Translation-based approaches to Conformant Planning

Héctor Luis Palacios Verdes

TESI DOCTORAL UPF / 2009

Thesis supervisor

Prof. Dr. Héctor Geffner,
Department of Information and Communication Technologies

UNIVERSITAT POMPEU FABRA

To my father

Acknowledgements

Looking back, I am surprised and at the same time very grateful for all what I have received throughout these years. It has certainly shaped me as a person and has led me where I am now. All these years of PhD studies are full of such gifts.

I want first to thank my advisor Hector Geffner for all the hope he has put on me, before I thought I could do any research at all. He has always encouraged me to live intensively, even when I was thinking about doing something else. During more than decade of knowing him, he has helped me to see life and science in their full depth, and taught me how to appreciate the good scientific work that helps other researchers to build on it. He has enlightened me through his wide knowledge of Artificial Intelligence and his deep intuitions about where it should go and what is necessary to get there.

I also thank Adnan Darwiche, Jerome Lang, and Pierre Marquis for their lectures in 2004, which gave me tools that turned out to be essential in my PhD research, and for answering quickly all the questions I had about topics of their expertise. Thanks also to Blai Bonet, Adnan Darwiche, Joërg Hoffmann and Malte Helmert for insightful discussions, and for making their software available so I was able to use it in this work. I also thank the authors of the SAT solvers `zChaff`, `Siege` and `relsat`, enabling their use in other settings. I also appreciate insightful discussions with Dan Bryce, David E. Smith, Enrico Giunchiglia, Jinbo Huang, Jorge Baier, Jussi Rintanen, Menkes van den Briel, Patrik Haslum, and Tran Cao Son. Our work greatly improved thanks to the comments of the anonymous reviewers and editors of the articles published with contributions of this dissertation.

I thank Maria Esther Vidal for her invaluable help on tracking the writing of this dissertation this year, when I was immersed in the chaos of moving back to Venezuela. I thank Blai Bonet for the research collaboration, his friendship, his encouragement, and for facilitating my landing in our department in the *Simón Bolívar University* (USB), and the rest of the AI group at the USB, Carolina Martínez and Carolina Chang. Special thanks to Adriana Ampueda, Alex Albore, Anders Johnsonn, Blai Bonet, Carlos Linares, Dani Martí, Eduardo Izquierdo, Eduardo Madrid, Emil Keyder, Menkes van Den Briel, Miquel Ramírez, Vicenç Gómez, and Victor Dias for their help in proof reading this document.

I thank the *Embedded Reasoning Group* of the *Palo Alto Research Center* – PARC for the wonderful experience of being a summer intern. Thanks to Minh B. Do and Rong Zhou for the advise, to Johan de Kleer, Haitham Hindi, Lukas Kuhn. Also thanks to Menkes, Pew and Brynn for the wonderful chat and friendship. Thanks to Mali for her support. Thanks to David E. Smith and Jeremy Frank for arranging a visit and talk at *NASA Ames* while I was a PARC intern.

It was a pleasure to share doctoral studies and life with wonderful people like Vicenç, Andreas and Leticia, my first office mates, and with Dani, Enric, Gabriele, Laura, Oscar among others who are very close friends now.

I thank the rest of the Artificial Intelligence group at the UPF including both the faculty and the students. Thanks to Anders for his friendship, to Victor and Hubie for their example on being a computer science researcher, and to Alex, Miquel, Emil and Nir for sharing the glory and sadness of conferences deadlines and day-to-day research.

During the development of this dissertation I received support of my advisor's grants TIN2005-09312-C03-03 and TIN2006-15387-C03-03 from MEC/Spain. I was partially supported by a FPI fellowship from MEC/Spain and by a teaching assistant position at the *Department of Information and Communication Technologies* of the *Pompeu Fabra University* (UPF).

The years spent in Barcelona would not have been as wonderful without my Venezuelan friends, including Flavia, Haydee, Janzet & Alejandro and Lucas, Javier, Julio Cesar, Mónica, Marisol, Mimo & Nino, Nataly & Lao and their daughters, Rafael & Marialejandra, Ruben, Miguel & Andreina, Yensi & Jose Vicente. I met some of them in the *Altosf* theater group, who I also thank for helping me to *Work* better. Special thanks to its director Juan Carlos De Petre.

I thank my group and the people in the *Teresian Institution* that allow me to stay centered and rediscover what is important and why I am doing what I am doing. I thank Bea & Gonzalo, Carles, Carmen María, Cris, Eugenia, Gemma & Justo, Helen, Juan, Meritxel & Miguel, Miqui, Nani, Oscar, Ramon, Roger, Santi, Teo, and also Amparo, Ana Almuni and Anna de Guia from the *Passatge Center.* I also thank the *Christianism and Justice Center* for the many insightful courses I took with them, providing me, for example, with new elements to see the importance of PhD studies and my role in society.

I also thank other people I met in Barcelona for the wonderful friendship they offered me, like Aina, Marina and Joan Lluis. Ricardo & Iris at Venezuela were a real support even while I was in Barcelona. So was Yosmar. Thank to the Sampayo-Cortes family for their hospitality in hosting me during the last stage this work.

Thank also to my family, specially my mother Margarita and my brother Gabo, who also accomplished without complaints the endless errands that I asked him to do, even when he was on peaks of stress and lack of sleep because of his job and studies. Thanks to Sandro Faedi for such a lasting gift. Thanks to my father, whose memory has only increased after so many years of his death, when I was a boy wishing to save the world. I am working on it, still.

Last but not least, a big thank you to my wife, Neritza. Without her I would be a very different person today, and it would have been certainly much harder to finish a PhD. Still today, learning to love her and to receive her love makes me a better person. Special thanks to her also for helping me with the figures and epigraph of this dissertation.

I finish with a final silence of gratitude for my life.

Abstract

Conformant planning is the problem of finding a sequence of actions for achieving a goal in the presence of incomplete information in the initial state and in the state transitions. While few practical problems are purely conformant, the ability to find conformant plans is needed in planning with observations where conformant situations are an special case and where relaxations into conformant planning yield useful heuristics. In this dissertation, we tackle the conformant planning problem with deterministic actions by using translations. On the one hand, we propose a translation into propositional logic and two schemes for obtaining conformant plans for it: one based on boolean operations of projection and model counting, and the other based on projection and satisfiability. On the other hand, we define translations of the conformant planning problem into classical problems that are solved by a modern classical planners. We also analyze the formal properties of the translations and evaluate the performance of the resulting planners.

Resumen

La planificación conformante es el problema de encontrar una secuencia de acciones para lograr un objetivo en presencia de información incompleta sobre el estado inicial y en las transiciones entre estados. Aunque pocos problemas son de carácter puramente conformante, la posibilidad de encontrar planes conformantes es necesaria en planificación con observaciones, donde las situaciones conformantes son un caso particular, y donde las relajaciones a planificación conformante dan heurísticas útiles. En esta tesis atacamos el problema de la planificación conformante con acciones determinísticas mediante dos formulaciones basadas en traducciones. Por un lado, proponemos una traducción a lógica proposicional y dos esquemas para obtener planes conformantes a partir de estas, uno basado en operaciones booleanas de projección y conteo de modelos, y otro basado en projección y satisfacción proposicional. Por otro lado, introducimos traducciones que permiten transformar un problema de planificación conformante en un problema de planificación clásica que es luego resuelto usando planificadores clásicos. También analizamos las propiedades formales de las traducciones y evaluamos el rendimiento de los planificadores obtenidos.

Preface

Classical planning is the problem of finding a sequence of actions that achieves a goal, starting from a particular initial situation. A wide range of problems can be expressed in this form. Classical planning can be cast as a path finding problem in a graph whose nodes are the possible states and whose edges are the transitions that are possible with the actions available. State-of-the-art approaches to classical planning use heuristic search in the state graph or map the problem of finding an N-time-step plan into a propositional satisfiability problem (SAT).

The model underlying classical planning can be extended to take into account partial information, probabilities, cost, time, or resources. Conformant planning is planning over a model in which the goal is to be achieved from an uncertain initial situation using actions with non-deterministic effects. A conformant plan is a sequence of actions that achieves the goal for any possible initial state and any possible state transition. Conformant planning is computationally harder than classical planning, as even under polynomial restrictions on plan length, plan verification remains hard.

While few practical problems are purely conformant, the ability to find conformant plans is needed in planning with observations, where conformant situations are a special case and provide useful relaxations and heuristics. Techniques used for conformant planning have been used, indeed, for finding contingent plans in problems with sensing and for deriving finite-state controllers in problems where sensing is available (Albore, Palacios, and Geffner, 2009; Bonet, Palacios, and Geffner, 2009).

An example of a conformant planning problem is a robot required to clean up a square grid without knowing which cells are dirty. A conformant plan would involve visiting all the cells of the grid, cleaning any dirt that may be found in them. After such a plan the room will be clean with certainty. Another example involves a patient that after some tests, is known to suffer from one of a set of possible illnesses. If there is a sequence of treatments that would heal the patient without having to isolate the exact illness, such sequence will constitute a conformant plan for curing the patient. As a final example, consider a device made of a set of components that can be defective, preventing the device from working properly. Even if no information is available to determine which component is faulty, a plan that replaces all components will be a conformant plan for fixing the device.

Conformant planning has been addressed computationally as a path-finding problem in a graph whose nodes are sets of possible states (*belief state*) and whose edges are the transitions among belief resulting of applying actions. Most conformant planners use this approach together with an effective belief state representation and an informative heuristics for guiding the search. In this dissertation we propose an alternative approach based on translations.

In the first part of the dissertation, we translate the conformant problem into a *Conjunctive Normal Form* formula (CNF), whose models capture the possible N-time-step plans. Each such plan works for some possible initial state and state transitions. The conformant plans are the ones that work for all of them. For computing such plans we carry an DPLL-like search over the action variables, pruning the assignments than cannot lead to a conformant plan. This pruning is achieved by means of *model-counting* and *projection* operations that are made efficient by compiling the CNF formula into *Deterministic Decomposable Negational Normal Form* (d-DNNF Darwiche, 2001a). Another approach considered in this part of the dissertation uses the same d-DNNF compilation to obtain a new CNF formula whose models capture exactly the possible *conformant* plans, such that a conformant plan can be obtained calling a standard SAT *solver* once upon the new formula.

In the second part of the dissertation we introduce an alternative translation where conformant problems are mapped into classical ones that are solved by using an state-of-the-art *classical planner*. In the worst case this translation is exponential, but for a large collection of problems it can be shown to be polynomial and complete. The complexity of the complete translation is exponential in a *conformant width* parameter that for most conformant benchmarks turns out to be bounded and equal to one. The conformant planner T_0 –the best performing planner in the Conformant Track of the 2006 *International Planning Competition* (IPC-2006)– is based on a complete translation for problems of width equal to one, but is effective for other problems as well.

The results presented in the dissertation have been published in the following articles:

- Héctor Palacios, Blai Bonet, Adnan Darwiche, and Héctor Geffner. *Pruning conformant plans by counting models on compiled d-DNNF representations.* In *Proceedings of the 15th International Conference on Planning and Scheduling (ICAPS-05)*, pages 141–150. AAAI Press, 2005. [Chapter 3]

- Héctor Palacios and Héctor Geffner. *Mapping conformant planning to sat through compilation and projection.* In *Current Topics in Artificial Intelligence*, volume 4177, pages 311–320, Berlin, Germany, 2006. Springer Berlin / Heidelberg. Selected Papers from the *11th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2005).* [Chapter 4]

- Héctor Palacios and Héctor Geffner. *Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes).* In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-2006)*, pages 900–905. AAAI Press, 2006. [Chapter 5 and Section 8.2]

- Héctor Palacios and Héctor Geffner. *From conformant into classical planning: Efficient translations that may be complete too.* In *Proceedings of the 17th International Conference on Planning and Scheduling (ICAPS-07)*, pages 264–271. AAAI Press, 2007. [Chapters 6 and 7]

- Héctor Palacios and Héctor Geffner. *Compiling uncertainty away in conformant planning problems with bounded width.* In *Journal of Artificial Intelligence Research (JAIR)*, volume 35, pages 623–675, 2009. [Chapters 6 and 7]

The research done during this dissertation has received a number of distinctions. The work presented at *CAEPIA-2005* was a runner up for the *Best Research Article Award*, and the article presented at *ICAPS-07* received the *Best Student Paper Award*. In addition, the conformant planner T_0 was the winner winner of the Conformant track of the IPC-2006, and the runner up of Conformant track of the IPC-2008.

Contents

List of Figures

List of Tables

Part I

Background

Chapter 1

Classical Planning

Alegremente se perdieron en el [laberinto], al principio como si condescendieran a un juego y después no sin inquietud …

Cheerfully they lost themselves in [the labyrinth]-at first as though condescending to a game, but then not without some uneasiness …

Parable of the Palace.
Short story by Jorge Luis Borges

In this chapter we introduce the Classical Planning problem and review the main computational approaches to deal with it. We start with a motivation and define the classical planning problem formally. We illustrate the notation and basic concepts using a simple example and briefly discuss the complexity of the classical planning task. We then comment on two of the main approaches for classical planning: heuristic search and propositional satisfiability. Both are relevant to our formulation of conformant planning.

1.1 Introduction

A classical planning problem consists of an initial state, a set of goal states, and actions that change the states. The goal of classical planning is to find effective methods for obtaining a plan, *i.e.* a sequence of actions, that applied at the initial situation achieves a goal state.

Many different problems can be expressed in classical planning. For example, a logistics problem involving the pickup and delivery of packages can be modeled as follows. The initial situation describes the initial location of the packages, trucks, planes. The actions include loading and unloading a package from a train or a plane, and moving trucks and planes between locations and cities. The goal encodes the final desired position of the packages. A classical planner use such encoding of the problem to return a sequence of actions that finally deliver the packages to their desired destinations. In a similar way, robot navigation and puzzle problems can be modeled as well.

Classical planning has seen great advances in the last two decades (Blum and Furst, 1995; Kautz and Selman, 1996; Bonet and Geffner, 2001a). The most successful approach is to use heuristic search on the underlying state model, guided by a heuristic extracted from the classical problem. Another approach that has been successful is to try and find a plan of N time steps by creating a set of propositional formulas that contain such plans.

In the rest of the chapter, we provide a formal definition of classical planning, see an example and review some complexity results and state-of-the-art planners.

1.2 Model

Solving a classical planning problem involves selecting actions to achieve a goal from an fully known initial state.

The model underlying classical planning can be described by the tuple $\mathcal{S} = \langle S, s_0, S_G, A, f \rangle$, consisting of

- a finite and discrete state space S,
- an *initial state* $s_0 \in S$,
- a set $S_G \subseteq S$ of goal states,
- the actions $A(s) \subseteq A$ that are applicable in each $s \in S$,
- a *deterministic transition function* $s' = f(a, s)$ for $a \in A(s)$, and

A *classical plan* in this model is a sequence of actions $a_0, \ldots, a_n$ that generates a state sequence $s_0, s_1, \ldots, s_{n+1}$ such that a_i is applicable in the state s_i and results in a state $s_{i+1} = f(a_i, s_i)$, and s_{n+1} is a goal state.

In this work we assume that action cost are uniform; *i.e.* $c(a, s) = 1$. The cost of a plan is the sum of the action costs, that corresponds to the plan length, denoted $|\pi|$. A classical plan is *optimal* if it has minimum cost.

Classical planners accept a compact description of the above models as input for automatically producing a plan. Let us first consider a syntax for expressing a planning problems, and then provide a semantic in terms of state models.

1.3 Syntax

A classical planning problems P is expressed as a tuple of the form $P = \langle F, I, O, G \rangle$ where F stands for the fluent symbols in the problem, I is a set of literals over F defining the initial situation, O stands for a set of operators or actions, and G is a set of literals over F defining the goal. Every action $a \in O$ has a precondition $Pre(a)$ given by a set of literals, and a set of conditional effects $C \rightarrow L$ where C is a set of literals and L is a single literal. This definition corresponds to the STRIPS language extended with conditional effects and negations (Fikes and Nilsson, 1971; Nebel, 2000).

We refer to the conditional effects $C \rightarrow L$ of an action a as the *rules* associated with a, and sometimes write them as $a : C \rightarrow L$. Sometimes we group rules $a : C \rightarrow L$ for different literals L into one rule $a : C \rightarrow E$, and write $C \rightarrow L$ as $true \rightarrow L$ when C is empty. When all the effects of an action a are of the form $true \rightarrow L$, we will say that the effect of a is simply the set of literals L. Finally, for a literal L, $\neg L$ denotes the complement of L.

1.4 Semantics

Given a problem $P = \langle F, I, O, G \rangle$, the corresponding state model $\mathcal{S}(P) = \langle S, s_0, S_G, A, f \rangle$, consists of the following.

- The set of states S, where a state $s \in S$ is a set of positive literals in F so that a positive literal p is in s if and only if the fluent p is true in the state s.[1]
- The initial state s_0 is I.
- The set of goal states S_G, such that $G \subseteq s_g$ for each $s_g \in S_G$.
- The actions a applicable in s, $A(s)$, are the ones in O such that $Pre(a) \subseteq s$,
- The state-transition function $f(a, s)$ maps the action a applied to the state s into the successor state s_a. A rule $a : C \rightarrow L$ is said to *apply* in an state s if all the positive literals of C are in s, and all the negative literals of C are not in s. The state s_a contains the same atoms as s, except that for all rules $a : C \rightarrow L$ that *applies* in s, then L is in s_a if L is a positive literal, and L is not in s_a if L is a negative literal.

Following the state model, an action sequence $\pi = \{a_0, a_1, \ldots, a_n\}$ is a *classical plan* for P if π is executable in the initial state s_0 and achieves a goal state $s_g \in S_G$. An action a_i is executable if the preconditions of a_i are true in s_i. The state that results from executing action a_i in s_i is s_{i+1}. π is a classical plan if all goal literals are true in s_{n+1}.

1.5 Example

Consider the problem of a robot in a $N \times N$ grid, with $N = 8$, that has to go from I to G (Fig. 1.1). This problem can be modeled as a classical planning problems $P = \langle F, I, O, G \rangle$, with:

- Fluents F: `at-x`(p_i), `at-y`(p_j), for any i, j such that $1 \leq i, j \leq N$.
- Initial situation I: `at-x`(p_2), `at-y`(p_2).
- Actions O:

[1]This corresponds to the so called *closed world assumption*, as any atom not mentioned in an state is assume to be false.

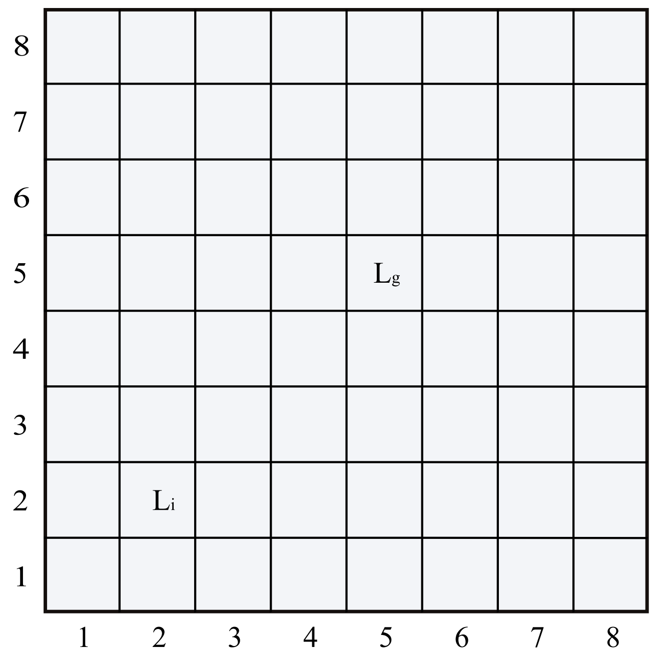

Figure 1.1: Navigation problem in a simple square grid. The robot starts at *Li* and should go to *Lg*.

- `up`(p_j) for any $j < N$.
  Precondition: `at-y`(p_j). Effect: `at-y`(p_{j+1}), $\neg$`at-y`(p_j).
- `down`(p_j) for any $j > 1$.
  Precondition: `at-y`(p_j). Effect: `at-y`(p_{j-1}), $\neg$`at-y`(p_j).
- `left`(p_i) for any $i > 1$.
  Precondition: `at-x`(p_i). Effect: `at-x`(p_{i-1}), $\neg$`at-x`(p_i).
- `right`(p_i) for any $i < N$.
  Precondition: `at-x`(p_i). Effect: `at-x`(p_{i+1}), $\neg$`at-x`(p_i).

• Goal G: `at-x`(p_5), `at-y`(p_5).

A solution for this problem is the sequence.

{ `up`(p_2), `up`(p_3), `up`(p_4), `right`(p_2), `right`(p_3), `right`(p_4) }

Note that the same problem could also be encoded in a slightly different way. Instead of 28 actions with one conditional effect each, we could use 4 actions with 7 conditional effects each, as follows:

• Actions O:

- `up`(). Precondition: None.
  Effect: `at-y`(p_j) $\rightarrow$ `at-y`(p_{j+1}), $\neg$`at-y`(p_j), for each $j < N$.
- `down`(). Precondition: None.
  Effect: `at-y`(p_j) $\rightarrow$ `at-y`(p_{j-1}), $\neg$`at-y`(p_j), for each $j > 1$.
- `left`(). Precondition: None.
  Effect: `at-x`(p_i) $\rightarrow$ `at-x`(p_{i-1}), $\neg$`at-x`(p_i), for each $i > 1$.

– `right`(). Precondition: None.
Effect: `at-x`(p_i) $\rightarrow$ `at-x`(p_{i+1}), $\neg$`at-x`(p_i), for each $i < N$.

In this case, a solution is:

{ `up`, `up`, `up`, `right`, `right`, `right` }

The second problem allows the execution of actions in states where the first does not, making it a different problem. However, the two given solutions behave equally when executed. We will go back to the second formulation when talking about conformant planning, where conditional effects are needed for modeling problems.

1.6 The Planning Domain Definition Language – PDDL

Most work in classical planning has been done for problems expressed in the STRIPS language (Fikes and Nilsson, 1971). More recently, however, the *Planning Domain Definition Language*, PDDL (McDermott et al., 1998), has become the standard *de facto*, mainly because of its use in the *International Planning Competitions* (IPC) (McDermott, 2000; Bacchus, 2001; Fox and Long, 2003; Hoffmann and Edelkamp, 2005; Gerevini et al., 2009; Helmert et al., 2008). PDDL allows to specify STRIPS problems as well as extensions to the STRIPS language. Figures 1.2 and 1.3 show the navigation examples expressed in PDDL.

State-of-the-art planners ground the actions of a PDDL problem before attempting to solve it, *i.e.* they transform predicates, objects, and constants into a propositional representation, like our definition of Section 1.3.

