is changed from *object* to *action* and these predicates are initialized by sentences $\mathit{scan}(p, \mathbf{s_0}) \leftrightarrow p = A_0()$ and $\neg \mathit{star}(p, \mathbf{s_0})$. In addition, we prevent the application of actions A_0 and $\mathit{succ}(a)$ with the successor state axioms $\neg \mathit{poss}(A_0())$ and $\neg \mathit{poss}(\mathit{succ}(a))$.

With these modifications, we can for a given Turing machine specify a $\mathrm{RBAT}_{\mathit{actions}}$ task that is solvable iff the Turing machine halts. Since only the initial state specification, the task-specific constants and the goal depend on the given Turing machine, the domain is fixed. If there was a compilation scheme, there would be a fixed PDDL domain that could be used to decide the halting problem. □

We have argued before that considering the restrictions on sort action individually is not interesting. However, we would like to examine a particular formalism that will allow us to generalize the previous compilability result for successor state axioms to the most general form of successor state axioms. In this formalism $\mathrm{RBAT}_{\textit{actions-in-conditions}}$, we only loosen restriction R4.2 and only for the action precondition axioms and the successor state axioms.

This leads to a similar scenario as relinquishing the domain closure axiom: There can be infinitely many objects (now of sort *action*) but we cannot refer to them by constants or function terms. Indeed, we can compile this formalism to RBAT in a very similar way as we compiled $\mathrm{RBAT}^{0}_{nodca}$. Since we already know that $\mathrm{RBAT}_{ssa} \preccurlyeq^{\mathrm{e}}_{p} \mathrm{RBAT}$, we avoid technical complications, which would be necessary to maintain the specific syntactic form of the successor state axioms, and present instead a compilation scheme from $\mathrm{RBAT}_{\textit{actions-in-conditions}}$ to RBAT_{ssa}.

Theorem 16 ($\mathrm{RBAT}_{\textit{actions-in-conditions}} \preccurlyeq^{\mathrm{e}}_{p} \mathrm{RBAT}_{ssa}$)**.** *Omitting restriction R4.2 for the action precondition axioms and the successor state axioms does not increase the expressive power of* $RBAT_{ssa}$*:* $RBAT_{\textit{actions-in-conditions}} \preccurlyeq^{\mathrm{e}}_{p} RBAT_{ssa}$*.*

Proof. The desired compilation scheme is essentially the same as in the proof of Theorem 10: we need to explicitly represent a number of worlds, each containing a different number of additional action objects.

We do not want to repeat the compilation scheme here but only address the issues that need to be adapted. The only critical aspect is that we need to represent an exponential number of action objects that can be denoted by action functions with a polynomial number of constants of sort *object*.

We add for each A in the set $\mathcal{A}$ of all action function symbols of the domain a new constant $\mathbf{o_A}$ of sort *object*. In addition there are new constants $\mathbf{act^j_i}$ that are used for the unnamed action objects in the universe (analogously to the objects $\mathbf{c^j_i}$ in the proof of Theorem 10). The parameter k is now the maximal quantifier rank of quantifiers over sort *action* occurring in the domain description and we use the analogous world constants $\mathbf{w_0}, \ldots, \mathbf{w_k}$.

To identify the new objects that represent actions we add two predicates *action-function*(x) and *unnamed-action*(x) and extend the initial database with sentences

$$\textit{action-function}(x) \leftrightarrow \bigvee\nolimits_{A \in \mathcal{A}} x = \mathbf{o_A}. \tag{6.67}$$

and

$$\textit{unnamed-action}(x) \leftrightarrow \bigvee\nolimits_{1 \le j \le i \le k} x = \mathbf{act^j_i}. \tag{6.68}$$

Predicate *in-world*(x, w) is used analogously to the earlier proof but not covering the objects that can be denoted by action functions:

$$\begin{aligned} \textit{in-world}(x, w) \leftrightarrow\ & (x = \mathbf{act^1_1} \wedge w = \mathbf{w_1}) \vee \\ & (x = \mathbf{act^1_2} \wedge w = \mathbf{w_2}) \vee (x = \mathbf{act^2_2} \wedge w = \mathbf{w_2}) \vee \\ & \vdots \\ & (x = \mathbf{act^1_k} \wedge w = \mathbf{w_k}) \vee \cdots \vee (x = \mathbf{act^k_k} \wedge w = \mathbf{w_k}) \end{aligned} \tag{6.69}$$

Let m be the largest arity associated with any function symbol in $\mathcal{A}$. In the following, we use an abbreviation *represents-action*$(o, x_1, \ldots, x_m)$ that stands for

$$\textit{unnamed-action}(o) \vee (\textit{action-function}(o) \wedge \bigwedge_{i \in \{1,\ldots,m\}} \textit{constant}(x_i)).$$

In addition we will temporarily introduce a function

$$\textit{real-action} : \textit{object}^{m+1} \rightarrow \textit{action}$$

with the intended interpretation that *real-action*$(\mathbf{o_A}, x_1, \ldots, x_m)$ denotes the action $A(x_1, \ldots, x_{\mathrm{ar}(A)})$ (ignoring the parameters $x_{\mathrm{ar}(A)+1}, \ldots, x_m$) and that *real-action*$(\mathbf{act_i^j}, x_1, \ldots, x_m)$ denotes the j-th unnamed action object in a universe with i such objects (ignoring all parameters $x_1, \ldots, x_m$). We will later replace the occurrences of *real-action* accordingly.

Let φ be a formula that is uniform in s. We define formula φ_w as the formula we get after the following transformations:

1. Replace each occurrence of a relational fluent $F(\bar{x}, s)$ with predicate *F-in-world*$(\bar{x}, w, s)$.

2. Replace each subformula $\exists a(\psi)$ with

$$\begin{aligned} \exists o_a, y_1, \ldots, y_m((&\textit{in-world}(o_a, w) \vee \\ &\textit{action-function}(o_a) \wedge \textit{represents-action}(o_a, y_1, \ldots, y_m)) \\ &\wedge \psi[a/\textit{real-action}(o_a, y_1, \ldots, y_m)]), \end{aligned} \tag{6.70}$$

   where $o_a, y_1, \ldots, y_m$ are new variables.

3. Replace each subformula $\forall a(\psi)$ with

$$\begin{aligned} \forall o_a, y_1, \ldots, y_m((&\textit{in-world}(o_a, w) \vee \\ &\textit{action-function}(o_a) \wedge \textit{represents-action}(o_a, y_1, \ldots, y_m)) \\ &\rightarrow \psi[a/\textit{real-action}(o_a, y_1, \ldots, y_m)]), \end{aligned} \tag{6.71}$$

   where $o_a, y_1, \ldots, y_m$ are again new variables.

We apply this transformation on the action precondition axioms, the successor state axioms, and the goal. Note that for the goal this only renames predicates and adds a parameter for the world because there cannot be quantifiers over actions in this formula (by the part of restriction R4.2 that we maintained). On the other formulas, the transformations eliminate all quantifications over sort *action*. Afterwards all terms of sort *action* in φ_w are either functions or variables occurring freely in φ_w (and φ). Due to restriction R4.1, such terms t_i can only occur in equalities as $t_i = t_j$.

In a second step, we eliminate the temporarily introduced action function *real-action* from the action precondition axioms and the successor state axioms and replace equalities of action terms as follows depending on the form of the terms:

1. *real-action*$(o, x_1, \ldots, x_m) =$ *real-action*$(o', y_1, \ldots, y_m)$ becomes

$$(o = o') \wedge \big(\textit{unnamed-action}(o) \vee \bigvee_{A \in \mathcal{A}} \big((o = \mathbf{o_A}) \wedge (x_1 = y_1) \wedge \cdots \wedge (x_{\mathrm{ar}(A)} = y_{\mathrm{ar}(A)})\big)\big). \tag{6.72}$$

2. *real-action*$(o, x_1, \ldots, x_m) = A(y_1, \ldots, y_l)$ becomes

$$(o = \mathbf{o_A}) \wedge (x_1 = y_1) \wedge \cdots \wedge (x_k = y_l). \tag{6.73}$$

3. $A(x_1, \ldots, x_l) = A(y_1, \ldots, y_l)$ becomes

$$(x_1 = y_1) \wedge \cdots \wedge (x_l = y_l). \tag{6.74}$$

4. $A(x_1, \ldots, x_l) = A'(y_1, \ldots, y_{l'})$ for different action function symbols A and A' becomes some unsatisfiable expression (e. g., $\neg(\mathbf{o}_A = \mathbf{o}_A)$).

5. $a =$ *real-action*$(o, x_1, \ldots, x_m)$ becomes

$$\bigvee_{A \in \mathcal{A}} \big((o = \mathbf{o_A}) \wedge (a = A(x_1, \ldots, x_{\mathrm{ar}(A)})\big)\,. \tag{6.75}$$

6. $a = A(x_1, \ldots, x_l)$ is not modified.

For all transformations but the fifth it is obvious that they implement the intended meaning of *real-action*. For transformation 5 this is indeed only the case if the interpretation maps a to an object to which we also can refer by an action function. However, case 5 can only occur in the transformation of the right-hand side of an action precondition axiom or a successor state axiom, where a denotes the action applied in state s. Since the unnamed actions do not exist in every model of the theory, they cannot be part of a plan and therefore these axioms are irrelevant if a denotes an unnamed action.

Overall, we reduced quantifications and equalities over unnamed actions to quantifications and equalities over sort *object*. With these modifications, the idea of the compilation scheme from Theorem 10 also works for sort *action*: Models with different numbers of unnamed objects (or in this case actions) are represented explicitly in the theory and are therefore reliably simulated. The argument why it is sufficient to only consider a finite number of such worlds directly carries over from the proof of Theorem 10: models with more unnamed actions than the maximal quantifier rank cannot be distinguished by the theory.

From the same argumentation as in the proof of Theorem 10 it follows that the compilation scheme preserves plan size exactly. Moreover, it is easy to check that it can be computed in polynomial time. □

6.9 Omitting More than One Restriction

We have shown that some of the restrictions of the RBAT formalism are necessary to stay within the expressivity of PDDL while others can be removed or weakened:

- Functions of more general sorts are permissible if their initial values are completely specified for the initial situation.
- The domain closure axiom may be omitted.
- General successor state axioms can be permitted if their right-hand side contains no quantifiers over sort *action*.
- Action precondition axioms and successor state axioms may contain arbitrary symbols of sort *action*.

Each of these results has been achieved under the assumption that all other restrictions are retained. What happens if we omit more than one restriction at once?

Our main motivation for examining the last restriction was a generalization of the result on successor state axioms. We can indeed combine the corresponding compilation schemes.

Theorem 17. *Removing restriction R2 on successor state axioms and at the same time weakening R4.2 so that action symbols can be used in action precondition axioms and successor state axioms does not increase the expressive power of RBAT.*

Proof. In the proof of Theorem 16 we presented a compilation scheme that removes all action symbols forbidden by R4.2. Since this compilation scheme does not depend on a particular syntactic form of the successor state axioms, we can equally apply it to the formalism we consider here. In a second step, we can apply the compilation scheme from RBAT_{ssa} to PDDL. Since both these compilation schemes are polynomial-time and preserve plan size exactly, we also can combine them into one polynomial-time compilation scheme from the source formalism to PDDL preserving plan size exactly (by Theorem 5). □

We also can weaken the restriction on functions *and* give up the requirement of a domain closure axiom without increasing the expressive power.

Theorem 18. *Extending restricted basic action theories with functions as in $RBAT_{functions}$ and at the same time omitting the requirement of a domain closure axiom does not increase the expressive power of the formalism.*

Proof. If more general functions are allowed (R1.2 is weakened), the necessary modification to restriction R3.2 states that all initial function values must be enumerated explicitly (Equation 6.64):

$$\begin{aligned} f(x_1, \ldots, x_n, \mathbf{s_0}) = y \leftrightarrow \\ ((x_1 = \mathbf{c}_{11}) \wedge \ldots \wedge (x_n = \mathbf{c}_{1n}) \wedge (y = \mathbf{c}_1)) \vee \ldots \vee \\ ((x_1 = \mathbf{c}_{m1}) \wedge \ldots \wedge (x_n = \mathbf{c}_{mn}) \wedge (y = \mathbf{c}_m)) \,. \end{aligned}$$

where the $\mathbf{c}_i$ are the named constants. This specification *necessarily implies* that there are no objects besides the named constants. If there existed an unnamed object o in a given model, then the above equation implies $f(o, \ldots, o, s_0) \neq y$ for all objects y. However, functions must have a defined value. Hence, no such model exists.

Therefore, we can weaken R1.2 and remove the domain closure axiom (R3.4) at the same time. Either the basic action theory contains no functions of the form forbidden by R1.2 (in which case it does not matter that R1.2 was weakened), or the removal of R3.4 does not matter because the domain closure axiom is implied. □

Actually, we also can loosen all four restrictions together.

Theorem 19. *We can weaken the RBAT formalism at the same time in all four ways for which we presented positive compilability results without exceeding the expressive power of PDDL.*

Proof. We need to check that the correctness of the individual compilation schemes does not depend on one of the other restrictions we want to weaken and that the compilation schemes do not interfere with each other.

If there are functions we can always first replace them with predicates as in the compilation scheme of Theorem 13.

In a next step, we would apply a combination of the compilation schemes from Theorems 11 and 16. Both compilation schemes simulate a number of worlds with different numbers of unnamed objects (of sort *object* and *action*, respectively). Let k be the maximal number of unnamed objects of sort *object*, and k' be the maximal number of unnamed objects of sort *action* that these compilation schemes would consider. We now need to distinguish worlds $\mathbf{w_i^j}$ ($0 \leq i \leq k$ and $0 \leq j \leq k'$) for all possible settings with up to k unnamed objects of sort *object* and up to k' unnamed objects of sort *action*. Since we also represent unnamed objects of sort *action* with constants of sort *object*, we need to ensure that these representatives do not interfere. This can easily be done by protecting the type of the objects by means of two new predicates $\mathit{object}(x)$ and $\mathit{action}(x)$, e.g. an existentially quantified subexpression $\exists y(\psi)$ over type *object* is replaced with $\exists y(\mathit{object}(y) \wedge \mathit{in\text{-}world}(y, w) \wedge \psi_w)$ instead of only $\exists y(\mathit{in\text{-}world}(y, w) \wedge \psi_w)$. Since these compilation schemes do not depend on a particular form of the successor state axiom, also their combination is applicable in the general case.

After this step, the task is in the form required by formalism RBAT_{ssa}. Therefore, we can apply the compilation scheme from Definition 28 to receive a corresponding PDDL task.

All three compilation steps are polynomial-time computable and preserve plan size exactly. Therefore, we can combine them into a single polynomial-time compilation scheme from the source formalism to PDDL, which also preserves plan size exactly. □

Overall, we have seen that we can freely combine all four changes without adding expressive power.

6.10 Summary of Theoretical Results

We took the restrictions on basic action theories by Eyerich et al. (2006) as basis for our formalism RBAT (Definition 26) and examined for each of the restrictions whether it is necessary for the same expressive power as PDDL.

R1.1: Situation-independent object functions

Restriction R1.1 forbids all situation-independent object functions apart from constants.

We considered a more general formalism $\text{RBAT}_{\textit{sit-independent-functions}}$ that allows functions of sort *object* $\rightarrow$ *object* and generalizes the domain closure axioms so that these functions are not only synonyms for constant symbols. We showed that $\text{RBAT}_{\textit{sit-independent-functions}} \not\preceq^{\text{u}}$ PDDL (Theorem 12).

R1.2: Functional fluents

Restriction R1.2 does not allow the usage of functional fluents of sort $\textit{object}^n \times \textit{situation} \rightarrow \textit{object}$.

If we generalize the domain closure axiom the same way as for situation-independent object functions, the negative result from there directly carries over to functional fluents. Therefore, the formalism $\text{RBAT}_{\textit{functions}}$ introduces functional fluents only as (fluent) synonyms of constant symbols. This does not add expressive power to RBAT and it holds that $\text{RBAT}_{\textit{functions}} \preceq_p^{\text{e}}$ PDDL (Theorem 13).

R2: Syntactic form of successor state axioms

Restriction R2 requires a specific syntactic form of the successor state axioms. We examined a generalized formalism RBAT_{ssa} that allows successor state axioms of the general form $F(\bar{x}, do(a, s)) \leftrightarrow \Phi_F(\bar{x}, a, s)$ if $\Phi_F(\bar{x}, a, s)$ does not contain variables of sort *action* besides a. We showed that $\text{RBAT}_{ssa} \preceq_p^{\text{e}}$ PDDL (Corollary 4) and generalized this result to the entirely unrestricted form of successor state axioms (Theorem 17).

R3.1 and R3.2: Predicate specification in the initial database

Restrictions R3.1 and R3.2 require that the initial database enumerates all initially true ground atoms for the relational fluents and the situation-independent predicates.

$$P(x_1, \ldots, x_n, \mathbf{s_0}) \leftrightarrow (x_1 = \mathbf{c}_{11} \wedge \cdots \wedge x_n = \mathbf{c}_{1n}) \vee \cdots \vee (x_1 = \mathbf{c}_{m1} \wedge \cdots \wedge x_n = \mathbf{c}_{mn}), \tag{6.4}$$

We examined two formalisms that loosen this restriction.

The first formalism allows a space-saving representation with expressions $P(x_1, \ldots, x_n\mathbf{s_0}) \leftrightarrow \varphi_P(x_1, \ldots, x_n)$ as long as these specifications are acyclic. For this generalized formalism $\text{RBAT}_{\textit{compact}}$ it holds that $\text{RBAT}_{\textit{compact}} \not\preceq^{\text{r}}$ PDDL unless $\mathsf{PSPACE} = \mathsf{P}$ (Theorem 7). This is already the case if the compact representation is only allowed for a single predicate of arity 0.

The second formalism, $\text{RBAT}_{\textit{incomp}}$, admits that the truth values of a single unary situation-independent predicate are not specified in the initial database. We showed that $\text{RBAT}_{\textit{incomp}} \not\preceq^{\text{u}}$ PDDL (Theorem 8).

R3.3: Unique names axioms for constant symbols

Restriction R3.3 requires that the initial database contains a unique names axiom $\mathbf{c_i} \neq \mathbf{c_j}$ for each pair of different constant symbols $\mathbf{c_i}, \mathbf{c_j}$.

We studied the formalism RBAT_{noUNA} that allows to omit these axioms for some pairs of constant symbols and showed that $\mathrm{RBAT}_{noUNA} \not\preccurlyeq^{\mathrm{u}}$ PDDL (Theorem 9).

R3.4: Domain closure axiom

Restriction R3.4 requires a domain closure axiom $x = \mathbf{c_1} \vee \cdots \vee x = \mathbf{c_n}$ in the initial database.

We first considered a family of RBAT-variants $\mathrm{RBAT}^{r}_{nodca}$ that do not require a domain closure axiom but therefore set a bound r on the quantifier rank of the goal formula. We have shown that $\mathrm{RBAT}^{r}_{nodca} \preccurlyeq^{\mathrm{e}}_{p}$ RBAT (Theorem 10).

In the next step, we examined the variant $\mathrm{RBAT}^{\infty,=}_{nodca}$ that allows an arbitrary quantifier rank but no predicate symbols except $=$ in the goal formula. Also for this formalism we could show that there is a polynomial-time compilation scheme to RBAT, preserving plan size exactly: $\mathrm{RBAT}^{\infty,=}_{nodca} \preccurlyeq^{\mathrm{e}}_{p}$ RBAT (Lemma 8).

In the last step, we finally considered the formalism $\mathrm{RBAT}^{\infty}_{nodca}$ that is like RBAT but without a domain closure axiom in the initial database. We were able to suitably combine the compilation schemes of $\mathrm{RBAT}^{r}_{nodca}$ and $\mathrm{RBAT}^{\infty,=}_{nodca}$ into a polynomial-time compilation scheme to RBAT preserving plan size exactly: $\mathrm{RBAT}^{\infty}_{nodca} \preccurlyeq^{\mathrm{e}}_{p}$ RBAT (Theorem 11).

It follows directly that $\mathrm{RBAT}^{\infty}_{nodca} \preccurlyeq^{\mathrm{e}}_{p}$ PDDL (Corollary 3).

R4: Usage of sort *action*

Restriction R4 limits the usage of sort *action*. We considered the RBAT variant $\mathrm{RBAT}_{actions}$ that omits restriction R4 and showed that $\mathrm{RBAT}_{actions} \not\preccurlyeq^{\mathrm{u}}$ PDDL (Theorem 15).

Omitting More than One Restriction

Each of the previous results has been achieved under the assumption that all other restrictions are retained. However, we have shown that we can freely combine the changes for which we received positive compilability results without increasing the expressive power of the base formalism (Theorem 19).

7
Experimental Results

We have seen that it is possible to translate a certain fragment of basic action theories to PDDL and that the resulting plan has a direct correspondence to an action sequence in the theory. We now want to examine whether it is worth to integrate a planning system in a Golog system, i.e., whether the expected better runtime of an external planner (compared to the internal search mechanisms) pays of for the translation overhead.

For the integration, we chose the IndiGolog (de Giacomo and Levesque, 1999) framework because it supports interesting features such as sensing actions for gathering additional information and exogenous actions that change the dynamic environment but are not under the control of the agent. This framework allows us to specify tasks that are beyond the scope of classical planning.

The IndiGolog (de Giacomo and Levesque, 1999) implementation[1] we use already supports an operator `achieve` that uses iterative deepening search (courtesy of Hector Levesque) to find a plan for a given goal formula with a given set of actions. Semantically, `achieve` implements the following program construct **solve**.

$$\textbf{solve}(\varphi, [a_1, \ldots, a_n]) := \Sigma(\textbf{while } \neg\varphi \textbf{ do } (\pi a.(a \in [a_1, \ldots, a_n])?; a) \textbf{ endWhile})$$

We extended IndiGolog with an analogous operator `achieve_ext` that implements the same semantics by translating the given subproblem to PDDL, calling an external planning system, and extracting the resulting plan.

For our experiments, we further have extended the IndiGolog framework by a simple simulator that plays the role of the outside world. It runs in a separate instance of PROLOG and communicates with Golog via TCP/IP sockets. The basic idea is to keep track of the (relevant part of the) world state using PROLOG's `assert` and `retract` mechanism. The simulator specifies the outcome of sensing actions according to the true simulated world state. It also can trigger exogenous actions at prespecified time points or if the simulated world state satisfies a given condition.

This chapter is based on joint work with Jens Claßen, Viktor Engelmann and Gerhard Lakemeyer (Claßen et al., 2008) but in this earlier work we only

[1] http://www.cs.toronto.edu/cogrobo/main/systems/

ran experiments with the satisficing FF planning system (Hoffmann and Nebel, 2001). Since the internal planner gives an optimality guarantee, we now use the Fast Downward Planning System[2] (Helmert, 2006) that supports optimal as well as satisficing configurations. We also improved some of the domain formulations.

In the *optimal* planner configuration, we use A* search (Hart et al., 1968) with the LM-Cut heuristic (Helmert and Domshlak, 2009). Since the original LM-Cut heuristic does not support conditional effects, we considered two recent extensions: the basic variant by Keyder et al. (2012) and the variant based on context-splitting by Röger et al. (2014). While context-splitting leads to a better heuristic guidance and better theoretical dominance properties, the basic variant can be computed faster and leads to a higher coverage (Röger et al., 2014). Therefore, we decided to use this latter version in our experiments.

In the *satisficing* configuration of Fast Downward, we use greedy best-first search (Pearl, 1984) with the FF heuristic (Hoffmann and Nebel, 2001) and its preferred operators.