1.7 Complexity

Given a classical problem P, the decision problem `PlanEx` is defined by the question *Is there a plan for P?*. For a constant k, the decision problem `PlanLen` is defined by the question *Is there a plan* π *for P with* $|\pi| \leq k$*?* The first problem is related to the complexity of obtaining any plan reaching a goal state, while the second is related to obtaining an optimal plan. Both problems are PSPACE-complete (Bylander, 1994), *i.e.* the class of problems that can be solved in polynomial space with no restrictions on running time. For constant k, the `PlanLen` problem is NP-complete.

1.8 Classical Planning as Heuristic Search

In this and the following section we review the state-of-the-art approaches to classical planning that have been developed in recent years: *heuristic search* and *propositional satisfiability*. It should be noted that all the winning planning systems from the IPC in the *satisficing* track[2] have been based on planning as heuristic search. Planning as

[2]The track where planners are intended to obtain a solution, even though it is not optimal

```
(define (domain square)
  (:requirements :typing)
  (:types pos)
  (:constants p1 p2 p3 p4 p5 p6 p7 p8 - pos)
  (:predicates (x ?p - pos) (y ?p - pos))
  (:action up-p1
     :precondition (y p1)
     :effect (and (not (y p1)) (y p2))
  )
  ...
  (:action up-p7
     :precondition (y p7)
     :effect (and (not (y p7)) (y p8))
  )
  (:action down-p2
     :precondition (y p2)
     :effect (and (not (y p2)) (y p1))
  )
  ...
  (:action left-p2
     :precondition (x p2)
     :effect (and (not (x p2)) (x p1))
  )
  ...
  (:action right-p1
     :precondition (x p1)
     :effect (and (not (x p1)) (x p2))
  )
  ...
)
(define (problem square-8)
  (:domain square)
  (:init (and (x p2) (y p2)))
  (:goal (and (x p5) (y p5)))
)
```

Figure 1.2: PDDL encoding of a navigation problem in a simple square grid. The agent in a 8×8 grid starts at $(2, 2)$ and must get to $(5, 5)$.

```
(define (domain square)
  (:requirements :typing :conditional-effects)
  (:types pos)
  (:constants p1 p2 p3 p4 p5 p6 p7 p8 - pos)
  (:predicates (x ?p - pos) (y ?p - pos))
  (:action up
     :effect (and
     (when (y p1) (and (not (y p1)) (y p2)))
     ...
     (when (y p7) (and (not (y p7)) (y p8)))
  ))
  (:action down
     :effect (and
     (when (y p2) (and (not (y p2)) (y p1)))
     ....
     (when (y p8) (and (not (y p8)) (y p7)))
  ))
  (:action left
     :effect (and
     (when (x p2) (and (not (x p2)) (x p1)))
     ...
     (when (x p8) (and (not (x p8)) (x p7)))
  ))
  (:action right
     :effect (and
     (when (x p1) (and (not (x p1)) (x p2)))
     (when (x p2) (and (not (x p2)) (x p3)))
     ....
     (when (x p7) (and (not (x p7)) (x p8)))
  ))
)
(define (problem square-8)
  (:domain square)
  (:init (and (x p2) (y p2)))
  (:goal (and (x p5) (y p5)))
)
```

Figure 1.3: PDDL encoding of a navigation problem in a simple square grid. The agent in a 8×8 grid starts at $(2, 2)$ and must get to $(5, 5)$. In contrast to Fig. 1.2 on the preceding page, this encoding uses conditional effects.

propositional satisfiability is a competitive approach for optimal classical planning, even though such approaches minimize the parallel length, *makespan*, instead of the number of actions in the plan.

Given a classical planning task P and its corresponding state space $\mathcal{S}(P)$, a plan can be obtain by finding a path from the initial state into a goal state. In principle, any standard graph-search algorithm (Cormen et al., 1990) can be used, but blind search turn out to be ineffective as the size of the state space is exponential on the number of fluents of the problem P. Instead, heuristic search algorithms have been found to be very effective when appropriate heuristics are used. Many successful heuristics are based on relaxations of the original problems that are easier to solve (Pearl, 1983). Useful relaxations of a planning problem are, for example, to ignore the negative effects of actions or to ignore some of the preconditions. Assuming that the cost of achieving a set of literals is equal to the sum of the costs for achieving each literal independently. Planning as heuristic search was introduced independently by McDermott (1996) and Bonet, Loerincs, and Geffner (1997). Their work uses relaxations and assumptions to obtain informative heuristics, that combined with suitable algorithms, lead to very effective planners.

Some very successful classical planners are HSP (Bonet and Geffner, 2001a, 1999), FF (Hoffmann and Nebel, 2001), *FastDownward* (Helmert, 2006), SGPlan (Chen et al., 2006; Wah and Chen, 2006) and LAMA (Richter et al., 2008), all of them based on heuristic search.

The heuristic-search based planner HSP uses a heuristic extracted from the planning problem by *ignoring the negative effects* and assuming that achieving a set of literals is equivalent to achieving each one of them independently. HSP uses a *Weighted-*A* search algorithm that biases the selection of nodes. This biases the selection of nodes to the heuristic criterion, getting in many cases solutions in less time even at a reasonable detriment of their quality.

Another successful classical planner is FF, that takes ideas from HSP, but uses as the heuristic the length of a *relaxed plan* obtained from a similar problem without negative effects. FF first tries to reach the goal using an incomplete *greedy search*, and switches back to a complete search algorithm when the greedy search fails. Such greedy search is incomplete but quite effective, as it continues as far as it can improve the heuristic value by considering actions that are part of the relaxed plan.

Planning as Heuristic Search is sound and complete by construction, as far as the used search algorithm is complete, given that the state space contains exactly all the possible plans as paths from the initial state to any goal state.

1.9 Classical Planning as Propositional Satisfiability

The SAT-based approach to classical planning (Kautz and Selman, 1992, 1996), maps the problem of finding a plan of N time steps into the problem of finding a model of a suitable propositional formula. This way, a planning algorithm may proceed by generating propositional theories for a problem P and an horizon N, increasing N until a plan is found. Since the value of N is unknown, the algorithm start with $N = 0$.

The SAT approach has become feasible after the great advances in the area of propositional satisfiability. SAT solvers accept propositional formulas in *Conjunctive Normal Form* (CNF), represented as a list of clauses. State-of-the-art sat solvers are able to deal thousand of variables and hundreds of thousand of clauses. The basic algorithm underlying state-of-the-art SAT solvers, called *DPLL*, is based on very simple ideas (Davis et al., 1962).

A variable is selected and is assigned to true or false. After each variable assignation, a limited but efficient form of reasoning called *unit propagation* is run on the problem, allowing to set other variables to true or false. Modern SAT solvers also learn new clauses when a contradiction is found, and use effective techniques for variable and value selection. Modern SAT solvers are being used for both industrial application and other research problems, like classical planning.

The propositional formula $T_N(P)$ encodes the plans of N time steps for classical problem P, involves propositional variables x_i, where i is a temporal index, and i is in $0, \ldots, N$ for fluents of problem and i is in $0, \ldots, N-1$ for actions. For a formula B, B_i refers to the formula obtained by replacing each variable x in B by its time-stamped counterpart x_i. For the encoding $T_N(P)$ of a classical planning problem $P = \langle F, I, O, G \rangle$ we extend the encoding of Kautz and Selman (1996) for supporting problems with conditional effects. Given a horizon N, the CNF theory $T_N(P)$ is defined as follows.

Definition 1.1. *The propositional theory $T_N(P)$ for a classical planning problem $P = \langle F, I, O, G \rangle$ and an horizon N is given by the following set of clauses:*

1. **Init:** a literal L_0 for each literal $L \in I$.
2. **Goal:** a literal L_N for each literal $L \in G$.
3. **Actions:** For $i = 0, 1, \ldots, N-1$ and $a \in O$:

$$\begin{array}{rcll} a_i & \supset & Pre(a)_i & \text{(preconditions)} \\ C_i \wedge a_i & \supset & E_{i+1} & \text{(for each rule } a : C \rightarrow E\text{)} \end{array}$$

4. **Frame:** for $i = 0, 1, \ldots, N-1$, and each fluent literal L

$$L_i \wedge \bigwedge_{a:C \rightarrow \neg L} \neg[C_i \wedge a_i] \quad \supset \quad L_{i+1}$$

   where the conjunction ranges over the rules $a : C \rightarrow \neg L$
5. **Exclusion:** $\neg a_i \vee \neg a'_i$ for $i = 0, \ldots, N-1$ if a and a' are incompatible

The meaning of the clauses in *Init*, *Goal*, and *Actions* is straightforward. *Frame* expresses the persistence of fluents in the absence of actions that may affect them. Finally, *Exclusion* forbids the concurrent execution of actions that are deemed incompatible.

For obtaining a *sequence* of actions achieving the goal, pair of actions cannot be executed at the same time, and should be regard as incompatible by using the *exclusions* clauses. However, the SATPLAN approach to classical planning handle parallelism naturally if two actions are deemed compatible when the sets of boolean variables in their effects are disjoint.[3] Such parallelism often lead to more compact formulas for obtaining plans, allowing to scale up to larger planning problems.

Thus, a *parallel plan* is a sequence of actions sets that maps the initial state into a goal state, where the set of actions is applicable in a state when the preconditions of all such actions holds, and the resulting state of applying the set of actions A in an state s has the same literals of s except for literals $\neg L$, if C is satisfied by s and there is rule $a : C \rightarrow L$, for some a in A. Normal sequence of actions that achieve the goal are called *serial plans*.

For a given N, if the propositional theory $T_N(P)$ is satisfiable, the sequence of actions sets that are true in the model encodes a parallel plan. Vice versa, for any parallel plan of P with makespan N, there is exactly one model of $T_N(P)$ encoding both the actions executed and their effects, starting at the initial state until a goal state.

A parallel classical plan π for P is *optimal* if there is no other plan π' such that $|\pi'| < |\pi|$. Optimal parallel plans are said to have *minimal makespan*, as there is no other parallel plan achieving the goal in less time steps. An optimal parallel plan can be found by setting the horizon N to 0, and increasing it one by one, until a plan is found. Other strategies for obtaining optimal plans are possible (Rintanen et al., 2005).

1.10 Syntactic variants

Many of the problems we will be dealing with have a set of clauses instead of a set of literals, a goal state being a state that satisfies such clauses. Those problems can be converted into pure STRIPS in a standard way. Each goal clause $C : L_1 \vee \cdots \vee L_m$ is modeled by a new goal atom G_C, and a new action that can be executed once is added with rules $L_i \rightarrow G_C$, $i = 1, \ldots, m$. An alternative way to represent such CNF goals is by converting them into DNF, discarding unreachable terms, and having an action *End* map each terms into a dummy goal L_G. For the first approach, planners that use additive heuristics (Bonet and Geffner, 2001a), as HSP or FF, may fail to realize that there are some combinations of literals that are not reachable and get lost while searching. On the other hand, using the second approach may lead to an exponential number of DNF terms, making the problem in practice unsolvable.

Other features of PDDL cannot be handle by compiling them away into pure strips. The language for classical planning defined in Section 1.3 includes conditional effects, in contrast with the STRIPS language where the effects are unconditionally applied (Fikes and Nilsson, 1971). Indeed, some modern classical planners does not support conditional effects or provide limited support for them. Conditional effects can be compiled away into pure STRIPS but at the cost of causing an exponential blow up in size of the problem or a polynomial increase in the plan length (Rintanen, 2003), that may harm the heuristics used by state-of-the-art classical planners. In general, classical planners perform better if they do not explicitly compile conditional effects into STRIPS, but represent them implicitly and, in the case of heuristic-search based algorithms, extend their heuristics accordingly.

[3]Other definitions of compatible actions are possible. We choose this one for simplicity on its treatment in this dissertation and in the implementation of our conformant planning algorithms.

Chapter 2

Conformant Planning

De noche iremos, de noche,
que para encontrar la fuente,
sólo la sed nos alumbra.

By night, we hasten in darkness,
to search for living water,
only our thirst leads us onwards

Taizé Community's song. Based on a poem
by John of the Cross[1]

In this chapter we define the conformant planning problem and review some of the previous approaches. We start by motivating the problem and providing some examples. Then, we define formally *conformant planning*, a planning problem that is similar to classical planning, except for the uncertainty in the initial state and action effects. In the rest of the chapter we discuss the computational complexity of the problem and review current approaches to conformant planning.

2.1 Introduction

A conformant planning problem is like a classical planning problem but the initial situation is not fully known, and may have non-deterministic effects (Goldman and Boddy, 1996; Smith and Weld, 1998). Since there are no observations in a conformant planning problem, conformant plans are sequence of actions like in classical planning. This plan must ensure that the goal is achieved with certainty regardless of the actual initial states and possible transitions.

While few practical problems are purely conformant, the ability to find conformant plans is needed in planning with sensing, that extends the conformant case by allowing observations. Indeed, relaxations of planning with sensing into conformant

[1]From `http://www.taize.fr`. ©Ateliers & Presses de Taizé, Communauté de Taizé, 71250 Taizé, France.

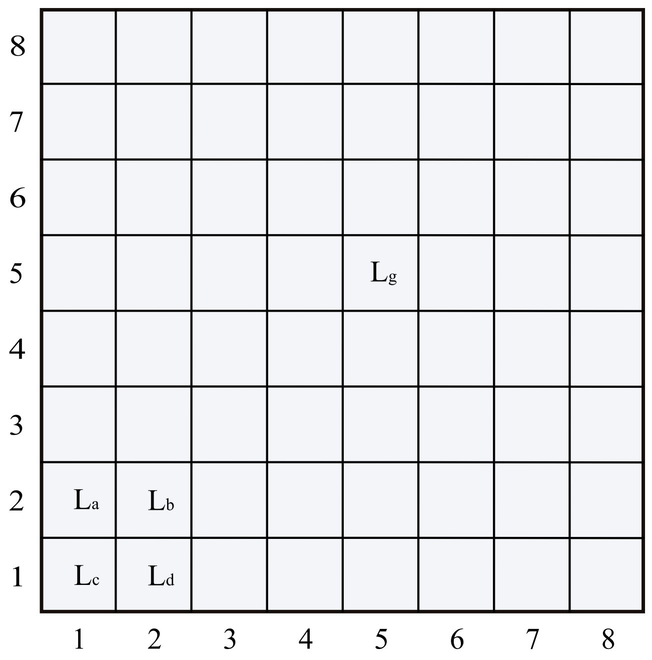

Figure 2.1: Conformant Planning problem: Robot in a room with incomplete information about the initial position. The robot starts at *La*, *Lb*, *Lc*, or *Ld*, and should go to *Lg*.

planning yield useful heuristics (Hoffmann and Brafman, 2005; Albore et al., 2009). In general, the research in conformant planning is relevant to any form of planning involving reasoning about actions executed on belief states.

Some applications of conformant planning are automatic web service composition (McDermott, 2007; Pistore et al., 2004; Bertoli et al., 2006; Hoffmann et al., 2007, 2009) and bioinformatics (Bryce and Kim, 2007).

2.2 Examples

As an illustration, consider a variation of the classical planning problem presented in the previous chapter (Fig. 1.3 on page 9). In the problem depicted in Figure 2.1, the initial situation is that the agent could be in any of the positions $(1, 1)$, $(1, 2)$, $(2, 1)$ or $(2, 2)$. In the classical problem illustrated in Fig. 1.3 on page 9, the initial situation is that the agent is in position $(2, 2)$ with certainty.

First, we observe that the solution to a conformant planning problem may be totally different from the solution to a classical planning problem. The solution to the classical planning example in Fig. 1.3 on page 9

$$\{\ \texttt{up}, \texttt{up}, \texttt{up}, \texttt{right}, \texttt{right}, \texttt{right}\ \}$$

is not a solution to the conformant planning example in Fig. 2.1. After executing the classical plan in Fig. 2.1, the agent may end up in positions $(4, 4)$, $(4, 5)$, $(5, 4)$ or $(5, 5)$, and thus not achieve the goal that requires the agent to be in position $(5, 5)$ with certainty. In contrast, the sequence of actions

$$\{\ \texttt{down}, \texttt{left},\ \texttt{up}, \texttt{up}, \texttt{up}, \texttt{up},\ \texttt{right}, \texttt{right}, \texttt{right}, \texttt{right}\ \} \tag{2.1}$$

is a conformant plan, as the first two actions cause the agent to be in position $(1,1)$ with certainty, and then moving to the goal, position $(5,5)$, where the agent ends with certainty.

This example may suggest that the heuristics used in classical planning may be difficult to adapt to the conformant case. Heuristics based on reducing the number of possible states may be useful but can also be misleading. For example, if the goal were at $(4,4)$, $(4,5)$, $(5,4)$ or $(5,5)$ then it would be a good idea to go directly to the goal.

This example could be formulated as a conformant problem $P = \langle F, I, O, G\rangle$ with

- Fluents F: `at-x`(p_i), `at-y`(p_j), for any i, j such that $1 \leq i, j \leq N$.
- Initial situation I: `oneof(at-x`(p_1), `at-x`(p_2)`)`, `oneof(at-y`(p_1), `at-y`(p_2)`)`
- Actions O:
  - `up`(). Precondition: None.
    Effect: `at-y`(p_j) $\rightarrow$ `at-y`(p_{j+1}), $\neg$`at-y`(p_j), $j < N$.
  - `down`(). Precondition: None.
    Effect: `at-y`(p_j) $\rightarrow$ `at-y`(p_{j-1}), $\neg$`at-y`(p_j), $j > 1$.
  - `left`(). Precondition: None.
    Effect: `at-x`(p_i) $\rightarrow$ `at-x`(p_{i-1}), $\neg$`at-x`(p_i), $i > 1$.
  - `right`(). Precondition: None.
    Effect: `at-x`(p_i) $\rightarrow$ `at-x`(p_{i+1}), $\neg$`at-x`(p_i), $i < N$.
- Goal G: `at-x`(p_5), `at-y`(p_5)

where the expressions `oneof(at-x`(p_1), `at-x`(p_2)`)` and `oneof(at-y`(p_1), `at-y`(p_2)`)` means that agent is at one of two coordinates along each dimension. We can verify that the sequence of actions (2.1) solves the conformant problem P, as it achieves the literals `at-x`(p_5) and `at-y`(p_5) for all the possible initial states described in I, given that the four actions are deterministic. The encoding of these problems in PDDL, a language accepted by modern conformant planners, is shown in Fig. 2.2.

A more challenging conformant planning problem is the following. A robot is initially at a certain position `I` in a 8×8 grid and the location of an object is unknown. The goal is that the object ends up in a position `T`, the trash. The robot can move around the grid, pick up any object at the current position, and release any object that it is holding. The robot can only pick up one object at a time. Also, when the robot executes the `pick-up` action, any object in its location or in the eight neighbor cells becomes held. Fig. 2.3 illustrates a plan obtained by T_0, a conformant planner we introduce in Chapter 7, where the numbers in circles indicate the order and place of the pick-up actions.[2]

Following most work on conformant planning until now, we assume in this dissertation that actions are deterministic. For an exception see the work of Cimatti et al. (2004).[3]

[2] A PDDL encoding of this problem can be found in appendix C.5 on page 177.

[3] In Section 8.3 on page 127, however, we discuss a limited extension of one of our algorithms to actions with non-deterministic effects.

```
(define (domain square-center)
  (:requirements :typing :conditional-effects)
  (:types pos)
  (:constants p1 p2 p3 p4 p5 p6 p7 p8 - pos)
  (:predicates (x ?p - pos) (y ?p - pos))
  (:action right
     :effect (and
     (when (x p1) (and (not (x p1)) (x p2)))
     (when (x p2) (and (not (x p2)) (x p3)))
     ....
     (when (x p7) (and (not (x p7)) (x p8)))
  ))
  (:action left
     :effect (and
     (when (x p2) (and (not (x p2)) (x p1)))
     ...
     (when (x p8) (and (not (x p8)) (x p7)))
  ))
  (:action down
     :effect (and
     (when (y p1) (and (not (y p1)) (y p2)))
     ...
     (when (y p7) (and (not (y p7)) (y p8)))
  ))
  (:action up
     :effect (and
     (when (y p2) (and (not (y p2)) (y p1)))
     ....
     (when (y p8) (and (not (y p8)) (y p7)))
  ))
)
(define (problem square-center-8)
 (:domain square-center)
 (:init (and
   (oneof (x p1) (x p2) (x p3) (x p4) (x p5) (x p6) (x p7) (x p8))
   (oneof (y p1) (y p2) (y p3) (y p4) (y p5) (y p6) (y p7) (y p8))
 ))
 (:goal (and (x p5) (y p5)))
)
```

Figure 2.2: PDDL for a conformant planning problem. A robot in a 8×8 grid starts at an unknown position $(1,1)$, $(1,2)$, $(2,1)$ or $(2,2)$. The goal is to get to $(5,5)$.

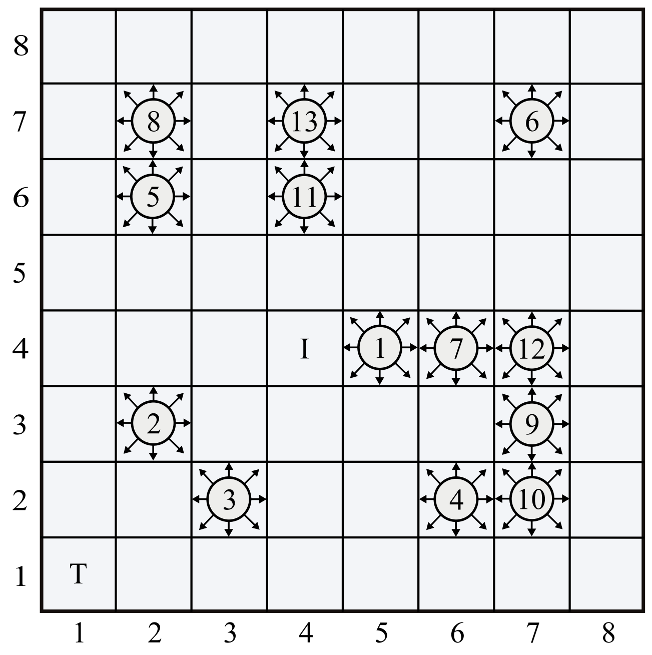

Figure 2.3: A solution to the Look-and-Grab conformant problem. A object with unknown location in a 8×8 grid must be collected by a robot whose gripper can hold one object at a time. The robot has an action that picks up the objects that are sufficiently close, if any, and after each pick-up the agent must dump the collected object into the trash before continuing. The numbers in circles indicates the order and location of the pick-up actions.

2.3 Model

Just like classical planning, solving a conformant planning problem involves selecting actions to achieve a goal state. However, unlike classical planning, such a plan should do so from each possible initial state.