For the experiments, we designed three example application domains, which are all representatives of so-called transportation domains (Helmert, 2001). The characterizing property of such problems is that there are portables that should be transported from their origin to their destination location using mobiles that can move between some of the locations. This type of problem is especially interesting because such tasks arise very often in practice. This is probably also the reason why a large fraction of the benchmarks used in the International Planning Competitions is among these domains. The most interesting aspect of our last example domain is that it in addition involves sensing.

All experiments where conducted on an Intel Core i7-2640M CPU running at 2.80GHz with a timeout (per instance) of 300 seconds and a 2GB memory limit.

7.1 Elevator Domain

The first test domain has been inspired by the Miconic elevator domains of the International Planning Competition in 2000 and is very similar to our example from Figure 4.1. There is an elevator moving between the floors of a building. At some floors passengers are waiting and should be transported to their respective destination floor. The main difference to our previous example domain is that during the program execution new passengers arrive randomly. Since an elevator can move faster if it does not have to stop at each floor, there are additional actions for fast movements that overcome two floors within one step.

In the elevator domain, we defined a control program using prioritized interrupts:

proc *mainControl*
  ⟨ *unservedPassengers* ⟶ *servePassengers* ⟩ ⟫
  ⟨ ¬ *finished* ⟶ `wait` ⟩
endProc

[2] http://www.fast-downward.org/

In each cycle of the (implicit) main loop, if there are passengers that have not been served yet, compute a plan to bring them to their destination floors and execute it. If this is not the case but execution is not yet finished, do nothing for one cycle. Otherwise terminate. Here, *finished* is a fluent that serves as a flag for signaling when program execution is supposed to halt. This is necessary to be able to perform finite experiments for a task that is in principle open-ended.

Testing whether there are unserved passengers is done by a straight-forward procedure *unservedPassengers*:

proc *unservedPassengers*
  $\exists p : passenger.\ \neg served(p)$
endProc

We use exogenous actions of the form `new_ride(p,f1,f2)` to represent newly arriving passengers.

The procedure *servePassengers* uses planning to find a suitable action sequence for transporting all unserved passengers to their destination:

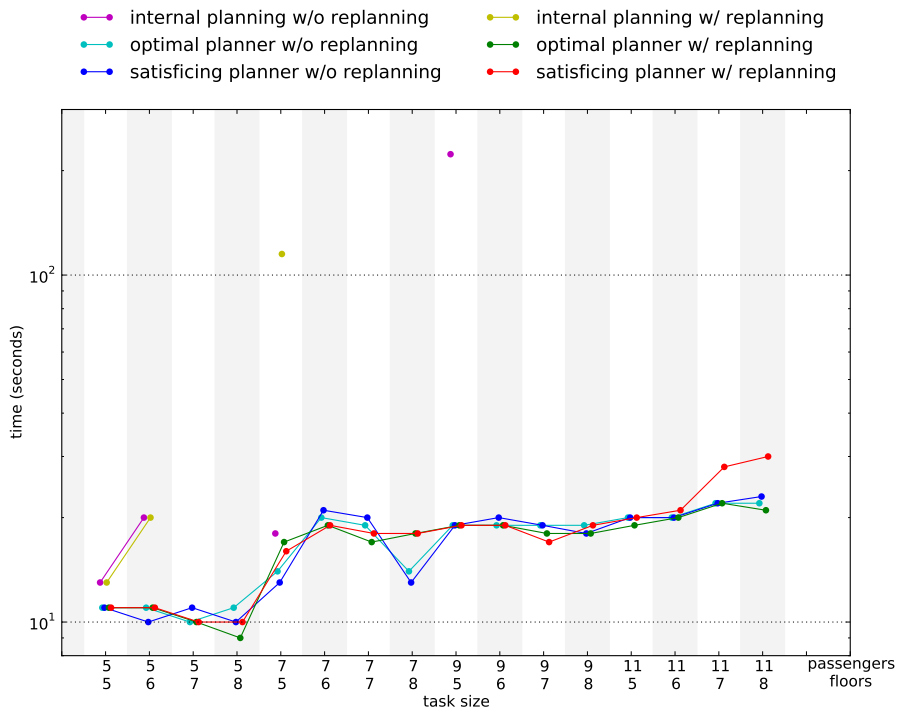
proc *servePassengers*
  solve($\forall p : passenger.\ served(p)$, `[move_fast,move,stop]`) **endProc**

The first argument of **solve** is the goal formula and the second one the list of actions the planner has to consider. In our experiments, we tested two different versions of **solve**: one calling the external planning system via `achieve_ext`, the other, `achieve`, being an internal, PROLOG-implemented construct of the IndiGolog framework that basically performs an iterative deepening search. The two planners were in each case given the same amount of information: a list of available actions, the fluent predicates involved (including their initial values) and the types of objects and action parameters. Whereas the internal `achieve` construct directly uses the appropriate part of the Golog domain axiomatization, the external planner is provided with the corresponding PDDL translation that is created within the `achieve_ext` call.

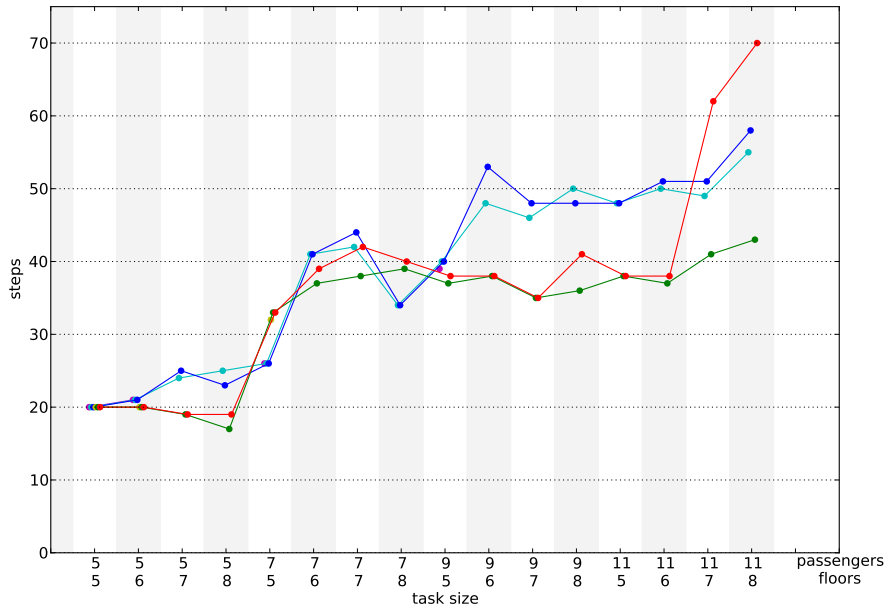
For both of the planners, we considered two variants. In the first one, once a plan is found it gets executed entirely. Passengers arriving during that time can be transported as side-effects of the plan execution but they are only considered actively with the next call of the planner after the current execution ended. In the other variant, the system aborts the current plan execution and performs a re-planning after each `new_ride` event.

For the benchmark instances of this domain we always start with two initially waiting passengers and let the number of new passengers vary among 3, 5, 7 and 9. The number of floors varies between 5 and 8. We created one instance for each combination, i.e., a total of 16 tasks. The origin and destination floors of the passengers are chosen randomly and the intervals between newly arriving passengers lie between 2 and 8 steps, where one step corresponds to the execution of a primitive, non-exogenous action.

For each planner variant and domain instance we measured the overall runtime of the system and the number of steps (minus the number of `wait` actions) that were taken until termination. Runs that did not terminate within 300 seconds were aborted. The runtime includes a wait interval of 0.3 seconds



(a) Runtime in seconds.



(b) Number of steps.

Figure 7.1: Experimental results for elevator tasks. The individual plots are slightly shifted along the x-axis to make overlapping lines visible.

		no replanning			replanning		
		internal	external		internal	external	
Passengers	Floors	opt	opt	sat	opt	opt	sat
5	5	13	**11**	**11**	13	**11**	**11**
5	6	20	11	**10**	20	11	11
5	7	–	**10**	11	–	**10**	**10**
5	8	–	11	10	–	**9**	10
7	5	18	14	**13**	115	17	16
7	6	–	20	21	–	**19**	**19**
7	7	–	19	20	–	**17**	18
7	8	–	14	**13**	–	18	18
9	5	223	**19**	**19**	–	**19**	**19**
9	6	–	**19**	20	–	**19**	**19**
9	7	–	19	19	–	18	**17**
9	8	–	19	**18**	–	**18**	19
11	5	–	20	20	–	**19**	20
11	6	–	**20**	**20**	–	**20**	21
11	7	–	**22**	**22**	–	**22**	28
11	8	–	22	23	–	**21**	30

Table 7.1: Runtime on elevator tasks (in seconds).

after each executed action which was reserved to handle the communication with and the state update of the simulator.

Even our comparatively simple test instances are challenging for the internal planner: while all variants with an external planner easily solve all the tasks, only 4 of them can be solved with the internal search when we do not replan after the arrival of a new passenger and only 3 when we use replanning. All unsolved instances are due to timeouts; the memory consumption was uncritical.

Figure 7.1a and Table 7.1 show the overall runtime for each configuration and each task (with and without replanning). Even if the internal planner solves an instance, the runtime can be significantly larger than with the external planning system (note the a logarithmic scale of the y-axis).

Although it is not the main topic of our experiments, it attracts attention that the results for the external planner appear not to be affected much by the replanning strategy and by the choice of a satisficing or optimal configuration. We suspect that the actual planning tasks are too simple for state-of-the-art planning systems so that the overhead of the IndiGolog system dominates the runtime results. Indeed, the reported total time of each Fast Downward run was 0 seconds, i.e., too low to be measured reliably.

Figure 7.1b and Table 7.2 show the total number of `move_fast`, `move` and `stop` action applications for every configuration (i.e. we omit the wait actions and semaphore actions that were required to handle the replanning strategy because they would induce a negative bias against better plans).

		no replanning			replanning		
		internal	external		internal	external	
Passengers	Floors	opt	opt	sat	opt	opt	sat
5	5	20	20	20	20	20	20
5	6	21	21	21	**20**	**20**	**20**
5	7	–	24	25	–	**19**	**19**
5	8	–	25	23	–	**17**	19
7	5	**26**	**26**	**26**	32	33	33
7	6	–	41	41	–	**37**	39
7	7	–	42	44	–	**38**	42
7	8	–	**34**	**34**	–	39	40
9	5	39	40	40	–	**37**	38
9	6	–	48	53	–	**38**	**38**
9	7	–	46	48	–	**35**	**35**
9	8	–	50	48	–	**36**	41
11	5	–	48	48	–	**38**	**38**
11	6	–	50	51	–	**37**	38
11	7	–	49	51	–	**41**	62
11	8	–	55	58	–	**43**	70

Table 7.2: Number of steps on elevator tasks.

The number of steps required with the internal solver is similar to the number required with the external planning system. Overall, replanning tends to lead to better results, except for the last two instances where the satisficing planner makes some very bad decisions.

It is striking that the optimality guarantee of the internal and the external optimal planner does not always translate to a better (or equal) overall number of steps than we achieve with the satisficing planner.

With replanning, this happens when a current plan is discarded to serve a new request, and when later another new request is made it turns out that following the original plan would have been less costly. However, due to the side-effects of the plan execution on passengers that only arrived after the planning phase (and maybe a luckier choice of the final elevator position in each phase), this happens also without replanning.

7.2 Logistics Domain

Also our second example application domain falls in the class of so-called transportation domains and shall serve as a representative for all kinds of logistics applications. The task is to transport packages to their destination locations, using a number of trucks which can hold up to two packages that can only be unloaded in the reverse order as they have been loaded. The direct connections between locations form a (not necessarily complete) connected graph structure. The domain has the dynamic aspect that new packages keep arriving at run-

time, represented by exogenous actions, and have to be picked up and delivered in turn.

One difference to the elevator domain is that load and unload actions always only affect a single package, so there are no side-effects on newly arriving packages by the current plan application when we do not use replanning. There is no longer a single vehicle but the planner has to decide which trucks to use. Also the capacity restriction of the truck with the constraints on the load and unload order makes this domain different. Moreover, the domain allows more complicated graph structures that the vehicles can traverse.

The experimental setting for the logistics domain is analogous to the one of the elevator domain and uses the following main program.

proc *mainControl*
  $\langle$ *undeliveredPackages* $\longrightarrow$ *deliverPackages* $\rangle\rangle$
  $\langle \neg$ *finished* $\longrightarrow$ `wait` $\rangle$
endProc

The program is to be understood as follows. In each cycle of the (implicit) main loop, if there are packages that have not been delivered yet, compute a plan to deliver them and execute it. If this is not the case but execution is not yet finished, do nothing for one cycle. Otherwise terminate.

Fluent *finished* is again a flag for signaling when program execution is supposed to halt. While delivering the currently pending packages, new delivery requests keep arriving, each of which being modeled by an exogenous action `new_package(p,l,d)` that sets the current location of package `p` to `l`, its destination to `d` and marks the package as undelivered. Since the system does not know in advance when and how many new package arrivals will occur, a special exogenous action `no_more_packages` is used to set *finished* to `TRUE` after the last `new_package` event indicating that the experiment ends at this point.

Testing whether there are still packages to be delivered is done by the simple procedure *undeliveredPackages*:

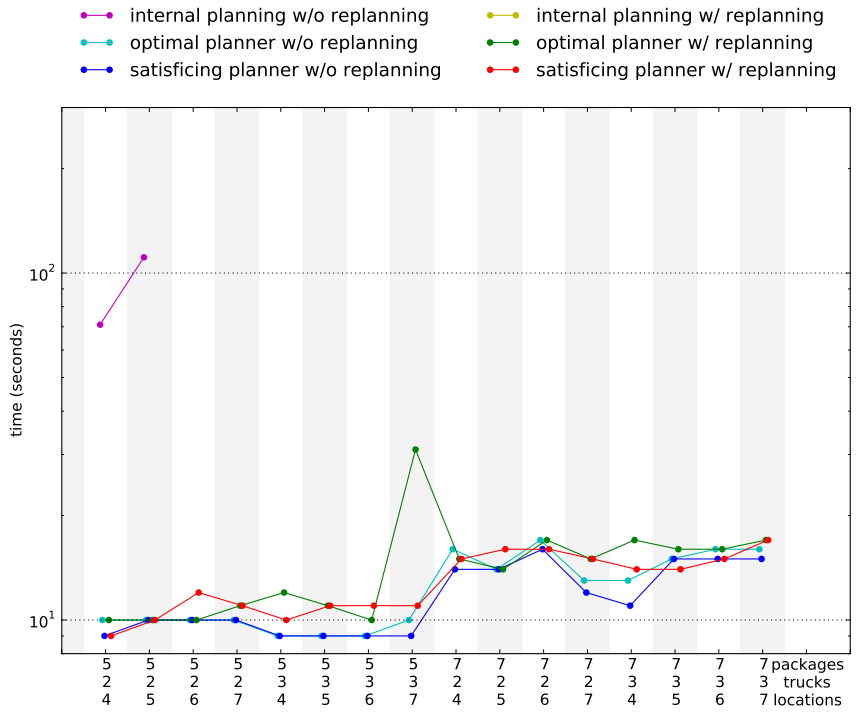
proc *undeliveredPackages*
  $\exists p : package.\ undeliveredPackage(p)$
endProc

The *deliverPackages* procedure is in charge of planning the delivery of packages:

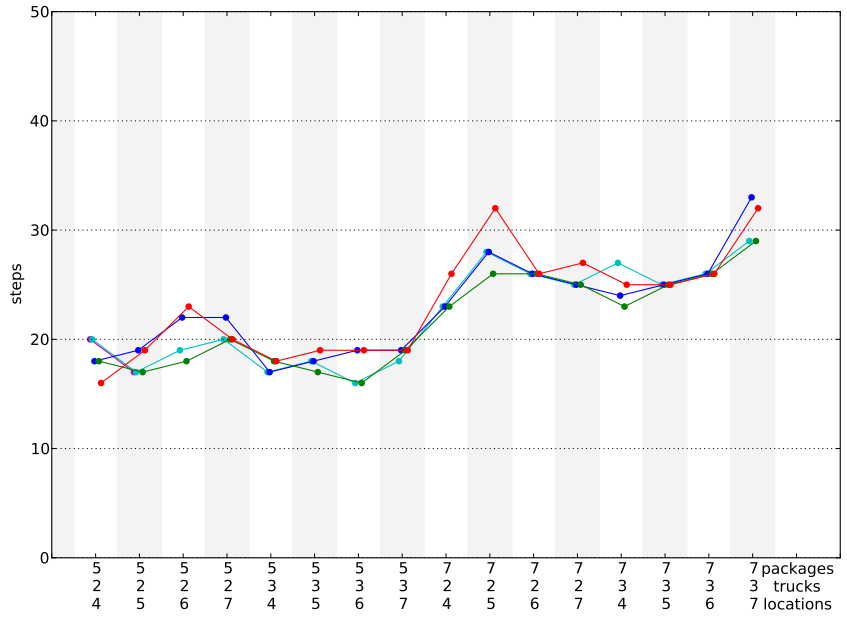
proc *deliverPackages*
  solve($\forall p : package.\ \neg undeliveredPackage(p)$, `[load,unload,drive]`)
endProc

Again, we tested two versions of **solve**, one calling the internal planner, the other calling the external planning system, each with and without re-planning on the arrival of a new package.

We performed a series of experiments where the number of locations varied between 4 to 7, the number of trucks between 2 and 3 and the number of dynamically arriving packages among 3 and 5. Initially, there are always 2 additional packages waiting for delivery. The initial locations of trucks and packages as well as the destinations of the packages are chosen randomly. Two locations are connected with a probability of 50%; additional random edges



(a) Runtime in seconds.



(b) Number of steps.

Figure 7.2: Experimental results for logistics tasks. The individual plots are slightly shifted along the x-axis to make overlapping lines visible.

			no replanning			replanning		
			internal	external		internal	external	
Pack.	Trucks	Locations	opt	opt	sat	opt	opt	sat
5	2	4	71	10	**9**	–	10	**9**
5	2	5	111	**10**	**10**	–	**10**	**10**
5	2	6	–	**10**	**10**	–	**10**	12
5	2	7	–	**10**	**10**	–	11	11
5	3	4	–	**9**	**9**	–	12	10
5	3	5	–	**9**	**9**	–	11	11
5	3	6	–	**9**	**9**	–	10	11
5	3	7	–	10	**9**	–	31	11
7	2	4	–	16	**14**	–	15	15
7	2	5	–	**14**	**14**	–	**14**	16
7	2	6	–	17	**16**	–	17	**16**
7	2	7	–	13	**12**	–	15	15
7	3	4	–	13	**11**	–	17	14
7	3	5	–	15	15	–	16	**14**
7	3	6	–	16	**15**	–	16	**15**
7	3	7	–	16	**15**	–	17	17

Table 7.3: Runtime on logistics tasks (in seconds).

ensure that the roadmap graph forms a single connected component. The intervals between the arrival times of new packages vary between 2 and 8 steps.

The interval between action executions was again set to 0.3 seconds.

As with the elevators domain, it was hard to find sufficiently easy instances that could also be solved by the internal planning mechanism. Therefore only two of our generated tasks were solved by the internal search, both without replanning.

Figure 7.2a and Table 7.3 show the runtimes. For the two instances solved with the internal `achieve` mechanism, the process took about a magnitude longer than with the external planner. There is a tendency that solving the tasks with replanning takes a bit longer, but the difference is in a range of 3 seconds, except for one task that required with the optimal planner configuration 31 seconds with replanning compared to only 10 seconds without. Of these 31 seconds, 8.1 seconds relate to the sleep intervals between action executions and 20.33 seconds to the four planner calls (where the last one required 18.59 seconds). For most other tasks, the optimal planner calls took a fraction of a second, except for very few that were still solved withing 2 seconds. With the satisficing configuration, no Fast Downward call took more than 0.03 seconds. The time required for translating from the basic action theory to PDDL and extracting the plan is negligible. Overall, we again observe that tasks that are not challenge for an external planning system are by far out of reach of the internal search.

Figure 7.2b and Table 7.4 show the total number of `load`, `unload` and

			no replanning			replanning		
			internal	external		internal	external	
Pack.	Trucks	Locations	opt	opt	sat	opt	opt	sat
3	2	4	20	20	18	–	18	**16**
3	2	5	**17**	**17**	19	–	**17**	19
3	2	6	–	19	22	–	**18**	23
3	2	7	–	**20**	22	–	**20**	**20**
3	3	4	–	**17**	**17**	–	18	18
3	3	5	–	18	18	–	**17**	19
3	3	6	–	**16**	19	–	**16**	19
3	3	7	–	**18**	19	–	19	19
5	2	4	–	**23**	**23**	–	**23**	26
5	2	5	–	28	28	–	**26**	32
5	2	6	–	26	26	–	26	26
5	2	7	–	**25**	**25**	–	**25**	27
5	3	4	–	27	24	–	**23**	25
5	3	5	–	25	25	–	25	25
5	3	6	–	26	26	–	26	26
5	3	7	–	**29**	33	–	**29**	32

Table 7.4: Number of steps on trucks tasks.

`drive` actions. Neither optimal planning nor replanning gives consistently a lower number of steps so for our benchmark tasks the additional overhead does not pay off in the presence of unpredictable exogenous events.

7.3 Mail Delivery Robot

The third domain is a variant of a common application example (Tam et al., 1997) of a mobile robot operating in an office environment, where it has to deliver letters and parcels between the workers' mailboxes. Here, the structure of the building is assumed to consist of a number of hallways, which are connected (e.g. by an elevator) to other hallways, and where there is a certain number of offices at each hallway. Each office may contain one or multiple different mailboxes, each of which serving for both incoming and outgoing mails.

This domain involves sensing since the robot must look into a mailbox in order to find out how many and which letters it currently contains. Furthermore, before the agent actually knows where to deliver a letter, it has to pick it up and read off the addressee. For this domain, we assume that no new letters appear online as it would for example be the case if the robot delivered the letters at night when no-one is working in the office. An extension to newly arriving letters could be implemented analogously to the previous domains.

The mail delivery robot is controlled as follows:

proc *mainControl*
  while $(\exists m : mailbox.\ \neg cleared(m))$ **do**
    $(\pi\ m : mailbox.\ ?(\neg cleared(m)); getLettersFrom(m));$
    deliverLetters
  endWhile
endProc

As long as the robot is not sure that all mailboxes are cleared, it randomly picks a candidate mailbox and collects all letters from there with the procedure *getLettersFrom*.

proc *getLettersFrom*(m)
  $(\pi\ l : location.$
    $at(m, l)?;$
    solve$(robotAt(l),$ `[move]`$));$
    `look_into`$(m);$
    takeAllLetters(m)
endProc

The path to the chosen mailbox is determined by means of planning. The robot then looks into the mailbox (a sensing action) and takes out all letters from this mailbox with the following procedure:

proc *takeAllLetters*(m)
  while $(\exists l : letter.\ in(l, m))$
    $(\pi\ l : letter.$
      $in(l, m)?;$
      `take_out`$(l, m);$
      `look_at`$(l))$
    `look_into`(m)
  endWhile
endProc

The outcome of the sensing action `look_into`(m) is a constant l denoting one of the letters in the box (i.e. the robot can always only "see" the topmost one). Thus, the agent gets to know that fluent $in(l, m)$ is currently true. In case the mailbox is empty, the return value is instead simply the special constant "`empty`" and $cleared(m)$ is set to true. After picking up l, action `look_at`(l) is applicable and causes $addressee(l, m')$ to become known to the agent for some mailbox m', which is the destination of letter l. If the addressee is not the current mailbox, the action makes atom $to\_deliver(l)$ true. For delivering the letters obtained like this, we conceptually would like to use the planner with a procedure like this:

proc *deliverLetters*$'$
  solve$(\forall l : letter.\ to\_deliver(l) \rightarrow delivered(l),$ `[put_in,move]`$)$
endProc

The disadvantage of this program is that the robot would visit many mailboxes, putting letters in but not picking up letters on the go. However, the intended behavior of the robot for checking the mailboxes is almost deterministic (in the sense that the actual choice of the nondeterministic aspects does

not affect the outcome or the quality of the plan): If the robot puts a letter in a mailbox, it should look into the mailbox. If it is not empty, it should collect all letters with the procedure *takeAllLetters*. Since this behavior requires sensing, we cannot integrate it in the call of the planner. However, collecting letters from the visited mailboxes does not invalidate the plan for delivering the original letters. We therefore extended the implementation of **solve** so that we can specify for each action a macro that is executed instead of each actually planned action. In this example, we use for each planned action $\texttt{put\_in}(l, m)$ the macro $\texttt{put\_in}(l, m); \texttt{look\_into}(m)$. The `look_into` action detects if there is a letter in the mailbox. In this case, we can start the *takeAllLetters* procedure by means of prioritized interrupts:

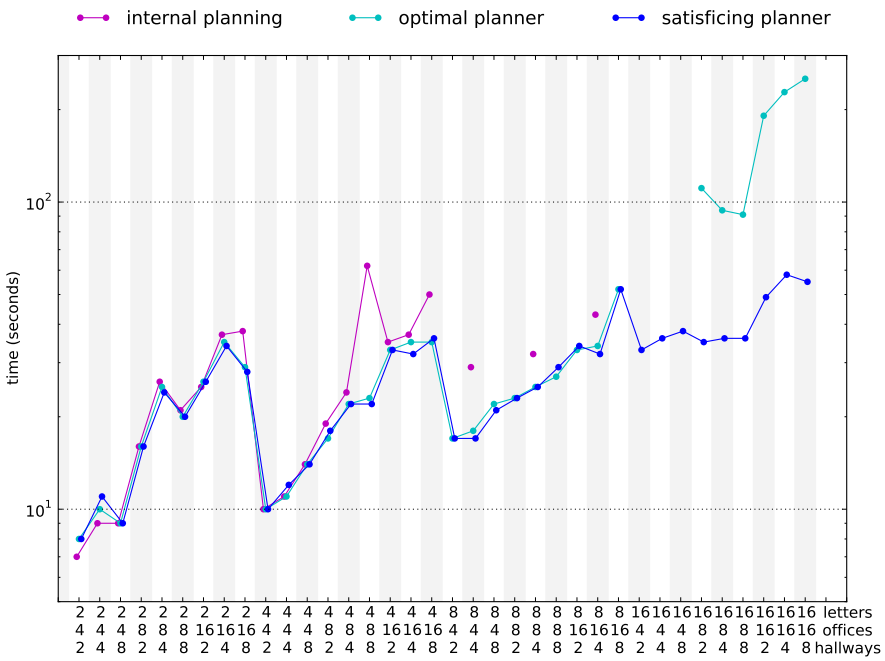
proc *deliverLetters*
  $\langle\ \exists l : \mathit{letter},\ m : \mathit{mailbox}.\ \mathit{in}(l, m) \longrightarrow$
    $(\pi\ m : \mathit{mailbox}.\ ?(\exists l : \mathit{letter}.\ \mathit{in}(l, m)); \mathit{takeAllLetters}(m))\ \rangle$
  $\rangle\rangle$ **solve**$(\forall l : \mathit{letter}.\ \mathit{to\_deliver}(l) \rightarrow \mathit{delivered}(l),$ `[put_in,move]`$)$
endProc

Again, we study the system's behavior for the case in which **solve** uses the internal planner and for the case where an external planning system is called instead.

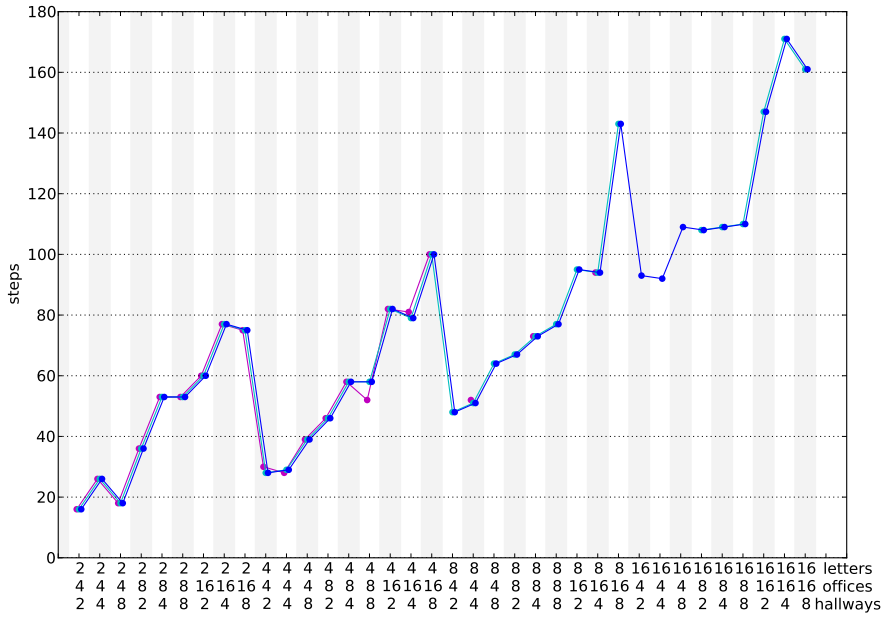
In our benchmark scenarios the number of offices varies among 4, 8 and 16, the number of hallways among 2, 4 and 8, and the number of letters among 2, 4, 8 and 16. As in the other domains we created 10 instances for each combination, the offices being connected randomly to some hallway and hallways being connected to one another in a tree-like fashion. There are as many mailboxes as offices, but they are placed randomly. Therefore, it is possible that an office contains multiple mailboxes, only one, or even none at all. The origins and addressees of the letters are also chosen randomly. We again used a timeout of 300 seconds and an action execution interval of 0.3 seconds.

On this domain, the IndiGolog interpreter solved 21 of the 36 tasks using the internal mechanism, all but three tasks using the external optimal planner, and all tasks using the external satisficing planner. All unsolved tasks were due to timeouts.

The runtimes for each method are given in Table 7.5 and shown graphically in Figure 7.3a. One aspect of the mail delivery domain that can affect the planning time a lot, is how many letters the robot has to deliver when *deliverLetters* is called. While this number is not very critical for a satisficing planner, it is problematic for the approaches that need to give an optimality guarantee. This makes the instances with 16 letters and 4 mailboxes (= number of offices) especially hard, and indeed only the satisficing variant is able to solve these instances. For the easiest instances, the overhead of the IndiGolog interpreter is so large that it dominates the overall runtime by far. On some instances, the execution interval of 0.3 seconds per step already explains most of the runtime. On the intermediately hard instances with 8 letters, we can see the difference between the internal and external optimal solvers. While the external search can still easily solve these instances, the internal variant only solves instances sporadically, albeit then with not much more required time. For the hard instances with 16 letters also the external optimal solver



(a) Runtime in seconds.



(b) Number of steps.

Figure 7.3: Experimental results for mail delivery tasks. The individual plots are slightly shifted along the x-axis to make overlapping lines visible.

			steps			runtime		
			internal	external		internal	external	
Letters	Offices	Hallways	opt	opt	sat	opt	opt	sat
2	4	2	16	16	16	**7**	8	8
2	4	4	26	26	26	9	10	11
2	4	8	18	18	18	9	9	9
2	8	2	36	36	36	16	16	16
2	8	4	53	53	53	26	25	**24**
2	8	8	53	53	53	21	**20**	**20**
2	16	2	60	60	60	**25**	26	26
2	16	4	77	77	77	37	35	**34**
2	16	8	75	75	75	38	29	**28**
4	4	2	30	**28**	**28**	10	10	10
4	4	4	**28**	29	29	**11**	**11**	12
4	4	8	39	39	39	14	14	14
4	8	2	46	46	46	19	**17**	18
4	8	4	58	58	58	24	**22**	**22**
4	8	8	**52**	58	58	62	23	**22**
4	16	2	82	82	82	35	**33**	**33**
4	16	4	81	**79**	**79**	37	35	**32**
4	16	8	100	100	100	50	**35**	36
8	4	2	–	**48**	**48**	–	**17**	**17**
8	4	4	52	**51**	**51**	29	18	**17**
8	4	8	–	**64**	**64**	–	22	**21**
8	8	2	–	**67**	**67**	–	**23**	**23**
8	8	4	73	73	73	32	**25**	**25**
8	8	8	–	**77**	**77**	–	**27**	29
8	16	2	–	**95**	**95**	–	**33**	34
8	16	4	94	94	94	43	34	**32**
8	16	8	–	**143**	**143**	–	**52**	**52**
16	4	2	–	–	**93**	–	–	**33**
16	4	4	–	–	**92**	–	–	**36**
16	4	8	–	–	**109**	–	–	**38**
16	8	2	–	**108**	**108**	–	111	**35**
16	8	4	–	**109**	**109**	–	94	**36**
16	8	8	–	**110**	**110**	–	91	**36**
16	16	2	–	**147**	**147**	–	191	**49**
16	16	4	–	**171**	**171**	–	228	**58**
16	16	8	–	**161**	**161**	–	252	**55**

Table 7.5: Number of steps and runtime (in seconds) on mail delivery robot tasks.

is pushed towards its limits, while the satisficing variant solves each of this instances within a minute.

The number of steps for each method is also included in Table 7.5. A graphical representation can be found in Figure 7.3b. There is no large difference between the variants, which is not surprising because the Golog program and the basic action theory already fix most of the steps: there is exactly one `take_out`, one `put_in`, one `look_at`, and one `look_into` action for each letter, and there is always one additional `look_into` action for each mailbox. The only way the planners can influence the overall number of steps is by choosing better routes. But even then, an optimal system is not guaranteed result in a lower number of steps because it can be unlucky if there are several possible optimal routes for delivering the current letters but one ends in a more favorable location. This also explains the cases where the number of steps differs for the two optimal variants.

We observed in all three domains that the external planner scales much further than the internal solver and that it provides better runtimes except for the most trivial instances. In the elevators and the trucks domain, the planning subtasks where so simple that the optimal and the satisficing configuration of the external planner lead to similar results, both in terms of runtime and number of steps. However, in the presence of unpredictable exogenous events or sensing, the optimality guarantee of the subplanner does not translate to an overall optimality guarantee. Due to their better scaling behavior, we therefore expect that using satisficing planners is in most scenarios the better choice. This is also indicated by the results on the mail delivery domain, where the optimal configuration was not able to solve all tasks. Nevertheless, there might be applications where optimal planning is more suitable, e.g. because there are no exogenous events and sensing is not required, or because the actual execution of an action takes a lot of time.

The main advantage of our approach is that the IndiGolog system uses the common input language PDDL for communicating with the external planner. Therefore, we can for each application plug in the planning system that is best-suited for the given scenario, whether it is suboptimal or not, always using the latest state-of-the-art planning techniques.

Part II

Potential and Limitations of Heuristic Search for Planning

8

Heuristic Search in Automated Planning

The predominant approach for classical planning is *heuristic state space search*. In the context of optimal planning, the most widely used algorithm is A*(Hart et al., 1968, 1972) in combination with an admissible heuristic. Indeed, all awards in the sequential optimization track of the last International Planning Competition (IPC 2011) went to planners using A* (all implemented on top or within the Fast Downward planning system): The winner *Fast Downward Stone Soup* is a portfolio planner combining several A* searches (with different heuristics) and one configuration using the A* variant LM-A* (Karpas and Domshlak, 2009), which is optimized for the use with landmarks. There were two runner-ups ex-aequo: *SelMax* (Domshlak et al., 2011, 2012a) also runs the A* variant LM-A* and learns online which heuristic to evaluate in each state. The *Merge-and-Shrink planner* (Nissim et al., 2011b) uses A* with merge-and-shrink abstraction heuristics (Nissim et al., 2011a)[1].

In satisficing planning, where we are only interested in finding good but not necessarily optimal plans, there are several heuristic search algorithms in use. The most common approaches are probably greedy best-first search (Pearl, 1984) and weighted A*(Pohl, 1970), which are used by many planning systems like for example the winner *LAMA* (Richter and Westphal, 2010; Richter et al., 2011) and the runner-up *Fast Downward Stone Soup* of the International Planning Competition in 2011. An alternative heuristic search algorithm in satisficing planning is *Enforced Hill-Climbing*, used by the very successful planning system *FF* (Hoffmann and Nebel, 2001).

Since heuristic state space search is so important for classical planning, we examine in the following the potential but also the limitations of heuristic search in planning. The emphasis lies not on the development of entirely new approaches, but rather on the question of how far we can get with standard heuristic search and how we can get the best out of existing techniques.

In the first section, we concentrate on the potential of heuristic search algorithms such as A* and IDA* (Korf, 1985) for obtaining *optimal* sequential solutions. Theoretical analyses, such as the well-known results of Pohl (1977), Gaschnig (1977) and Pearl (1984), suggest that such heuristic search algorithms

[1] To do justice to other approaches: *Gamer* (Edelkamp and Kissmann, 2008), the winner of the sequential optimization track of the IPC 2008, runs a symbolic search rather than an A* variant.

can obtain better than exponential scaling behavior, provided that the heuristics are accurate enough. We show that for a number of common planning benchmark domains, including ones that admit optimal solution in polynomial time, general search algorithms such as A* must *necessarily* explore an exponential number of search nodes even under the optimistic assumption of *almost perfect* heuristic estimators, whose heuristic error is bounded by a small additive constant.

In the second section we concentrate on the *satisficing* setting. The main reason for the limitations of heuristic search in the optimal setting is that the search needs to process a large part of the search space to prove the optimality of the solution. Therefore, the results from the first section do not carry over to planning without quality guarantees and it looks more effective there to reach for better heuristic guidance. One possible approach is to develop new, stronger estimators, an alternative one to use *multiple* existing heuristics concurrently with the idea of exploiting their complementary strengths.

While the problem of effectively combining multiple heuristic estimators has been studied extensively in the context of optimal planning, this is not the case for the satisficing setting. To narrow this gap, we empirically examine several ways of exploiting the information of multiple heuristics in a satisficing best-first search algorithm, comparing their performance in terms of coverage, plan quality and runtime. Our empirical results indicate that using multiple heuristics for satisficing search is indeed useful and that the best results are not obtained by the most obvious combination methods.

9 Limitations of Pure Heuristic Search

Optimal sequential planning is harder than satisficing planning. While there is no difference in theoretical complexity in the general case (Bylander, 1994), many of the classical planning domains are provably easy to solve sub-optimally, but hard to solve optimally (Helmert, 2003).

Moreover, strikingly different scaling behavior of satisficing and optimal planners has been observed in practice (Hoffmann and Edelkamp, 2005). In fact, this disparity even extends to planning domains which are known to be *easy* to solve optimally in theory. If we apply two state-of-the-art optimal planning algorithms (Haslum et al., 2007; Helmert et al., 2007) to the GRIPPER domain, neither of them can optimally solve more than 8 of the standard suite of 20 benchmarks within reasonable run-time and memory limits, whereas the whole suite is solved in a few milliseconds by satisficing planners like FF (Hoffmann and Nebel, 2001). Moreover, those 8 tasks are quickly solved by breadth-first search, showing no significant advantage of sophisticated heuristic methods over brute force.

Why is this the case? One possible explanation is that the heuristic estimators of these planning systems may be grossly misleading for GRIPPER tasks. However, we do not believe that this is the case – the GRIPPER domain in particular has resisted many attempts by optimal heuristic planners, hinting at a different, more fundamental problem. In this chapter, we argue the following claim:

> For many, maybe *all* of the standard benchmark domains in planning, standard heuristic search algorithms such as A* quickly become prohibitively expensive even if *almost perfect heuristics* are used.

In the following, we will formally define what we mean with the notion of *almost perfect* heuristics and discuss related work. Afterwards, we will present theoretical results for problem families in three standard IPC benchmark domains. An empirical evaluation will show that we can observe the same problematic behavior on the actual IPC benchmark tasks. We conclude with some ideas how this limitation of pure heuristics search could be overcome by means of complementary techniques.

9.1 Almost Perfect Heuristics

The performance of heuristic search is commonly measured by the number of performed node expansions. Of course, this measure depends on the search algorithm used; for example, A* with admissible and consistent heuristics will usually explore fewer states than IDA* in the same search space, and never more (assuming that successors are ordered in the same way).

Here, we consider lower bounds for node expansions of the A* algorithm with admissible and consistent heuristics and with full duplicate elimination. Results for this algorithm immediately apply to other search algorithms that rely exclusively on node expansions and admissible heuristic estimates to guide search, such as IDA*, A* with partial expansion (Yoshizumi et al., 2000), breadth-first heuristic search (Zhou and Hansen, 2006), and many more. However, they do *not* apply to algorithms that use additional information for state pruning, such as symmetry reduction (Fox and Long, 1999; Rintanen, 2003; Pochter et al., 2011; Domshlak et al., 2012b), and neither to algorithms that use fundamentally different techniques to find optimal plans, such as symbolic breadth-first search (Edelkamp and Helmert, 2001; Kissmann, 2012) or SAT planning (Kautz and Selman, 1999; Rintanen, 2010).

How many nodes does A* expand for a planning task $\mathcal{T}$, given an admissible heuristic h? Clearly, this depends on the properties of $\mathcal{T}$ and h. It also depends on some rather accidental features of the search algorithm implementation, in particular on the order in which nodes with identical f values are explored. Because we are interested in lower bounds, one conservative assumption is to estimate the solution effort by the number of states s with the property $f(s) := g(s) + h(s) < h^*(\mathcal{T})$, where $g(s)$ is the cost (or distance – we assume a unit cost model) for reaching s from the initial state, $h(s)$ is the heuristic estimate for s, and $h^*(\mathcal{T})$ is the optimal solution cost for $\mathcal{T}$. All states with this property *must* be considered by A* in order to guarantee that no solutions of length below $h^*(\mathcal{T})$ exist. In practice, a heuristic search algorithm will also expand *some* states with $f(s) = h^*(\mathcal{T})$, but we ignore those in our estimates.

Our aim is to demonstrate fundamental limits to the scaling possibilities of optimal heuristic search algorithms when applied to planning tasks. For this purpose, we show that A* search effort already grows extremely fast for a family of very powerful heuristic functions. More precisely, we consider heuristics parameterized by a natural number $c \in \mathbb{N}_1$ where the heuristic estimate of state s is defined as $\max(h^*(s) - c, 0)$, with $h^*(s)$ denoting the length of an optimal solution from state s as usual. In the following, we use the notation "$h^* - c$" to refer to this heuristic (not reflecting the maximization operation in the notation). In other words, $(h^* - c)(s) := \max(h^*(s) - c, 0)$.

We call heuristics like $h^* - c$, which only differ from the perfect heuristic h^* by an additive constant, *almost perfect*. Almost perfect heuristics are unlikely to be attainable in practice in most planning domains. Indeed, for any of the planning benchmark domains from IPC 1998–2004 that are NP-hard to optimize, if there exists a polynomial-time computable almost perfect heuristic, then APX = PTAS and hence P = NP (Helmert et al., 2006).

Throughout our analysis, we use the notation $N^c(\mathcal{T})$ to denote the A* node expansion lower bound for task $\mathcal{T}$ when using heuristic $h^* - c$. In other words, $N^c(\mathcal{T})$ is the number of states s with $g(s) + (h^* - c)(s) < h^*(\mathcal{T})$. This can be equivalently expressed as the number of states with $f^*(s) := g(s) + h^*(s) <$

$h^*(\mathcal{T}) + c$ and $g(s) < h^*(\mathcal{T})$. The objective of our analysis is to provide results for $N^c(\mathcal{T})$ for planning tasks $\mathcal{T}$ drawn from the standard IPC benchmark domains, focusing on the "easiest" domains, namely those that fall within the approximation class APX (Helmert et al., 2006). These domains are particularly relevant because they would intuitively appear most likely to be within reach of optimal planning techniques.

9.2 Related Work

There is quite a bit of literature on the computational costs of A* and related algorithms. A widely cited result by Pohl (1977) considers precisely the kind of heuristics we study in this paper, i.e., those with constant absolute error. He proves that the A* algorithm requires a linear number of node expansions in this case. However, the analysis relies on certain critical assumptions which are commonly violated in planning tasks. First, it assumes that the branching factor of the search space is constant across inputs. If the branching factor is polynomially related to input size, as is often the case in planning, the number of node expansions is still polynomial in the input size for a fixed error c, but of an order that grows with c. More importantly, the analysis requires that there is only a single goal state and that the search space contains no transpositions, i.e., every state can only be reached by a single path. The latter assumption, critical to Pohl's tractability result, is violated in all common benchmark tasks in planning. For example, in the following section we show that the GRIPPER domain requires an exponential number of node expansions even with very low heuristic inaccuracies due to the large number of transpositions. Pohl also considers the case of heuristics with a constant *relative* error, i.e., $h(s) \approx c \cdot h^*(s)$ for some constant $c < 1$. However, as our negative results already apply to the much more optimistic case of constant absolute error, we do not discuss this analysis.

Gaschnig (1977) extends Pohl's analysis to logarithmic absolute error (i.e., $h^*(s) - h(s) = O(\log(h^*(s)))$), showing that A* also requires a polynomial number of expansions under this less restrictive assumption. However, his analysis requires the same practically rare search space properties (constant branching factor, no transpositions, a single goal state).

Both Pohl's and Gaschnig's work is concerned with worst-case results. Pearl (1984) extends their analyses by showing that essentially the same complexity bounds apply to the *average case*.

More recently, Dinh et al. (2007) consider heuristics with constant relative error in a setting with multiple goal states. While this is a significant step towards more realistic search spaces, absence of transpositions is still assumed.

In addition to these analyses of A*, the literature contains a number of results on the complexity of the IDA* algorithm. The article by Korf et al. (2001) is particularly relevant in the planning context, because it presents an analysis which has recently been used to guide heuristic selection in the very effective[1] optimal sequential planner of Haslum et al. (2007). Korf et al. show that IDA* expands at most $\sum_{i=0}^{k} N_i P(k-i)$ nodes to prove that no solution of length at most k exists (or to find such a solution, if it exists). Here, N_i is the

[1] With an efficient implementation of the underlying pattern database heuristics (Sievers et al., 2012).

number of nodes at depth i in the search tree, and P is the *equilibrium distribution* with $P(m)$ being the probability that a randomly drawn node "deep" in the brute-force search tree has a heuristic value of at most m. However, the formula only applies to IDA*, and only in the limit of large k. We are interested in lower bounds on complexity for *any* general heuristic search algorithm, including ones that perform complete duplicate elimination. In that case, the number of states N_i at depth i eventually becomes zero, so that it is not clear what results that apply "in the limit of large k" signify. Moreover, with complete duplicate elimination, there is no reasonable way of defining the equilibrium distribution P. Zahavi et al. (2010) refined the analysis by Korf et al., based on a *conditional* distribution of heuristic values that also takes properties of the neighborhood of a search node into consideration. Since with duplicate elimination there is no reasonable way of defining the conditional or unconditional distribution, we do not discuss these complexity results further.

9.3 Theoretical Results

There are several ways of obtaining $N^c(\mathcal{T})$ estimates. One way is to *measure* them empirically for particular values of c and particular tasks $\mathcal{T}$ by using an algorithm that conducts A* searches with the $h^* - c$ heuristic. One advantage of this method is that it is fully general – it can be directly applied to arbitrary planning tasks, and any value $c \in \mathbb{N}_1$. We present some results obtained by this method in the following section.