The model underlying conformant planning can thus be described as a state space defined as a tuple $\mathcal{S} = \langle S, S_0, S_G, A, f \rangle$, consisting of

- a finite and discrete state space S,
- a set of *possible* initial states $S_0 \subseteq S$,
- a set $S_G \subseteq S$ of goal states,
- the actions $A(s) \subseteq A$ that are applicable in each $s \in S$,
- a *nondeterministic* transition function $S' = f(a, s)$ for each $a \in A(s)$, and

A *conformant plan* is a sequence of actions $a_0, \ldots, a_n$ that for any possible initial state s_0, and for any possible transition of the sequence of actions, generates a state sequence $s_0, s_1, \ldots, s_{n+1}$ such that a_i is applicable in state s_i and results in a state $s_{i+1} \in f(a_i, s_i)$, and s_{n+1} is a goal state.

In this work we assume that action cost are uniform; *i.e.* $c(a, s) = 1$. The cost of an action sequence π is the sum of the action costs, that corresponds to the plan length denoted $|\pi|$. An action sequence π is an *optimal* conformant plan for P if the action sequence π is a conformant plan, and there is no other plan π' such that $|\pi'| < |\pi|$.

As for classical planners, conformant planners accept a compact description of the above models as an input for producing a plan.

2.4 Syntax

Conformant planning problems P are expressed as tuples of the form $P = \langle F, I, O, G \rangle$ where F stands for the fluent symbols in the problem, I is a set of clauses over F defining the initial situation, O stands for a set of (ground) operators or actions, and G is a set of literals over F defining the goal.[4] Every action a has a precondition $Pre(a)$ given by a set of fluent literals, and a set of conditional effects $C \to L$ where C is a set of fluent literals and L is a single fluent literal.

All actions are assumed to be *deterministic* and hence all uncertainty lies in the initial situation. Hence, the language for conformant planning problems excluding the uncertainty in the initial situation, is equivalent to the STRIPS language for classical planning, presented in Section 1.3 on page 4, extended with conditional effects and negation. Moreover, if there is no uncertainty in the initial situation, as when I consist only of unit clauses, P is equivalent to a classical planning problem.

As we did for classical planning, we use $C \to L$ to refer to the conditional effects of an action a as the *rules* associated with a, and sometimes write them as $a : C \to L$.

[4]All planning systems presented in this document support clauses in the Goal by means of different transformations (see Section 1.10). For simplicity in presentation, unless stated otherwise, the goal is considered to be a set of literals.

When convenient, we also join several effects associated with the same action and condition as in $a : C \rightarrow L \wedge L'$ and write $C \rightarrow L$ as $true \rightarrow L$ when C is empty. Finally, for a literal L, $\neg L$ denotes the complement of L.

It is convenient to allow expressions $oneof(l_1, \ldots, l_n)$ in the initial situation, that are interpreted as a clause $X_1 \vee \cdots \vee X_n$ and a set of binary clauses $\neg X_i \vee \neg X_j$ for each $i \neq j$, encoding that exactly one of the l_i literals is true in the initial situation.

2.5 Semantics

Now we show how the syntax defined in the previous section can be interpreted to encode an state model. Given a conformant problem $P = \langle F, I, O, G \rangle$, the state model $\mathcal{S}(P)$ is obtained in the following way.

- The states s of the state space S are set of literals that represents truth-assignment over the fluents F in P, *i.e.* for every fluent L in F either L or $\neg L$ must belong to s.[5]
- The set of possible initial states S_0 are the states of S that satisfy the clauses in I.
- The goal states $s_g \in S_G$ are those such that $G \subseteq s_g$.
- The actions a applicable in s, $A(s)$, are the ones in O such that $Prec(a) \subseteq s$.
- The *deterministic* state-transition function $f(a, s)$ that maps the action a applied to the state s into the successor state s_a. The state s_a contains the same literals as s, except that s_a contains the literal L if $C \subseteq s$ and there is a rule $a : C \rightarrow L$.

Given that we restrict the syntax to deterministic actions, a conformant plan is a sequence of actions $a_0, \ldots, a_n$ that maps any possible initial state s_0 into a goal state, using the deterministic transition function.

Let us write $I(s)$ to refer to the set of literals that are true in a state s (*i.e.*, $l \in I(s)$ iff l is true in s), and P/s to refer to the *classical planning problem* $P/s = \langle F, I(s), O, G \rangle$ which is like the conformant problem P except for the initial state that is fixed to s. Observe that an action sequence π is a *conformant plan* for P iff π is a classical plan for P/s, for every possible initial state s of P.

Throughout this document we assume that I is logically consistent, so that the set of possible initial states is not empty, and that P itself is *consistent*, in the sense that the effects triggered by an action a in a reachable state s must be logically consistent, in the sense that no pairs of effects cancel each other out.

Conformant planning semantics can also be defined in terms of a state-model whose nodes are sets of possible states (*belief state*) and the plans are paths starting at the initial belief state, and ending in a state where the goal is satisfied with certainty (Bonet and Geffner, 2000). Even though both semantics are equivalent, in this

[5]For conformant planning we do not follow a convention that is common in planning of assuming that fluents not mentioned in I are false in the initial situation. As a result, if we want a fluent p to be false in the initial situation, we must explicitly add the literal $\neg p$ to I.

dissertation we choose to take advantage of the assumption of action determinism to a simpler semantic. This simplifies the formal treatment of our results as they rely on algorithms for classical planning.

2.6 Complexity

Given a conformant problem P, the decision problem `ConfPlanEx` is defined by the question *is there a plan for P?*. `ConfPlanEx` is PSPACE-complete for a language similar to propositional STRIPS with arbitrary preconditions (Haslum and Jonsson, 1999; Rintanen, 2004b; Littman et al., 1998). In general, however, conformant planning is computationally harder than classical planning as plan verification remains hard even under polynomial restrictions on plan length. Indeed, while determining the existence of a classical plan with length at most k is NP-complete if k is assumed to be polynomial in the size of the problem (Kautz and Selman, 1996), but under the same conditions conformant plan existence is Σ_2^P-complete (Turner, 2002). The class Σ_2^P stands for NP^{NP}; *i.e.* that the problem is in NP only if we allow the use of an NP oracle (Papadimitriou, 1994).

Verifying a classical plan of polynomial length is polynomial in time as it is enough to apply the actions starting from the initial state. In contrast, a polynomial length sequence of actions is a conformant plan if it is a classical plan for a possible exponential number of initial states. Indeed, conformant planning verification of polynomially-length plans is NP-complete by itself (Turner, 2002).

2.7 Conformant Planning as Heuristic Search in Belief Space

The most common approach to conformant planning is based on the *belief state* formulation (Bonet and Geffner, 2000). A belief state b is the non-empty set of states that are deemed possible in a given situation, and every action a executable in b, maps b into a new belief state b_a

$$b_a = \{s' \mid \text{ such that } s' = f(a,s) \text{ for some } s \in b\}$$

where $f(a,s)$ is the state transition function that maps an action a and a state s into a new state s'.

A conformant planning task can be solved as a path-finding problem in a graph where the nodes are belief states b, the source node b_0 is the belief state corresponding to the initial situation, and target belief states b_G are those where all the goals are true. A formula is true in a belief state b if it is true in every state s in b, and an action a is executable in a belief state b if its preconditions are true in every state s in b.

This formulation, which underlies most current conformant planners (Hoffmann and Brafman, 2006; Bryce et al., 2006; Cimatti and Roveri, 2000; Cimatti et al., 2004; Tran et al., 2009), must address two problems: the problem of representing beliefs in a compact way, and the problem of obtaining effective heuristics over beliefs. The first problem has been approached through logical representations that make use of SAT or *Ordered Binary Decision Diagrams* (OBDD) technology (Bryant, 1992)[6], that while intractable in the worst case, scale up better than fully enumerated state representations. The second problem, on the other hand, has been more complex, with heuristics for searching in belief space not being as successful so far as the heuristics developed for classical planning (Bonet and Geffner, 2001a; Hoffmann and Nebel, 2001), in the sense that different heuristics tend to perform very well in some family of problems but not in others.

Note that heuristic search in belief space uses a *directional* branching scheme that searches for plans by applying actions either forward or backwards. In classical planning, *non-directional* branching has also been useful, for example in SATPLAN (Kautz and Selman, 1996; Hoffmann and Geffner, 2003).

2.8 Belief Space Representation and Heuristics

There are two issues that are crucial for belief space planners to scale up: one is the heuristic, and the other is the belief space representation and update. In this section we review the state of the art in conformant planning based on search over belief state, and look at different approaches to these two critical issues.

The first planner to use explicit search in belief space is GPT (Bonet and Geffner, 2000), where the search for a goal belief state from a given initial belief state is carried out by means of the A* algorithm with a *heuristic* function obtained from a suitable relaxation of the problem. This relaxation retains the uncertainty in the model but *assumes full observability*, resulting in a heuristic function that is useful in certain problems, but not in problems where reasoning by cases is not appropriate. For example, if an agent does not know whether it is at a distance one or two from the goal, reasoning by cases, the agent will conclude that it is best to move towards the goal, yet a move in a different direction might help the agent find its true location, as the problem depicted in Fig. 2.1 on page 14. In general, the assumption of full observability yields a heuristic that is not well informed for problems that include an information-gathering component, a feature that is present in many conformant planning problems even if they do not involve observations.

The *cardinality heuristic*, used by the planner HSCP (Bertoli et al., 2001), selects actions that reduce the size the most of the current belief state, that is useful in many situations. There are problems, however, where it might be difficult to find actions that reduce the belief state size or where it is not a good idea to immediately try to reduce it.For example, in a problem of getting to a certain position in a grid (Fig. 2.1 on page 14) where the possible initial states were any position on the grid, the *cardinality heuristic* would do very well. In that case, there is almost always a movement that reduces the size of the belief state and after reaching a corner it is easier to get to the goal position. But if the possible initial positions are only a few ones, moves in any direction will lead to a new belief state of the same size, until one of the walls is reached.

To address the limitations of the reachability heuristic in GPT and the cardinality heuristic used in HSCP, the planner KACMBP uses the notion of *necessary knowledge*

[6]OBDD is a normal form for propositional logic that while being cost to generate, make tractable some queries and operations that are costly for general propositional formulas.

(Cimatti et al., 2004). This planner alternates between the “acquire knowledge” mode, when it detects that necessary knowledge could be acquire, and the “reach goal” mode that uses the GPT heuristic.

Based on the success of *GraphPlan* algorithm for classical planning, some attempts of using a similar structure for conformant planning have been made. The modern planner POND (Bryce et al., 2006) uses a structure called the *Labeled Uncertainty Graph* (LUG) that represents the same *GraphPlan*-like information for different initial states, *i.e.* mutexes, support from preconditions to actions, and from actions to effects. For example, the LUG considers multiple supports for a literal when they are necessary across different initial states, but does not overcount when the same action is used as a support for a literal given different initial states.

Good heuristics need a compact representation to speed up the node generation rate, and reduce the memory footprint, allowing to scale to larger problems. The planner GPT represents belief states as an explicit enumeration of possible sets. Influenced by the use of OBDD (Bryant, 1992) as a compact representation of formulas in model verification, many modern conformant planners use OBDD for representing belief states and computing their heuristics.

The current distribution of the planner MBP integrates CMBP (Cimatti and Roveri, 2000), HSCP (Bertoli et al., 2001), and KACMBP (Cimatti et al., 2004), using OBDDs for belief state representation. In the case of HSCP, the cardinality heuristic is calculated easily given that OBDD supports model counting in linear time in the size of the formula. The planner POND also uses an OBDD-based representation. An efficient construction of the LUG depends on the compactness for representing a forest of OBDDs, as the nodes in the LUG are labeled with formulas for the possible initial states where they apply.

Another representation and heuristic was proposed in Conformant-FF as an extension to FF (Hoffmann and Brafman, 2006). In Conformant-FF the belief states are represented implicitly by a sequence of actions that leads to them. For verifying whether a literal is known to be true in all possible states $s \in b$ of a belief state b, it needs a SAT solver call over a propositional theory encoding the initial belief state and the sequence of actions that leads to s. For the heuristic, it uses an idea based on FF (Hoffmann and Nebel, 2001), leading to a relaxed plan for the conformant planning problem. The key step is to project the CNF theory of the conformant problem to one having clauses of size 2. This way, it is very efficient to approximate the consequences of an action during the calculation of the conformant relaxed plan.

2.9 Other approaches

Another approach to planning with incomplete information is PKS (Petrick and Bacchus, 2002) where belief states are represented by more complex formulas which may include disjunctions. The language used in PKS explicitly represents the knowledge of the agent about the truth assignment of the literals in the problem, and the actions modify this knowledge directly. It allows to encode domain-specific information, but PKS ends up depending on blind search for finding a solution. PKS is sound but incomplete, as it fails to find a solution for some problems.

A way to trade off completeness for efficiency in conformant planning results from approximating belief states or transitions. For example, the *0-approximation* introduced by Baral and Son (1997) represents belief states b by means of two sets: the set of literals that are true in b, and the set of literals that are false in b. Variables which do not appear in either set are unknown. In this representation, checking whether an action a is applicable becomes tractable. Later on, Son and Tu (2006) introduced a complete algorithm based on 0-approximation. Their idea is to create a *set of partial states*, such that a plan that conforms with all them, would be conformant with the original problem. The size of the set of partial states could be exponentially smaller than the corresponding belief state. The `CpA` conformant planner, winner of the conformant track of the IPC-2008 (Bryce and Buffet, 2008), relies on such a complete algorithm, using a heuristic based on the cardinality of the belief state and the number of subgoals achieved (Tran et al., 2009).

All algorithms mentioned so far rely on forward or backward search, on an explicit or implicit belief space. In classical planning, some successful algorithms do non-directional search, typically in the space of possible plans. However, the results have not been as successful, in part due to the higher complexity of the problem (Haslum and Jonsson, 1999; Rintanen, 2004b), and in part due to the lack of sufficiently strong pruning criterion.

There are other works that do not fit in the previous classification. For example, Finzi et al. (2000) present an algorithm for planning based on situational calculus for the case where the initial state is not closed, having to deal with the possible fully specified states. There are some incomplete conformant planners based on answer set programming (Eiter et al., 2003), and even though it can be related to other logic-based approaches, answer set programming is of a higher level than SAT. Along the line of 0-approximation, there has also been extension using answer set programming (Morales et al., 2007; Son et al., 2005a) but they do not scale well when the plan length increases.

Conformant probabilistic planning is also an active area of research. Majercik and Littman (1998) proposed an approach based on a CNF encoding. Later Hyafil and Bacchus (2003) proposed a constraint-based algorithm for this problem, and Huang (2006) proposed an algorithm based on a propositional normal form called *Deterministic Decomposable Negation Normal Form* (d-DNNF, Darwiche, 2001a).

See the article by Hoffmann and Brafman (2006) for a comprehensive review of previous approaches to conformant planning.

Part II

Conformant Planning into CNF

Chapter 3

Model-Counting Formulation

El universo (que otros llaman la Biblioteca) se compone de un número indefinido, y tal vez infinito, de galerías hexagonales.

The universe (which others call the Library) is composed of an indefinite and perhaps infinite number of hexagonal galleries.

The Library of Babel.
Short story by Jorge Luis Borges

The SAT approach to classical planning is based on a correspondence between the plans of length N for a problem P and the models that satisfy a propositional formula $T_N(P)$ obtained from P. Using this correspondence, the plans for P with length N can be obtained by running a SAT solver on $T_N(P)$, if there is any. A SAT-based classical planner starts with $N = 0$, generates a theory $T_N(P)$, trying to obtain a plan of such length, or increasing N by one and trying again.

The same idea, however, does not work for conformant planning. For a conformant problem P, the models of a propositional encoding similar to $T_N(P)$ are in correspondence, not with the plans for P with length N that conform with every possible initial state of P, but with the plans for P that conformant with *some* possible initial state, which are thus not necessarily conformant.

In this chapter and in the next, we introduce conformant planning algorithms that use the propositional encoding $T_N(P)$ and a family of logical operations for ensuring that the obtained plans conform with *all* the possible initial states. In both cases such operations can be computationally expensive, representing a challenge to be addressed.

The conformant planning algorithm introduced in this chapter uses search with a criterion for pruning branches that are deemed *invalid.* Given a propositional encoding of plans of length N for a conformant problem, an *action set* is a partial truth-assignment to the action variables of the theory. It is *complete* if it assigns a truth-value to every action variable and is *valid* if it is consistent with each possible initial state. A valid action set that is complete is guaranteed to encode a conformant plan, and vice versa. The searches proceeds incrementally until obtaining an action set that is both valid and complete

Checking validity can be very expensive in general but it can be verified fast if the propositional theory is in *Deterministic Decomposable Negation Normal Form* (d-DNNF Darwiche and Marquis, 2002), a normal form akin to OBDD (Bryant, 1992). For a given horizon N, stating at 0, the planning algorithm requires first to compile the propositional theory into d-DNNF, and the search algorithm to use operations on the d-DNNF formula to prune invalid branches and guide the search until a solution is found, or the horizon is increased to $N + 1$ for trying again.

This chapter is organized as follows. We extend the definition of conformant planning to allow action parallelism in the solutions and present a propositional theory $T_N(P)$ for conformant problems. Then we define validity and proof that valid actions sets encodes conformant plans and vice versa. Then we show of to verify validity using logical operations, and introduce d-DNNF as a mechanism for calculating them efficiently. We present the resulting conformant planner, report its performance and finish with a short discussion.

The content of this chapter is based on a paper published by Palacios, Bonet, Darwiche, and Geffner (2005).

3.1 Introduction

Optimal planners in the classical setting are built around two notions: branching and pruning. In state-based search, classical planners branch by applying actions (forward or backward) and prune by comparing estimated costs with a given bound. SAT-based planners, on the other hand, branch by trying the values of a selected variable, and prune by propagating constraints and checking consistency.

In principle, the same two notions can and have been used in the conformant setting (Goldman and Boddy, 1996; Smith and Weld, 1998) although the results have not been as strong, in part due to the higher complexity of the problem (Haslum and Jonsson, 1999; Rintanen, 2004b), in part, due to the lack of strong pruning criterion.

Conformant planning is usually solved by *directional* branching schemes that search for plans by applying actions either forward or backward. The problem of optimal conformant planning becomes a shortest path problem over a graph in which the nodes are sets of states or *belief states* (see a longer discussion in Section 2.8 and following, and also Bonet and Geffner, 2000).

The complexity of the search in belief space grows with two factors: the *size* of the states and the *number* of belief states. The first is exponential in the number of variables; the second in the number of states. The switch to symbolic representations, as done by (Cimatti and Roveri, 2000), where sets of states are represented by OBDDs, provides a handle on the first problem but not on the second that demands more informed admissible heuristic functions. Steps in this direction have been reported by Cimatti et al. (2004), Rintanen (2004a), and Bryce et al. (2006).

Conformant planning can also be approached from a logical perspective, working on the theory encoding the problem, and branching on action literals until a valid plan is found. This approach, however, while so successful in the classical setting (Kautz and Selman, 1996),[1] does not appear to work well in the conformant setting. Actually, an action set may complete and consistent with an initial state, but may fail to achieve the goal for other initial state, not being *valid* and, thus, not encoding a conformant plan. Checking validity, however, while useful for pruning can be very expensive. We show then that such validity checks can be performed in *linear time* provided that the theory T encoding the problem is transformed into a logically equivalent theory T' in *Deterministic Decomposable Negation Normal Form* (d-DNNF Darwiche, 2001a). In d-DNNF, validity checks can be reduced to two linear-time operations: projection (finding the strongest consequence of a formula over some of its variables) and model counting (finding the number of satisfying assignments).

We now define the propositional encoding of conformant problems, how to verify validity using logical operations, for then introduce d-DNNF and how to verify validity efficiently, obtaining an effective conformant planner.

3.2 Propositional Encoding of Conformant Tasks

Following the conformant planning task definition of Section 2.4 on page 18, we consider conformant planning problems P given by tuples of the form $P = \langle F, I, O, G\rangle$ where F stands for the fluent symbols f in the problem, O stands for a set of deterministic actions a with conditional effects $a : C \rightarrow L$, I is a set of clauses over the fluents in F encoding the initial, and G is a set of literals over the fluents in F encoding the goal situations.[2]

We also consider *parallel* plans to be a sequence $\{A^0, A^1, \ldots, A^{n-1}\}$ of *sets* of actions. Every pair of actions in each set A^i must be *compatible*, meaning that actions do not interfere each other. For more details about parallel plans see Section 1.9 on page 10.

We assume throughout that the planning problem is consistent in the sense that the set of possible initial states is not empty, and that for no pair of conflicting rules $a : C \rightarrow$ and $a : C' \rightarrow -L$, there is a state reachable from some initial state s_0 where both C and C', and the precondition of the action a are all true.

We build on the propositional encoding for classical planning presented in Section 1.9 on page 10. The encoding of a conformant planning problem $P = \langle F, I, O, G\rangle$ with horizon N is called $T_N(P)$. In this encoding there are variables x_i for fluents and actions x where i is a temporal index in $[0, N]$ for fluents and in $[0, N-1]$ for actions, for the problem of finding a plan for P within N time steps. The only difference in the encoding of a conformant planning is that now the *Init* part of $T_N(P)$ has a clause C_0 for each init clause $C \in I$.[3] Thus, the propositional encoding for a conformant problem $P = \langle F, I, O, G\rangle$ is as follows.

Definition 3.1. *The propositional theory $T_N(P)$ for a conformant planning problem $P = \langle F, I, O, G\rangle$ and horizon N is given by the following set of clauses:*

[1]See Giunchiglia et al. (1998) for a similar approach that only branches on action literals.
[2]See comments on non-deterministic actions in sections 2.4 on page 18
[3]If we allow clauses for encoding the goal of P, the **Goal** part of $T_N(P)$ may has a clause C_N for each goal clause $C \in G$. The conformant planner depicted in this chapter supports such goals, and some benchmarks (sortnet for example) are actually encoded with clauses in goal.

1. **Init:** a literal C_0 for each clause $C \in I$.
2. **Goal:** a literal L_N for each literal $L \in G$.
3. **Actions:** For $i = 0, 1, \ldots, N-1$ and $a \in O$:

$$\begin{array}{rcll} a_i & \supset & Pre(a)_i & \text{(preconditions)} \\ C_i \wedge a_i & \supset & E_{i+1} & \text{(for each rule } a : C \rightarrow E\text{)} \end{array}$$

4. **Frame:** for $i = 0, 1, \ldots, N-1$, each fluent literal L

$$L_i \;\wedge \bigwedge_{a:C\rightarrow\neg L} \neg[C_i \;\wedge\; a_i] \quad\supset\quad L_{i+1}$$

   where the conjunction ranges over the rules $a : C \rightarrow \neg L$
5. **Exclusion:** $\neg a_i \vee \neg a'_i$ for $i = 0, \ldots, N-1$ if a and a' are incompatible

Recall that for a formula B, B_i refers to the formula obtained by replacing each variable x in B by its time-stamped counterpart x_i, and that pair of actions are incompatible when serial plans are required or if their effects share some boolean variable.