However, the empirical approach has some drawbacks. Firstly, it is computationally expensive and thus limited to comparatively small planning tasks (in particular, a subset of those which we can solve optimally by heuristic search). Secondly, its results are fairly opaque. In addition to knowing lower bounds for a certain set of planning tasks, we would also like to know *why* they arise, and how to extrapolate them to instances of larger size. For these purposes, theoretical results are preferable.

In this section, we present such theoretical results for three planning domains: Gripper, Miconic-Simple-Adl, and Blocksworld. We assume familiarity with these domains and point to the literature (Helmert, 2008) for definitions and details. We choose these particular domains because we consider them fairly representative of the IPC domains in class APX. Similar theorems can be shown for most other APX domains such as Miconic-Strips, Logistics, ZenoTravel, Depots, and Schedule.

Our theorems take the form of *worst case* results: In each of the domains, we show that there exist tasks of scaling size for which the number of states expanded by $h^* - c$ grows exponentially. One problem with worst case considerations is that they might only apply in some fairly unusual "corner cases" that are unlikely to arise in practice. We partially avoid this problem by discussing families of tasks of different types. For example, we can observe exponential scaling of $N^i(\mathcal{T})$ both for families of Blocksworld tasks where initial and goal configurations consist of a single tower and for families of tasks consisting of a large number of small towers. Still, *average-case* results are clearly also of interest, and are left as a topic for future work.

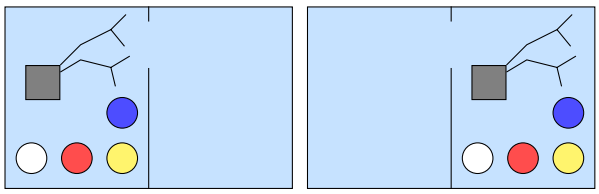


Figure 9.1: Initial state and goal of GRIPPER task $\mathcal{T}_4$.

Gripper

The GRIPPER domain was included in the IPC 1998 benchmark set because its tasks are trivially to solve for humans but lead to a combinatorial explosion with most state-space search methods (McDermott, 2000). We will show that this is indeed also the case with almost perfect heuristics.

In this domain, a robot with two arms needs to transport a number of balls from one room to another (Fig. 9.1). All balls and the two robot arms can be distinguished: for example, if two situations only differ in that the robot holds a specific ball once with the left arm and once with the right arm, these two situations correspond to two different states in the state space. Initially, the robot and the balls are always located in one room (*room A*). The robot can *move* between the rooms, and *pickup* or *drop* a ball with each gripper. In the goal, all balls must be in the other room (*room B*). The robot position does not matter for the goal.

Since GRIPPER tasks are completely characterized by the number of balls, there is no difference between worst-case and average-case results in this domain. We denote with $\mathcal{T}_n$ ($n \in \mathbb{N}^+$) the task with n balls. For $n \geq 2$, the total number of reachable states of $\mathcal{T}_n$ is $S_n := 2 \cdot (2^n + 2n2^{n-1} + n(n-1)2^{n-2})$. (The initial factor of 2 represents the two possible robot locations; the three terms of the sum correspond to the cases of 0, 1 and 2 carried balls, respectively, and represent the contents of the robot arms and the locations of the balls not being carried. Note that the two robot arms can be distinguished.)

A GRIPPER task can be solved optimally by repeating the following procedure until the task is solved (Helmert, 2008, p. 76):

1. Pickup any two balls (or one if only one is left in the room).
2. Move to room B.
3. Drop all carried balls.
4. If not all balls are in room B, move to room A.

So, for an even number n of balls, an optimal plan has length $h^*(\mathcal{T}_n) = 3n - 1$, for a odd number of balls, $h^*(\mathcal{T}_n) = 3n$

We now state our main result for GRIPPER.

Theorem 20. *Let $n \in \mathbb{N}_0$ with $n \geq 3$. If n is even, then $N^1(\mathcal{T}_n) = N^2(\mathcal{T}_n) = \frac{1}{2}S_n - 3$ and $N^c(\mathcal{T}_n) = S_n - 2n - 2$ for all $c \geq 3$. If n is odd, then $N^1(\mathcal{T}_n) = N^2(\mathcal{T}_n) = S_n - 3$ and $N^c(\mathcal{T}_n) = S_n - 2$ for all $c \geq 3$.*

Proof. For a GRIPPER state s, let A_s, B_s, and R_s be the set of balls which are in room A, in room B, and carried by the robot in state s, respectively. In the following, we will use the notation $B_s \prec R_s \prec A_s$ to indicate that whenever there is a choice between several balls (e. g., when picking them up or dropping them), those from B_s will be preferred to all others and balls from R_s will be preferred to balls from A_s.

We prove the two cases of the theorem separately.

1. *n is even*

   In the case where n is even, we call a state s *even* if the number of balls in each room is even, and *odd* otherwise. (Balls carried by the robot are counted towards the room in which the robot is located.)

   Exactly half of the S_n states are even, which can easily be seen by defining a bijection between the odd and the even states as follows: Let $<$ be an arbitrary total order on the balls. If the robot carries no or two balls in a state s, we map the state to the state which is identical to s except that the smallest (with respect to $<$) ball not carried by the robot is located in the other room. If the robot carries one ball in a state, we map this state to the identical state except that the robot (and the carried ball) is in the other room.

   a) *even states*

      All even states s apart from the two states where all balls are in the same room and the robot is in the other room are part of some optimal plan: we receive such a plan that leads through s from the procedure above, when preferring balls according to $B_s \prec R_s \prec A_s$ and picking up balls in R_s with the arm holding them in s. So all these states (except the reached goal state itself) are counted in $N^c(\mathcal{T}_n)$ (for all c).

      For the special state with all balls in room A and the robot in room B, the only applicable action is to move the robot to room A which results in the initial state, so the minimal goal distance is $3n$. Since the g value of this state is 1, its f-value with $(h^* - c)$ is $1 + \max(3n - c, 0)$, which is smaller than the optimal plan length $3n - 1$ iff $c > 2$.

      In the second special state all balls are in room B and the robot is in room A. This state can only be reached by moving the robot from room B to room A in the last step, and the predecessor state is the goal state. Hence, its g value exceeds $h^*(\mathcal{T}_n)$ and it will never be expanded by A*.

      Overall, all but 3 even states are counted in $N^1(\mathcal{T}_n)$ and $N^2(\mathcal{T}_n)$. For $c > 2$, $N^c(\mathcal{T}_n)$ additionally captures one of the two special states.

   b) *odd states*

      All odd states are parts of plans of length $h^*(\mathcal{T}_n) + 2$: For an odd state s, transport first one ball from A_s (or R_s if B_s is empty) to room B and move the robot back to room A. Afterwards follow the optimal procedure preferring balls according to $B_s \prec R_s \prec A_s$. The resulting plan has length $h^*(\mathcal{T}_n) + 2$ (two additional *move* operations).

However, the $2n$ odd states where all but one ball are in room B and the robot holds one ball in room A are still ruled out from $N^c(\mathcal{T}_n)$ because their g value is $h^*(\mathcal{T}_n)$: Moving two balls from room A to room B and going back to room A requires 6 steps (*pickup*, *pickup*, *move*, *drop*, *drop*, *move*), doing the same for one ball requires 4 steps (one *pickup* and one *drop* action less), so overall reaching these states requires $6(\frac{1}{2}n - 1) + 4 + 1 = h^*(\mathcal{T}_n)$ steps (the last one for picking up the last ball in room A). These are the only odd states with a g value larger than $h^*(\mathcal{T}_n)$ (all other odd states are reached within a smaller number of steps in the plan of length $h^*(\mathcal{T}_n) + 2$ described above), so all other odd states are counted in $N^c(\mathcal{T}_n)$ for $c > 2$.

Odd states are never part of a plan of length smaller than $h^*(\mathcal{T}_n)+2$: Since for an odd state it is necessary to move with a single ball to room B and to move the robot back afterwards in order to reach the goal, a plan traversing an odd state must always contain two additional *move* actions. So no odd state is counted in $N^1(\mathcal{T}_n)$ and $N^2(\mathcal{T}_n)$.

Summing up, $N^1(\mathcal{T}_n) = N^2(\mathcal{T}_n) = \frac{1}{2}S_n - 3$ (only even states) and for $c > 2$, $N^c(\mathcal{T}_n) = (\frac{1}{2}S_n - 2) + (\frac{1}{2}S_n - 2n) = S_n - 2n - 2$.

2. *n is odd*

   In the case where n is odd, all states except for the two states where all balls are in the same room and the robot is in the other room are part of some optimal plan. The special state with all balls in room A will be expanded by A* with $(h^* - c)$ iff $c > 2$, the other one will never be expanded because its g value exceeds the optimal plan length. The full proof for this case is analogous to the one of case 1a). Since the reached goal state (all balls and the robot in room B) is not counted by N^c, this results in $N^1(\mathcal{T}_n) = N^2(\mathcal{T}_n) = S_n - 3$ and $N^c(\mathcal{T}_n) = S_n - 2$ for all $c \geq 3$.

□

The theorem shows that there is little hope of achieving significant pruning for GRIPPER by using heuristic estimators, even in the easier case where the number of balls is even. With a heuristic error of 1, about half of all reachable states need to be considered already, because they all lie on optimal paths to the goal. Moreover, once the heuristic error is greater than 2, the A* algorithm has essentially no pruning power and offers no advantage over breadth-first search with duplicate elimination.

Miconic-Simple-Adl

In the MICONIC domain family, there is an elevator moving between the floors of a building. There are passengers waiting at some of the floors; the goal is to transport each passenger to their destination floor. In the MICONIC-SIMPLE-ADL domain variant, there are *movement* actions, which move the elevator from one floor to any other floor in a single step, and *stop* actions, which drop off all boarded passengers whose destination is the current floor and cause all passengers waiting to be served at that floor to enter.

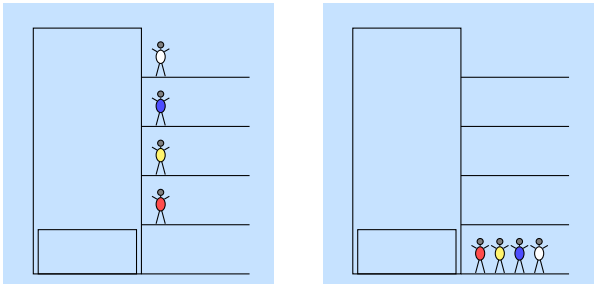


Figure 9.2: Initial state and goal of MICONIC task $\mathcal{T}_4$.

We consider the following family of MICONIC-SIMPLE-ADL tasks $\mathcal{T}_n$: There are $(n+1)$ floors and n passengers. The elevator is initially located at the bottom floor, which is also the destination of all passengers. There is one passenger waiting to be served at each floor except the bottom one (Fig. 9.2).

An optimal plan for $\mathcal{T}_n$ clearly needs $h^*(\mathcal{T}_n) = 2(n+1)$ steps: Move to each floor where a passenger is waiting and stop there. Once all passengers are boarded, move to the destination floor and stop again to drop off all passengers.

Altogether there are $S_n := 3^n(n+1)$ states in the search space (each passenger can be waiting, boarded, or served; the elevator can be at $(n+1)$ different locations). Some of these states are never expanded by A* with any heuristic (not even by $h = 0$) before the final search layer, because they cannot be reached within less than $h^*(\mathcal{T}_n)$ steps. However, the fraction of such states is fairly small, and *all other reachable states must be expanded by A* if the heuristic error is at least* 4.

Theorem 21. *For all $c \geq 4$, $N^c(\mathcal{T}_n) = S_n - (2^n - 1)(n+1)$.*

Proof. As noted above, $h^*(\mathcal{T}_n) = 2(n+1)$, which means that nodes with $g(s) \geq 2(n+1)$ cannot possibly be expanded before the final f layer of the A* search (which our definition of N^c optimistically excludes from consideration). These nodes can be exactly characterized: a state s requires at least $2(n+1)$ actions to reach iff no passenger is waiting and at least one passenger is served in s. (Under these conditions, the elevator must have moved to and stopped at each of the $(n+1)$ locations at least once, which requires $2(n+1)$ actions.) There are exactly $(2^n-1)(n+1)$ states of this kind, where the first term characterizes the $2^n - 1$ possible nonempty sets of served passengers, and the second term characterizes the possible elevator locations.

Now let s be an arbitrary state which is *not* of this form and where the elevator is at floor l. Consider the following plan: first collect all passengers which are served in s and drop them off at the bottom floor (if there are any such passengers), then collect all passengers boarded in s, then move to l if not already there. (At this point, we are in state s.) Finally, collect the remaining passengers and drop them off at the bottom floor. The plan visits s after less than $2(n+1)$ steps, and has length at most $h^*(\mathcal{T}_n) + 3$. Together, these facts imply that s will be counted towards $N^c(\mathcal{T}_n)$ for $c \geq 4$. □

We remark that the fraction of states *not* expanded by $h^* - 4$ according to our optimistic assumptions is almost exactly $(\frac{2}{3})^n$, which quickly tends towards

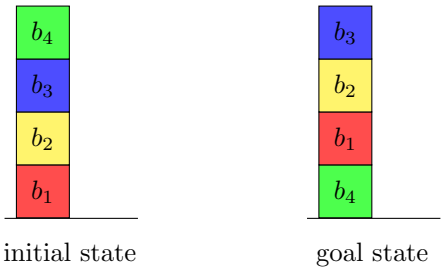


Figure 9.3: Initial and goal state of BLOCKSWORLD task $\mathcal{T}_4$.

zero as n increases. Thus, for larger values of n, heuristics with an error of 4 have no significant advantage over blind search. Since deciding whether there is a plan of a given maximal length for a given MICONIC-SIMPLE-ADL task is NP-complete (Helmert, 2001), computing the perfect heuristic h^* for this domain is NP-hard.

We also remark that a similar construction is possible with tasks where all origin and destination floors are disjoint.

Blocksworld

In the BLOCKSWORLD domain blocks stacked into towers must be rearranged by a robotic arm which can pick up the top block of a tower and place it on another tower or on the table.

One easy way of defining a family of BLOCKSWORLD tasks for which non-perfect heuristics must perform exponential amounts of search is to create a linear number of "independent" subproblems. (This is a general idea that works for a large number of benchmark domains, including many of those not discussed here.) One example is an initial state with n stacks of two blocks each, each of which needs to be reversed in the goal. Since the independent subplans can be interleaved arbitrarily, there are exponentially many states that are part of optimal plans, which implies that $N^1(\mathcal{T}_n)$ grows exponentially.

However, we can prove a similar result even in the case where all blocks are stacked into a single tower in the initial state and goal. Consider the class of BLOCKSWORLD instances $\mathcal{T}_n$ ($n \geq 2$) where one block shall be moved from the top of a tower with n blocks to the base of the tower (Fig. 9.3).

For these tasks, a heuristic planner needs to expand an exponential number of states – even if the heuristic only has an error of $c = 1$. The number $N^1(\mathcal{T}_n)$ of states that need to be expanded depends on the Bell numbers B_k (defined as the number of partitions of a set with k elements), which grow asymptotically faster than c^k for an arbitrary constant c (de Bruijn, 1958; Berend and Tassa, 2010). More precisely, we get the following result:

Theorem 22. $N^1(\mathcal{T}_n) = 4 \cdot \sum_{k=0}^{n-3} B_k + 3B_{n-2} + 1$

Proof. It is easy to see that $N^1(\mathcal{T}_n)$ is equal to the number of states that are part of some optimal plan, minus one. (Goal states themselves should not be counted, and in this case, all optimal plans obviously end in the same goal state.) We now compute this number by considering the states that are part of

some optimal plan. An optimal plan works in two phases: in the first phase it decomposes the source tower, and in the second phase it builds the goal tower.

First phase At the beginning of each optimal plan, b_n is picked up. Together with the initial state, this amounts to two states we need to count.

$\Rightarrow$ 2 states.

Now consider the states where $k \in \{1, \ldots, n-1\}$ blocks have already been moved. (Note: a block held by the robot arm does not count as having been moved.) There are $n-k$ blocks still on the source tower or held by the robot arm and block b_n lies on the table in a stack of its own. The other $k-1$ blocks are allocated into arbitrary stacks, but the order within each stack is fixed (blocks b_i lie above blocks b_j for $i < j$). Thus, the number of possible arrangements is equal to the number of partitions of a set with $k-1$ elements. This is given by the Bell number B_{k-1}. The next block to be moved, b_{n-k}, can either still be part of the source tower or be held by the robot arm, so there are two relevant states for each arrangement.

$\Rightarrow \sum_{k=1}^{n-1} B_{k-1} \cdot 2 = 2 \sum_{k=0}^{n-2} B_k$ states.

Second phase At the end of the first phase, b_1 is held by the robot arm. In the next action, b_1 must be stacked on top of b_n. There are B_{n-2} possible states after this action, corresponding to the possible arrangements of blocks $b_2, \ldots, b_{n-1}$.

$\Rightarrow B_{n-2}$ states.

After b_1 has been stacked on top of b_n, the remaining blocks need to be picked up and added to the goal tower, in sequence. Consider states where $k \in \{1, \ldots, n-3\}$ blocks (namely, blocks $b_{n-k}, \ldots, b_{n-1}$) are not yet in goal position and not held by the robot arm. As discussed for the first phase, there are B_k possible arrangements for these blocks. Moreover, there are two possibilities for the status of the robot arm; it is either free, or it holds block b_{n-k-1}.

$\Rightarrow 2 \sum_{k=1}^{n-3} B_k = 2 \sum_{k=0}^{n-3} B_k - 2$ states.

There are only two more possible states in an optimal plan: The goal state itself, and the state just before the goal state where b_{n-1} is held by the robot arm. As discussed above, we do not count the goal state.

$\Rightarrow$ 1 state.

Summary By adding up the individual counts, we obtain a total count of

$$2 + 2 \sum_{k=0}^{n-2} B_k + B_{n-2} + 2 \sum_{k=0}^{n-3} B_k - 2 + 1 = 4 \sum_{k=0}^{n-3} B_k + 3B_{n-2} + 1.$$

□

To get an impression of the rate of growth of $N^1(\mathcal{T}_n)$, values for $n = 2, \ldots, 15$ are shown in Fig. 9.4.

n	$N^1(\mathcal{T}_n)$	n	$N^1(\mathcal{T}_n)$
2	4	9	3748
3	8	10	17045
4	15	11	84626
5	32	12	453698
6	82	13	2605383
7	253	14	15924744
8	914	15	103071652

Figure 9.4: Lower bound of the number of expanded states in a BLOCKSWORLD task $\mathcal{T}_n$ with heuristic error $c = 1$.

9.4 Empirical Results

We have seen that the h^*-c family of heuristics has surprisingly bad theoretical properties in a number of common planning domains. This result immediately prompts the question: can we observe this behavior in practice?

To answer this question, we have devised an algorithm to compute $N^c(\mathcal{T})$ values and applied it to planning tasks from the IPC benchmark suite. One obvious problem in computing these values is that they are defined in terms of the perfect heuristic estimate of a state $h^*(s)$, which usually cannot be determined efficiently.

One possible way of computing $N^c(\mathcal{T})$ is to completely explore the state space of $\mathcal{T}$, then search backwards from the goal states to determine the $h^*(s)$ values. However, generating *all* states is not actually necessary. Recall that we are interested in $N^c(\mathcal{T})$, which is the number of states s with $g(s)+(h^*-c)(s) < h^*(\mathcal{T})$. Obviously, all these states are reachable within less than $h^*(\mathcal{T})$ steps from the initial state. Furthermore, they must have a descendant which is a goal state and has a depth of at most $h^*(\mathcal{T})+c-1$ (Fig. 9.5). Thus, for determining $h^*(s)$ for the relevant nodes s, it is sufficient to know all goal states until this depth. For this purpose, we employ a consistent and admissible heuristic to expand all nodes up to and including the f layer $h^*(\mathcal{T}) + c - 1$.

In summary, we use the following algorithm:

1. Perform an A* search with an arbitrary consistent, admissible heuristic until a goal state is found at depth $h^*(\mathcal{T})$.

2. Continue the search until all states in layer $h^*(\mathcal{T}) + c - 1$ have been expanded.

3. Determine the optimal goal distance $h^*(s)$ within the explored part of the search space for all expanded states s by searching backwards from the detected goal states.

4. Count the states with $g(s) + (h^* - c)(s) < h^*(\mathcal{T})$.

The outcome of the experiment is shown in Fig. 9.6. For all IPC tasks from the domains considered, we attempted to measure $N^c(\mathcal{T})$ for $c \in \{1, 2, 3, 4, 5\}$ using the technique outlined above. Instances within a domain are ordered by scaling size; in cases where the IPC suite contains several tasks of the same

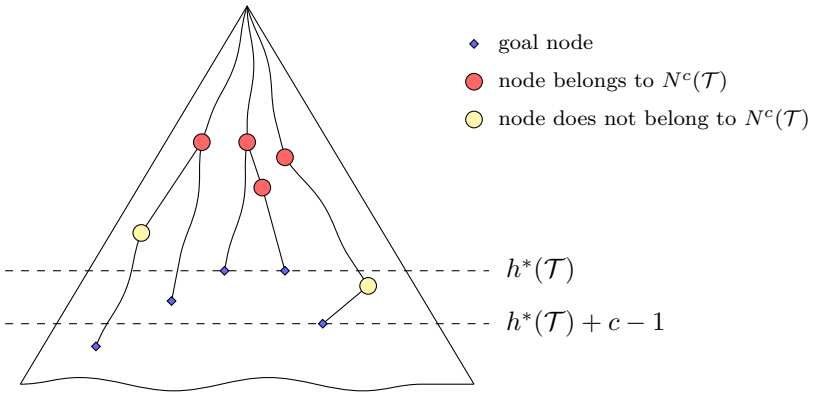


Figure 9.5: Schematic view of the search space

size, we only report data for the one that required the largest search effort. Results are shown up to the largest instance size for which we could reliably complete the computation for $c = 1$ within a 3 GB memory limit. (Blank entries for $c > 1$ correspond to cases where the memory limit was exceeded for these larger values.)

For the Gripper domain and both Miconic variants, the theoretical worst-case results are clearly confirmed by the outcome of the experiments. Even for $c = 1$, run-time of the algorithm scales exponentially with instance size. The data for Blocksworld and Logistics is less conclusive, but there appears to be a similar trend (for Blocksworld, this is most pronounced for $c = 5$).

Extrapolating from the expansion counts in the figure, it appears unlikely that a standard heuristic search approach can be used to solve Gripper tasks of size beyond 10–12, or reliably solve Miconic tasks of size beyond 16–18.

task	$h^*(\mathcal{T})$	$N^1(\mathcal{T})$	$N^2(\mathcal{T})$	$N^3(\mathcal{T})$	$N^4(\mathcal{T})$	$N^5(\mathcal{T})$
BLOCKSWORLD						
04-1	10	10	10	16	16	29
05-2	16	28	28	72	72	162
06-2	20	27	27	144	144	476
07-1	22	106	106	606	606	2244
08-1	20	66	66	503	503	2440
09-0	30	411	411	3961	3961	21135
GRIPPER						
01	11	125	125	246	246	246
02	17	925	925	1842	1842	1842
03	23	5885	5885	11758	11758	11758
04	29	34301	34301	68586	68586	68586
05	35	188413	188413	376806	376806	376806
06	41	991229	991229	1982434	1982434	1982434
07	47	5046269	5046269	10092510	10092510	10092510
LOGISTICS (IPC 2000)						
4-0	20	159	408	1126	1780	2936
5-0	27	459	2391	5693	14370	21124
6-0	25	411	2160	5712	14485	23967
7-1	44	17617	111756	427944	1173096	
8-1	44	4843	27396	157645	558869	
9-0	36	2778	15878	61507	183826	460737
10-0	45	10847				
11-0	48	10495				
MICONIC-SIMPLE-ADL						
1-0	4	4	4	4	4	4
2-1	6	6	22	26	26	26
3-1	10	58	102	102	102	102
4-2	14	148	280	470	560	560
5-1	15	209	759	1136	1326	1399
6-4	18	397	948	1936	2844	3436
7-4	23	3236	7654	11961	15780	16968
8-3	24	1292	5870	15188	25914	34315
9-3	28	20891	39348	39348	39348	39348
10-3	28	6476	16180	65477	129400	224495
11-3	32	58268	130658	258977	399850	497030
12-4	34	83694	181416	541517	970632	1640974
13-2	40	461691	947674	2203931	3443154	4546823
MICONIC-STRIPS						
1-0	4	4	4	4	4	4
2-1	7	18	29	34	37	37
3-1	11	70	138	195	241	251
4-4	15	166	507	814	1182	1348
5-4	18	341	1305	2708	4472	5933
6-4	21	509	2690	7086	13657	21177
7-4	25	3668	13918	32836	61852	95548
8-3	28	4532	35529	97529	205009	349491
9-3	32	25265	114840	321202	700640	1239599
10-3	34	8150	97043	423641	1151402	2505892

Figure 9.6: Empirical results for IPC benchmark tasks.

10 Combining Heuristic Estimators for Satisficing Planning

Heuristic forward search is also one of the most popular approaches in *satisficing* classical planning. However, the analysis in the previous chapter for *optimal* planning does not carry over to the satisficing case because the problems in the former case do not arise from finding a plan but merely from proving its optimality.

In the last decade, researchers have put a lot of effort into the development of new heuristics so that a wide range of heuristics are available these days. None of these heuristics consistently outperforms all others across all benchmark domains. Therefore, it appears worthwhile to use the information of several heuristics during search instead of only one.

In the case of optimal planning, which most commonly means using A* with an admissible heuristic, arbitrary admissible estimates can simply be combined by using their maximum. The resulting heuristic dominates all individual ones, which usually translates into a reduction of the state evaluations required to solve the task. Often, even better combinations are possible: using action-cost partitioning methods (Haslum et al., 2005; Katz and Domshlak, 2010), we can add heuristic estimates in an admissible way, dominating their maximum. The main drawback of these techniques is that efficiently finding good cost partitionings remains a widely open research problem despite significant recent progress (Katz and Domshlak, 2008).

In the case of satisficing planning, where greedy best-first search is the most common approach, the setting for combining heuristic values is quite different: the heuristics do not have to estimate the true distance to the goal in any quantitatively meaningful way, since greedy search only cares about their *relative* values: states further from the goal should receive larger estimates than states close to the goal. Since there is no need to respect a criterion like admissibility, we can combine estimates of several heuristics into a single numeric value in essentially arbitrary ways.

Combining several heuristic estimates in a satisficing planner can potentially lead to large performance and scalability improvements. Figure 10.1 shows a striking example of this. The graphs show the runtime, in seconds, for solving instances of the IPC 2000 Assembly domain using the FF heuristic h^{FF}

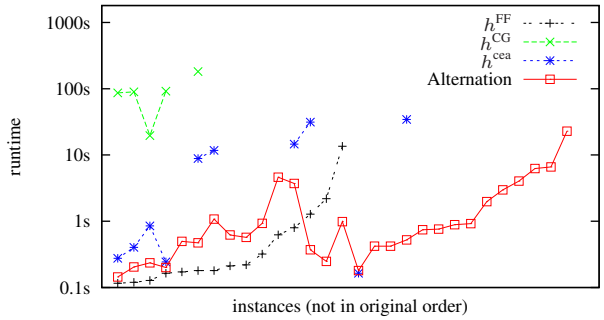


Figure 10.1: Runtimes in the Assembly domain. (Ordering of tasks does not correspond to the original benchmark suite.)

(Hoffmann and Nebel, 2001), the causal graph heuristic h^{CG} (Helmert, 2004), and the context-enhanced additive heuristic h^{cea} (Helmert and Geffner, 2008). None of the individual heuristics solves more than 15 instances. However, their combination (labeled "Alternation" in the figure) solves 29 out of 30 instances, including 13 instances not solved by any of the three heuristics it is based on.

The question, then, is *how* to combine the individual heuristic estimates to achieve the best possible performance. One obvious way to do so, by analogy to optimal planning, is to take their maximum or sum. However, for the Assembly example this does not turn out to be very useful: none of the heuristics that can be obtained by taking two or three of the candidate heuristics and computing their maximum or sum solves more than 13 of the 30 tasks within usual time and memory limits (30 minutes, 2 GB), so they are all outperformed by the FF heuristic used alone.

An alternative idea is to use *weighted* sums, but this immediately raises the question of how to determine suitable weights. In the given domain, we experimented with all 30 combinations of the form $h(s) = p \cdot h_1(s) + (1-p)h_2(s)$ where $p \in \{0, 0.1, 0.2, \ldots, 1.0\}$ and h_1 and h_2 are two heuristics from the given set. None of these combinations improves over the basic FF heuristic. It might be the case that better results could be obtained by using weighted sums of all three heuristics, but then the space of possible weights quickly explodes combinatorially.

So clearly, there are cases where maximization or summation is not the best way to combine heuristics for satisficing planning. Indeed, in Fig. 10.1, the *Alternation* method is vastly superior. This method is not new: it was introduced by Helmert (2006) under the name "multi-heuristic best-first search" (a term which we avoid in this thesis because it applies to any of the methods we discuss), and it is one of the ingredients underlying the Fast Downward (Helmert, 2006) and LAMA (Richter and Westphal, 2010) planners. However, neither Alternation nor any other method for combining heuristic estimates in satisficing planners has ever been evaluated in a principled way, and from the literature it is completely unclear *if*, *to what extent*, and *why* Alternation or any other method for combining heuristic values leads to better planner performance than just using a single heuristic.

In the following, we attempt to rectify this situation by giving detailed

Algorithm 2 Greedy best-first search (with duplicate detection)

```
1  open ← new open-list
2  open.insert(s_init)
3  closed ← ∅
4  while not open.empty() do
5      s ← open.remove-best()
6      if s ∉ closed then
7          closed ← closed ∪ {s}
8          if is-goal(s) then
9              return extract-solution(s)
10         foreach s' ∈ successors(s) do
11             if not is-dead-end(s') then
12                 open.insert(s')
13 return unsolvable
```

descriptions of several methods for combining heuristic estimates, providing a thorough experimental study on common planning benchmarks, and conducting targeted experiments in artificial search spaces to illustrate the benefits of using more than one heuristic for satisficing search.

One observation that motivates several of these methods is that maximization and summation are not very *robust* approaches: the overall heuristic can become wildly misleading as soon as a *single* estimator provides bad values. We will describe several methods that do not share this weakness and show that they convincingly outperform the non-robust approaches on typical planning benchmarks.

The rest of this chapter is structured as follows: after briefly introducing the greedy best-first search algorithm in the next section, we introduce several combination methods in the following four sections. We then shed some light on connections between these methods, followed by an extensive empirical evaluation and conclusion.

10.1 Greedy Search with Multiple Heuristics

All search methods presented in this chapter are variations of greedy best-first search (Pearl, 1984), differing only in the choice of which state to expand next. Greedy best-first search is a well-known algorithm, so we only present it briefly to introduce some terminology (Algorithm 2).

Starting from the initial state, the algorithm expands states until it has found a path to a goal state or until it has completely explored the state space. *Expanding* a state means generating its successors and adding them to the *open list*. The open list plays a very important role because its remove-best operation determines the order in which states are expanded. In single-heuristic search, it is usually simply a min-heap ordered by $s \mapsto h(s)$, where s is a search state and $h : s \to \mathbb{N}_0 \cup \{\infty\}$ estimates the length of the shortest path from s to any goal state. Hence, states with a low estimate are expanded first. If states share the same estimate, they are usually ordered according to the FIFO principle.

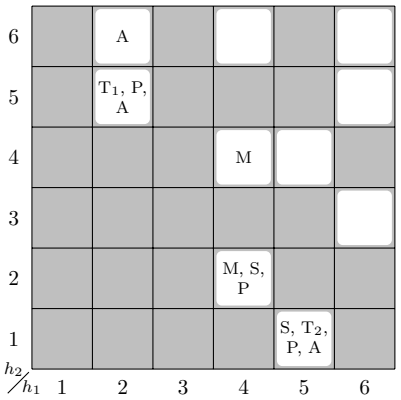


Figure 10.2: Buckets of an open list with heuristics h_1 and h_2. Each box represents a bucket that collects all entries with a certain combination of heuristic estimates, e. g., the top-left box would contain all entries s with an estimate of $h_1(s) = 1$ and $h_2(s) = 6$. White boxes indicate non-empty buckets, gray buckets are empty. The symbols within some of the buckets are explained later.

This chapter deals with the question of how to use the estimates of multiple heuristics $h_1, \ldots, h_n$ within this algorithm. In principle, the methods we present only differ in which states are selected by the *remove-best* operation.

We can see the open list as a collection of *buckets* (Fig. 10.2), each associated with an estimation vector $(e_1, \ldots, e_n)$ and containing all open states s with $(h_1(s), \ldots, h_n(s)) = (e_1, \ldots, e_n)$. (We assume that is-dead-end(s) evaluates to true iff any of the heuristic estimators regards s as a dead end by mapping it to ∞, so estimates e_i of states in the open list are always finite.) All combination approaches we present can be understood as first selecting a *bucket* to expand a state from, and then picking a state from this bucket according to the FIFO principle. Hence, an approach can be largely characterized in terms of its *candidate buckets*, i.e., the buckets that are possible candidates for expansion at each step.

For example, the candidate buckets for the *maximum* method are exactly those where $\max\{e_1, \ldots, e_n\}$ is minimized. In Fig. 10.2, this means that either the bucket with estimation vector $(4, 2)$ or the bucket with estimation vector $(4, 4)$ is chosen. Which of these buckets is actually selected again depends on FIFO tie-breaking: the bucket with the "oldest" state is given preference. Of course, an actual implementation of the method should not maintain separate buckets for each estimation vector, but rather use a one-dimensional vector of buckets indexed by $\max\{e_1, \ldots, e_n\}$.

10.2 Maximum and Sum

The first combination methods we discuss are the already mentioned *maximum* and *sum* approaches. The candidate buckets for the maximum approach are those which minimize $\max\{e_1, \ldots, e_n\}$, and the candidate buckets for the sum approach are those which minimize $e_1 + \cdots + e_n$. In the example of Fig. 10.2, these buckets are marked with an **M** for the maximum approach and **S** for the

sum approach. Among all states in these buckets, the oldest one is expanded first.

The maximum and sum methods are very easy to implement: since they reduce each estimation vector to a single numeric value, a standard single-heuristic open list can be used. However, we will later see in our experiments that maximum and sum are among the weakest methods for combining heuristic estimates and rarely offer a compelling advantage over using one of the component heuristics individually. One explanation for this is that they are easily misled by bad information. If one of the component heuristic provides very inaccurate values, then these inaccuracies affect every single search decision of the sum method, because each heuristic directly contributes to the final estimation. For the maximum method, *large* inaccurate values from one heuristic can completely cancel the information from all other heuristics.

Of course, one can try to balance a disproportionate influence of a single heuristic by applying weights to the different estimates, but it is not clear how reasonable weight values can be determined automatically, or if weighting can help overcome the fundamental problems of these methods at all. One approach we experimented with is to calculate weighted sums with weights determined from the estimates of the initial state, trying to "balance" the contribution of each heuristic. However, this approach did not show any positive effect on planning benchmarks. One possible explanation for this is that such a normalization not just levels the influence of bad estimates, but also of good estimates.

Because initial experiments were discouraging and it is not clear how to assign reasonable weights, our empirical evaluation does not include the case of weighted sums. However, we do report experiments with the unweighted sum and maximum methods, which serve as baselines for the other approaches, to be introduced next.

10.3 Tie-breaking

Our experience with the maximum and sum methods suggests that aggregating heuristic estimates into one value tends to dilute the quality and characteristics of the individual heuristics. Therefore, in the following we concentrate on methods that preserve the individual estimates. One obvious idea is to rank the heuristics by importance and use the less important ones only for breaking ties. With this approach, search is mainly directed by one good heuristic and only if there are several states with the same minimum estimate, the other heuristics are successively consulted to identify the most promising state. If two states have exactly the same estimation vector, they are again expanded according to the FIFO principle.

Tie-breaking always selects a single candidate bucket. In the example of Fig. 10.2, this bucket is labeled as $\mathbf{T}_1$ for the case where h_1 is the more important heuristic and h_2 is used to break ties, and it is labeled as $\mathbf{T}_2$ for the opposite case.

We considered two implementations of the tie-breaking method. One natural approach is to calculate only the main heuristic and to order the open list according to these estimates. Upon each remove-best operation, we check if several states share the same minimum estimate. Only then do we successively

calculate the tie-breaking heuristics, until we have identified a single state to expand. The advantage of this approach is that a heuristic estimate for a tie-breaking heuristic is never computed if it is never needed.

However, in typical planning tasks the range of encountered heuristic values is much smaller than the size of the search frontier, and there are usually many states with the same estimate of the main heuristic. Therefore, the disadvantage of this approach is that we must perform the same tie-breaking calculations again and again, which is costly even if heuristic values are cached after their first computation. While additional data structures may reduce the effort of these recomputations, this causes overhead, and it is not clear if it is worth the additional implementation complexity.

For this reason, we use a different implementation of tie-breaking: for each state inserted into the open list, we calculate the estimates of all heuristics and directly sort it to the appropriate position. With this approach, we can again implement the open list as a min-heap, ordering states lexicographically by their estimate vector. Our experimental data suggests that the cost of always computing all heuristics is not problematic at least in the cases we consider. (One important mitigating factor is that in our case, the main heuristic is more computationally intensive than the tie-breaking heuristics and hence tends to dominate overall runtime.)

Note that both implementations differ only in the time that is needed for inserting and removing states from the open list and in the space requirements for the open list data structure, but behave equivalently in all other aspects. In particular, there is no difference in the number of expanded states.

A major drawback of tie-breaking is that we have to define an importance ranking of the heuristics. For our experiments, we decided to order the heuristics according to their (empirical) quality in single-heuristic search. It is apparent that combining multiple heuristics via tie-breaking does not fully exploit the available information: we only use the additional estimates if the main heuristic does not distinguish two states. If it does, even if it performs very badly, we ignore the estimates of the additional heuristics. Hence, the approach is clearly not robust against bad estimates of the main heuristic.

Finally, we note that unlike the maximum and sum approaches, tie-breaking is unaffected by changing the "scale" of the component heuristics. Increasing estimates by an additive or multiplicative constant or applying any other strictly increasing transformation to a heuristic function does not affect the choices of the tie-breaking method. We see this as a strength rather than a weakness because it offers some resilience against systematic errors in heuristic estimates.

10.4 Selecting from the Pareto Set

We now present a method that, like tie-breaking, is robust to transformations of heuristic estimates, but does not require us to arbitrarily favour one heuristic over another. Such a method can be derived from the concept of Pareto optimality that is well-known in economics and game theory. Pareto optimality has been successfully applied in multi-objective search (Stewart and White, 1991; Tung and Chew, 1992), where the goal is finding a state that is good in terms of multiple objectives whose measures cannot be meaningfully compared.

In order to introduce this method, we need to define the notion of *dominance*. We say that a state s *dominates* a state s' if all heuristics consider s at least as promising as s' and there is at least one heuristic that strictly prefers s over s'.

Definition 29. *A state* s dominates *a state* s'*, written* $s < s'$*, with respect to heuristics* $h_1, \ldots, h_n$ *if* $h_i(s) \leq h_i(s')$ *for all* $i \in \{1, \ldots, n\}$ *and* $h_i(s) < h_i(s')$ *for at least one heuristic.*

It appears reasonable to require that if state s dominates s', then s should be expanded before s'. Hence, we are interested in the Pareto set of nondominated states, defined as

$$\text{nondom} \stackrel{\text{def}}{=} \{s \in \mathit{open} \mid \nexists s' \in \mathit{open} \text{ with } s' < s\}.$$

In the Pareto approach, the candidate buckets are exactly those buckets whose states belong to nondom. In the example in Fig. 10.2, these buckets are labeled with **P**. We see that the set includes many of the candidate buckets of the previous approaches, but not all of them. In particular, bucket $(4, 4)$ which is a candidate for the maximization approach is not Pareto-optimal because it is dominated by $(4, 2)$.

We experimented with two variants of the Pareto approach. Both variants first randomly select one of the candidate buckets and then expand the oldest state in that bucket. The two variants differ in how the random choice of buckets is performed: in the *uniform* approach, each candidate bucket is chosen with equal probability, while in the *weighted* approach each candidate bucket is chosen with probability proportional to the number of states it contains.

Note that all previous combination methods define a total preorder on the states. This is somewhat restricting because estimate vectors where neither dominates the other cannot always be reasonably compared. However, algorithmically it is very useful because it allows implementing the open list as a min-heap. This is not possible in the Pareto approach because the preorder is not total. For example, in a given situation the nondominated buckets might have associated estimate vectors of $(2, 4, 4)$ and $(4, 4, 2)$, so that the oldest states with these heuristic profiles, say s_1 and s_2, are candidates for expansion. Now assume that we insert a new state with heuristic profile $(2, 4, 3)$. This new states dominates s_1 but not s_2, so one of the previously "best" states remains a candidate for expansions, while another does not. Such effects complicate the open list implementation for the Pareto approach, and therefore this approach can carry a much larger search overhead than the others. Moreover, this overhead quickly increases with the number of heuristic estimators.

On the positive side, the Pareto method has none of the disadvantages of the previous approaches: we neither have to aggregate estimates in a non-robust way, nor do we have to fix a magic order of the heuristics. Instead, we use all available ordering information, and whenever we prefer a state over another one, we can theoretically justify this decision.

10.5 Alternation

The last approach we want to discuss is the *alternation* method. Like the Pareto method, it avoids aggregating the individual heuristic estimates and makes

equal use of all heuristics. The method gets its name because it alternates between heuristics across search iterations. The first time a state is expanded, it selects the oldest state minimizing h_1. On the next iteration, it selects the oldest state minimizing h_2, and so on, until all heuristics have been used. At this point, the process repeats from h_1. The candidate buckets for the alternation method are those whose estimate vectors minimize at least one component (labeled with **A** in Fig. 10.2).

As mentioned earlier, the alternation method was originally proposed by Helmert (2004, 2006) under the name *multi-heuristic best-first search*. It is built on the assumption that different heuristics might be useful in different parts of the search space, so each heuristic gets a fair chance to expand the state it considers most promising. One heuristic might provide good guidance in one part of the search space, but be weak in another. A second heuristic might have its strong and weak regions distributed differently in the search space. By alternating between the heuristics, it is always possible to escape a plateau as long as at least one heuristic can give good guidance. There are two important differences between alternation and the Pareto approach:

- Alternation only expands states that are considered *most promising* by some heuristic. The Pareto approach can also expand states which offer a good *trade-off* between the different heuristics, such as bucket $(4, 2)$ in Fig. 10.2.
- For states that *are* most promising to the currently used heuristic, the alternation method completely ignores all other heuristic estimates. The Pareto approach also attempts to optimize the other heuristics in such situations. For example, it would not consider bucket $(2, 6)$ in Fig. 10.2 because it is dominated by bucket $(2, 5)$.

Alternation can be efficiently implemented by maintaining a set of min-heaps, one ordered by each heuristic. The approach has been used by several successful planners, including Fast Downward (Helmert, 2006), using the causal graph and FF heuristics, and LAMA (Richter and Westphal, 2010), using the FF and landmark heuristics.

10.6 Combining Alternation and Tie-breaking

Before we move to the experimental evaluation, we observe that with the approaches we presented, the design space for heuristic combination methods is far from exhaustively covered. Indeed, one natural idea is to *combine* several of the methods we have introduced.

One particularly interesting case is the combination of the alternation and tie-breaking methods: one of the major drawbacks of tie-breaking is that we must define a ranking of the heuristics. We can try to escape this problem by alternating between all possible rankings. Combining alternation and tie-breaking in this fashion can be seen as a compromise between the pure alternation method and the Pareto approach: the combined approach only expands states deemed *most promising* by some heuristic, a property that it shares with alternation and that distinguishes it from the Pareto approach. However, like the Pareto method and unlike alternation, it does not base its decision on one heuristic alone, as states considered by tie-breaking are always Pareto-optimal.

By restricting itself to Pareto-optimal states, this combination method retains many of the characteristics of the Pareto approach. However, unlike that method, it can be implemented quite efficiently if the number of heuristics is not too large – for any fixed number of heuristics, the overhead compared to single-heuristic search is bounded by a constant factor.

10.7 Experiments

We now turn to the central questions of this chapter: is the use of multiple heuristics for satisficing best-first search actually beneficial? And if so, which combination method performs best? To answer these questions, we conducted two experiments. In the first experiment, we integrated the different combination methods into a state-of-the-art planning system, to investigate their effect on typical planning benchmarks. In the second experiment, we studied the behavior of the different methods on artificial search spaces, to get a cleanroom perspective of how factors like heuristic quality impact their relative performance.

We conducted all experiments on computers with 2.3 GHz AMD Opteron CPUs, setting a timeout of 30 minutes and a memory limit of 2 GB.

Experiment on IPC Benchmarks

In our first experiment, the benchmark suite consists of all planning tasks from the first five international planning competitions (IPC 1998–2006). We report results on coverage (number of solved instances), solution quality, speed, and heuristic guidance (number of state expansions). We consider three different heuristic estimators:

- h^{FF}: the FF heuristic (Hoffmann and Nebel, 2001),
- h^{CG}: the causal graph heuristic (Helmert, 2006), and
- h^{cea}: the context-enhanced additive heuristic (Helmert and Geffner, 2008).

We evaluate each approach on all two- and three-element subsets of these heuristics. For the tie-breaking approach we fixed the ranking of the heuristics as $h^{\mathrm{cea}} \succ h^{\mathrm{FF}} \succ h^{\mathrm{CG}}$ (so h^{cea} is given the highest priority) based on the coverage these heuristics achieve on the benchmark set in single-heuristic search. For the Pareto method we only report results for the *weighted* approach, because it performs slightly better than the uniform approach and the difference between these variants is low compared to the difference to other methods.

Our implementation is based on the Fast Downward planning system (Helmert, 2006), which we extended with implementations of the different combination approaches. As we are interested in measuring the impact of heuristic combinations, not other search enhancements, we did not use the preferred operator information provided by the heuristics. We have run experiments both with Fast Downward's deferred variant of greedy best-first search and with the textbook ("eager") algorithm (Richter and Helmert, 2009), with virtually identical results. Here, we report on the more standard eager algorithm. Results for lazy search are reported in a workshop paper (Röger and Helmert, 2009).

We first present the overall results, shown in Table 10.1. The table reports scores according to four metrics: coverage, (solution) quality, speed, and (heuristic) guidance. All scores are in the range 0–100, where larger values indicate better performance. For each metric, the score is computed by assigning a value between 0 and 100 to each task, then averaging the scores for the tasks of each domain to compute a domain score, and finally averaging the domain scores to compute an overall score. Unsolved tasks are always scored as 0, while the score for solved tasks depends on the metric:

- **Coverage:** Solved tasks receive a score of 100. This metric corresponds to the probability (in percent) that the approach solves a "typical" benchmark task.