Let us consider an example of a CNF encoding of a problem, for getting a better idea of the theories we are dealing with.

Example: a CNF encoding of a conformant problem

Consider the following conformant problem P:

Fluents p, q, r

Init $p \vee q, \neg r$

Actions aq and ar with no preconditions, but conditional effects

- $aq : p \rightarrow q$
- $ar : q \rightarrow r$

Goal r

Following the propositional encoding for conformant planning we show a theory $T_2(P)$ for horizon $N = 2$, for obtaining serial plans:

- Init: $p_0 \vee q_0$, $\neg r_0$
- Goal: r_2
- For $0 \leq i \leq 1$, axioms for:
  - Effects of actions:
    $p_i \wedge aq_i \supset q_{i+1}$
    $q_i \wedge ar_i \supset r_{i+1}$

– For each literal, the corresponding frame axiom:

p	$p_i \supset p_{i+1}$
$\neg p$	$\neg p_i \supset \neg p_{i+1}$
q	$q_i \supset q_{i+1}$
$\neg q$	$\neg q_i \wedge \neg(p_i \wedge aq_i) \supset \neg q_{i+1}$
r	$r_i \supset r_{i+1}$
$\neg r$	$\neg r_i \wedge \neg(q_i \wedge ar_i) \supset \neg r_{i+1}$

– Exclusion: $\neg aq_i \vee \neg ar_i$

Action Sets and Validity

Given the encoding $T_N(P)$, we will refer to collections of *action literals* as *action sets*, and denote them as T_A. We assume that no action set contains complementary or incompatible literals.

Definition 3.2. *An action set T_A is* complete *when it mentions all action variables in the theory $T_N(P)$*

We will refer to an action set that is complete, as a *complete plan* or simply *plans.*[4]

Definition 3.3. *An action set T_A is* consistent *if T_A is logically consistent with the theory $T_N(P)$.*

If P is a classical planning problem, a *consistent* action set that is *complete* encodes a plan for P, yet this is not true if P is conformant. Indeed, a consistent action set that is complete encodes a sequence of actions that conforms with some but not necessarily all possible initial states of P.

As an illustration, consider the following conformant problem P:

Fluents p, s, q, x

Init $p \vee \neg p, s, \neg q, \neg x$

Actions a, b, c and d with no preconditions, but conditional effects

- $a : s \rightarrow q$
- $b : p, s \rightarrow x$
- $c : \neg p, q \rightarrow x$
- $d : true \rightarrow \neg s$

Goal x

For a horizon $N = 3$, we can generate a propositional encoding $T_N(P)$ that will have actions variables for time steps 0, 1 and 2. A plan for this problem is $\{a_0, b_1, c_2\}$. The action set $\{d_0\}$ is inconsistent, while $\{b_0, d_1\}$ is consistent. It maybe extended up to a satisfying assignment of all fluents in $T_N(P)$ by assuming that p is true in the initial situation, but cannot be extended to a conformant plan. In contrast, the action set $\{b_0\}$ can be extended to a conformant plan, and we will say that such action sets are *valid.*

[4]A complete plan denotes a maximal consistent set of *action* literals

3.3 Searching for conformant plans in a CNF representation

Searching for plans requires, thus, to define a notion of validity for discarding invalid plans as soon as possible, and to be able to detect such plans efficiently. An action set can be defined as *valid* when it is logically consistent with the theory and *each* possible initial state. A valid action set that is complete is guaranteed to encode a conformant plan, and vice versa. Let us formalize this idea.

Let $T_0(P)$ refer to the slice of the theory $T_N(P)$ that represents the initial situation, let s_0 represent a state satisfying $T_0(P)$, and let $Lits(s_0)$ refer to the set of literals true in s_0.[5] Then we define the notion of *validity* of action sets in the conformant setting as follows:

Definition 3.4 (Validity). *An action set T_A is* valid *in the context of a theory $T_N(P)$, if and only if for* every *possible state s_0 satisfying $T_0(P)$, the set of formulas given by $T_A \cup T_N(P) \cup Lits(s_0)$ is logically consistent.*

This definition has two properties that we will exploit in the branching scheme used to search for conformant plans. The first is that a complete plan that is valid represents a conformant plan and, vice versa, a conformant plan represents a valid complete plan. The second is that an incomplete action set that is not valid cannot lead to a conformant plan.

We state these properties as follows:

Theorem 3.5. *A valid complete actions set T_A for $T_N(P)$ encodes a conformant plan π for P, where $\pi = \{A^0, \ldots, A^{N-1}\}$, and $a \in A^i$ iff $a_i \in T_A$.*

Theorem 3.6. *A conformant plan π for P, with $\pi = \{A^0, \ldots, A^{N-1}\}$, encodes a valid complete action set T_A for $T_N(P)$, where $a_i \in T_A$ iff $a \in A^i$, and $\neg a_i \in T_A$ iff $a \notin A^i$.*

These two results establish a correspondence between the conformant plans for P and the valid complete action sets for $T_N(P)$. In addition, we have another result that is fundamental for searching for valid action sets that are complete, and hence, for conformant plans:

Theorem 3.7. *An invalid action set for $T_N(P)$ cannot be extended into a* valid complete *action set for $T_N(P)$.*

With this result, we can search for valid partial action sets that are complete incrementally, pruning the partial action that are not valid. Now we prove formally these three theorems, but first we need a to prove a lemma that will be useful.

Lemma 3.8. *Let $\pi = \{A^0, \ldots, A^{N-1}\}$ a plan, and T_A a complete action set of a theory $T_N(P)$, such that $a_i \in T_A$ iff $a \in A^i$, and $\neg a_i \in T_A$ iff $a \notin A^i$. For an initial state s_0, π is a classical plan for the classical problem P/s_0 if and only if the complete action set T_A is consistent with $T_N(P) \cup Lits(s_0)$.*

[5]Each state s_0 corresponds to an initial state in the conformant problem P

Proof. If π is a classical plan for P/s_0, then exists a model M satisfying $T_N(P/s_0)$ because of the correspondence between plans and models of the SATPLAN encoding. Moreover, M is also a model of $T_N(P) \cup Lits(s_0)$. Thus, as the action set T_A is part of the model M, T_A is consistent with $T_N(P) \cup Lits(s_0)$. Besides, T_A is complete, proving one direction of the equivalence.

For the other direction, if T_A is a complete action set and is consistent with $T_N(P) \cup Lits(s_0)$, observe that T_A and $Lits(s_0)$ completely determine the theory $T_N(P)$. Thus, exists a model M for $T_N(P) \cup Lits(s_0)$, that is also a model of $T_N(P/s_0)$. Finally, the model M encodes a plan π defined in terms of the literals in T_A. □

Proof of theorem 3.5. Let s_0 be a possible initial state. From lemma 3.8, T_A encodes a classical plan π for the classical problem P/s_0. Thus, π conforms with any s_0, and hence π is conformant. □

Proof of theorem 3.6. The definition of conformant planning means that π is a plan of the classical problem P/s_0, for any initial state s_0. From lemma 3.8, T_A is complete and consistent with $T_N(P)$ and $Lits(s_0)$. Thus, T_A is valid. □

Proof of theorem 3.7. Let us assume that the action set T_A can be extended into a valid complete plan T'_A. From Definition 3.4 of validity, for any possible initial state s_0, the set $T'_A \cup T_N(P) \cup Lits(s_0)$ is logically consistent. Thus, the set $T_A \cup T_N(P) \cup Lits(s_0)$ is also logically consistent. Thus, T_A is valid. Contradiction. □

These properties ensure the soundness and completeness of a simple branch and prune algorithm that branches on action literals, prunes action sets that are not valid, and terminates when a non-pruned complete plan is found. Of course, this simple algorithm would not be necessarily efficient as it involves an expensive validity check in every node of the search tree, which if done naively, would involve a number of satisfiability tests linear in the number of possible initial states.

3.4 Pruning Action Sets by Model Counting and Projection

We focus now on the use of compilation techniques for making the validity checks efficiently. Indeed, while the validity checks in Definition 3.4 may be exponential in the clauses of the initial situation of P, it turns to be *polynomial* when $T_N(P)$ is in *deterministic decomposable negation normal form*, d-DNNF (Darwiche, 2001a). d-DNNF is a target language that renders a number of boolean operations and transformations tractable, including model counting and projection (Darwiche and Marquis, 2002). Indeed, model counting and projection are the two boolean operations needed in order to implement the validity tests.

Definition 3.9 (Model count)**.** *The* model count *of a formula* Δ*, denoted as* $MC(\Delta)$*, stands for the number of truth assignments that satisfy the formula.*

Intuitively, the projection of a formula Δ over a subset of its variables V allows to obtain a new formula Δ', in some sense equivalent to Δ as far as we refer to variables in V.

Definition 3.10 (Projection)**.** *The* projection *of a formula Δ over a subset V of its variables stands for the strongest formula over the variables V implied by Δ; i.e., $\Delta' = \mathsf{project}[\ \Delta\ ;\ V\ ]$ if Δ entails Δ' and for every formula Δ'' over V that is entailed by Δ, $Delta'$ entails Δ''.*

Projection is unique up to logical equivalence, *i.e.* if Δ_1 and Δ_2 are projections of Δ over V then $Delta_1$ and $Delta_2$ are logically equivalent. The projection operation is in turn a dual of variable elimination, usually called forgetting in the context of propositional formula (Lin and Reiter, 1994; Lang et al., 2003). Indeed, projection over V is equivalent to eliminating all the variable that are in Δ but not in V'. Moreover, it is well known that a boolean variable can be eliminated from a formula Δ by the conditioning operation. *Conditioning* of a theory Δ on a literal α results in a formula equivalent to $\Delta \wedge \alpha$, which models are the models of Δ consistent with the literal α. Here we follow the definition 5.4 provided by Darwiche and Marquis (2002).

Definition 3.11 (Conditioning)**.** *Let Δ be a propositional formula, and let α be a consistent term. The conditioning of Δ on α, noted $\Delta \,|\, \alpha$, is the formula obtained by replacing each variable X of Δ by true (resp. false) if X (resp. $\neg X$) is a positive (resp. negative) literal of α.*

We this definition of conditioning, forget a variable v from a formula Δ is equivalent to $\Delta \,|\, p \vee \Delta \,|\, \neg p$. Indeed, Lin and Reiter (1994) define forgetting in this way.

For using the projection operation, it might be useful to describe it in term of the models of a theory Δ and its projection over V. We establish such relation through the following lemma.

Lemma 3.12. *If $\Delta' = \mathsf{project}[\ \Delta\ ;\ V\ ]$, then M' is a model of Δ' if and only if M is a model of Δ and M' is the model M restricted to the variables V.*

Proof. [6] Let M' a models that satisfy Δ', $M' \models D'$. For contradiction, let us assume that there is no extension of M' satisfying Δ. Let us abuse slightly of the notation to consider the model M also as a conjunction of literals. If we can show that $\Delta \models \neg M'$ the proof will be complete. $\neg M'$ is a theory over the variables V and Δ' is the strongest, hence $\Delta' \models \neg M'$, but this contradicts $M' \models \Delta'$.

It left only to show $\Delta \models \neg M'$. Let us prove it for the case of projecting over only one var w not in V. Let $M'[w]$ the model M' extended with the positive literal w. Because hypothesis *ad absurdum*,

$$M'[\neg w] \models \neg\Delta \qquad \text{and} \qquad M'[w] \models \neg\Delta$$

then

$$\Delta \supset \neg(M'[\neg w]) \qquad \text{and} \qquad \Delta \supset \neg(M'[w])$$

[6] I thank Blai Bonet for his help with this proof.

then

$$\Delta \supset w \vee \neg M' \qquad \text{and} \qquad \Delta \supset \neg w \vee \neg M'$$

then

$$\Delta \supset \neg M'$$

then

$$\Delta \models \neg M'$$

□

Now, if we let F_0 refer to the fluent variables f_0 at time $i = 0$ and $T_0(P)$ refer to the slice of $T_N(P)$ encoding the initial situation, the validity check from Definition 3.4 can be rephrased as follows:

Theorem 3.13 (Validity by Projection and MC)**.** *An action set T_A is* valid *in the context of a theory $T_N(P)$ iff*

$$\mathsf{MC}(T_0(P)) = \mathsf{MC}(\mathsf{project}[\ T_N(P) \,|\, T_A\ ;\ F_0\ ])\,. \tag{3.1}$$

Proof. For simplicity, we use T_0 for referring to $T_0(P)$, and T' to the result of $\mathsf{project}[\ T_N(P) \,|\, T_A\ ;\ F_0\ ]$. We start with two observations.

Observation$_1$: *Each model of T' is also a model of T_0.* This holds because $T_N(P)$ includes the formula T_0 as a conjunct, T' variables are the same of T_0, and lemma 3.12 of projection. Observation$_1$ implies that $\mathsf{MC}(T') \leq \mathsf{MC}(T_0)$.

Observation$_2$: *Each model of T_0 is a model of T' iff $\mathsf{MC}(T_0) = \mathsf{MC}(T')$.* That is because if each model of T_0 is a model of T', because observation$_1$, it follows that $\mathsf{MC}(T_0) = \mathsf{MC}(T')$. On the other hand, if a model of T_0 is not a model of T', because observation$_1$, then $\mathsf{MC}(T')$ should be strictly smaller than $\mathsf{MC}(T_0)$.

Now we proceed with the proof, giving justifications between { *braces* }.

T_A is valid
iff { *Definition 3.4 (validity)* }
For any possible initial state s_0: $T_A \cup T_N(P) \cup \mathit{Lits}(s_0)$ is consistent
iff { *rephrase and conditioning* }
Each model s_0 of $T_0(P)$ is consistent with $T_N(P) \,|\, T_A$
iff { *Lemma 3.12 (projection)* }
Each model s_0 of $T_0(P)$ is a model of $\mathsf{project}[\ T_N(P) \,|\, T_A\ ;\ F_0\ ]$
iff { *observation$_2$* }
$\mathsf{MC}(T_0(P)) = \mathsf{MC}(\mathsf{project}[\ T_N(P) \,|\, T_A\ ;\ F_0\ ])$

□

The theorem reduces the validity check of an action set T_A to the comparison of two numbers: the number of possible initial states, and the number of initial states satisfying the theory and the commitments made in T_A. Clearly, the second number cannot be greater than the first, as $T_N(P)$ alone entails $T_0(P)$ ($T_0(P)$ is part of $T_N(P)$). Yet the second number can be smaller: this would happen precisely when some possible initial state s_0 is not compatible with $T_N(P)$ and T_A, which according to definition 3.4, is exactly the situation in which T_A is not a valid action set.

We turn now to the compilation of the propositional formula $T_N(P)$ for making the model count and projection operations tractable. Even though the compilation is intractable, it will be done once, and will allow to evaluate model counting and projection on each node of the search tree.

3.5 A Conformant Model-Counting Planner based on d-DNNF

Knowledge compilation is the area in AI concerned with the problem of mapping logical theories into suitable target languages that make certain desired operations tractable (Selman and Kautz, 1996; Cadoli and Donini, 1997). For example, propositional theories can be mapped into their set of Prime Implicates making the entailment test of clauses tractable (Reiter and de Kleer, 1987). Similarly, the compilation into *Ordered Binary Decision Diagrams* (OBDDs) renders a large number of operations tractable including model counting (Bryant, 1992). While in all these cases, the compilation itself is intractable, its cost may be justified if these operations are to be used a sufficiently large number of times in the target application. Moreover, while the compilation will run in exponential time and space in the worst case, it will not necessarily do so on average. Indeed, the compilation of theories into OBDDs has been found useful in formal verification (Clarke et al., 2000) and more recently in planning (Giunchiglia and Traverso, 1999). A more recent compilation language is Decomposable Negation Normal Form (DNNF) (Darwiche, 2001b). DNNFs support a rich set of polynomial–time operations, some of which are particularly suited to our application, like *projection* on an arbitrary set of variables, which can be performed simply and efficiently. A subset of DNNF, known as deterministic DNNF, also supports *model counting*

Deterministic Decomposable NNF (d-DNNF)

A propositional sentence is in negation normal form (NNF) if it is constructed from literals using only conjunctions and disjunctions (Barwise, 1977). A practical representation of NNF sentences is in terms of rooted directed acyclic graphs (DAGs), where each leaf node in the DAG is labeled with a literal, true or false; and each non-leaf (internal) node is labeled with a conjunction $\wedge$ or a disjunction $\vee$; see Figure 3.1. Decomposable NNFs are defined as follows:

Definition 3.14. *(Darwiche, 2001b) A* decomposable negation normal form (DNNF) *is a negation normal form satisfying the* decomposability property: *for any conjunction $\wedge_i \alpha_i$ in the formula, no variable appears in more than one conjunct α_i.*

The NNF in Figure 3.1 is decomposable. It has ten conjunctions and the conjuncts of each share no variables. Decomposability is the property which makes the satisfiability of DNNF tractable: a decomposable NNF formula $\wedge_i \alpha_i$ is indeed satisfiable iff every conjunct α_i is satisfiable, while $\vee_i \alpha_i$ is satisfiable iff some disjunct α_i is. The satisfiability of a DNNF can thus be tested in linear time by means of a single bottom up pass over its DAG.

The NNF $(A \vee B) \wedge (\neg A \vee C)$ is not decomposable since variable A is shared by the two conjuncts. Any such form, however, can be converted into DNNF. The main

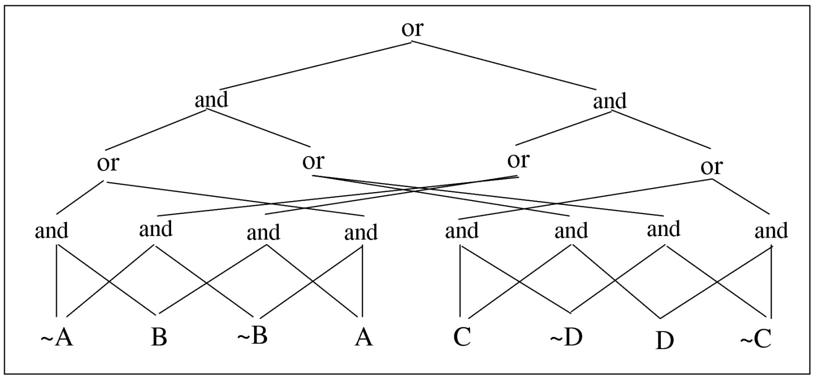

Figure 3.1: A negation normal form (NNF) represented as a rooted DAG. (from Darwiche, 2001a).

technique to use here is that of performing *case analysis* over the variables that violate the decomposition property, in this case A. Assuming that A is true, the NNF reduces to C, while if A is false, the NNF reduces to B. The result is the NNF $(A \wedge C) \vee (\neg A \wedge B)$ which is decomposable, and hence, a DNNF.[7]

The above principle can be formulated more precisely using the notion of *conditioning,* (Definition 3.11 on page 34). The conditioning of Δ on literal α, written $\Delta \,|\, \alpha$, is obtained by simply replacing each leaf α in the NNF DAG by true and each leaf $\neg\alpha$ by false. If $\Delta = (A \vee B) \wedge (\neg A \vee C)$, then $\Delta \,|\, A$ is $(\mathsf{true} \vee B) \wedge (\mathsf{false} \vee C)$ which simplifies to C. Similarly, $\Delta \,|\, \neg A$ is $(\mathsf{false} \vee B) \wedge (\mathsf{true} \vee C)$ which simplifies to B.

The case analysis principle can now be phrased formally as follows:[8]

$$\Delta \;\equiv\; (\Delta \,|\, A \;\wedge\; A) \vee (\Delta \,|\, \neg A \;\wedge\; \neg A) \tag{3.2}$$

This will actually be the pattern for decomposing propositional theories as we shall see later. For now though, we point out that the split exhibited by (3.2) above, leads to a second useful property called *determinism,* giving rise to the special class of *Deterministic* DNNF*s*:

Definition 3.15. *(Darwiche, 2001a) A* deterministic DNNF (d-DNNF) *is a* DNNF *satisfying the* determinism property: *for any disjunction $\vee_i \alpha_i$ in the formula, every pair of disjuncts α_i is mutually exclusive.*

Determinism is the property which makes *model counting* over DNNFs tractable: the number of models of a DNNF $\wedge_i \alpha_i$ is the *product* of the number of models of each conjunct α_i, while the number of models of a DNNF $\vee_i \alpha_i$ that satisfies determinism is the *sum* of the number of models of each disjunct. Actually, in order to get a normalized count it must be made sure that the same set of variables appear in the different disjuncts of the d-DNNF. However, this property called 'smoothness' is

[7]Disjunctive Normal Form (DNF), with no literal sharing in terms, is a subset of DNNF. The DNF language, however, is *flat* as the height of corresponding NNF DAG is no greater than 2. This restriction is significant as it reduces the succinctness of DNF as compared to DNNF. For example, it is known that the DNF language is incomparable to the OBDD language from a succinctness viewpoint, even though DNF and OBDD are both strictly less succinct than DNNF (Darwiche and Marquis, 2002).

[8]This principle is also known as Boole's expansion and Shannon's expansion.

easily enforced (Darwiche, 2001a). In the rest of the chapter and this document, we will assume that d-DNNFs are smooth, denoted as sd-DNNF.

The final key operation on DNNFs that we need is *projection.* As we comment on Section 3.4, theorem 3.13, the projection of a theory Δ on a set of variables V is the strongest sentence implied by Δ over those variables. This sentences is unique up to logical equivalence and is denoted by $\mathsf{project}[\ \Delta\ ;\ V\ ]$. Projection is dual to *elimination* or *forgetting* (Lin and Reiter, 1994): that is, projecting Δ on V is equivalent to eliminating (existentially quantifying) all variables that are *not* in V from Δ. Like satisfiability on DNNFs, and model counting on d-DNNFs, projection on DNNFs can be done in linear time (Darwiche, 2001b). Specifically, to project a DNNF on a set of variables V, all we have to do is replace every literal in Δ by true if that literal mentions a variable outside V. For example, the projection of DNNF $(A \wedge \neg B) \vee C$ on variables B and C is the DNNF $(\mathsf{true} \wedge \neg B) \vee C$, which simplifies to $\neg B \vee C$. Moreover, the projection on variable C only is the DNNF $(\mathsf{true} \wedge \mathsf{true}) \vee C$, which simplifies to true.

Testing plan validity using d-DNNF

The use of d-DNNFs in this chapter has been motivated by the desire to make the validity test for action sets tractable and efficient. The test, captured by the equation (3.1) on page 35, involves computing the model count of a projection. We have seen that we can count the models of d-DNNF and take the projection of a DNNF in linear time. This may suggest that we can render the validity test for action sets to be linear in the size of the d-DNNF representation. This however is not true in general; the problem is that the linear–time projection operation given above is guaranteed to preserve decomposability but not necessarily determinism. This means that while we can model count and project a deterministic DNNF in linear time, we cannot always model count the projection of a deterministic DNNF in linear time, which is precisely what we want. There are however two conditions under which the projection $\mathsf{project}[\ \Delta\ ;\ V\ ]$ of a deterministic DNNF Δ on variables V can be guaranteed to remain deterministic and allow for model counting in linear time. The first condition is that variables $\overline{V}$, which are projected away, are *determined* in Δ by variables V that are kept (i.e., the values of variables $\overline{V}$ in any model of Δ are determined by the values of variables V). This condition holds in our setting for the fluent variables f_i for $i > 0$ which are determined by the initial fluent variables f_0 and the action variables a_i, $i = 1, \ldots, N-1$. We actually use this result to project the compiled d-DNNF on the initial state variables and on action variables, leaving out all other variables at the outset. The second condition relates to an ordering restriction that we can impose on the splits given by (3.2) above; we discuss this restriction in the following section.

Compiling planning theories into d-DNNF

A propositional theory Δ can be compiled into d-DNNF by simply ordering the variables appearing in Δ in a sequence $x_1, \ldots, x_n$, and then splitting Δ first on x_1 leading to $(\Delta \,|\, x_1 \wedge x_1) \vee (\Delta \,|\, \neg x_1 \wedge \neg x_1)$, and then compiling recursively each of the conditioned theories $\Delta \,|\, x_1$ and $\Delta \,|\, \neg x_1$ using the sub-order $x_2, \ldots, x_n$. Coupled with a

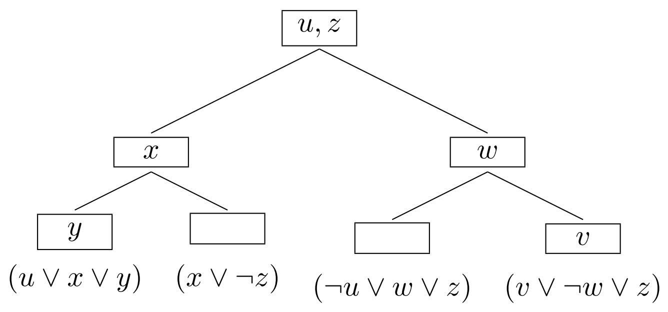

Figure 3.2: A decomposition tree for a CNF.

caching scheme to avoid compiling the same theory multiple times, the above technique will lead to d-DNNFs that are isomorphic to OBDDs. In fact, this particular method for compiling OBDDs, which deviates from the vast tradition on this subject, was explored in Huang and Darwiche (2004). Deterministic DNNFs, however, are known to be strictly more space efficient than OBDDs (Darwiche and Marquis, 2002), and indeed a more efficient compilation scheme is possible (Darwiche, 2004). In particular, if during this top–down compilation process one gets to an instantiated theory Δ' of the form $\Delta'_1 \wedge \Delta'_2$ such that Δ'_1 and Δ'_2 share no variables, then the compilation of Δ' can be *decomposed* into the conjunction of the compilation of Δ'_1 and the compilation of Δ'_2. Moreover, one does not need to use a fixed variable order as required by OBDDs, but can choose variables dynamically to split on, typically, to try to maximize the opportunities for decomposition.

Although dynamic variable ordering and decomposition appear to be the reasonable strategy to adopt in this context, experience has shown that it may incur in an unjustifiable overhead. The d-DNNF compiler we use is instead based on semi–dynamic variable orderings, which are obtained by a pre–processing step to reduce the overhead during compilation (Darwiche, 2004). In particular, before the compilation process starts, one constructs a *decomposition tree (dtree)* as shown in Figure 3.2 (Darwiche, 2001b). This is simply a binary tree whose leaves are tagged with the clauses appearing in the CNF to be compiled. Each internal node in the dtree corresponds to a subset of the original CNF and is also tagged with a *cutset:* a set of variables whose instantiation is guaranteed to decompose the CNF corresponding to that node. The compiler will then start by picking up variables from the root cutset to split on until the CNF corresponding to the root is decomposed. It will then recurse on the left and right children of the root, repeating the same process again. Within each cutset (which can be arbitrary large), the compiler chooses variable order dynamically. Note also that the dtree imposes only a partial order on cutsets, not a total order.

A number of additional features are incorporated into the d-DNNF compiler we use (Darwiche, 2004). Examples include the use of caching to avoid compiling identical theories twice; unit propagation for simplifying theories; dependency directed backtracking; and clause learning.

The key benefit of d-DNNF compilers in relation to OBDD compilers is that the former makes use of *decomposition.* Hence, for example, the complexity of d-DNNF compilations are known to be exponential only in the treewidth of the theory, while OBDD compilations are exponential in the pathwidth, which is no less than the treewidth and usually much larger (McMillan, 1994; Darwiche, 2004; Dechter and Mateescu, 2007). Another advantage of using d-DNNFs over OBDDs is that d-DNNFs employ a more general form of determinism allowing us to use the linear–time projection operation discussed earlier, while still preserving both decomposability and determinism in some cases. This feature allowed us to project compiled planning theories on the initial state fluents and the actions in linear time. OBDDs do not support a linear–time operation for projection under the same conditions (Darwiche and Marquis, 2002).

We note that we had to use specific decomposition trees to allow us to generate d-DNNFs which can be projected on the initial state fluents *only* while preserving determinism—this is needed to implement the plan validity test in (3.1) which is critical for pruning during search. In particular, we had to construct dtrees in which initial state fluents are split on before any other variables are split during case analysis. This guarantees that determinism would be preserved when projecting the d-DNNF on initial state fluents as it guarantees that every remaining disjunctions will be of the form $(f_0 \wedge \alpha) \vee (\neg f_0 \wedge \beta)$ where f_0 is an initial state fluent.[9]

```
Input: Formula $\Delta$ in deterministic Decomposable Negated Normal Form
       d-DNNF
Input: Set of literals $S$ for conditioning
Output: Number of models of $\Delta \cup S$

(* from (Darwiche, 2001a) *)
function MC($\Delta$) begin
    if $\Delta$ is a labeled with literal $L$ and $\neg L \in S$ then
        return 0
    if $\Delta$ is labeled with literal $L$ and $\neg L \notin S$ then
        return 1
    if $\Delta$ is a labeled with AND then
        return $\Pi_i$ (MC ($\Delta_i$)) where each $\Delta_i$ is a child of $\Delta$
    if $\Delta$ is a labeled with OR then
        return $\Sigma_i$ (MC ($\Delta_i$)) where each $\Delta_i$ is a child of $\Delta$
end

return MC($\Delta$)          (* model counting of $\Delta$ conditioned to $S$ *)
```

Figure 3.3: Model Counting algorithm for d-DNNF, conditioned to S.

For computing the model-counting and projection in one pass in the d-DNNF we develop a variation of the model-counting algorithm as follow. Fig 3.3 shows the original algorithm for computing the number of models of a d-DNNF.[10] The intuition is that *decomposability* guarantee that no variable appears in more than one conjunct, thus the number of models is the product of the model counting of each node. Also, *determinism* guarantee that disjuncts are incompatible with each other, thus the number of models if the sum of the model counting of each disjunct.

As mentioned above, in order to perform the pruning test (3.3) efficiently in every node n, we must ensure that the projection operation preserves determinism. This is

[9] A similar technique can be used if one compiles into OBDDs, by having initial state fluents first in the OBDD order.

[10] Again, we assume that d-DNNF is actually smooth.

ensured by compiling the CNF theory $T_N(P)$ using a decomposition tree in which the splits on the variables belonging to F_0 (the fluent variables f_0 for the initial situation) are done before any other splits. Also, for having an equivalent but smaller d-DNNF we project away all fluent variables f_i from the theory for $i > 0$ at compilation time. Such fluents are not needed, and their elimination satisfies the other condition above: they are determined by the initial fluent and action variables that are kept. This lead us to an algorithm for doing projection and model counting in one pass in the d-DNNF, allowing to compute:

$$\mathsf{MC}(\mathsf{project}[\ \Delta\,|\,T_A\ ;\ F_0\ ])$$

The idea of the algorithm shown in Figure 3.4 is to project lower sections of the d-DNNF DAG, corresponding to theories with fluents for time > 0, action variables. Subtheories inconsistent with current assignation return zero, but consistent one return *FORGET*, meaning that are consistent, but will not add up for model counting. The first time a non-to-forget theory is combined with another to be forgotten, the decision is more subtle. Basically, a forget branch counts as one, unless the node is an OR and the remaining node is zero. In such case, that OR node can be consider to be just the *FORGET* branch, and thus returns *FORGET*.

The Conformant Planner

We have implemented a validity-based optimal conformant planner called vplan that accepts the description P of a conformant planning problem and a planning horizon N, and then produces a valid conformant plan in N time steps at most if one exists, otherwise reports failure. If the horizon is incremented by 1 starting from $N = 0$, the first plan found is guaranteed to have minimal makespan. The planner can be run in serial or parallel mode according to the sets of concurrent actions allowed. vplan translates P into a theory T_c in d-DNNF and then performs a backtrack search for a valid conformant plan by performing operations on the theory: branching on action literals, pruning invalid sets of action literals (action sets), and terminating when a non-pruned complete plan is found. More precisely, the planner can be characterized by the following aspects:

- **Preprocessing:** the problem P with a given horizon N is translated into a CNF theory $T_N(P)$, which is compiled into a d-DNNF theory T_c which is associated with the root node n_r of the search tree; i.e. $T(n_r) = T_c$.[11]

- **Branching:** at a node n in the search tree, the planner branches by selecting an undetermined action variable a_i and trying each of its possible values; namely, two d-DNNF theories T_{n_1} and T_{n_2} are created for the children nodes n_1 and n_2 of n that correspond to $T_n|a_i$ and $T_n|\neg a_i$. This process continues depth-first until a node is pruned, resulting in a backtrack, or all action variables are determined, resulting in a valid conformant plan.

[11]A subtlety in the translation from P to the CNF theory $T_N(P)$ are the frame axioms which may generate an exponential number of clauses. In order to avoid such explosion, each conjunction $c^k(a)_i \wedge a_i$ of a condition $c^k(a)_i$ and the corresponding action a_i, is replaced by a new auxiliary variable z_i and the equivalence $z_i \equiv c^k(a)_i \wedge a_i$ is added to the theory. Such auxiliary variables do not affect the compilation into d-DNNF as they are 'implied variables' that are safely projected away when the CNF theories are compiled into d-DNNF.

```
Input: Formula $\Delta$ in d-DNNF
Input: Set of literals $S$ to for conditioning
Input: Set of literals $V$ to project
Output: Number of models of project[ $\Delta \cup S$ ; $V$ ]
(* For simplicity of the presentation, we assume nodes have at
   most two children $\Delta_1$ and $\Delta_2$ *)
function Calc-MC-and-P($\Delta$) begin
    if $\Delta$ is a labeled with literal $L$ and $\neg L \in S$ then
        return 0
    if $\Delta$ is a labeled with literal $L$ and $var(L) \notin V$ then
        return FORGET
    if $\Delta$ is labeled with literal $L$ and $\neg L \notin S$ then
        return 1
    if $\Delta$ is a labeled with AND then
        Val$_i$ ← Calc-MC-and-P ($\Delta_i$) for each $i \in \{1, 2\}$
        if both Val$_i$ = FORGET then
            return FORGET
        if both Val$_i$ ≠ FORGET then
            return Val$_1$× Val$_2$
        if some Val$_i$ ≠ FORGET and the other Val$_j$ = FORGET then
            return Val$_i$              (* Consider FORGET branch as one *)
    if $\Delta$ is a labeled with OR then
        Val$_i$ ← Calc-MC-and-P ($\Delta_i$) for each $i \in \{1, 2\}$
        if both Val$_i$ = FORGET then
            return FORGET
        if both Val$_i$ ≠ FORGET then
            return Val$_1$+ Val$_2$
        if some Val$_i$ ≠ FORGET and the other Val$_j$ = FORGET then
            if Val$_i$ = 0 then
                return FORGET    (* If $\Delta_i$ is inconsistent, but $\Delta_j$ is
                to be forget, keep forgetting *)
            else
                return Val$_i$
end
return Calc-MC-and-P($\Delta$)
```

Figure 3.4: Algorithm MC-and-P for simultaneous Model Counting and Projection for d-DNNF Δ. Also condition Δ on a set of literals. Returns MC(project[$\Delta \mid S$; V]). Requires some properties on Δ to be sound.

		CNF theory		d-DNNF theory		
problem	N^*	vars	clauses	nodes	edges	time/acc
blocks-2	2	34	105	61	97	0.03/0.06
blocks-3	9	444	2913	4672	20010	0.25/1.13
blocks-4	26	3036	40732	225396	913621	77.5/752.65
square-center-4	8	200	674	1000	2216	0.1/0.39
square-center-8	20	976	3642	9170	19555	0.7/6.7
square-center-16	44	4256	16586	79039	164191	31.17/512.54
ring-3	8	209	669	2753	6161	0.11/0.48
ring-4	11	364	1196	13239	29295	0.62/2.52
ring-5	14	561	1874	60338	132045	3.68/16.4
ring-6	17	800	2703	254379	551641	23.77/120.58
ring-7	20	1081	3683	1018454	2195393	221.58/1096.7
ring-8	23	1404	4814	3928396	8406323	2018.32/12463.3
sortnet-3	3	51	122	133	230	0.03/0.09
sortnet-4	5	150	409	1048	2325	0.04/0.19
sortnet-5	9	420	1343	7395	17823	0.51/1.4
sortnet-6	12	813	3077	30522	77015	1.28/7.12
sortnet-7	16	1484	6679	116138	294840	8.29/56.61
sortnet-8	19	2316	12364	369375	931097	56.73/427.58
sortnet-9	25	3870	24414	1264508	3075923	780.77/6316.53

Table 3.1: Compilation data for serial planning. N^* is the optimal planning horizon. Nodes and edges refer to the DAG-representation of the generated d-DNNF. Time refers to the compilation time for the theory with horizon N^*, and 'acc' to the sum of all compilation times for horizons $N = 0, \ldots, N^*$. All times are in seconds.

- **Pruning:** a node n is pruned when the d-DNNF theory T_n associated with n fails the *validity test*:

$$\mathsf{MC}(T_0) = \mathsf{MC}(\mathsf{project}[\ T_n\ ;\ F_0\ ]) \tag{3.3}$$

  where T_0 stands for the slice of the theory $T_N(P)$ encoding the initial situation, MC stands for the model count operator, and F_0 stands for the fluent variables in the initial situation. The model count and the projection are done in linear time by means of a single bottom-up pass over the DAG representation as show in the algorithm of Figure 3.4 on the facing page. The model count over T_0 is done once, and measures the number of possible initial states.

- **Selection Heuristics and Propagation:** The undetermined action variable a_i for branching in node n is selected as the positive action literal a_i that occurs in the greatest number of models of T_n; this ranking being obtained by means of a single model count $\mathsf{MC}(T_n)$ implemented so that with just two passes over the DAG-representation of T_n (one bottom up, another top-down; see (Darwiche, 2001a)), it yields model counts $\mathsf{MC}(T_n \wedge l)$ for all literals l in the theory. Moreover, when for a yet undetermined action literal l this model count yields a number which is smaller than the number of initial states $\mathsf{MC}(T_0)$, then its complement $\neg l$ is set to true by conditioning T_n on $\neg l$. This process is iterated until no more literals can be so conditioned on in T_n. This inference cannot be captured by performing deductions on T_n as this could only set a literal $\neg l$ to true when the model count $\mathsf{MC}(T_n \wedge l)$ is exactly 0. The inference, however, follows from the QBF *formula* associated with T_n encoding not only the planning *domain* but the planning *task* (Rintanen, 1999).[12]

```
Input: Conformant Problem $P = \langle F, O, I, G \rangle$
Input: Horizon $N$
Output: Print conformant plan $\pi$ for $P$ or return FALSE

$T_N(P) \leftarrow$ propositional translation of $P$ for $N$ time steps plans
$\Delta \leftarrow$ d-DNNF compilation of $T_N(P)$
$\#S_0 \leftarrow$ number of initial states of $P$

procedure Closure($T_A$):
  repeat
    MC-L($\Delta$,$T_A$) (* calculates MC($\Delta$,$T_A \cup \{a\}$) for each literal $a$ in
    two passes over $\Delta$ (Darwiche, 2001a)  *)
    foreach $a$ such that MC($\Delta$,$T_A \cup \{a\}$) $\leq \#S_0$ do
      $T_A \leftarrow T_A \cup \{\neg a\}$
  until no change

function PickAction($T_A$):
  return action literal $a$ with larger MC($\Delta$,$T_A \cup \{a\}$)

function Search(action set $T_A$):
begin
   $\phi \leftarrow$ MC-and-P($\Delta$,$T_A$,$F_0$)        (* returns MC(project[$\Delta \mid T_A$ ; $F_0$]) *)
                                             (* See Figure 3.4 on page 42 *)
   if $\phi < \#S_0$ then
      return FALSE
   else if Complete($T_A$) then
      PrintPlan ($T_A$) and abort
   else
      begin
         Closure($T_A$)
         $a \longleftarrow$ PickAction($T_A$)
         if not Search($T_A \cup \{a\}$) then
            if not Search($T_A \cup \{\neg a\}$) then
               return FALSE
      end
end

return Search($\emptyset$)                  (* start with empty action set *)
```

Figure 3.5: vplan algorithm for finding a conformant plan of N time steps using compilation to d-DNNF, model counting and projection.

[12]The QBF formulation of conformant planning reflects that we are not looking for models of T_n but for interpretations over the action variables that can be extended into models of T_n for any choice of the initial fluent variables compatible with T_0. The QBF formula encoding the planning task over the theory $T_N(P)$ will imply that an action literal a_i that does not participate in any conformant plan that solves P needs to be false. This, however, does not imply that $\neg a_i$ is a deductive consequence of the planning theory $T_N(P)$; it is rather a deductive consequence of the QBF formula encoding the planning *task.* This distinction does not appear in the classical setting, where the same formula encodes the planning theory and the planning task; it is however important in the conformant setting where this is no longer true. In Section 9.1 we comment more about the QBF for a conformant planning task in a more appropriate context.

Summarizing, the vplan algorithm is shown in Figure 3.5. An optimization respect to the aspects explained before is that no new d-DNNF is calculated during search. Instead, the algorithm of Figure 3.4 is used to condition on S the theory Δ while doing model counting and projection upon vars in P, calculating $\mathsf{MC}(\mathsf{project}[\ \Delta \,|\, S\ ;\ V\ ])$.

3.6 Experimental Results

We performed the experiments on a Intel/Linux machine running at 2.80GHz and 2Gb of memory.[13] Times for the experiments were limited to 2 hours, in most cases, and memory to 1Gb. We used the same suite of problems as Rintanen (2004a). These are challenging problems that emphasize some of the critical aspects that distinguish conformant from classical planning; some of the problems are from Cimatti et al. (2004)

- **Ring:** There are n rooms arranged in a circle and a robot that can move either clockwise or counter-clockwise, one step at a time. The room features windows that can be closed and locked. Initially, the position of the robot and the status of the windows are not known. The goal is to have all windows closed and locked. The number of initial states is $n \times 3^n$ and the optimal plan has $3n - 1$ steps. The parameter n used ranges from 3 to 8.
- **Sorting Networks:** The task is to build a circuit made of compare-and-swap gates that maps an input vector of n boolean variables into the corresponding sorted vector. The compare-and-swap action compares two entries in the input vector and swaps their contents if not ordered. The optimal serial plan minimizes the number of gates, while the optimal parallel plans minimizes the 'time delay' of the circuit. Only optimal plans for small n are known (Knuth, 1973). The number of initial states is 2^n. The parameter n used ranges from 2 to 7.
- **Square-center:** A robot without sensors moves in a room to north, south, east, and west, and its goal is to get to the middle of the room. The optimal serial plan for a grid of size n with an unknown initial location is to do $n - 1$ moves in one direction, $n - 1$ moves in an orthogonal direction, and then from the resulting corner, $n-2$ moves to the center for a total of $3 \times n-4$ steps. In the parallel setting, pairs of actions that move the robot in orthogonal directions are allowed. There are $2n$ initial states. The parameter n used ranges from 2^2 to 2^4.[14]
- **Blocks:** Refers to blocksworld domain with move-3 actions but in which the initial state is completely unknown. Actions are always applicable but have an effect only if their normal 'preconditions' are true. The goal is to get a fixed ordered stack with n blocks. The parameter n used ranges from 2 to 4, and the number of initial states is 3, 13 and 73 respectively.

[13]The algorithm presented on this chapter can be run using the option 'mc' of the conformant planner 'Translator' available at `http://www.ldc.usb.ve/~hlp/software`.

[14]This problem was called 'emptyroom' in the experiments reported by Rintanen (2004a). As described, this 'square-center' is the same used in the experiments of part III about translation from conformant into classical planning.

			search at horizon k			search at horizon $k-1$	
problem	N^*	$\#S_0$	time	backtracks	#act	time	backtracks
blocks-2	2	3	0	1	2	0	1
blocks-3	9	13	0.02	7	9	144.45	248619
blocks-4	26	73	> 2h	> 76029		> 2h	> 78714
square-center-4	8	16	0	0	8	0.02	243
square-center-8	20	64	0.05	0	20	> 2h	> 3741672
square-center-16	44	256	> 2h	> 188597		> 2h	> 191030
ring-3	8	81	0	0	8	0	5
ring-4	11	324	0.06	1	11	0.02	5
ring-5	14	1215	0.71	2	14	0.16	5
ring-6	17	4374	3.49	4	17	0.69	5
ring-7	20	15309	24.48	5	20	3.35	5
ring-8	23	52488	128.64	7	23	13.08	5
sortnet-3	3	8	0	0	3	0	5
sortnet-4	5	16	0	0	5	0.05	421
sortnet-5	9	32	0.02	0	9	> 2h	> 4845305
sortnet-6	12	64	0.2	1	12	> 2h	> 458912
sortnet-7	16	128	> 2h	> 102300		> 2h	> 104674

Table 3.2: Search data for serial planning for optimal horizon N^* (left) and and suboptimal horizon $N^* - 1$ (right). The columns show the optimal horizon, number of possible initial states, search time in seconds, number of backtracks, and number of actions in the plan. Rows with '$> 2h$' mean the search reached the cutoff time of 2 hours. All times are in seconds.

None of the problems feature preconditions, and only sorting and square-center admit parallel solutions (recall that we only allow parallel actions whose effects, ignoring their conditions, affect different variables).