- **Quality:** Solved tasks receive a score of $100 \cdot c^*/c$, where c is the cost of the generated solution and c^* is the cost of the best solution generated by any of the approaches.

- **Speed:** Tasks solved within one second receive a score of 100, and tasks that require the full 1800 seconds receive a score of 0. Between these extremes, scores are interpolated logarithmically, so that doubling the runtime decreases the score by about 9.25.

- **Guidance:** Tasks solved within 100 state expansions receive a score of 100, and tasks solved with more than 1,000,000 expansions receive a score of 0. Between these extremes, scores are interpolated logarithmically, so that doubling expansions decreases the score by about 7.53.

We now turn to the interpretation of the results of Table 10.1.

Comparison between combination approaches. There is a clear classification of the different combination methods into three groups.

Alternation generally performs best: it gives the best results in terms of coverage and quality on all four heuristic sets, and is best in terms of speed and guidance in all cases except for one where its combination with tie-breaking and the Pareto approach are slightly better.

Alternation combined with tie-breaking and the Pareto method perform similarly to each other and always outperform the remaining approaches in terms of speed and guidance. In terms of coverage and quality, the maximum and sum approaches sometimes obtain comparable results.

The remaining three techniques, maximum, sum and tie-breaking, perform quite similarly to each other and are clearly worst overall. In terms of coverage, speed and guidance, the sum method appears to slightly outperform the other two approaches; for quality, sum and maximum are too close to each other to pick a winner. The tie-breaking method appears to be weakest overall. In particular, it is almost always the worst method in terms of coverage (except for the combination of all three heuristics where the maximum method performs worse).

Comparison to single-heuristic methods. Another clear outcome of the experiment is that using multiple heuristics can give considerable benefits, especially with the alternation method. For any set of heuristics and any of the four metrics, the alternation method improves the performance over the

	Coverage	**Quality**	**Speed**	**Guidance**
h^{cea}	74.62	68.67	65.27	65.65
h^{FF}	73.85	70.55	66.81	64.07
h^{CG}	72.66	65.36	64.16	60.43
h^{cea}, h^{FF}				
Maximum	72.69	67.26	62.15	64.02
Sum	73.75	68.42	63.75	*65.67
Tie-breaking	72.44	67.14	62.90	64.67
Pareto	*76.20	*70.71	66.32	*68.90
Alternation	***77.95**	***73.70**	***67.84**	***70.14**
Alternation-TB	*75.42	70.21	66.23	*68.48
h^{FF}, h^{CG}				
Maximum	*74.76	68.76	65.29	*65.08
Sum	*75.01	67.99	65.41	*65.35
Tie-breaking	72.59	66.13	64.66	*64.41
Pareto	*74.93	67.84	65.87	*66.19
Alternation	***78.73**	***73.28**	***69.22**	***69.28**
Alternation-TB	*74.75	67.45	66.06	*66.18
h^{cea}, h^{CG}				
Maximum	74.06	67.95	63.63	65.51
Sum	*74.76	67.70	64.12	*65.67
Tie-breaking	73.78	67.41	63.36	64.99
Pareto	74.52	67.70	64.48	*66.52
Alternation	***75.20**	***69.18**	64.42	*66.39
Alternation-TB	74.58	67.79	**64.59**	***66.59**
h^{cea}, h^{FF}, h^{CG}				
Maximum	72.21	66.54	61.13	63.71
Sum	73.47	67.52	62.98	65.24
Tie-breaking	72.49	66.95	61.90	64.34
Pareto	*76.29	70.16	66.01	*69.18
Alternation	***79.80**	***74.62**	***68.56**	***71.91**
Alternation-TB	*76.05	70.15	65.83	*69.16

Table 10.1: Overall result summary. The best combination method for a given set of heuristics and metric is highlighted in bold. Entries marked with a star indicate results that are better than all respective single-heuristic approaches.

best single heuristic from the set, with only one small exception (speed for the combination of h^{cea} and h^{CG}).

Indeed, adding more heuristics is almost universally a good idea for the alternation method in our experiment. There are nine ways to choose a single heuristic or two-heuristic set and a new heuristic to add, and there are four metrics to measure. In 34 of these 36 cases, the *marginal contribution* of adding the new heuristic is positive.

For the Pareto method and the combination of alternation and tie-breaking, the comparison to single-heuristic search gives somewhat mixed results. While

both approaches lead to better results in terms of coverage (except for the combination of h^{cea} and h^{CG} where they perform slightly worse) and guidance, their results in terms of quality and speed are worse than those of the best individual heuristics.

For the maximum and sum methods, it is hard to argue that they offer any compelling advantage over single-heuristic search, and the tie-breaking method is clearly not worth using in this setting. It consistently performs worse on all metrics than just using the main heuristic on its own, with only one exception.

Domain	h^{cea}	h^{FF}	h^{CG}	**Max.**	**Sum**	**Tie-br.**	**Pareto**
Airport	–3	+7	+18	+2/–1	+3/–2	+7	+6
Assembly	+20	+15	+24	+20	+20	+20	+14
Depot	+2	–1	+3/–1	+2	+3/–1		–2
Driverlog	+1	+1	+1	+2	+1	+2	+2
FreeCell	+1/–1	+3/–2	+10/–1	+4/–1	+3/–1	+5	–1
Grid	+1	+1	+1	+1		+1	
Logistics-1998	–4	+4	–4			–1	
Miconic-FullADL	–1	+4/–1	+2	–1	+1/–1	–1	+1
MPrime		+8	–1	+6			–1
Mystery	+1	+3/–1		+1			–1
Openstacks	+5		+4	+5	+5	+5	
OpticalTelegraphs			+3				+2
Pathways	+5	+7	+4	+5	+6	+5	+5
Pipesw.-NoTank.	+13	+7	+15/–1	+14	+12	+12	+9/–1
Pipesw.-Tankage	+4/–1	+4/–3	+7/–3	+4	+4/–1	+5/–2	+2/–2
PSR-Large	–2	+1/–1	–2	+1	+3	+3	+2
PSR-Middle					+1	+1	+1
Rovers	+7	+5	+7	+7	+8	+8	+7
Satellite	–3	+1	–9				
Schedule	+9	+3/–12	+9	+9	+9	+9	+9
Storage	+2	–1	+1	+2		+2	
TPP	+3/–4	+3	+1	+3	+3	+5	+2/–4
Trucks		+2	+4/–1		+2		+1/–1
Total	+74/–19	+79/–22	+114/–23	+88/–3	+84/–6	+90/–4	+63/–13

Table 10.2: Tasks solved by Alternation compared to single heuristics and other combination approaches. Entry $+x/-y$ means that Alternation solves x tasks not solved by the other approach and fails to solve y tasks solved by the other approach. Domains where all methods solve the same set are omitted. All combination methods use all three heuristics.

Coverage details. We have established that we obtain the best results when using the alternation method applied to all three heuristics. Hence, we conclude our discussion of the planning experiment with some detailed data for this particular approach, in order to see whether its benefits are limited to a few benchmark domains or distributed more evenly.

Firstly, we remark that using the same nonparametric test that Hoffmann and Nebel (2001) employ in their comparison of FF and HSP, the improvement of coverage of the alternation method compared to any of the other combination methods or individual heuristics is statistically significant at a level of $p \leq 0.001$. (The same is true for the use of alternation with two heuristics, except for the combination of h^{cea} and h^{CG}, where the significance is lower.)

Secondly, to provide some more detail Table 10.2 reports, for all IPC 1998–2006 benchmark domains, in which ways the set of tasks solved by the alterna-

tion method differs from other approaches. We compare to all single heuristics and to all pure combination methods that use the same (full) set of heuristics. We omit the comparison to the combination of alternation and tie-breaking, for which the results are very similar to the Pareto approach. The table shows that the improvements are spread over many domains. Moreover, there are very few cases where the alternation method fails to solve a substantial number of tasks solved by one of the single heuristics, indicating that it is indeed very robust.

There are only five domains in which *any* of the single heuristics outperforms the alternation technique by more than one instance, and all of these are (perhaps not coincidentally) among the IPC domains with the largest instances. There are only three domains where the approach performs worse than the average of the three heuristics it combines, *Logistics-1998*, *PSR-Large* and *Satellite*. These are domains where heuristic guidance is generally near-perfect, but raw search speed matters a lot due to the size of the tasks. On the largest Satellite instances, even a perfect heuristic must evaluate several hundred thousand states because optimal plan length is in the range of 300–500 steps and the branching factor exceeds 1000.

Controlled Experiments

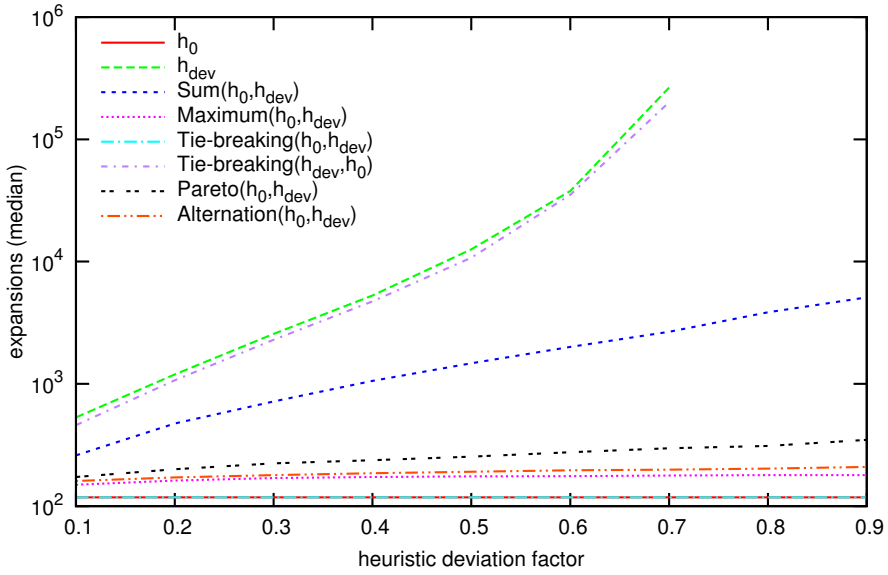
In the second set of experiments, we investigate the behavior of the combination approaches in a manually designed search space. The aim of the experiments is to study some aspects of the algorithms in a controlled way. In particular, we are interested in how heuristic quality affects the performance of the algorithms and how the algorithms behave on instances of scaling size. We use a tree-shaped infinite search space with uniform branching, following the controlled experiments in the evaluation of preferred operators and deferred evaluation by Richter and Helmert (2009)[1].

Every state is characterized by a single value, its *approximate goal distance* (*agd*), which defines the typical distance to the goal. States with an *agd* of 0 are goal states. In the first set of experiments, all initial states have an approximate goal distance of 75; in the second set, we vary the *agd* of initial states in the range 50–500. All states with *agd* $n > 0$ have 15 successors, whose *agd* is chosen independently at random in such a way that on average, every state has one successor closer to the goal (*agd* $n-1$), ten successors at the same distance to the goal (*agd* n), and four successors further away from the goal (*agd* $n+1$).

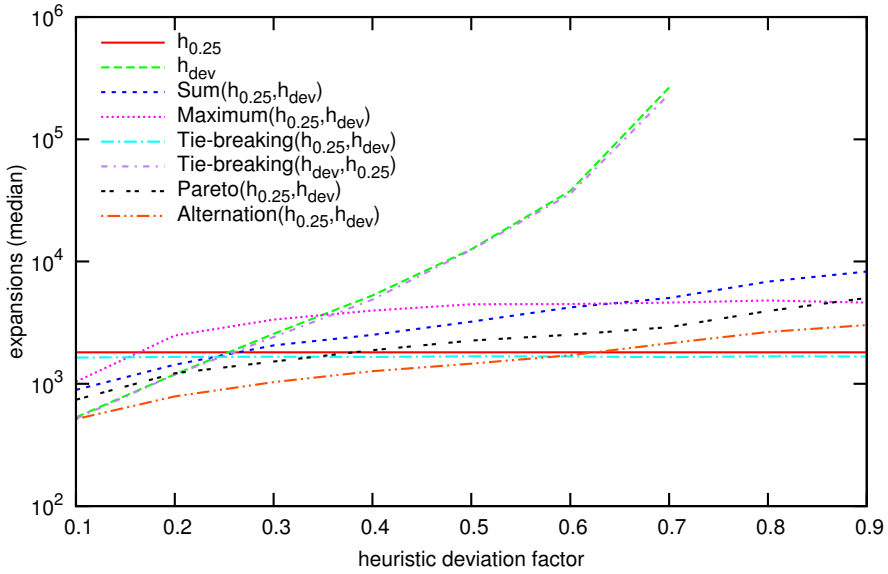
Preliminary tests showed that greedy best-first search performs very poorly on the artificial problems (and indeed, this algorithm is not complete for infinite search spaces of this kind). Therefore, all experiments on artificial search spaces used the weighted A* algorithm with a weight of 10 for the heuristics, which is still quite greedy, but complete.

To control the quality of the heuristic, we use a family of heuristics h_{dev} that deviate by a factor $0 \leq dev < 1$ from the approximate goal distance. More precisely, the estimate for a state with *agd* n is chosen uniformly from the range $[n(1-dev), n(1+dev)]$, rounding down to a natural number.

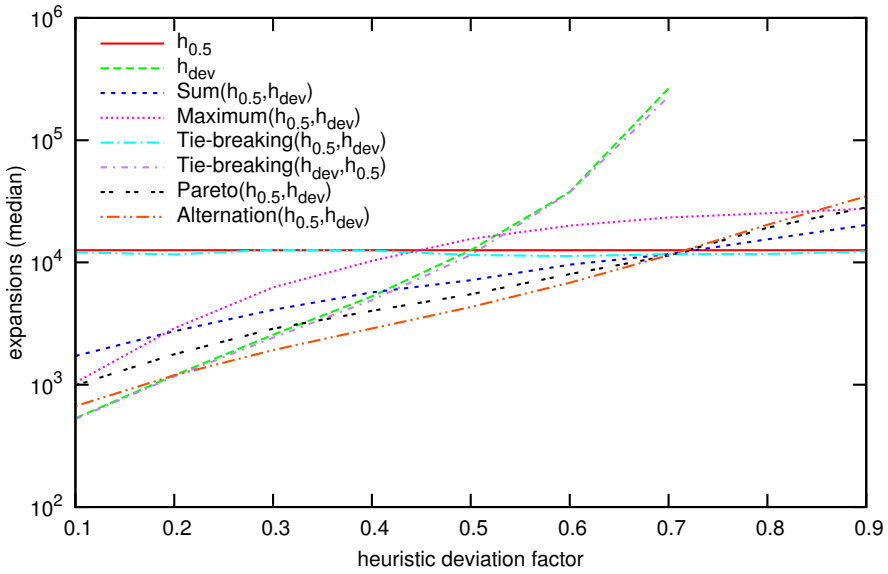
[1] We thank Silvia Richter for making the code for the controlled experiments available to us.



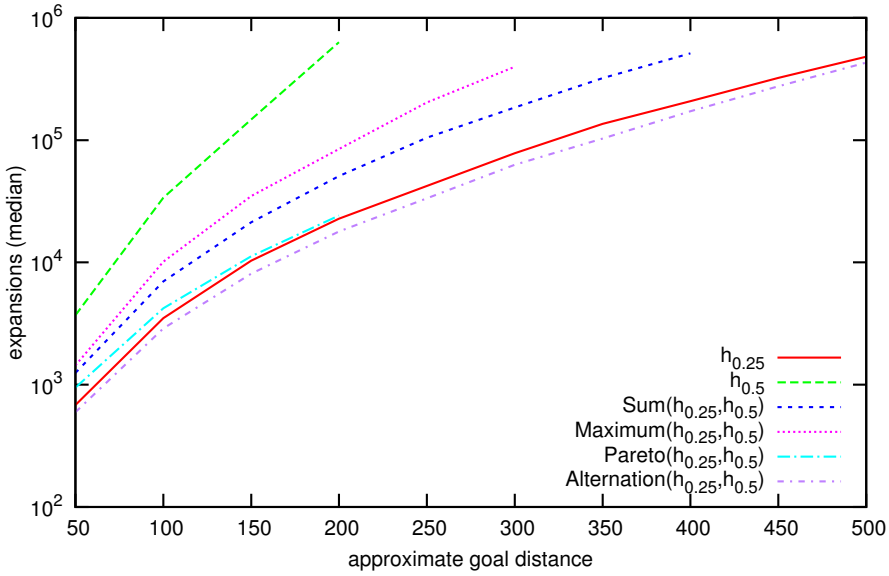
(a) near-perfect fixed heuristic (deviation 0)



(b) mediocre fixed heuristic (deviation 0.25)



(c) bad fixed heuristic (deviation 0.5)



(d) instances of scaling size

Figure 10.3: Experiments in an artificial search space. In panels (a)–(c), the quality of one heuristic is fixed while the quality of the second heuristic varies. Panel (d) shows how the approaches scale with the size of the search space.

In addition to the runtime and memory limits, we aborted all runs that generated more than 10^7 states.

Influence of heuristic quality. In the first experiment we examine the impact of heuristic quality on the performance of the different combination approaches. Figures 10.3(a)–(c) show the results for combinations of two heuristics, where we fix the deviation of one heuristic and vary the deviation of the other heuristic in the range 0.1–0.9. The graphs report the median number of expansions based on 100 runs. (Other order statistics, such as the 25th or 75th percentiles, produce very similar graphs.)

The alternation method provides the best guidance in wide parts of the realistic settings where every involved heuristic has some deviation from the real goal distance. As long as at least one heuristic is reasonably good, the approach provides a clear advantage over single-heuristic search, as its graph runs below both graphs of the involved heuristics. The only exception to this is when one of the heuristics is *really* good, but even then the alternation method demonstrates its robustness against bad estimates of the second heuristic.

The Pareto method shows similarly good robustness properties, but its guidance is slightly worse than for alternation. Nevertheless, it still can have some advantage over single-heuristic search. However, since we only measure the number of expansions here, the graphs do not take into account the relatively high per-state overhead of the approach.

Tie-breaking leads to almost identical results to the respective main heuristic. One reason for this is that in the experiment setting the estimates deviate symmetrically from the approximate goal distance. But even if we use the real approximate goal distance for tie-breaking (Fig. 10.3(a): *tie-breaking*(h_{dev}, h_0)), we can observe only a very low positive impact on the number of expansions.

The sum method can easily be misled by bad estimates of one heuristic, even if the other heuristic provides almost accurate estimates. If both heuristics have a similar quality, the sum method has some advantage in this experimental setting: for each state, the two heuristics select randomly from the same range around the (approximately) perfect estimate, so errors tend to cancel out. The maximum method tends to do well when one heuristic is near-perfect, but is among the worst methods in the more challenging settings (Fig. 10.3(b,c)).

Scaling behavior. Figure 10.3(d) explores the scaling behavior of the different approaches. We use two heuristics with deviation factors 0.25 and 0.5 and vary the approximate goal distance of the initial state between 50 and 500. The graph again shows the median number of expansions based on 100 runs. To keep it legible, we omit the values for the tie-breaking approaches, which are again almost identical to those of the respective main heuristics.

Alternation emerges as the clear winner of the comparison. Not only does it solve almost all instances (for *agd* 500 it solves 97 of the 100 instances, and for lower values it solves all of them), it also requires the lowest number of expansions. It also offers consistent improvements over the results of the better heuristic $h_{0.25}$, unlike the other combination approaches.

The Pareto approach performs quite competitively in terms of expansions, but times out on the harder instances: due to the wide spread of heuristic values on these tasks and the weak correlation of the two component heuristics, the number of estimate buckets to keep track of is very large, and the overhead for maintaining the set of nondominated buckets grows with the square of the *agd*.

The sum and maximum methods provide much worse guidance and exceed the node limit on the harder instances.

11 Conclusion

In the first part of this thesis, we focused on the integration of Golog and planning.

We make planning systems directly available to the Golog interpreter as a kind of subsolver, building on previous work by Eyerich et al. (2006) who formulated restrictions on the situation calculus that directly mimic the characteristics of the planning language PDDL. However, it was entirely unclear which of these restrictions are really necessary and which could safely be dropped. Our contribution on the theoretical side is that we clarified exactly this question.

Indeed, not all original restrictions are required to preserve the same expressive power as PDDL and many results were not obvious at all, like for example that the domain closure axiom can be dropped. However, we also gained valuable insights from non-compilability results, revealing interesting connections between features that are beyond the expressivity of PDDL.

The advantage of the positive results is that they directly define an algorithm that allows to translate from basic action theories to PDDL and that there is a trivial correspondence between the resulting plans and the action sequences required by the Golog interpreter. Therefore, it is easily possible to bring the theoretical results to a practical application.

Our empirical evaluation of the approach demonstrates how impressively Golog systems can benefit from the integration of a planning system. Indeed, it was almost impossible to find tasks that were solvable by the pure Golog system but not entirely trivial for the system with an integrated planner.

The integration is designed in a way that allows to plug in arbitrary planning systems that support the standard input language PDDL. This way, we can always use the planning system that is best suited for the given application and if there is a new and better planner on the market, we can make it available to the Golog system within a minute of work.

In the second part of the thesis we focus on the actual planning problem and its predominant approach – heuristic search.

We first have presented an analysis of heuristic search algorithms for optimal sequential planning in some standard planning domains under the assumption of *almost perfect heuristics*. Theoretical and empirical results show an exponential cost increase as tasks grow larger. In many cases, such as the Gripper

domain and a family of MICONIC tasks, there is no significant difference in node expansions between A* with an almost perfect heuristic and breadth-first search.

We argue that this is not just a theoretical problem. There is a barrier to the scaling capabilities of A*-family algorithms, and current optimal heuristic planners are pushing against it. To break the barrier, other ideas are needed.

One possible source of such ideas is the literature on *domain-dependent (optimal) search*. For example, Junghanns and Schaeffer (2001) observe that their Sokoban solver *Rolling Stone* only solves 5 out of 90 problem instances from the standard benchmark suite when using only the basic heuristic search algorithm with transposition tables. When coupled with other, domain-specific search enhancements, the total number of solved problem instances increases to 57 out of 90. Many of the techniques they present easily generalize to domain-independent planning.

However, several of the search enhancements they consider would not improve our analysis, which already makes a number of optimistic assumptions. For example, their use of *move ordering* only helps in the last search layer, which we optimistically ignored in our analysis. Their use of *deadlock tables* to detect and prune states with infinite heuristic values cannot improve the performance of our almost perfect heuristics, which by definition detect all infinite-heuristic states reliably (otherwise the heuristic error of these states would exceed the given constant). Other search enhancements, such as *overestimation* and *relevance cuts*, lose optimality or even completeness of the search algorithm.

A few techniques remain that may reduce the numbers in our analysis. These are mostly *forward pruning* techniques, which limit the set of allowed interleavings of independent parts of the solution (e. g., *tunnel macros*). However, these techniques are the ones that are most Sokoban-specific, and finding a widely useful generalization appears challenging. Some of these ideas are closely related to the concept of *partial-order reduction* in model checking (Valmari, 1989; Godefroid, 1996).

Several researchers in the planning community have picked up this line of research since the first publication of our work in 2008:

The expansion core method (Chen and Yao, 2009; Xu et al., 2011) reduces the search space of a planning task and can be cast (Wehrle and Helmert, 2012) as an instance of strong stubborn sets (Valmari, 1989), which originally have been introduced in the area of Petri nets. Wehrle et al. (2013) presented an actual instantiation of strong stubborn sets that is as efficiently computable as expansion core but provides a strictly higher pruning power. Wehrle and Helmert (2014) generalize the strong stubborn set approach and examine several instantiation strategies.

Stratified planning (Chen et al., 2009; Xu et al., 2011) is a recent transition reduction technique. However, it has been shown that its pruning power is dominated by the much older commutativity pruning method (Haslum and Geffner, 2000), which fixes the application order of commutative actions. Commutativity pruning is in turn dominated (Wehrle and Helmert, 2012) by sleep sets (Godefroid, 1996), which are known from model checking.

Move pruning (Burch and Holte, 2012) eliminates redundant operator sequences, but it is not yet clear how it can be effectively used with duplicate pruning.

One specific type of forward pruning that has been studied before in the context of domain-independent planning is symmetry reduction (Fox and Long, 1999). For example, by detecting and exploiting the equivalence of the balls of a GRIPPER task, we can easily solve arbitrarily large tasks in this domain in low-order polynomial time. Recently, Pochter et al. (2011) proposed a successful symmetry reduction technique based on automorphisms in the state transition graph, which was extended by Domshlak et al. (2012b) to larger symmetry classes. These techniques also proved to be useful for satisficing planning (Domshlak et al., 2013).

Another approach that we deem to be promising that has not been thoroughly explored yet is *problem simplification* (Haslum, 2007). The main difficulty in optimal sequential planning does not lie in finding the optimal plan; it lies in proving that no shorter plan exists. Cutting down the number of possibilities for "wasting time" by performing irrelevant actions may be a key idea for this. Also the recent bounded intention planning (Wolfe and Russell, 2011) can be seen as work along these lines. It augments the planning task with so-called *intention variables*, representing how the original variables should be used or changed next. While the approach is interesting, it is currently only applicable to *unary* SAS$^+$ tasks, in which each operator affects a single variable. So, problem simplification is a research area that still is wide open for future work.

Beyond a certain point, only trying to improve a heuristic search algorithm by refining its heuristic estimates is basically fruitless, and, indeed, several of the recent non-heuristic techniques mentioned in this section lead to a much more impressive progress than usually seen from heuristic improvements.

We therefore suggest to investigate such orthogonal performance enhancements further to improve the scaling behavior of optimal planners – unless one can reach the extremely ambitious goal of deriving *perfect*, and not merely *almost perfect* heuristics, this way rendering the search trivial.

In another contribution, we have argued that the problem of combining heuristic estimates for satisficing planning calls for different approaches than the problem of combining heuristic estimates for optimal planning. We have presented five different basic combination methods and compared them experimentally.

The *alternation* method, which performs best in our experiments, is not new: under the name *multi-heuristic best-first search*, it has been used in the Fast Downward and LAMA planners. However, prior to our experiments, the alternation method has never been systematically evaluated, and it was not clear to what extent it contributes to the performance of these planners. Moreover, it has never been compared to other approaches for combining heuristic estimates.

Our results show that aggregating different heuristic estimates into a single numeric value through arithmetic operations like taking the maximum or sum is not a good idea, even though it is the common approach for optimal planning. Our explanation for this is that such aggregation methods are easily led astray even if only one heuristic generates bad distance estimates. The Pareto and alternation approaches are much more robust to such misleading estimates.

In future work, it would be interesting to see if even better results can be obtained by including yet more estimators such as the additive (Bonet and

Geffner, 2001) or landmark heuristic (Richter et al., 2008), or if performance begins to degrade when four or more estimators are used. Another interesting question is whether *adaptive* techniques that acquire information about the heuristic during search (as for example done in the work by Domshlak et al. (2012a) for optimal planning) can improve over the performance of the alternation approach.

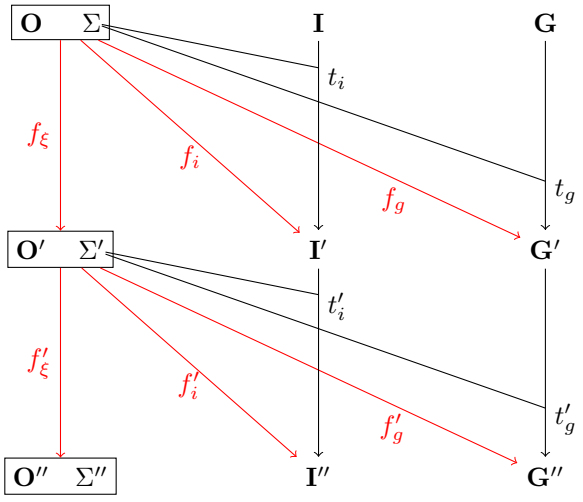
A Appendix

A.1 Proof of Theorem 4

On page 36 we introduced Theorem 4 (without proof) as follows:

Theorem 4. *If $\mathcal{X} \preceq^x \mathcal{Y}$ with a modularity-preserving compilation and $\mathcal{Y} \preceq^x \mathcal{Z}$ with a modularity-preserving compilation then $\mathcal{X} \preceq^x \mathcal{Z}$ with a modularity-preserving compilation.*

Proof. Let $\mathcal{X}, \mathcal{Y}$, and $\mathcal{Z}$ be propositional planning formalisms and let $\mathbf{f} = \langle f_\xi, f_i, f_g, t_i, t_g \rangle$ be a modularity-preserving compilation scheme from $\mathcal{X}$ to $\mathcal{Y}$ and $\mathbf{f}' = \langle f'_\xi, f'_i, f'_g, t'_i, t'_g \rangle$ be a modularity-preserving compilation scheme from $\mathcal{Y}$ to $\mathcal{Z}$. Consider an $\mathcal{X}$-instance $\Pi = \langle \Xi, \mathbf{I}, \mathbf{G} \rangle$ with $\Xi = \langle \Sigma, \mathbf{O} \rangle$. We can depict the dependencies of the compilation schemes as follows, where the red lines indicate that the respective function needs not to be computable in polynomial time.



We define a compilation scheme $\mathbf{f}'' = \left\langle f''_\xi, f''_i, f''_g, t''_i, t''_g \right\rangle$ from $\mathcal{X}$ to $\mathcal{Z}$ that is equivalent to the concatenation of $\mathbf{f}$ and $\mathbf{f}'$ via formalism $\mathcal{Y}$. We will use the notation from the picture to denote the components of the result of this

concatenation. In addition, we will use f_σ and f_o do denote the first and the second component of the result of f_ξ, i.e., $f_\xi(\Xi) = \langle f_\sigma(\Xi), f_o(\Xi)\rangle$ (for the primed functions analogously).

The definition of f''_ξ is trivial: we can define $f''_\xi(\Xi) := f'_\xi(f_\xi(\Xi))$ because

$$\langle \Sigma'', \mathbf{O}''\rangle = f'_\xi(\langle \Sigma', \mathbf{O}'\rangle) = f'_\xi(f_\xi(\langle \Sigma, \mathbf{O}\rangle)).$$

Let l_i and l_g be the modularity-preserving functions of $\mathbf{f}$ and l'_i and l'_g be the modularity-preserving functions of $\mathbf{f}'$ as specified in Definition 23.

To see, how we can define the functions f''_i and t''_i, we first show that we can rewrite $t_i(\Sigma', \mathbf{I}')$ as

$$\begin{aligned}
t'_i(\Sigma', \mathbf{I}') &= t'_i(\Sigma', f_i(\langle \Sigma, \mathbf{O}\rangle) \cup t_i(\Sigma, \mathbf{I})) \\
&\overset{(1)}{=} t'_i(\Sigma' \setminus l_i(\Sigma), (f_i(\langle \Sigma, \mathbf{O}\rangle) \cup t_i(\Sigma, \mathbf{I})) \cap (\widehat{\Sigma' \setminus l_i(\Sigma)})) \cup \\
&\quad\ t'_i(l_i(\Sigma), (f_i(\langle \Sigma, \mathbf{O}\rangle) \cup t_i(\Sigma, \mathbf{I})) \cap \widehat{l_i(\Sigma)}) \\
&\overset{(2)}{=} t'_i(\Sigma' \setminus l_i(\Sigma), f_i(\langle \Sigma, \mathbf{O}\rangle)) \cup t'_i(l_i(\Sigma), t_i(\Sigma, \mathbf{I})) \\
&\overset{(3)}{=} (t'_i(\Sigma' \setminus l_i(\Sigma), f_i(\langle \Sigma, \mathbf{O}\rangle)) \cap \widehat{l'_i(\{\})}) \cup \\
&\quad\ (t'_i(\Sigma' \setminus l_i(\Sigma), f_i(\langle \Sigma, \mathbf{O}\rangle)) \setminus \widehat{l'_i(\{\})}) \cup \\
&\quad\ (t'_i(l_i(\Sigma), t_i(\Sigma, \mathbf{I})) \cap \widehat{l'_i(\{\})}) \cup \\
&\quad\ (t'_i(l_i(\Sigma), t_i(\Sigma, \mathbf{I})) \setminus \widehat{l'_i(\{\})}) \\
&\overset{(4)}{=} t'_i(\{\}, \{\}) \cup (t'_i(\Sigma' \setminus l_i(\Sigma), f_i(\langle \Sigma, \mathbf{O}\rangle)) \setminus \widehat{l'_i(\{\})}) \cup \\
&\quad\ t'_i(\{\}, \{\}) \cup (t'_i(l_i(\Sigma), t_i(\Sigma, \mathbf{I})) \setminus \widehat{l'_i(\{\})}) \\
&\overset{(5)}{=} (t'_i(\Sigma' \setminus l_i(\Sigma), f_i(\langle \Sigma, \mathbf{O}\rangle)) \setminus \widehat{l'_i(\{\})}) \cup \\
&\quad\ t'_i(l_i(\Sigma), t_i(\Sigma, \mathbf{I}))
\end{aligned}$$

with the following justifications:

(1) modularity of t'_i,

(2) Definition 23: conditions 1 and 2,

(3) set operations,

(4) Definition 23: condition 5, and

(5) set operations.

Based on this, we define f''_i and t''_i for $\Xi = \langle \Sigma, \mathbf{O}\rangle$ as

$$\begin{aligned}
f''_i(\Xi) &:= f'_i(f_\xi(\Xi)) \cup (t'_i(f_\sigma(\Xi) \setminus l_i(\Sigma), f_i(\Xi)) \setminus \widehat{l'_i(\{\})}) \text{ and} \\
t''_i(\Sigma, S) &:= t'_i(l_i(\Sigma), t_i(\Sigma, S)).
\end{aligned}$$

This leads to the correct result which is easy to see by applying the previous reformulation of $t'(\Sigma', \mathbf{I}')$ and using that $\Sigma' = f_\sigma(\Xi)$:

$$\begin{aligned}\mathbf{I}'' &= f'_i(\langle\Sigma', \mathbf{O}'\rangle) \cup t'_i(\Sigma', \mathbf{I}') \\ &= f'_i(f_\xi(\langle\Sigma, \mathbf{O}\rangle)) \cup (t'_i(\Sigma' \setminus l_i(\Sigma), f_i(\langle\Sigma, \mathbf{O}\rangle)) \setminus \widehat{l'_i(\{\})}) \cup \\ &\quad t'_i(l_i(\Sigma), t_i(\Sigma, \mathbf{I})) \\ &= f''_i(\Xi) \cup t''_i(\Sigma, \mathbf{I})\end{aligned}$$

With an analogous argument, we can define f''_g and t''_g for $\Xi = \langle\Sigma, \mathbf{O}\rangle$ as

$$\begin{aligned}f''_g(\Xi) &:= f'_g(f_\xi(\Xi)) \cup (t'_g(f_\sigma(\Xi) \setminus l_i(\Sigma), f_g(\Xi)) \setminus \widehat{l'_g(\{\})}) \text{ and} \\ t''_g(\Sigma, S) &:= t'_g(l_g(\Sigma), t_g(\Sigma, S)).\end{aligned}$$

We have seen that the given definitions lead to the correct results but still need to show that they define a modularity-preserving compilation scheme from $\mathcal{X}$ to $\mathcal{Z}$.

It is trivial that the functions preserve plan existence and satisfy the restrictions on the plan length because $\mathbf{f}$ and $\mathbf{f}'$ do so. It is also easy to check that the place and time restrictions of compilation schemes are fulfilled. So it is only left to show that the state translation function t''_i and t''_g are modular and that the compilation is modularity-preserving. We start with the former: Let $\Sigma = \Sigma_1 \cup \Sigma_2$ and $S \subseteq \hat{\Sigma}$. Then it holds for $x = i, g$ that

$$\begin{aligned}t''_x(\Sigma, S) &= t'_x(l_x(\Sigma), t_x(\Sigma, S)) \\ &= t'_x(l_x(\Sigma_1) \cup l_x(\Sigma_2), t_x(\Sigma, S)) \\ &= t'_x(l_x(\Sigma_1), t_x(\Sigma, S) \cap \widehat{l_x(\Sigma_1)}) \cup \\ &\quad t'_x(l_x(\Sigma_2), t_x(\Sigma, S) \cap \widehat{l_x(\Sigma_2)}) \\ &= t'_x(l_x(\Sigma_1), t_x(\Sigma_1, S \cap \hat{\Sigma}_1)) \cup t'_x(l_x(\Sigma_2), t_x(\Sigma_2, S \cap \hat{\Sigma}_2)) \\ &= t''_x(\Sigma_1, S \cap \hat{\Sigma}_1) \cup t''_x(\Sigma_2, S \cap \hat{\Sigma}_2).\end{aligned}$$

In order to show that the compilation is modularity-preserving, we define the modularity-preserving functions l''_i and l''_g of $\mathbf{f}''$ as

$$l''_x(\Sigma) = l'_x(l_x(\Sigma)).$$

This definition satisfies the conditions of Definition 23 (for $x = i, g$):

1. $t''_x(\Sigma, S) = t'_x(l_x(\Sigma), t_x(\Sigma, S)) \subseteq \widehat{l'_x(l_x(\Sigma))} = \widehat{l''_x(\Sigma)}$
2. Since $f''_x(\Xi) = f'_x(f_\xi(\Xi)) \cup (t'_x(f_\sigma(\Xi) \setminus l_x(\Sigma), f_x(\Xi)) \setminus \widehat{l'_x(\{\})})$ for $\Xi = \langle\Sigma, \mathbf{O}\rangle$, we can prove that $f''_x(\Xi) \cap \widehat{l''_x(\Sigma)} = \emptyset$ by showing that

$$\begin{aligned}f'_x(f_\xi(\Xi)) \cap \widehat{l''_x(\Sigma)} = \emptyset, \text{ and} \\ (t'_x(f_\sigma(\Xi) \setminus l_x(\Sigma), f_x(\Xi)) \setminus \widehat{l'_x(\{\})}) \cap \widehat{l''_x(\Sigma)} = \emptyset.\end{aligned}$$

   a) As $\mathbf{f}'$ is modularity-preserving, $f'_x(f_\xi(\Xi)) \cap \widehat{l'_x(f_\sigma(\Xi))} = \emptyset$. Since $l_x(\Sigma) \subseteq f_\sigma(\Xi)$, it holds that $l''_x(\Sigma) = l'_x(l_x(\Sigma)) \subseteq l'_x(f_\sigma(\Xi))$ and we can conclude that $f'_x(f_\xi(\Xi)) \cap \widehat{l''_x(\Sigma)} = \emptyset$.

b) Since $\mathbf{f}'$ is modularity-preserving, it holds that

$$t'_x(f_\sigma(\Xi) \setminus l_x(\Sigma), f_x(\Xi)) \subseteq l'_x(f_\sigma(\widehat{\Xi) \setminus l_x}(\Sigma)).$$

As for the same reason $l'_x(f_\sigma(\Xi) \setminus l_x(\Sigma)) \cap l'_x(l_x(\Sigma)) = l'_x(\{\})$ is true, we know that $t'_x(f_\sigma(\Xi) \setminus l_x(\Sigma), f_x(\Xi)) \cap \widehat{l''_x(\Sigma)} \subseteq \widehat{l'_x(\{\})}$. Thus, we can conclude that $(t'_x(f_\sigma(\Xi) \setminus l_x(\Sigma), f_x(\Xi)) \setminus \widehat{l'_x(\{\})}) \cap \widehat{l''_x(\Sigma)} = \emptyset$.

3. For Σ_1, Σ_2 such that $\Sigma = \Sigma_1 \cup \Sigma_2$ it holds that

$$\begin{aligned} l''_x(\Sigma) &= l'_x(l_x(\Sigma)) \\ &= l'_x(l_x(\Sigma_1) \cup l_x(\Sigma_2)) \\ &= l'_x(l_x(\Sigma_1)) \cup l'_x(l_x(\Sigma_2)) \\ &= l''_x(\Sigma_1) \cup l''_x(\Sigma_2). \end{aligned}$$

4. For $\Sigma_1 \cap \Sigma_2 = \emptyset$ it holds that

$$\begin{aligned} l''_x(\Sigma_1) \cap l''_x(\Sigma_2) &= l'_x(l_x(\Sigma_1)) \cap l'_x(l_x(\Sigma_2)) \\ &= l'_x((l_x(\Sigma_1) \setminus l_x(\Sigma_2)) \cup (l_x(\Sigma_1) \cap l_x(\Sigma_2))) \cap \\ &\quad l'_x((l_x(\Sigma_2) \setminus l_x(\Sigma_1)) \cup (l_x(\Sigma_2) \cap l_x(\Sigma_1))) \\ &= l'_x((l_x(\Sigma_1) \setminus l_x(\Sigma_2)) \cup l_x(\{\})) \cap \\ &\quad l'_x((l_x(\Sigma_2) \setminus l_x(\Sigma_1)) \cup l_x(\{\})) \\ &= (l'_x(l_x(\Sigma_1) \setminus l_x(\Sigma_2)) \cup l'_x(l_x(\{\}))) \cap \\ &\quad (l'_x(l_x(\Sigma_2) \setminus l_x(\Sigma_1)) \cup l'_x(l_x(\{\}))) \\ &= (l'_x(l_x(\Sigma_1) \setminus l_x(\Sigma_2)) \cap l'_x(l_x(\Sigma_2) \setminus l_x(\Sigma_1))) \cup \\ &\quad (l'_x(l_x(\Sigma_1) \setminus l_x(\Sigma_2)) \cap l'_x(l_x(\{\}))) \cup \\ &\quad (l'_x(l_x(\Sigma_2) \setminus l_x(\Sigma_1)) \cap l'_x(l_x(\{\}))) \cup l'_x(l_x(\{\})) \\ &= l'_x(\{\}) \cup l'_x(\{\}) \cup l'_x(\{\}) \cup l'_x(l_x(\{\})) \\ &= l'_x(l_x(\{\})) = l''_x(\{\}). \end{aligned}$$

5. For $\Sigma' \subseteq \Sigma$ (1) it holds that $l_x(\Sigma') \subseteq l_x(\Sigma)$ (2), so we can conclude (using condition 5 of Definition 23) that

$$\begin{aligned} t''_x(\Sigma', S \cap \hat{\Sigma}') &= t'_x(l_x(\Sigma'), t_x(\Sigma', S \cap \hat{\Sigma}')) \\ &\overset{(1)}{=} t'_x(l_x(\Sigma'), t_x(\Sigma, S) \cap \widehat{l_x(\Sigma')}) \\ &\overset{(2)}{=} t'_x(l_x(\Sigma), t_x(\Sigma, S)) \cap l'_x\widehat{(l_x(\Sigma'))} \\ &= t''_x(\Sigma, S) \cap \widehat{l''_x(\Sigma')}. \end{aligned}$$

To sum up: If $\mathcal{X} \preceq^x \mathcal{Y}$ with a modularity-preserving compilation scheme $\mathbf{f} = \langle f_\xi, f_i, f_g, t_i, t_g \rangle$ and $\mathcal{Y} \preceq^x \mathcal{Z}$ with a modularity-preserving compilation $\mathbf{f}' = \langle f'_\xi, f'_i, f'_g, t'_i, t'_g \rangle$ then $\mathcal{X} \preceq^x \mathcal{Z}$ with compilation $\mathbf{f}'' =$

$\langle f''_\xi, f''_i, f''_g, t''_i, t''_g \rangle$ whose components are defined as

$$f''_\xi(\Xi) = f'_\xi(f_\xi(\Xi)), \tag{A.1}$$

$$f''_i(\Xi) = f'_i(f_\xi(\Xi)) \cup (t'_i(f_\sigma(\Xi) \setminus l_i(\Sigma), f_i(\Xi)) \setminus \widehat{l'_i(\{\})}), \tag{A.2}$$

$$f''_g(\Xi) = f'_g(f_\xi(\Xi)) \cup (t'_g(f_\sigma(\Xi) \setminus l_g(\Sigma), f_g(\Xi)) \setminus \widehat{l'_g(\{\})}), \tag{A.3}$$

$$t''_i(\Sigma, S) = t'_i(l_i(\Sigma), t_i(\Sigma, S)), \text{ and} \tag{A.4}$$

$$t''_g(\Sigma, S) = t'_g(l_g(\Sigma), t_g(\Sigma, S)), \tag{A.5}$$

where l_i, l_g, l'_i, and l'_g are the modularity-preserving functions of $\mathbf{f}$ and $\mathbf{f}'$, respectively. Moreover, $\mathbf{f}''$ is modularity-preserving with functions

$$l''_i(\Sigma) = l'_i(l_i(\Sigma)) \tag{A.6}$$

$$l''_g(\Sigma) = l'_g(l_g(\Sigma)). \tag{A.7}$$

□

Bibliography

Bacchus, F., Halpern, J. Y., and Levesque, H. J. (1995). Reasoning about noisy sensors in the situation calculus. In Martin and Ralescu (1995), pages 1933–1940.

Bäckström, C. (1995). Expressive equivalence of planning formalisms. *Artificial Intelligence*, 76:17–34.

Bäckström, C. and Nebel, B. (1995). Complexity results for SAS^+ planning. *Computational Intelligence*, 11(4):625–655.

Baier, J. A., Fritz, C., Bienvenu, M., and McIlraith, S. A. (2008). Beyond classical planning: Procedural control knowledge and preferences in state-of-the-art planners. In Fox and Gomes (2008), pages 1509–1512.

Baier, J. A., Fritz, C., and McIlraith, S. A. (2007). Exploiting procedural domain control knowledge in state-of-the-art planners. In Boddy et al. (2007), pages 26–33.

Berend, D. and Tassa, T. (2010). Improved bounds on Bell numbers and on moments of sums of random variables. *Probability and Mathematical Statistics*, 30(2):185–205.

Blom, M. (2011). *Decoupling Preference and Planning through Argumentation*. PhD thesis, The University of Melbourne.

Blom, M. and Pearce, A. (2010). Relaxed regression for a heuristic GOLOG. In Ågotnes, T., editor, *Proceedings of the Fifth Starting AI Researchers' Symposium (STAIRS 2010)*. IOS Press.

Blum, A. L. and Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1–2):281–300.

Boddy, M., Fox, M., and Thiébaux, S., editors (2007). *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling (ICAPS 2007)*. AAAI Press.

Bonet, B. and Geffner, H. (2001). Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33.

Borrajo, D., Felner, A., Korf, R., Likhachev, M., Linares López, C., Ruml, W., and Sturtevant, N., editors (2012). *Proceedings of the Fifth Annual Symposium on Combinatorial Search (SoCS 2012)*. AAAI Press.

Borrajo, D., Kambhampati, S., Oddi, A., and Fratini, S., editors (2013). *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*. AAAI Press.

Boutilier, C., editor (2009). *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*.

Burch, N. and Holte, R. C. (2012). Automatic move pruning revisited. In Borrajo et al. (2012), pages 18–24.

Burgard, W., Cremers, A. B., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., and Thrun, S. (1998). The interactive museum tour-guide robot. In Rich, C. and Mostow, J., editors, *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI 1998)*, pages 11–18. AAAI Press.