The results that we report are collected in three tables. Table 3.1 reports the data corresponding to the compilation of the theories for serial planning. The last column shows the time taken for compiling each theory with the optimal horizon N^*, and the accumulated time for compiling each theory with horizon $N = 0, \ldots, N^*$. All theories compile: most in a few seconds, some in a few minutes, and only two of them – the largest ring and sort instance – take 33 and 13 minutes respectively. The accumulated times are also largest for these two instances, taking a total time of 3.4 and 1.7 hours. It is quite remarkable that all these theories actually compile; they are not trivial theories, with many featuring several thousands of variables and clauses, producing large d-DNNFs with millions of nodes in some cases (e.g., the largest sort and two largest ring instances). The largest instances, except for the ring instances, are probably beyond the reach of most conformant planners at the moment of doing this experiments (Palacios et al., 2005), whether optimal or not, with the planner in Rintanen (2004a) producing apparently the best results and solving most instances, except for the 3 most difficult sorting problems. From the point of view put forward in this chapter, this means that the compilation is not the bottleneck for solving these problems, but the search. However, in those cases where the d-DNNFs obtained are very large, the advantage of an informative and linear pruning criterion decreases, as the operations are linear on a structure that is very large (of course, there is no escape from this in the worst case, as checking the validity of a candidate conformant plan is hard). We also note though that some instances are quite challenging for conformant planning even if the size of their corresponding d-DNNFs are not that

			search at horizon k			search at horizon $k-1$	
problem	N^*	$\#S_0$	time	backtracks	#act	time	backtracks
square-center-4	4	16	0	54	8	0	9
square-center-8	10	64	1.26	952	20	39.82	20773
square-center-16	22	256	> 2h	> 235696		> 2h	> 240089
sortnet-4	3	16	0.01	38	6	0	29
sortnet-5	5	32	0.03	0	10	53.63	61469
sortnet-6	5	64	380.15	23884	14	> 2h	> 634880
sortnet-7	6	128	3.48	0	18	> 2h	> 84881

Table 3.3: Search data for parallel planning for optimal horizon N^* (left) and and suboptimal horizon $N^* - 1$ (right). The columns show the optimal horizon, number of possible initial states, search time in seconds, number of backtracks, and number of actions in the plan. Rows with '$> 2h$' mean the search reached the cutoff time of 2 hours. All times are in seconds.

large; this includes sortnet-6 and square-center-16.

Table 3.2 reports the data for the search for plans in the serial setting. On the left we show the results for the optimal horizon N^*, while on the right, for the horizon $N^* - 1$ for which there is no solution. In a sense, the first results show the difficulty of finding conformant plans; the second, the difficulty of proving them optimal. There are actually several examples in which plans are found for the optimal horizon which cannot be proved optimal in the immediately lower horizon; for example, square-center-8 and sortnet-6. The problems that are solved most easily are the ring problems that actually are the ones that have the largest d-DNNF representation. The reason is that the pruning criterion enables the solution of such instances with very few backtracks. On the other hand, the hardest block, square-center, and sortnet problems are not solved. By looking at the table, it appears that problems are solved with a few backtracks or are not solved at all. In principle it may be thought that this is because the pruning criterion is too expensive, and node generation rate is very low. Yet, the number of backtracks in some of the problems suggest the opposite: e.g., sortnet-5 cannot be proved optimal after almost 5 million backtracks, and similarly square-center-8. The complexity of the pruning operation, that grows linearly with the size of the d-DNNF representation explains however why the large unsolved instances reach the cutoff time with a smaller number of backtracks than the small unsolved instances.

Table 3.3 reports the data for the search for plans in the parallel setting. Given our simple model of parallelism where the only compatible actions are the ones that involve disjoint set of variables, it turns out that only the square-center and sortnet problems admit parallel solutions. In the first, one can take orthogonal directions at the same time; in the second, one can compare disjoint pairs of wires concurrently. The instances that get solved do not change significantly with respect to the serial setting, yet there are three interesting exceptions. One is square-center-8 which could not be solved before for the horizon $N^* - 1$ now is solved in less than 40 seconds with a relatively large number of backtracks: 20773 (by solving a problem in the horizon $N^* - 1$ we mean proving failure). This breaks the pattern observed earlier where problems were solved almost backtrack-free or not solved at all. In the same instance, the solution found for the optimal horizon N^* is obtained in a slightly more time, but with many more backtracks: 952. A possible explanation for this is that parallelism removes some symmetries in the problem, leaving an smaller space with fewer solutions. Thus, the proof for solutions become more difficult but the proofs for non-solutions become simpler. At the same time though, the parallel formulation makes sortnet-7 solvable in the optimal horizon. We are not solving sortnet-7 fully either; it is solved for the optimal horizon N^*, but not for $N^* - 1$ (which is always the most difficult horizon for proving the lack of solutions). Instead, sortnet-5 gets now solvable also for the horizon $N^* - 1$ taking under 54 seconds and a large number of backtracks: 61469.

From the benchmarks considered and reported in various papers, it was not simple to assess the performance of the proposed planner in relation to existing optimal and non-optimal ones at the moment of performing this evaluation (Palacios et al., 2005). It seems that various planners do well for some types of problems but not for others. In particular, the GPT planner (Bonet and Geffner, 2000) does well in problems where the *size* of the belief states that are reachable is small, and the heuristic V^*_{dp} that relaxes the problem assuming full observability, remains well informed (this also requires that the size of the state space not be too large either). The planner MBP reported in Cimatti et al. (2004) extends the scope of heuristic search planners by representing belief states symbolically as OBDDs; so it is not affected necessarily by the size of the state space nor by the size of the belief states. Still, when running in optimal model, MBP depends on the quality of a heuristic function similar to V^*_{dp}, and then it is also somewhat bound to problems where the assumption of full observability does not simplify the problem too much (for such problems actually a better informed admissible heuristic, although not fully automated is discussed in Cimatti et al. (2004)). The Conformant-FF planner recently introduced in Brafman and Hoffmann (2004) seems to perform best in problems that add a small amount of uncertainty in otherwise large classical planning problems, where the proposed, novel heuristic appears to work best. The problems that we have selected, which correspond to those considered in Rintanen (2004a), do not appear to exhibit these features, and gathering from the reported papers, it does not seem that these planners would do well on them, or even as well as vplan (with the exception of the ring problems, that involve large state spaces and large belief states, but where the heuristic V^*_{dp} remains well informed making the symbolic heuristic-search approach particularly suitable). In particular, we considered the 'cube-center' problem, a problem that extends the 'square-center' problem with another dimension. In Brafman and Hoffmann (2004) cubes of sizes up to $m = 3$ are reported solvable by Conformant-FF and cubes of sizes up to $m = 5$ are reported solvable by MBP. We ran this benchmark for vplan and as for square-center, we obtained better results in the parallel formulation, where some symmetries are broken. In this way, we were able to solve cube-center for $m = 7$ in less than a minute for the optimal horizon $N^* = 8$, proving the lack of solutions for the horizon $N^* - 1$ in 485 seconds.

The planner reported in Rintanen (2004a) does particularly well on the suite of problems considered, solving most of them very fast. The planner is a heuristic-search planner based on OBDD representations of belief states that uses a novel heuristic function obtained from relaxations that do not assume full observability. The relaxations appear to stand for conformant planning problems that differ from the original instance in their initial belief state. Rintanen solves the problem for all possible initial belief states with two states at most, and stores the resulting costs in memory. Then, the heuristic $h(b)$ of a belief state b is set to $\max_{b' \subseteq b} h'(b')$ where h' is the stored cost function, and b' is a belief state with at most two states in b. Unfortunately, Rintanen does not report data on the costs of preprocessing, but the results suggest that the heuristic obtained, while being more expensive, is better suited than the simpler heuristic V^*_{dp} for problems that involve some form of epistemic reasoning.

3.7 Discussion

We have developed an algorithm for conformant planning with deterministic actions that operates over logical encodings in which action literals are selected for branching, and branches that encode invalid action sets are pruned. The validity test checks at every node of the search tree whether the accumulated set of commitments (or action set) is consistent with each possible initial state and the planning theory. This ensures that the planner is sound and complete. Validity test, however, are informative but expensive. We showed how they can be reduced to projection and model count operations that can be carried out efficiently in the d-DNNF representation of the planning theory. The empirical results are encouraging, although we believe that there is still a lot of room for improvement.

Some goals for the future are: better ways for dealing with symmetries in the search space (there are plenty), better preprocessing (e.g., inference in the style of the planning graph capturing that certain actions literals cannot participate in a conformant plan), better criteria for selecting the action on which to branch (the planner is sensitive to this choice, and we should probably explore the use of one criterion for finding plans, and another one for proving optimality as done often in CSPs), incremental model counts in the search tree (taking advantage of the count performed in the parent node of the search tree), and other ways for using the d-DNNF representation to cut the search for plans even further.

The use of d-DNNF for conformant planning is further discussed at the end of the next chapter, after observing the performance of the algorithm proposed there.

Chapter 4

SAT Formulation

Sediento de saber lo que Dios sabe,
Judá León se dio a permutaciones
de letras y a complejas variaciones
y al fin pronunció el Nombre que es la Clave

Thirsty to see what God would see,
Judah Loew gave in to permutations
with letters in such complex variations
that he at last uttered the Name that is Key.

El Golem. Poem by Jorge Luis Borges[1]

In the previous chapter, we introduced an algorithm that starts by transforming the conformant planning problem P into a propositional theory $T_N(P)$ for an horizon N. We observed that verifying consistency of $T_N(P)$ through a SAT solver was not enough for discarding partial plans that cannot be extended to conformant plans. This prompted the introduction of a stronger criterion for discarding such partial plans, by model counting and projection over propositional theories, operations are made efficient by transforming the theory $T_N(P)$ into *Deterministic Decomposable Negational Normal Form* (d-DNNF, Darwiche and Marquis, 2002).

In this chapter we use the same propositional theory $T_N(P)$ and transform it into d-DNNF but, in contrast, we used it to obtain a new propositional theory $T_N(P)'$ such that getting a conformant plan requires *a single* SAT *call over such new theory.*

The chapter is organized as follows. We give a general introduction to the approach, referring to previous chapters for further details on conformant planning task definition, propositional encoding of planning theories, logic operations on propositional theories, and d-DNNF. Then, we study how to obtain a formula whose models corresponds to conformant plans, by using projection as a logical operation, and the use of d-DNNF as a compiled normal form that supports projection in linear time. Finally, we present the conformant planning algorithm, the experimental results, and a final discussion.

[1]Translation to English from http://www.syntheticzero.com/?p=629

The content of this chapter is based on a paper published by Palacios and Geffner (2006b).

4.1 Introduction

Conformant planning is computationally harder than classical planning, and unlike classical planning, cannot be reduced polynomially to SAT. Other SAT approaches to conformant planning follow a generate-and-test strategy (Ferraris and Giunchiglia, 2000): the models of the theory are generated one by one using a SAT solver (assuming a given planning horizon), and from each such model, a candidate conformant plan is extracted and tested for validity using another SAT call. This works well when the theory has few candidate plans and models, but otherwise is too inefficient. In this chapter, we propose a different use of a SAT engine where conformant plans are computed by means of *a single* SAT *call over a transformed theory.* This transformed theory is obtained by *projecting* the original theory over the action variables. Projection is the dual of variable elimination (also called forgetting or existential quantification): the projection of a formula over a subset of variables is the strongest formula over those variables; e.g., the projection of $((x \wedge y) \vee z)$ over $\{x, z\}$ is $x \vee z$. While projection is intractable, it can be done efficiently provided that the theory is in a certain canonical form such as *deterministic Decomposable Negated Normal Form* (d-DNNF Darwiche and Marquis, 2002), a form akin to OBDDs (Bryant, 1992).

Our scheme for planning is thus based on the following three steps: the planning theory in CNF is first compiled into d-DNNF, the compiled theory is then transformed into a new theory over the action variables only, and finally the conformant plan, if there is one, is obtained from this theory by a single invocation of a SAT engine. The experiments that are reported show that this COMPILE-PROJECT-SAT planner is competitive with state-of-the-art optimal conformant planners and improves upon the planner reported in the previous chapter.

Two optimal conformant planners are by Rintanen (2004a) and the one presented in Chapter 3 (Palacios et al., 2005). The first performs heuristic search in belief space with a powerful, admissible heuristic obtained by precomputing distances over belief states with at most two states. The second is a branch-and-prune planner that prunes partial plans that cannot comply with some possible initial state. This is achieved by performing model-count operations in linear-time over the d-DNNF representation of the theory. Both schemes assume that all uncertainty lies in the initial situation and that *all actions are deterministic.* In this chapter, we maintain this simplification which is not critical as non-deterministic effects can be eliminated by adding a polynomial number of hidden fluents, if the length of the plan is bounded. An appealing feature of the new conformant planning scheme is that it is based on the *two off-the-shelf components:* a d-DNNF compiler and a SAT solver.

We refer the reader to Chapter 2 for the definition of the conformant planning problem, and sections 3.2 to 3.5 for further details about propositional encoding of conformant problems, the projection logical operator, and the use of d-DNNF for computing such operations.

4.2 Conformant Planning and Models

In classical planning the relation between a problem P and its propositional encoding $T_N(P)$ is such that the models of $T_N(P)$ are in one-to-one correspondence with the plans that solve P (for the given horizon.In conformant planning, this correspondence no longer holds: the models of $T_N(P)$ encode 'optimistic plans', plans that work for *some* initial states and transitions but may fail to work for others, and hence are not conformant. However, we will see that it is possible to transform the theory $T_N(P)$ so that the models of the resulting theory are in correspondence with the conformant plans for P.

Let *Plan* denote a collection of *action literals* such that it mentions all action variables in theory $T_N(P)$, let *Init* denote the fragment of $T_N(P)$ encoding the initial situation, and let s_0 refer a possible initial state which we denote as $s_0 \in \mathit{Init}$[2]. Then for a *classical planning* problem P, *Plan* is a solution if and only if

$$T_N(P) \;\wedge\; \mathit{Plan} \qquad \text{is satisfiable.} \tag{4.1}$$

For a *conformant problem P with deterministic actions only*, on the other hand, *Plan* is a solution if and only if

$$\forall\; s_0 \in \mathit{Init} : \;\; T_N(P) \;\wedge\; \mathit{Plan} \;\wedge\; s_0 \qquad \text{is satisfiable.} \tag{4.2}$$

In other words, in the conformant setting, *Plan* must work for all possible initial states.

In order to find a *Plan* that complies with (4.1) it is enough to find a model of $T_N(P)$, and then set *Plan* to the set of action literals that are true in the model. On the other hand, for finding a *Plan* that complies with (4.2) this is not enough. As we will show, however, this will be enough when the theory $T_N(P)$ is transformed in a suitable way. As a first approximation, consider the problem of finding a *Plan* that complies with

$$T_N(P)' \;\wedge\; \mathit{Plan} \qquad \text{is satisfiable.} \tag{4.3}$$

where $T_N(P)'$ is a conjunction that takes into account *all* the initial states

$$T_N(P)' \quad = \quad \bigwedge_{s_0 \in \mathit{Init}} T_N(P) \,|\, s_0 \tag{4.4}$$

Here $T \,|\, X$ refers to theory T with variables x in T replaced by the value they have in state X: true if $x \in X$, and false if $\neg x \in X$. This operation is known as value substitution or *conditioning* (Definition 3.11 in Section 3.4 on page 33).

If equations (4.3) and (4.4) provided a correct formulation of conformant planning, we could obtain a conformant plan by finding a model for $T_N(P)'$, and extracting *Plan* from the value of the action variables in that model.

The formulation (4.3–4.4), however, is *not* correct. The reason is that the theory $T_N(P)$ contains fluent variables f_i for times $i > 0$ which are neither in *Init* nor in *Plan*. In (4.2), these variables can take different values for each s_0, while in (4.3–4.4), these variables are *forced* to take the *same* value over all possible s_0.

[2] s_0 denote a maximal consistent set of fluent literals f_0 compatible with *Init*

We can modify, however, the definition of $T_N(P)'$ in (4.4) for obtaining a correct SAT formulation of conformant planning. For this we need to *eliminate* or *forget* the fluent variables f_i, for $i > 0$, from each conjunct $T_N(P) \,|\, s_0$ in (4.4).

The forgetting of a set of variables S from a theory T (Lin and Reiter, 1994), also called elimination or existential quantification, is the dual operation to *Projection* of T over the *rest* of variables V; $V = vars(T) - S$. The projection of T over V, denoted $\mathsf{project}[\ T\ ;\ V\ ]$, refers to a theory over the variables V whose models are exactly the models of T restricted to those variables. For example, if $\phi = (a_1 \wedge f_1) \vee a_2$ then $\mathsf{project}[\,\phi;\ \{a_1, a_2\}\,] = a_1 \vee a_2$, which can also be understood as $\exists f_1 \phi \ = \ (\phi \,|\, f_1 = \mathsf{true}) \vee (\phi \,|\, f_1 = \mathsf{false}) \ = \ ((a_1 \wedge \mathsf{true}) \vee a_2) \vee ((a_1 \wedge \mathsf{false}) \vee a_2) \ = \ (a_1 \vee a_2)$. Projection was discussed in detail in section 3.4.

Getting rid of the fluent variables f_i for $i > 0$ in the conjuncts $T_N(P) \,|\, s_0$ in (4.4) simply means to project such formulas over the action variables, as the variables in $T_N(P) \,|\, s_0$ are either action variables or fluent variables f_i for $i > 0$ (the fluent variables f_i for $i = 0$ have been substituted by the their values in s_0).

The result is that the transformed theory $T_N(P)'$ becomes:

$$T_{\text{cf}}(P) \quad = \quad \bigwedge_{s_0 \in Init} \mathsf{project}[\ T_N(P) \,|\, s_0\ ;\ Actions\ ] \tag{4.5}$$

for which we can prove:

Theorem 4.1. *The models of $T_{cf}(P)$ in (4.5) are in one-to-one correspondence with the conformant plans for the problem P.*

Proof. Justifications and comments are enclosed by { *braces* }. Remember that the given a truth assignment for all action variables and variables of the initial situation, the theory $T_N(P)$ has either one model or none. Let T_A a model on $T_{\text{cf}}(P)$.

T_A satisfy $\bigwedge_{s_0 \in Init} \mathsf{project}[\ T_N(P) \,|\, s_0\ ;\ Actions\ ]$

iff { *Expanding. Observe that T_A contains exactly all action variables* }

For all $s_0 \in Init$: T_A satisfies $\mathsf{project}[\ T_N(P) \,|\, s_0\ ;\ Actions\ ]$

iff { *Lemma 3.12 (projection). T_A is unique given M because s_0 and action literals determine the theory $T_N(P)$* }

For all $s_0 \in Init$: Exists M that satisfies $T_N(P) \,|\, s_0$ such that $T_A \subseteq M$

iff { *Properties of conditioning* }

For all $s_0 \in Init$: Exists M that satisfies $T_N(P) \cup Lits(s_0)$ s.t. $T_A \subseteq M$

iff { *Lemma 3.8 on page 32 given that T_A is complete and consistent with $T_N(P) \cup Lits(s_0)$. Observe that π can be obtained from T_A and vice versa* }

For all $s_0 \in Init$: π is a classical plan for P/s_0

iff { *Definition of conformant planning* }

π is a conformant plan of P

□

Equation (4.5) suggests a simple *scheme* for conformant planning: construct the formula $T_{\text{cf}}(P)$ according to (4.5), and then feed this theory into a state-of-the-art SAT solver. The crucial point is the generation of $T_{\text{cf}}(P)$ from the original theory

$T_N(P)$: the transformation involves *conditioning* and *conjoining* operations, as well as *projections.* The key operation that is intractable is projection. Nevertheless, it is well known that projection, like many other intractable boolean transformations, can be performed in polynomial time provided that the theory is in a suitable compiled form (Darwiche and Marquis, 2002). Of course, the compilation itself may run in exponential time and space, yet this will not be necessarily so on average. We will actually show that the theory $T_{\mathrm{cf}}(P)$ in (4.5) can be obtained in time and space that is *linear* in the size of the d-DNNF compilation of $T_N(P)$. We refer the reader to Section 3.5 in the previous chapter for d-DNNF definition, and its support for operations like projection and conditioning.

4.3 A Conformant Planner based on SAT

Integrating the previous observations, the proposed conformant planner involves the following steps, given a horizon N.

1. A CNF theory $T_N(P)$ for horizon N is obtained from a PDDL-like description of the planning problem.
2. The theory $T_N(P)$ is **compiled** into the d-DNNF theory $T_{\mathrm{c}}(P)$
3. From $T_{\mathrm{c}}(P)$, the transformed theory

$$T_{\mathrm{cf}}(P) \quad = \bigwedge_{s_0 \in Init} \mathsf{project}[\ T_{\mathrm{c}}(P) \,|\, s_0 \ ;\ Actions\ ]$$

   is obtained by operations that are linear in time and space in the size of the DAG representing $T_{\mathrm{c}}(P)$. The resulting theory $T_{\mathrm{cf}}(P)$ is in NNF but is *not* decomposable due to the conjunction used to combine each formula $\mathsf{project}[\ T_{\mathrm{c}}(P) \,|\, s_0 \ ;\ Actions\ ]$, as the formulas share variables.

4. The NNF theory $T_{\mathrm{cf}}(P)$ is converted into CNF and a **sat solver** is called upon it.

This sequence of operations are summarized in Figure 4.1 on the following page and repeated starting from a planning horizon $N = 0$ which is increased by 1 until a solution is found.

Some of the details of the generation of the target theory $T_{\mathrm{cf}}(P)$ from the compiled theory $T_{\mathrm{c}}(P)$ are important. Normally, a simple way to compile an expression to d-DNNF is by recursively applying Shannon expansion over all variables in a theory T, as describe in Section 3.5:

$$T = (T \,|\, a) \vee (T \,|\, \neg a)$$

The first expansion leading to a d-DNNF appears in fig. 4.2, where $T \,|\, a$ and $T \,|\, \neg a$ are to be transformed using the same algorithm. Even though the process of compilation to d-DNNF includes many optimizations, the compiler can be set to do exactly that for some variables.

In particular for our algorithm, $T_N(P)$ is compiled into $T_{\mathrm{c}}(P)$ using an ordering of variables that expands the *Init* variables first; this is so that the DAG representing

```
Input: Conformant Problem $P = \langle F, O, I, G \rangle$
Input: Horizon $N$
Output: Print conformant plan $\pi$ for $P$ or return FALSE
$T_N(P) \leftarrow$ propositional translation of $P$ for $N$ time steps plans
$\Delta \leftarrow$ d-DNNF compilation of $T_N(P)$
$T_{\text{cf}} \leftarrow \bigwedge_{s_0 \in Init}$ project[ $\Delta \mid s_0$ ; $Actions$ ]
$T_{cnf} \leftarrow T_{\text{cf}}$ converted from NNF into CNF
$Model \leftarrow$ SAT($T_{cnf}$)
if $Model$ then
    PrintPlan($Model$)
else
    return FALSE
```

Figure 4.1: Algorithm for finding a conformant plan of N time steps using compilation to d-DNNF, projection, and one SAT solver call.