Burgard, W., Cremers, A. B., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., and Thrun, S. (1999). Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 114(1–2):3–55.

Bylander, T. (1994). The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69(1–2):165–204.

Carbonell, J. G., Blythe, J., Etzioni, O., Gil, Y., Joseph, R., Kahn, D., Knoblock, C., Minton, S., Perez, A., Reilly, S., Veloso, M., and Wang, X. (1992). Prodigy 4.0: The manual and tutorial. Technical Report CMU-CS-92-150, Carnegie Mellon University.

Chapman, D. (1987). Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377.

Chen, Y., Xu, Y., and Yao, G. (2009). Stratified planning. In Boutilier (2009), pages 1665–1670.

Chen, Y. and Yao, G. (2009). Completeness and optimality preserving reduction for planning. In Boutilier (2009), pages 1659–1664.

Claßen, J., Hu, Y., and Lakemeyer, G. (2007). A situation-calculus semantics for an expressive fragment of PDDL. In Holte and Howe (2007), pages 956–961.

Claßen, J. and Lakemeyer, G. (2006). A semantics for ADL as progression in the situation calculus. In Dix, J. and Hunter, A., editors, *Proceedings of the 11th Workshop on Nonmonotonic Reasoning (NMR 2006)*, pages 334–341. Institut für Informatik, TU Clausthal.

Claßen, J., Engelmann, V., Lakemeyer, G., and Röger, G. (2008). Integrating Golog and planning: An empirical evaluation. In *Proceedings of the 12th International Workshop on Nonmonotonic Reasoning (NMR 2008)*, pages 10–18.

Claßen, J., Eyerich, P., Lakemeyer, G., and Nebel, B. (2007). Towards an integration of Golog and planning. In Veloso (2007), pages 1846–1851.

Davis, E. (1990). *Representations of commonsense knowledge*. Morgan Kaufmann.

de Bruijn, N. G. (1958). *Asymptotic Methods in Analysis*, volume 4 of *Bibliotheca mathematica: a series of monographs in pure and applied mathematics*. North-Holland, Amsterdam.

de Giacomo, G., Lespérance, Y., and Levesque, H. J. (2000). ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1–2):109–169.

de Giacomo, G. and Levesque, H. J. (1999). An incremental interpreter for high-level programs with sensing. *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*, pages 86–102.

Dinh, H., Russell, A., and Su, Y. (2007). On the value of good advice: The complexity of A* search with accurate heuristics. In Holte and Howe (2007), pages 1140–1145.

Doherty, P., Gustafsson, J., Karlsson, L., and Kvarnström, J. (1998). TAL: Temporal Action Logics Language. *Electronic Transactions on Artificial Intelligence*, 2(3–4):273–306.

Doherty, P. and Kvarnström, J. (2001). TALplanner: A temporal logic based planner. *AI Magazine*, 22(3):95–102.

Domshlak, C., Helmert, M., Karpas, E., and Markovitch, S. (2011). The SelMax planner: Online learning for speeding up optimal planning. In *IPC 2011 planner abstracts*, pages 108–112.

Domshlak, C., Karpas, E., and Markovitch, S. (2012a). Online speedup learning for optimal planning. *Journal of Artificial Intelligence Research*, 44:709–755.

Domshlak, C., Katz, M., and Shleyfman, A. (2012b). Enhanced symmetry breaking in cost-optimal planning as forward search. In McCluskey et al. (2012).

Domshlak, C., Katz, M., and Shleyfman, A. (2013). Symmetry breaking: Satisficing planning and landmark heuristics. In Borrajo et al. (2013), pages 298–302.

Edelkamp, S. and Helmert, M. (2001). The model checking integrated planning system (MIPS). *AI Magazine*, 22(3):67–71.

Edelkamp, S. and Hoffmann, J. (2004). PDDL2.2: The language for the classical part of the 4th International Planning Competition. Technical Report 195, Albert-Ludwigs-Universität Freiburg, Institut für Informatik.

Edelkamp, S. and Kissmann, P. (2008). GAMER: Bridging planning and general game playing with symbolic search. IPC 2008 short papers http://ipc.informatik.uni-freiburg.de/Planners.

Erol, K., Hendler, J., and Nau., D. S. (1994). UMCP: A sound and complete procedure for hierarchical task-network planning. In Hammond, K., editor, *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems (AIPS 1994)*, pages 249–254. AAAI Press.

Erol, K., Nau, D. S., and Subrahmanian, V. S. (1995). Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76(1–2):65–88.

Eyerich, P. (2006). *Zu ADL gleichausdrucksstarke Basic-Action-Theorien im Situationskalkül*. Term paper (Studienarbeit), University of Freiburg, Freiburg, Germany.

Eyerich, P., Nebel, B., Lakemeyer, G., and Claßen, J. (2006). Golog and PDDL: What is the relative expressiveness? In *Proceedings of the International Symposium on Practical Cognitive Agents and Robots (PCAR 2006)*, pages 93–104. University of Western Australia Press.

Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208.

Finzi, A. and Pirri, F. (2004). Flexible interval planning in concurrent temporal Golog. In *Working Notes of the Fourth International Cognitive Robotics Workshop (CogRob 2004)*.

Fox, D. and Gomes, C. P., editors (2008). *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*. AAAI Press.

Fox, M. and Long, D. (1999). The detection and exploitation of symmetry in planning problems. In Dean, T., editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 1999)*, pages 956–961. Morgan Kaufmann.

Fox, M. and Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124.

Gaschnig, J. (1977). Exactly how good are heuristics? Toward a realistic predictive theory of best-first search. In Reddy, R., editor, *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI 1977)*, pages 434–441. William Kaufmann.

Geffner, H. (2000). Functional Strips: A more flexible language for planning and problem solving. In Minker, J., editor, *Logic-Based Artificial Intelligence*, volume 597 of *Kluwer International Series In Engineering And Computer Science*, chapter 9, pages 187–209. Kluwer, Dordrecht.

Gelfond, M. and Lifschitz, V. (1993). Representing action and change by logic programs. *Journal of Logic Programming*, 17(2):301–322.

Gelfond, M. and Lifschitz, V. (1998). Action languages. *Electronic Transactions of Artificial Intelligence*, 2(3–4):193–210.

Gerevini, A., Howe, A., Cesta, A., and Refanidis, I., editors (2009). *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*. AAAI Press.

Gerevini, A. and Long, D. (2005). Plan constraints and preferences in PDDL3. Technical Report R. T. 2005-08-47, Dipartimento di Elettronica per l'Automazione, Università degli Studi di Brescia.

Godefroid, P. (1996). *Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer-Verlag.

Green, C. (1969). Application of theorem proving to problem solving. In Walker, D. E. and Norton, L. M., editors, *Proceedings of the First International Joint Conference on Artificial Intelligence (IJCAI 1969)*, pages 219–239. William Kaufmann.

Grosskreutz, H. (2002). *Towards More Realistic Logic-Based Robot Controllers in the GOLOG Framework*. PhD thesis, RWTH Aachen, Aachen, Germany.

Haas, A. (1987). The case for domain-specific frame axioms. In *Proceedings of the Workshop on the Frame Problem in Artificial Intelligence*, pages 343–348.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1972). Correction to a formal basis for the heuristic determination of minimum cost paths. *SIGART Newsletter*, 37:28–29.

Haslum, P. (2007). Reducing accidental complexity in planning problems. In Veloso (2007), pages 1898–1903.

Haslum, P., Bonet, B., and Geffner, H. (2005). New admissible heuristics for domain-independent planning. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005)*, pages 1163–1168. AAAI Press.

Haslum, P., Botea, A., Helmert, M., Bonet, B., and Koenig, S. (2007). Domain-independent construction of pattern database heuristics for cost-optimal planning. In Holte and Howe (2007), pages 1007–1012.

Haslum, P. and Geffner, H. (2000). Admissible heuristics for optimal planning. In Chien, S., Kambhampati, S., and Knoblock, C. A., editors, *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2000)*, pages 140–149. AAAI Press.

Helmert, M. (2001). On the complexity of planning in transportation domains. In Cesta, A. and Borrajo, D., editors, *Pre-proceedings of the Sixth European Conference on Planning (ECP 2001)*, pages 349–360, Toledo, Spain.

Helmert, M. (2003). Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2):219–262.

Helmert, M. (2004). A planning heuristic based on causal graph analysis. In Zilberstein et al. (2004), pages 161–170.

Helmert, M. (2006). The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246.

Helmert, M. (2008). *Understanding Planning Tasks – Domain Complexity and Heuristic Decomposition*, volume 4929 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag.

Helmert, M. (2009). Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence*, 173:503–535.

Helmert, M., Do, M., and Refanidis, I. (2008). Changes in PDDL 3.1.

Helmert, M. and Domshlak, C. (2009). Landmarks, critical paths and abstractions: What's the difference anyway? In Gerevini et al. (2009), pages 162–169.

Helmert, M. and Geffner, H. (2008). Unifying the causal graph and additive heuristics. In Rintanen et al. (2008), pages 140–147.

Helmert, M., Haslum, P., and Hoffmann, J. (2007). Flexible abstraction heuristics for optimal sequential planning. In Boddy et al. (2007), pages 176–183.

Helmert, M., Mattmüller, R., and Röger, G. (2006). Approximation properties of planning benchmarks. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006)*, pages 585–589.

Helmert, M. and Röger, G. (2008). How good is almost perfect? In Fox and Gomes (2008), pages 944–949.

Hernádvölgyi, I. T. and Holte, R. C. (1999). PSVN: A vector representation for production systems. Technical Report tr-99-04, University of Ottawa, Computer Science Department.

Hoffmann, J. and Edelkamp, S. (2005). The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research*, 24:519–579.

Hoffmann, J. and Nebel, B. (2001). The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302.

Holte, R. C. and Howe, A. E., editors (2007). *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22–26, 2007, Vancouver, British Columbia, Canada*. AAAI Press.

Jonsson, P. and Bäckström, C. (1998). State-variable planning under structural restrictions: Algorithms and complexity. *Artificial Intelligence*, 100(1–2):125–176.

Junghanns, A. and Schaeffer, J. (2001). Sokoban: Enhancing general single-agent search methods using domain knowledge. *Artificial Intelligence*, 129(1–2):219–251.

Karp, R. M. (1972). Reducibility among combinatorial problems. In Miller, R. E. and Thatcher, J. W., editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press.

Karpas, E. and Domshlak, C. (2009). Cost-optimal planning with landmarks. In Boutilier (2009), pages 1728–1733.

Katz, M. and Domshlak, C. (2008). Optimal additive composition of abstraction-based admissible heuristics. In Rintanen et al. (2008), pages 174–181.

Katz, M. and Domshlak, C. (2010). Optimal admissible composition of abstraction heuristics. *Artificial Intelligence*, 174(12–13):767–798.

Kautz, H. and Selman, B. (1999). Unifying SAT-based and graph-based planning. In Dean, T., editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI 1999)*, pages 318–325. Morgan Kaufmann.

Keyder, E., Hoffmann, J., and Haslum, P. (2012). Semi-relaxed plan heuristics. In McCluskey et al. (2012), pages 128–136.

Kissmann, P. (2012). *Symbolic Search in Planning and General Game Playing*. PhD thesis, Universität Bremen, Germany.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109.

Korf, R. E., Reid, M., and Edelkamp, S. (2001). Time complexity of iterative-deepening A*. *Artificial Intelligence*, 129:199–218.

Kowalski, R. and Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing*, 4(1):67–94.

Lakemeyer, G. (1999). On sensing and offline-interpreting in GOLOG. *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter*, pages 173–187.

Lakemeyer, G. (2010). The situation calculus: A case for modal logic. *Journal of Logic, Language and Information*, 19(4):431–450.

Levesque, H., Pirri, F., and Reiter, R. (1998). Foundations for a calculus of situations. *Electronic Transactions of Artificial Intelligence*, 2(3–4):159–178.

Levesque, H., Reiter, R., Lespérance, Y., Lin, F., and Scherl, R. (1997). GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1–3):59–83.

Lifschitz, V. (1987). On the semantics of STRIPS. In Georgeff, M. and Lansky, A., editors, *Reasoning about Actions and Plans*, pages 1–9. Morgan Kaufmann.

Lin, F. (1995). Embracing causality in specifying the indirect effects of actions. In Martin and Ralescu (1995), pages 1985–1993.

Lin, F. and Reiter, R. (1997). How to progress a database. *Artificial Intelligence*, 92(1–2):131–167.

Martin, T. P. and Ralescu, A. L., editors (1995). *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 1995)*. Morgan Kaufmann.

McCarthy, J. (1959). Programs with common sense. In *Proceedings of the Symposium on the Mechanisation of Thought Processes*, pages 77–84.

McCarthy, J. (1963). Situations, actions, and causal laws. Technical report, Artificial Intelligence Project, Stanford University.

McCarthy, J. (1968). Situations, actions, and causal laws. In Minsky, M. L., editor, *Semantic Information Processing*, pages 410–417. The MIT Press.

McCarthy, J. and Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502.

McCluskey, L., Williams, B., Silva, J. R., and Bonet, B., editors (2012). *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*. AAAI Press.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL – The Planning Domain Definition Language – Version 1.2. Technical Report CVC TR-98-003, Yale Center for Computational Vision and Control.

McDermott, D. V. (1996). A heuristic estimator for means ends analysis in planning. In Drabble, B., editor, *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems (AIPS 1996)*, pages 142–149. AAAI Press.

McDermott, D. V. (2000). The 1998 AI planning systems competition. *AI Magazine*, 21(2):35–55.

Miller, R. and Shanahan, M. (1999). The event-calculus in classical logic – alternative axiomatizations. *Electronic Transactions of Artificial Intelligence*, 3(1):77–105.

Moore, R. C. (1979). *Reasoning about Knowledge and Action*. PhD thesis, Massachusetts Institute of Technology.

Nebel, B. (1999). What is the expressive power of disjunctive preconditions? In Biundo, S. and Fox, M., editors, *Proceedings of the 5th European Conference on Planning (ECP 1999)*, volume 1809 of *Lecture Notes in Computer Science*, pages 294–307. Springer-Verlag.

Nebel, B. (2000a). On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12:271–315.

Nebel, B. (2000b). On the expressive power of planning formalisms: Conditional effects and boolean preconditions in the STRIPS formalism. In Minker, J., editor, *Logic-Based Artificial Intelligence*, pages 469–490. Kluwer Academic Publishers.

Nebel, B., Dimopoulos, Y., and Koehler, J. (1997). Ignoring irrelevant facts and operators in plan generation. In Steel, S. and Alami, R., editors, *Proceedings of the 4th European Conference on Planning (ECP 1997)*, volume 1348 of *Lecture Notes in Computer Science*, pages 338–350. Springer-Verlag.

Nissim, R., Hoffmann, J., and Helmert, M. (2011a). Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In Walsh (2011), pages 1983–1990.

Nissim, R., Hoffmann, J., and Helmert, M. (2011b). The Merge-and-Shrink planner: Bisimulation-based abstraction for optimal planning. In *IPC 2011 planner abstracts*, pages 106–107.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley.

Pednault, E. (1989). ADL: Exploring the middle ground between STRIPS and the situation calculus. In Brachman, R. J., Levesque, H. J., and Reiter, R., editors, *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning (KR 1989)*, pages 324–332. Morgan Kaufmann.

Penberthy, J. S. and Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. In Nebel, B., Rich, C., and Swartout, W., editors, *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR 1992)*, pages 103–114. Morgan Kaufmann.

Pinto, J. (1994). *Temporal Reasoning in the Situation Calculus*. PhD thesis, University of Toronto.

Pirri, F. and Reiter, R. (1999). Some contributions to the metatheory of the situation calculus. *Journal of the ACM*, 46(3):325–362.

Pochter, N., Zohar, A., and Rosenschein, J. S. (2011). Exploiting problem symmetries in state-based planners. In Burgard, W. and Roth, D., editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*, pages 1004–1009. AAAI Press.

Pohl, I. (1970). Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3–4):193–204.

Pohl, I. (1977). Practical and theoretical considerations in heuristic search algorithms. In Elcock, E. W. and Michie, D., editors, *Machine Intelligence 8*, pages 55–72. Ellis Horwood.

Reiter, R. (1991). The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honour of John McCarthy*, pages 359–380. Academic Press Professional, Inc.

Reiter, R. (1996). Natural actions, concurrency and continuous time in the situation calculus. In Aiello, L. C., Doyle, J., and Shapiro, S. C., editors, *Proceedings of the 5th International Conference on the Principles of Knowledge Representation and Reasoning (KR 1996)*, pages 2–13. Morgan Kaufmann.

Reiter, R. (1998). Sequential, temporal GOLOG. In Cohn, A. G., Schubert, L. K., and Shapiro, S. C., editors, *Proceedings of the Sixth International Conference on the Principles of Knowledge Representation and Reasoning (KR 1998)*, pages 547–13. Morgan Kaufmann.

Reiter, R. (2001). *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press.

Richter, S. and Helmert, M. (2009). Preferred operators and deferred evaluation in satisficing planning. In Gerevini et al. (2009), pages 273–280.

Richter, S., Helmert, M., and Westphal, M. (2008). Landmarks revisited. In Fox and Gomes (2008), pages 975–982.

Richter, S. and Westphal, M. (2010). The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39:127–177.

Richter, S., Westphal, M., and Helmert, M. (2011). LAMA 2008 and 2011 (planner abstract). In *IPC 2011 planner abstracts*, pages 50–54.

Rintanen, J. (2003). Symmetry reduction for SAT representations of transition systems. In Giunchiglia, E., Muscettola, N., and Nau, D., editors, *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003)*, pages 32–40. AAAI Press.

Rintanen, J. (2004). Complexity of planning with partial observability. In Zilberstein et al. (2004), pages 345–354.

Rintanen, J. (2005). *Automated Planning: Algorithms and Complexity*. Habilitation thesis, Albert-Ludwigs-Universität Freiburg.

Rintanen, J. (2010). Heuristics for planning with SAT. In Cohen, D., editor, *Proceedings of the 16th International Conference on Principles and Practice of Constraint Programming*, pages 414–428. Springer-Verlag.

Rintanen, J., Nebel, B., Beck, J. C., and Hansen, E., editors (2008). *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS 2008)*. AAAI Press.

Röger, G. and Helmert, M. (2009). Combining heuristic estimators for satisficing planning. In *ICAPS 2009 Workshop on Heuristics for Domain-Independent Planning*, pages 43–48.

Röger, G. and Helmert, M. (2010). The more, the merrier: Combining heuristic estimators for satisficing planning. In Brafman, R., Geffner, H., Hoffmann, J., and Kautz, H., editors, *Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)*, pages 246–249. AAAI Press.

Röger, G., Helmert, M., and Nebel, B. (2008). On the relative expressiveness of ADL and Golog: The last piece in the puzzle. In Brewka, G. and Lang, J., editors, *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 544–550. AAAI Press.

Röger, G. and Nebel, B. (2007). Expressiveness of ADL and Golog: Functions make a difference. In Holte and Howe (2007), pages 1051–1056.

Röger, G., Pommerening, F., and Helmert, M. (2014). Optimal planning in the presence of conditional effects: Extending LM-Cut with context splitting. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*. To appear.

Sacerdoti, E. D. (1975). The nonlinear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI 1975)*, pages 206–214.

Sandewall, E. (1994). *Features and Fluents: A Systematic Approach to the Representation of Knowledge about Dynamical Systems*, volume 1. Oxford University Press.

Savitch, W. J. (1970). Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192.

Scherl, R. and Levesque, H. (1993). The frame problem and knowledge-producing actions. In Fikes, R. and Lehnert, W., editors, *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI 1993)*, pages 689–695. AAAI Press.

Schiffel, S. and Thielscher, M. (2006). Reconciling situation calculus and fluent calculus. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI 2006)*, pages 287–292. AAAI Press.

Schiffer, S., Ferrein, A., and Lakemeyer, G. (2012). Caesar – an intelligent domestic service robot. *Intelligent Service Robotics*, 5(4):259–273.

Shanahan, M. (1995). A circumscriptive calculus of events. *Artificial Intelligence*, 77(2):249–284.

Sievers, S., Ortlieb, M., and Helmert, M. (2012). Efficient implementation of pattern database heuristics for classical planning. In Borrajo et al. (2012), pages 105–111.

Sohrabi, S., Prokoshyna, N., and McIlraith, S. A. (2009). Composition via the customization of Golog programs with user preferences. In Borgida, A., Chaudhri, V., Giorgini, P., and Yu, E., editors, *Conceptual Modeling: Foundations and Applications*, volume 5600 of *Lecture Notes in Computer Science*, pages 319–334. Springer-Verlag.

Stewart, B. S. and White, III, C. C. (1991). Multiobjective A*. *Journal of the ACM*, 38(4):775–814.

Stockmeyer, L. J. and Meyer, A. R. (1973). Word problems requiring exponential time. In *Proceedings of the 5th Symposium on Theory of Computing*, pages 1–9. ACM.

Tam, K., Lloyd, J., Lespérance, Y., Levesque, H. J., Lin, F., Marcu, D., Reiter, R., and Jenkin, M. R. M. (1997). Controlling autonomous robots with GOLOG. In Pollack, M. E., editor, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 1–12. Morgan Kaufmann.

Thiébaux, S., Hoffmann, J., and Nebel, B. (2003). In defense of PDDL axioms. In Gottlob, G. and Walsh, T., editors, *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, pages 961–968. Morgan Kaufmann.

Thiébaux, S., Hoffmann, J., and Nebel, B. (2005). In defense of PDDL axioms. *AIJ*, 168(1–2):38–69.

Thielscher, M. (1999). From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, 111(1–2):277–299.

Thielscher, M. (2005). FLUX: A logic programming method for reasoning agents. *Theory and Practice of Logic Programming*, 5(4–5):533–565.

Tung, C. and Chew, K. (1992). A multi-criteria Pareto-optimal path algorithm. *European Journal of Operational Research*, 62:203–209.

Valmari, A. (1989). Stubborn sets for reduced state space generation. In Rozenberg, G., editor, *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets (APN 1989)*, volume 483 of *Lecture Notes in Computer Science*, pages 491–515. Springer-Verlag.

Veloso, M. M., editor (2007). *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*.

Walsh, T., editor (2011). *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*.

Wehrle, M. and Helmert, M. (2012). About partial order reduction in planning and computer aided verification. In McCluskey et al. (2012).

Wehrle, M. and Helmert, M. (2014). Efficient stubborn sets: Generalized algorithms and selection strategies. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS 2014)*. AAAI Press.

Wehrle, M., Helmert, M., Alkhazraji, Y., and Mattmüller, R. (2013). The relative pruning power of strong stubborn sets and expansion core. In Borrajo et al. (2013), pages 251–259.

Wilkins, D. (1988). *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann.

Wolfe, J. and Russell, S. J. (2011). Bounded intention planning. In Walsh (2011), pages 2039–2045.

Xu, Y., Chen, Y., Lu, Q., and Huang, R. (2011). Theory and algorithms for partial order based reduction in planning. *CoRR*, abs/1106.5427.

Yoshizumi, T., Miura, T., and Ishida, T. (2000). A* with partial expansion for large branching factor problems. In Kautz, H. and Porter, B., editors, *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI 2000)*, pages 923–929. AAAI Press.

Zahavi, U., Felner, A., Burch, N., and Holte, R. C. (2010). Predicting the performance of IDA* using conditional distributions. *Journal of Artificial Intelligence Research*, 37:41–83.

Zhou, R. and Hansen, E. A. (2006). Breadth-first heuristic search. *Artificial Intelligence*, 170(4–5):385–408.

Zilberstein, S., Koehler, J., and Koenig, S., editors (2004). *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS 2004)*. AAAI Press.

List of Figures