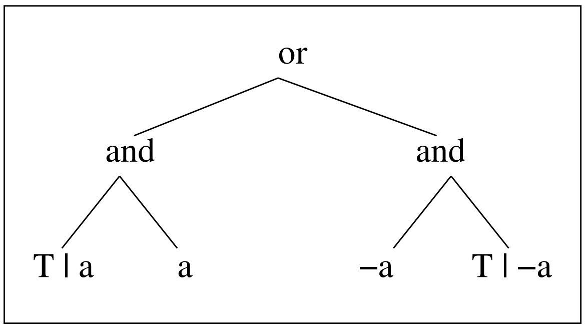

Figure 4.2: Partial compilation of T to d-DNNF using Shannon expansion

the d-DNNF subtheories $T_c(P) \mid s_0$ for each possible initial state s_0, all correspond to (non-necessarily disjoint) fragments of the DAG representing the compiled d-DNNF theory $T_c(P)$. Then the DAG representing the target NNF theory $T_{\text{cf}}(P)$ is obtained by conjoining these fragments. This last step requires to project the theory T over actions and variables of the initial situation $T_0(P)$, ensuring that the remaining variable after conditioning over states s_0 are only action variables. The compiler to d-DNNF supports such projection during compilation[3].

For example, if a and b were the only two variables appearing in *Init*, and there were exactly four possible initial states $\{a, b\}$, $\{a, \neg b\}$, $\{\neg a, b\}$ and $\{\neg a, \neg b\}$, the four subformulas corresponding to the conjuncts of equation (4.5) can be extracted as a subgraph, as illustrated in fig. 4.3.

The conjunction of these subgraph is in NNF *negational normal form*, but is no longer *decomposable* neither *deterministic*. In such case, verifying satisfiability would have been verifiable in linear time. Instead, we consider the DAG as a circuit and generate an equivalent CNF formula. This is done by creating propositional variables for each node of the DAG, and adding clauses for encoding the relations between nodes. If a parent node is labeled with *and*, then whenever all its children are true, the parent node should be true and vice versa. *Or* nodes are translated similarly.

[3]As in the algorithm in the previous chapter, all variables but actions and initial states were also forgot during compilation

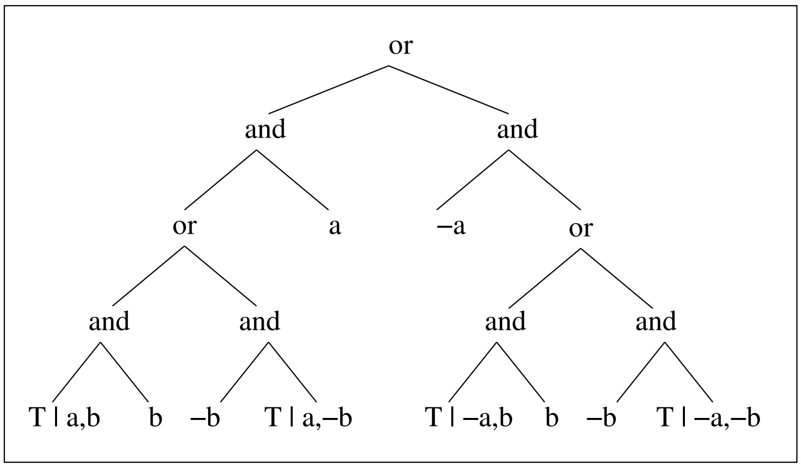

Figure 4.3: Partial compilation of T to d-DNNF for an initial state with variables a and b, and four possible initial states

4.4 Experimental Results

We performed experiments testing the proposed optimal conformant planner on a Intel/Linux machine running at 2.80GHz with 2GB of memory, in the same machine used in the experiments of previous chapter. We call satconf the planner based on the algorithm of this chapter.[4] Runs of the d-DNNF compiler and the SAT solver were limited to 2 hours and 1.8GB of memory. The d-DNNF compiler is Darwiche's c2d v2.18 (Darwiche, 2004), while the SAT solver is siege_v4 except for very large CNFs that would not load, and where *zChaff* was used instead. We used the same suite of problems used in the previous chapter and Rintanen (2004a). We provide a brief definition of the domains.

- **Ring:** A robot can move in n rooms arranged in a circle. The goal is to have all windows closed and locked.
- **Sorting Networks:** The task is to build a circuit of compare-and-swap gates to sort n boolean variables.
- **Square-center:** A robot without sensors can move in a grid of $n \times n$, and its goal is to get to the middle of the room. For this, it must first locate itself into a corner.
- **Cube-center:** Like the previous one, but in three dimensions.
- **Blocks:** Refers to the blocks-world domain with move-3 actions but in which the initial state is completely unknown. Actions are always applicable but have an effect only if their normal 'preconditions' are true. The goal is to get a fixed ordered stack with n blocks. None of the problems feature preconditions, and only sorting, square-center, and cube-center admit parallel solutions.

We report compilation and search times. The first is the time taken by the d-DNNF compiler; the second is the time taken by the SAT solver. For the search part, we

[4]The algorithm presented on this chapter can be run using the option 'sat' of the conformant planner 'Translator' available at `http://www.ldc.usb.ve/~hlp/software`.

		CNF theory		d-DNNF theory			$T_{\rm cf}(P)$	
problem	N^*	vars	cls	nodes	edges	time	vars	cls
ring-7	20	1081	3683	11M	3M	192.2	977k	3106k
ring-8	23	1404	4814	4M	9M	1177	4M	12M
blocks-3	9	444	2913	5242	21k	0.3	4667	24k
blocks-4	26	3036	41k	227k	889k	124.5	224k	1105k
square-center-8	20	976	3642	12k	23k	1.1	9664	28k
square-center-16	44	4256	17k	91k	175k	47.1	82k	239k
cube-center-9	33	2700	11k	283k	575k	98.9	277k	840k
cube-center-11	42	4191	17k	659k	1331k	371.6	648k	1959k
sortnet-7	16	1484	6679	116k	284k	12.4	113k	391k
sortnet-8	19	2316	13k	364k	896k	77.2	360k	1247k

Table 4.1: Compilation data and resulting CNF for serial formulation and optimal horizon N^*. On the left, the size of the theories $T_N(P)$ encoding the conformant planning problems, on the center, the size of the DAGs representing the compiled theories $T_c(P)$ and the times spent in the compilation; on the right, the size of the target theories $T_{\rm cf}(P)$ in CNF that are passed to the SAT engine. *cls* means the number of clauses. Suffix k and M means that the number has been divided by 1000 and 1000×1000, respectively, and rounded up.

show the results for both the optimal horizon N^* and $N^* - 1$. The first shows the difficulty of finding conformant plans; the second, the difficulty of proving them optimal. These times dominate the times consumed in the previous iterations.

In Table 4.1, we show results of the compilation for optimal horizons in the serial setting. The compilation of theories for smaller horizons or parallel formulations is normally less expensive. The table shows the optimal horizon N^* for each problem, the size of the original CNF theory $T_N(P)$, the size of the DAG representing the compiled theory $T_c(P)$ with the time spent in the compilation, and finally the size of the target theory $T_{\rm cf}(P)$ in CNF that is fed to the SAT solver. The first thing to notice is that all the problems considered by Rintanen (2004a) compile properly. Thus, as in Chapter 3, *the compilation is not the bottleneck.*

Table 4.2 shows the results of the SAT solver over the transformed theory $T_{\rm cf}(P)$ for both the optimal horizon N^* and $N^* - 1$, and for both the serial and parallel formulations. While not all problems are solved, the results improve upon those reported in Chapter 3, solving one additional instance in square-center and sorting, both in the serial and parallel setting. This represents an *order of magnitude* improvement over these domains. In blocks, on the other hand, there is no improvement, while the largest ring instances resulted in very large CNF theories that could not be loaded into Siege but were loaded and solved by zChaff (except for ring-r8 under the optimal planning horizon). In contrast, the algorithm based on model-counting did not exhaust the memory in the experiments reported.

4.5 Discussion

We presented a COMPILE-PROJECT-SAT scheme for computing optimal conformant plans. The scheme is simple and uses two off-the-shelf components: a d-DNNF compiler and a SAT solver. Given a conformant problem P and a horizon N, it first generate a propositional encoding $T_N(P)$, compile it into d-DNNF and generate a

			search with horizon k			horizon $k-1$	
problem	N^*	$\#S_0$	time	#dec	#act	time	#dec
serial theories							
ring-7	20	15309	° 2.1	2	20	° 0.8	0
ring-8	23	52488	> 1.8GB			° 2.4	0
blocks-3	9	13	0.1	1665	9	0.2	3249
blocks-4	26	73	$> 2h$			$> 2h$	
square-center-8	20	64	18.8	53k	20	207.4	208k
square-center-16	44	256	5184.4	1097k	44	$> 2h$	
cube-center-7	24	343	3771.5	579k	24	5574.2	737k
cube-center-9	33	729	$> 2h$			$> 2h$	
sortnet-5	9	32	0.0	352	9	22.0	35053
sortnet-6	12	64	40.0	35k	12	$> 2h$	
sortnet-7	16	128	3035.6	526k	16	$> 2h$	
sortnet-8	19	256	$> 2h$			$> 2h$	
parallel theories							
square-center-8	10	64	0.5	2737	20	0.3	1621
square-center-16	22	256	423.1	245k	44	1181.5	440k
cube-center-7	8	343	6.1	4442	24	2.9	1892
cube-center-9	11	729	114.6	28k	33	156.0	32760
cube-center-11	14	1331	> 1.8GB			181.5	13978
sortnet-7	6	128	46.1	19k	18	355.4	48264
sortnet-8	6	256	° 4256.6	534k	23	$> 2h$	

Table 4.2: Results for the Search: SAT calls over the transformed theory $T_{\rm cf}(P)$ for the optimal horizon N^* (left) and N^*-1 (right), both for serial and parallel formulations (when they differ). We show the number of initial states, the time spent on the SAT call, the number of decisions made, and the number of actions in the plan found. Entries '$> 2h$' and '> 1.8GB' mean time or memory exceeded. The sign ° indicates that the SAT solver used was *zChaff*, as `siege_v4` could not load $T_{\rm cf}(P)$ due to its size. Times are in seconds. A suffix k and M means that number has been divided by 1000 and 1000 × 1000, respectively, and rounded up.

new propositional formula whose models are all the conformant plans of N times steps for P. Calling a SAT-solver upon such new formula returns a conformant plan, if there is one for such N. We have shown that it improves on performance upon the model-counting-based algorithm of the previous chapter.

We have also explored a variation of the COMPILE-PROJECT-SAT scheme that may be more suitable for dealing with problems that are not that far from classical planning, such as those considered by Brafman and Hoffmann (2004). When the number of possible initial states s_0 is small, rather than getting rid of the fluent variables f_i, $i > 0$, by projection (as in Equation 4.5)

$$L \quad = \bigwedge_{s_0 \in Init} \mathsf{project}[\, T_N(P) \,|\, s_0 \ ; \ Actions \,], \tag{4.6}$$

it may be more convenient to introduce copies of them, one for each possible initial state $s0$, resulting in the formula

$$L' \quad = \bigwedge_{s_0 \in Init} [\, T_N(P)^{s_0} \,|\, s_0 \,] \tag{4.7}$$

where each $T_N(P)^{s_0}$ denotes a theory which is like $T_N(P)$ except that the fluent variables f_i, $i > 0$, are replaced by fresh copies $f_i^{s_0}$. Action variables, on the other hand, remain shared among all these theories. It can be shown that models of L' as well as the models of L, are in one-to-one correspondence with the conformant plans that solve the problem. The latter approach, which does not require projection or compilation, may work better when the number of possible initial states is low, and collapses to the standard SAT approach to classical planning when there is uncertainty. However, the results were not competitive in the limited tests we performed. This idea is related to the translation $K_0(P)$ of conformant problems into classical ones proposed in Chapter 6, were each atom is labelled with all possible initial states.

We transform the resulting formula in NNF to CNF, as SAT solvers for CNF are capable of solving industrial size formulas, albeit we tried with a non-clausal solver that accepts circuits (Thiffault et al., 2004) but we had syntactic problems that could not be overcome. It is encouraging the SAT competition includes now a track on solving circuits (Sinz and Jain, 2008), as in the future the resulting products of knowledge compilation operation over d-DNNF and OBDD may be feed directly into non-clausal SAT solvers, hopefully leading to better performance.

In this and the previous chapters the d-DNNF compilation step was not a bottleneck for the performance of the proposed algorithms. However, in both cases we refined the algorithms for model-counting, projection and conditioning in order to achieve good performance. The main reason is that even though these operations can be performed in time linear in the size of the d-DNNF graph, these graphs can be quite big. In general, the effective use of d-DNNF depends on being able to compile the propositional formulas, and obtaining a small enough compiled formula. Our compilation strategy, following the chronological order of the propositional variables, was critical for achieving good performance.

As in many other problem solving areas, symmetries appear in planning. For example, given two simple conformant planning problems, adding them up in a single up will cause quadratic increase on the number of possible initial states. Both algorithms presented in this part may find the new problem much harder than the two simple ones. Symmetries seems to play a role in the bad performance of both algorithms in the ring problem, for example.

Part III

Conformant Planning into Classical Planning

Chapter **5**

A Basic Translation to Classical Planning

But it happened that after walking for a long time through sand, and rocks, and snow, the little prince at last came upon a road. And all roads lead to the abodes of men.

The Little Prince.
Novel by Antoine de Saint-Exupéry

In this part of the dissertation we introduce an alternative approach to conformant planning where problems are automatically compiled into classical problems and solved by a classical planner. This approach provides an implicit solution to the two problems faced by conformant planners that search in belief space (Bonet and Geffner, 2000): the belief representation and the heuristic over beliefs (see Section 2.8). In the translation approach to classical planning, belief states are represented as plain states, allowing standard classical planning heuristics to be used.

In the translation-based approach to conformant planning considered in this chapter, beliefs are represented by literals KL that aim to represent that a literal L is known to be true with certainty. In addition, and since belief states are represented as plain states, the heuristic over beliefs is a classical heuristic. From a computational point of view, though, there is no explicit search in belief-space: conformant problems P are converted into classical problems $K_0(P)$ at the 'knowledge-level' (Petrick and Bacchus, 2002), whose solutions, computed by a classical planner, encode the conformant solutions for P. However, the translation of this chapter is incomplete, meaning that a problem P may have a conformant plan with no corresponding classical plan for its translation $K_0(P)$. In the next chapter we extend the translation $K_0(P)$ to make complete.

The content of this chapter is based on ideas first published by Palacios and Geffner (2006a).

5.1 Introduction

The problem of conformant planning can be formulated as a deterministic search problem in belief space, where a sequence of actions that map a given initial belief state bel_0 into a target set of beliefs is sought. A belief state bel represents the set of states s that are deemed possible, and actions a, whether deterministic or not, deterministically map one belief state bel into another, denoted as bel_a (Bonet and Geffner, 2000) (more details in Section 2.7). Since the number of belief states is exponential in the number of states, it is clear that the search for conformant plans takes place in a space that is exponentially larger than the search for classical plans (see section Complexity, 2.6). .

A way to trade off completeness for efficiency in conformant planning results from approximating belief states or transitions. For example, the 0-approximation introduced by Baral and Son (1997) represents belief states bel by means of two sets: the set of literals that are true in bel, and the set of literals that are false in bel. Variables which do not appear in either set are unknown. In this representation, checking whether an action a is applicable in bel, computing the next belief state bel_a, and verifying polynomial length plans are all polynomial time operations. Roughly, a fluent literal L makes it into bel_a iff a) action a has some conditional effect $C \rightarrow L$ such that all literals in C are in bel, or b) L is in bel and for all conditional effects $C' \rightarrow \neg L$ of action a, the complement of some literal $L' \in C'$ is in bel.

Conformant planning under the 0-approximation is thus no more complex, theoretically, than classical planning. The problem however is that the 0-approximation is strongly incomplete, as it does not capture any non-trivial form of disjunctive inference. For example, given a disjunction $p \vee q$ and an action a that maps either p or q into r, the semantics will not validate a as a conformant plan for r. Indeed, disjunctions that are not tautologies are thrown away. The 0-semantics does capture, on the other hand, situations in which the information that is missing is not relevant. For example, if there are actions that can make a variable p true or false, then uncertainty in the initial state of p would not hurt. Classical planners, on the other hand, cannot handle such situations.

Another sound but incomplete approach to planning with incomplete information is presented by (Petrick and Bacchus, 2002) where belief states bel are represented by more complex formulas which may include disjunctions. Yet in order to make belief updates efficient several approximations are introduced, and in particular, while existing disjunctions can be carried from one belief state to the next and can be simplified, no new disjunctions are added. This too imposes a serious limitation in the type of problems that can be handled.

Expressivity, however, is not the only problem; *efficiency* or control is the other. Indeed, it is not enough to introduce restrictions that under polynomial length constraints bring the complexity of conformant planning to that of classical planning or SAT; the control knowledge needed for solving the resulting problem must be made available as well. The approach by (Petrick and Bacchus, 2002) leave this problem largely unaddressed relying on a blind search over compact belief representations and efficient update rules. Recent elaborations of the 0-approximation by (Son et al., 2005b; Tran et al., 2009) rely in turn on a fixed heuristic function that counts the number of goals achieved, which applies well to some problems but not to others.

In this part of the dissertation, we aim to address both problems, expressivity and control, by introducing incomplete and complete mappings to classical planning. We refer the reader to Chapter 2 on page 13 for the definition of the conformant planning task. The formulations in this part of the dissertation are limited to conformant problems that are *deterministic* and where all uncertainty lies in the initial situation.

5.2 A Basic Translation K_0

A simple translation of the conformant problem P into a classical problem $K(P)$ can be obtained by replacing the literals L by literals KL and $K\neg L$ aimed at capturing whether L is known to be true and known to be false respectively.

Definition 5.1 (Translation K_0)**.** *For a conformant planning problem $P = \langle F, I, O, G\rangle$, the translation $K_0(P) = \langle F', I', O', G'\rangle$ is a classical planning problem with*

- $F' = \{KL, K\neg L \mid L \in F\}$
- $I' = \{KL \mid L$ *is a unit clause in* $I\}$
- $G' = \{KL \mid L \in G\}$
- $O' = O$ *but with each precondition L for $a \in O$ replaced by KL, and each conditional effect $a : C \rightarrow L$ replaced by $a : KC \rightarrow KL$ and $a : \neg K\neg C \rightarrow \neg K\neg L$,*

where the expressions KC and $\neg K\neg C$ for $C = L_1, L_2 \ldots$ are abbreviations of the formulas $KL_1, KL_2 \ldots$ and $\neg K\neg L_1, \neg K\neg L_2 \ldots$ respectively.

The intuition behind the translation is simple: first, the literal KL is true in the initial state I' if L is known to be true in I; otherwise it is false. This removes all uncertainty from $K_0(P)$, making it into a classical planning problem. In addition, for soundness, each rule $a : C \rightarrow L$ in P is mapped into *two* rules: a **support rule** $a : KC \rightarrow KL$, that ensures that L is known to be true when the condition is known to be true, and a **cancellation rule** $a : \neg K\neg C \rightarrow \neg K\neg L$ that guarantees that $K\neg L$ is deleted (prevented to persist) when action a is applied and C is not known to be false. The use of support and cancellation rules for encoding the original rules at the 'knowledge-level' is the only subtlety in the translation.

The translation $K_0(P)$ is sound as every classical plan that solves $K_0(P)$ is a conformant plan for P, but is incomplete, as not all conformant plans for P are classical plans for $K(P)$. The meaning of the KL literals follows a similar pattern: if a plan achieves KL in $K_0(P)$, then the same plan achieves L with certainty in P, yet a plan may achieve L with certainty in P without making the literal KL true in $K_0(P)$.[1]

Proposition 5.2 (Soundness of $K_0(P)$)**.** *If π is a classical plan for $K_0(P)$, then π is a conformant plan for P.*

As an illustration, consider the conformant problem $P = \langle F, I, O, G\rangle$ with $F = \{p, q, r\}$, $I = \{q\}$, $G = \{p, r\}$, and actions $O = \{a, b\}$ with effects

$$a : q \rightarrow r \ , \ a : p \rightarrow \neg p \ , \ b : q \rightarrow p \ .$$

[1] Formal proofs for this chapter can be found in the appendix A, since page 155

For this problem, the action sequence $\pi = \{a, b\}$ is a conformant plan for P while the action sequence $\pi' = \{a\}$ is not. Indeed, π is a classical plan for P/s for any possible initial state s, while π' is not a classical plan for the possible initial state s' where p is true (recall that s is a possible initial state of P if s satisfies I so that neither p nor r are assumed to be initially false in this problem).

From Definition 5.1, the translation $K_0(P) = \langle F', I', O', G'\rangle$ is a classical planning problem with fluents $F' = \{Kp, K\neg p, Kq, K\neg q, Kr, K\neg r\}$, initial situation $I' = \{Kq\}$, goals $G' = \{Kp, Kr\}$, and actions $O' = \{a, b\}$ with effects

$$a : Kq \rightarrow Kr\ ,\ a : Kp \rightarrow K\neg p\ ,\ b : Kq \rightarrow Kp,$$

that encode supports, and effects

$$a : \neg K\neg q \rightarrow \neg K\neg r\ ,\ a : \neg K\neg p \rightarrow \neg Kp\ ,\ b : \neg K\neg q \rightarrow \neg K\neg p,$$

that encode cancellations.

Proposition 5.2 implies, for example, that $\pi' = \{a\}$, which is not a conformant plan for P, cannot be a classical plan for $K(P)$ either. This is easy to verify, as while the support $a : Kq \rightarrow Kr$ achieves the goal Kr as Kq is true in I', the cancellation $a : \neg K\neg p \rightarrow \neg Kp$ associated with the same action, preserves Kp false for the other goal p.

While the translation K_0 is not complete, meaning that it fails to capture all conformant plans for P as classical plans, its completeness can be assessed in terms of a weaker semantics. In the so-called 0-approximation semantics (Baral and Son, 1997), belief states b are represented by 3-valued states where fluents can be true, false, or unknown. In this incomplete belief representation, checking whether an action a is applicable in a belief state b, computing the next belief state b_a, and verifying polynomial length plans are all polynomial time operations. In particular, a literal L is true it the next belief state b_a iff a) action a has some effect $C \rightarrow L$ such that all literals in C are true in b, or b) L is true in b and for all effects $C' \rightarrow \neg L$ of action a, the complement of some literal $L' \in C'$ is true in b. An action sequence π is then *a conformant plan for P according to the 0-approximation semantics* if the belief sequence generated by π according to the 0-approximation semantics makes the action sequence applicable and terminates in a belief state where the goals are true. It is possible to prove then that:

Proposition 5.3 ($K_0(P)$ and 0-Approximation)**.** *An action sequence π is a classical plan for $K_0(P)$ iff π is a conformant plan for P according to the 0-approximation semantics.*

This correspondence is not surprising though as both the 0-approximation semantics and the $K_0(P)$ translation throw away the disjunctive information and restrict the plans to those that make no use of the uncertain knowledge. Indeed, the states s_0, s_1, ... generated by the action sequence $\pi = \{a_0, a_1, \ldots\}$ over the classical problem $K_0(P)$ encode precisely the literals that are known to be true according to the 0-approximation; namely, L is true at time i according to the 0-approximation iff the literal KL is true in the state s_i.

Proposition 5.3 does not mean that the translation K_0 and the 0-approximation semantics are equivalent but rather that they both rely on equivalent belief representations. The translation K_0 delivers also a way to get valid conformant plans using a classical planner. The translation-based approach thus addresses both the representational and the heuristic issues that arise in conformant planning.

As an illustration of Proposition 5.3, given a conformant problem P with $I = \{p, r\}$ and actions a and b with effects $a : p \rightarrow q$, $a : r \rightarrow \neg v$, and $b : q \rightarrow v$, the plan $\pi = \{a, b\}$ is valid for achieving the goal $G = \{q, v\}$ according to both $K_0(P)$ and the 0-approximation, while the plan $\pi = \{b\}$ is not valid according to either. At the same time, if the initial situation is changed to $I = \{p \vee q\}$, neither approach sanctions the plan $\pi = \{a\}$ for $G = \{q\}$, even if it is a valid conformant plan. For this, some ability to reason with disjunctions is needed.

We postpone to Section 8.2 an extension to the basic translation K_0 that allows a limited form of disjunctive reasoning. That extension is based on the introduction of new literals L/X_i used for encoding the conditionals $X_i \supset L$ (Palacios and Geffner, 2006a). In the next chapter, K_0 is extended in a different manner that ensures both tractability and completeness over a large class of problems in a very simple way (Palacios and Geffner, 2007, 2009).

Chapter **6**

Complete Translations to Classical Planning

Viajan conmigo mis amigos muertos.
Adónde llego, van por todas partes,
apresurados me siguen, me preceden,
gentiles, cómodos e incómodos,
en grupos, solos, conversando, paseando.

My friends who are dead are travelling with me.
Each place I go they're there, they are all around,
dashing wildly to catch up, they're way ahead,
in comfort, in discomfort, in great style,
in groups, alone, talking, out walking.

The absent ones.
Poem by Eugenio Montejo[1]

In last chapter we introduced a scheme where conformant problems P are automatically converted into classical ones $K_0(P)$ and solved by an *off-the-shelf* classical planner. In this chapter we extend such translation by mapping literals L and sets of assumptions t about the initial situation, into new literals KL/t that represent that L *must be true if* t *is initially true.* We lay out a general translation scheme that is sound and establish the conditions under which the translation is also complete. We show that the complexity of the complete translation is exponential in a parameter of the problem that we call the *conformant width*, which for most benchmarks will turn out to be bounded.

This chapter is based on a paper published by Palacios and Geffner (2009), a revision and extension of the formulation originally presented by Palacios and Geffner (2007).

[1]English translation from "The trees: selected poems, 1967-2004" By Eugenio Montejo. Cambridge, U.K. : Salt, 2004

6.1 Introduction

In this chapter we present a new translation that maps sets of literals t about the initial situation and literals L into new literals KL/t that express that

if t is true in the initial situation, L must be true.

We lay out first a general translation scheme that is sound and then establish the conditions under which the translation is also complete. Also, we show that the complexity of the complete translation is exponential in a parameter of the problem that we call the *conformant width*, which for most benchmark domains is bounded, implying that the complete translation in those cases is polynomial. The planner based on this translation exhibits good performance in comparison with existing conformant planners and is the basis for T_0, the best performing planner in the Conformant Track of the 2006 *International Planning Competition* (IPC-2006).

The translation-based approach provides a solution to the two problems faced by conformant planners that search in belief space: the belief representation and the heuristic over beliefs. In the translation-based approach, the beliefs are represented by the literals KL/t that stand for conditionals, a representation that is polynomial and complete for conformant problems with bounded width. In addition, and since belief states are represented as plain states, the heuristic over beliefs is a classical heuristic. From a computational point of view, though, there is no explicit search in belief-space: conformant problems P are converted into classical problems $K(P)$ at the 'knowledge-level' (Petrick and Bacchus, 2002), whose solutions, computed by a classical planner, encode the conformant solutions for P.

Even though this formulation is limited to conformant problems that are *deterministic* and where all uncertainty lies in the initial situation, in Section 8.3 on page 127 we address the issues that must be handled in order to generalize the approach presented in this chapter to non-deterministic domains, and report empirical results over non-deterministic domains as well.

The chapter is organized as follows. Based on the sound but incomplete translation K_0 (Chapter 5), we consider a more general translation scheme $K_{T,M}$ where T and M are two parameters, a set of tags t encoding assumptions about the initial situation, and a set of merges m encoding valid disjunctions of tags (Section 6.2), and analyze several instances of this scheme that follow from particular choices of the sets of tags and merges: a complete but exponential translation K_{S0} where tags are associated with the possible initial states of the problem (Section 6.3), and a polynomial translation K_i for a fixed integer $i \geq 0$ that is complete for problems with conformant width bounded by i (Section 6.4). We provide then an alternative explanation for this compact but complete translation by showing that in problems in problems with bounded width, the exponential number of possible initial states S_0 includes always a polynomial number of 'critical' initial states S'_0 such that plans that conform with S'_0, conform also with S_0 (Section 6.5). Formal proofs are leave to the appendix A. We left for the next chapter the presentation of the conformant planner T_0 based on the translation $K_{T,M}(P)$.

6.2 General Translation Scheme $K_{T,M}$

The basic translation K_0 is extended now into a general translation scheme $K_{T,M}$ where T and M are two parameters: a set of *tags* t and a set of merges m. We will show that for suitable choices of these two parameters, the translation $K_{T,M}$, unlike the translation K_0, can be both sound and complete.

A tag $t \in T$ is a set (conjunction) of literals L from P whose truth value in the initial situation is not known. The tags t are used to introduce a new class of literals KL/t in the classical problem $K_{T,M}(P)$ that represent the conditional 'if t is true initially, then L is true', an assertion that could be written as $K(t_0 \supset L)$ in a temporal modal logic. We use the notation KL/t rather than L/t as used by Palacios and Geffner (2006a), because there is a distinction between $\neg KL/t$ and $K\neg L/t$: roughly $\neg KL/t$ means that the conditional $K(t_0 \supset L)$ is not true, while $K\neg L/t$ means that the conditional $K(t_0 \supset \neg L)$ is true.

Likewise, a merge m is a non-empty collection of tags t in T that stands for the Disjunctive Normal Form (DNF) formula $\bigvee_{t \in m} t$. A merge m is *valid* when one of the tags $t \in m$ must be true in I; i.e., when

$$I \models \bigvee_{t \in m} t \quad .$$

A merge m for a literal L in P will translate into a 'merge action' with a single effect

$$\bigwedge_{t \in m} KL/t \quad \rightarrow \quad KL$$

that captures a simple form of reasoning by cases.

While a valid merge can be used for reasoning about any literal L in P, computationally it is convenient (although not logically necessary) to specify that certain merges are to be used with some literals L and not with others. Thus, formally, M is a collection of pairs (m, L), where m is a merge and L is a literal in P. Such a pair means that m is a merge for L. We group all the merges m for a literal L in the set M_L, and thus, M can be understood as the collection of such sets M_L for all L in P. For simplicity, however, except when it may cause a confusion, we will keep referring to M as a plain set of merges.

We assume that the collection of tags T always includes a tag t that stands for the empty collection of literals, that we call the *empty tag* and denote it as $\emptyset$. If t is the empty tag, we denote KL/t simply as KL.

The translation $K_{T,M}(P)$ is the basic translation $K_0(P)$ 'conditioned' with the tags t in T and extended with the actions that capture the merges in M:

Definition 6.1 (Translation $K_{T,M}$). *Let $P = \langle F, I, O, G\rangle$ be a conformant problem, then $K_{T,M}(P)$ is the* classical planning problem *$K_{T,M}(P) = \langle F', I', O', G'\rangle$ with*

- $F' = \{KL/t, K\neg L/t \mid L \in F \text{ and } t \in T\}$
- $I' = \{KL/t \mid \ I, t \models L\}$
- $G' = \{KL \mid L \in G\}$

- $O' = \{a : KC/t \to KL/t,\ a : \neg K\neg C/t \to \neg K\neg L/t \mid a : C \to L \text{ in } P\} \ \cup$ $\{a_{m,L} : [\bigwedge_{t\in m} KL/t] \to KL \wedge XL \mid L \in P, m \in M_L\}$

where KL is a precondition of action a in $K_{T,M}(P)$ if L is a precondition of a in P, the conditions KC/t and $\neg K\neg C/t$ stand for $KL_1/t, KL_2/t, \ldots,$ and $\neg K\neg L_1/t, \neg K\neg L_2/t, \ldots$ respectively, when $C = L_1, L_2, \ldots,$ and XL stands for $\bigwedge_{L'} K\neg L'$ with L' ranging over the literals L' mutex with L in P.

The translation $K_{T,M}(P)$ reduces to the basic translation $K_0(P)$ when M is empty and T contains only the empty tag. The extra effects $XL = \bigwedge_{L'} K\neg L'$ in the merge actions $a_{m,L}$ are needed only to ensure that the translation $K_{T,M}(P)$ is consistent when P is consistent, and otherwise can be ignored. Indeed, if L and L' are mutex in a consistent P, the invariant $KL/t \supset K\neg L'/t$ holds in $K_{T,M}(P)$ for non-empty tags t, and hence a successful merge for L can always be followed by a successful merge for $\neg L'$. In the rest of the chapter we will thus assume that both P and $K_{T,M}(P)$ are consistent, and ignore such extra merge effects. We refer to the appendix B where we prove the consistency of $K_{T,M}(P)$ from the consistency of P.

For suitable choices of T and M, the translation $K_{T,M}(P)$ will be *sound and complete.* Before establishing these results, however, let us make these notions precise.

Definition 6.2 (Soundness)**.** *A translation $K_{T,M}(P)$ is sound if for any classical plan π that solves the classical planning problem $K_{T,M}(P)$, the plan π' that results from π by dropping the merge actions is a conformant plan for P.*

Definition 6.3 (Completeness)**.** *A translation $K_{T,M}(P)$ is complete if for any conformant plan π' that solves the conformant problem P, there is a classical plan π that solves the classical problem $K_{T,M}(P)$ such that π' is equal to π with the merge actions removed.*

The general translation scheme $K_{T,M}$ is sound provided that all merges are valid and all tags are consistent (literals in a tag are all true in some possible initial state):[2]

Theorem 6.4 (Soundness $K_{T,M}(P)$)**.** *The translation $K_{T,M}(P)$ is sound provided that all merges in M are valid and all tags in T are consistent.*

Unless stated otherwise, we will assume that all merges are valid and all tags consistent, and will call such translations, *valid translations.*

As a convention for keeping the notation simple, in singleton tags like $t = \{p\}$, the curly brackets are often dropped. Thus, literals KL/t for $t = \{p\}$ are written as KL/p, while merges $m = \{t_1, t_2\}$ for singleton tags $t_1 = \{p\}$ and $t_2 = \{q\}$, are written as $m = \{p, q\}$.

Example. As an illustration, consider the problem of moving an object from an origin to a destination using two actions: *pick*(l), that picks up an object from a location if the hand is empty and the object is in that location, and *drop*(l), that drops the object at a location if the object is being held. For making the problem more interesting, let us also assume that the action *pick*(l) drops the object being

[2] Formal proofs for this chapter can be found in the appendix A, since page 155

held at l if the hand is not empty. These are all conditional effects and there are no action preconditions. Assuming that there is a single object, these effects can be written as:

$$\begin{aligned}
&\mathit{pick}(l) : \neg \mathit{hold}, \mathit{at}(l) \rightarrow \mathit{hold} \wedge \neg \mathit{at}(l) \\
&\mathit{pick}(l) : \mathit{hold} \rightarrow \neg \mathit{hold} \wedge \mathit{at}(l) \\
&\mathit{drop}(l) : \mathit{hold} \rightarrow \neg \mathit{hold} \wedge \mathit{at}(l) \ .
\end{aligned}$$

Consider now an instance P of this domain, where the hand is initially empty and the object, initially at either l_1 or l_2, must be moved to l_3; i.e., $P = \langle F, I, O, G\rangle$ with

$$I = \{\neg \mathit{hold}\,,\ \mathit{at}(l_1) \vee \mathit{at}(l_2)\,,\ \neg \mathit{at}(l_1) \vee \neg \mathit{at}(l_2)\,,\ \neg \mathit{at}(l_3)\}$$

and

$$G = \{\mathit{at}(l_3)\} \ .$$

The action sequence

$$\pi_1 = \{\mathit{pick}(l_1), \mathit{drop}(l_3), \mathit{pick}(l_2), \mathit{drop}(l_3)\}$$

is a conformant plan for this problem, where an attempt to pick up the object at location l_1 is followed by a drop at the target location l_3, ensuring that the object ends up at l_3 if it was originally at l_1. This is then followed by an attempt to pick up the object at l_2 and a drop at l_3.

On the other hand, the action sequence π_2 that results from π_1 by removing the first drop action

$$\pi_2 = \{\mathit{pick}(l_1), \mathit{pick}(l_2), \mathit{drop}(l_3)\}$$

is not a conformant plan, since if the object was originally at l_1, it would end up at l_2 after the action $\mathit{pick}(l_2)$. In the notation introduced above, π_1 is a classical plan for the classical problem P/s for the two possible initial states s, while π_2 is a classical plan for the problem P/s but only for the state s where the object is initially at l_2.

Consider now the classical problem $K_{T,M}(P) = \langle F', I', O', G'\rangle$ that is obtained from P when $T = \{\mathit{at}(l_1), \mathit{at}(l_2)\}$[3] and M contains the merge $m = \{\mathit{at}(l_1), \mathit{at}(l_2)\}$ for the literals hold and $\mathit{at}(l_3)$. From its definition, the fluents F' in $K_{T,M}(P)$ are of the form KL/t and $K\neg L/t$ for $L \in \{\mathit{at}(l), \mathit{hold}\}$, $l \in \{l_1, l_2\}$, and $t \in T$, while the initial situation I' is

$$I' = \left\{ \begin{array}{l} K\neg \mathit{hold}, K\neg \mathit{hold}/\mathit{at}(l), K\neg \mathit{at}(l_3), K\neg \mathit{at}(l_3)/\mathit{at}(l), \\ K\mathit{at}(l)/\mathit{at}(l), K\neg \mathit{at}(l')/\mathit{at}(l) \end{array} \right\}$$

for $l, l' \in \{l_1, l_2\}$ and $l' \neq l$, and the goal G' is

$$G' = \{K\mathit{at}(l_3)\} \ .$$

The effects associated to the actions $\mathit{pick}(l)$ and $\mathit{drop}(l)$ in O' are the support rules

$$\begin{aligned}
&\mathit{pick}(l) : K\neg \mathit{hold},\ K\mathit{at}(l) \ \rightarrow\ K\mathit{hold} \wedge K\neg \mathit{at}(l) \\
&\mathit{pick}(l) : K\mathit{hold} \ \rightarrow\ K\neg \mathit{hold} \wedge K\mathit{at}(l) \\
&\mathit{drop}(l) : K\mathit{hold} \ \rightarrow\ K\neg \mathit{hold} \wedge K\mathit{at}(l)
\end{aligned}$$

[3]The empty tag is assumed in every T and thus it is not mentioned explicitly.

for each one of the three locations $l = l_i$, that condition each rule in O with the empty tag, along with the support rules:

$$\begin{aligned}
&\mathit{pick}(l) : K\neg \mathit{hold}/\mathit{at}(l'),\ K\mathit{at}(l)/\mathit{at}(l') \ \rightarrow\ K\mathit{hold}/\mathit{at}(l') \wedge K\neg \mathit{at}(l)/\mathit{at}(l')\\
&\mathit{pick}(l) : K\mathit{hold}/\mathit{at}(l') \ \rightarrow\ K\neg \mathit{hold}/\mathit{at}(l') \wedge K\mathit{at}(l)/\mathit{at}(l')\\
&\mathit{drop}(l) : K\mathit{hold}/\mathit{at}(l') \ \rightarrow\ K\neg \mathit{hold}/\mathit{at}(l') \wedge K\mathit{at}(l)/\mathit{at}(l')
\end{aligned}$$

that condition each rule in O with the tags $\mathit{at}(l') \in T$, for $l' \in \{l_1, l_2\}$. The corresponding cancellation rules are:

$$\begin{aligned}
&\mathit{pick}(l) : \neg K\mathit{hold},\ \neg K\neg \mathit{at}(l) \ \rightarrow\ \neg K\neg \mathit{hold} \wedge \neg K\mathit{at}(l)\\
&\mathit{pick}(l) : \neg K\neg \mathit{hold} \ \rightarrow\ \neg K\mathit{hold} \wedge \neg K\neg \mathit{at}(l)\\
&\mathit{drop}(l) : \neg K\neg \mathit{hold} \ \rightarrow\ \neg K\mathit{hold} \wedge \neg K\neg \mathit{at}(l)
\end{aligned}$$

and

$$\begin{aligned}
&\mathit{pick}(l) : \neg K\mathit{hold}/\mathit{at}(l'), \neg K\neg \mathit{at}(l)/\mathit{at}(l') \ \rightarrow\ \neg K\neg \mathit{hold}/\mathit{at}(l') \wedge \neg K\mathit{at}(l)/\mathit{at}(l')\\
&\mathit{pick}(l) : \neg K\neg \mathit{hold}/\mathit{at}(l') \ \rightarrow\ \neg K\mathit{hold}/\mathit{at}(l') \wedge \neg K\neg \mathit{at}(l)/\mathit{at}(l')\\
&\mathit{drop}(l) : \neg K\neg \mathit{hold}/\mathit{at}(l') \ \rightarrow\ \neg K\mathit{hold}/\mathit{at}(l') \wedge \neg K\neg \mathit{at}(l)/\mathit{at}(l')\ .
\end{aligned}$$

In addition, the actions in O' include the merge actions $a_{m,\mathit{hold}}$ and $a_{m,\mathit{at}(l_3)}$ that follow from the merge $m = \{\mathit{at}(l_1), \mathit{at}(l_2)\}$ in M for the literals hold and $\mathit{at}(l_3)$:

$$\begin{aligned}
&a_{m,\mathit{hold}} : K\mathit{hold}/\mathit{at}(l_1), K\mathit{hold}/\mathit{at}(l_2) \ \rightarrow\ K\mathit{hold}\\
&a_{m,\mathit{at}(l_3)} : K\mathit{at}(l_3)/\mathit{at}(l_1), K\mathit{at}(l_3)/\mathit{at}(l_2) \ \rightarrow\ K\mathit{at}(l_3)\ .
\end{aligned}$$

It can be shown then that the plan

$$\pi_1' = \{\mathit{pick}(l_1), \mathit{drop}(l_3), \mathit{pick}(l_2), \mathit{drop}(l_3), a_{m,\mathit{at}(l_3)}\}$$

solves the classical problem $K_{T,M}(P)$ and hence, from Theorem 6.4, that the plan π_1 obtained from π_1' by dropping the merge action, is a valid conformant plan for P (shown above). We can see how some of the literals in $K_{T,M}(P)$ evolve as the actions in π_1' are executed:

0:	$K\mathit{at}(l_1)/\mathit{at}(l_1), K\mathit{at}(l_2)/\mathit{at}(l_2)$	true in I'
1:	$K\mathit{hold}/\mathit{at}(l_1), K\mathit{at}(l_2)/\mathit{at}(l_2)$	true after $\mathit{pick}(l_1)$
2:	$K\mathit{at}(l_3)/\mathit{at}(l_1), K\mathit{at}(l_2)/\mathit{at}(l_2)$	true after $\mathit{drop}(l_3)$
3:	$K\mathit{at}(l_3)/\mathit{at}(l_1), K\mathit{hold}/\mathit{at}(l_2)$	true after $\mathit{pick}(l_2)$
4:	$K\mathit{at}(l_3)/\mathit{at}(l_1), K\mathit{at}(l_3)/\mathit{at}(l_2)$	true after $\mathit{drop}(l_3)$
5:	$K\mathit{at}(l_3)$	true after merge $a_{m,\mathit{at}(l_3)}$.

We can also verify in the same manner that the action sequence π_2'

$$\pi_2' = \{\mathit{pick}(l_1), \mathit{pick}(l_2), a_{m,\mathit{hold}}, \mathit{drop}(l_3)\}$$

is not a classical plan for $K_{T,M}(P)$, the reason being that the atom $K\mathit{hold}/\mathit{at}(l_1)$ holds after the first pick up action but not after the second. This is due to the cancellation rule:

$$\mathit{pick}(l_2) : \neg K\neg \mathit{hold}/\mathit{at}(l_1) \rightarrow \neg K\mathit{hold}/\mathit{at}(l_1)\ \wedge\ \neg K\neg \mathit{at}(l_2)/\mathit{at}(l_1)$$

Problem	$\#S_0$	K_{S0} time	K_{S0} len	POND time	POND len	CFF time	CFF len
adder-01	18	$> 2h$		0,4	26	$> 2h$	
blocks-02	18	0,2	23	0,4	26	$> 2h$	
blocks-03	231	59,2	80	126,8	129	$> 2h$	
bomb-10-1	1k	5,9	19	1	19	0	19
bomb-10-5	1k	11,3	15	3	15	0	15
bomb-10-10	1k	18,3	10	8	10	0	10
bomb-20-1	1M	$> 2.1GB$		4139	39	0	39
coins-08	1k	20,2	27	2	28	0	28
coins-09	1k	19,9	25	5	26	0	26
coins-10	1k	21,5	31	5	28	0,1	38
coins-11	1M	$> 2.1GB$		$> 2h$		1	78
comm-08	512	18,3	61	1	53	0	53
comm-09	1k	77,7	68	1	59	0	59
comm-10	2k	$> 2.1GB$		1	65	0	65
corners-square-16	4	0,2	102	1131	67	13,1	140
corners-square-24	4	0,7	202	$> 2h$		321	304
corners-square-28	4	1,2	264	$> 2h$		$> 2h$	
corners-square-116	4	581,4	3652	$> 2h$		$> 2h$	
corners-square-120	4	$> 2.1GB$		$> 2h$		$> 2h$	
square-center-16	256	13,1	102	1322	61	$> 2h$	
square-center-24	576	$> 2.1GB$		$> 2h$		$> 2h$	
log-2-10-10	1k	183,5	85	$> 2h$		1,6	83
log-3-10-10	59k	$> 2h$		$> 2h$		4,7	108
ring-5	1,2k	12,6	17	6	20	4,3	31
ring-6	4,3k	$> 2.1GB$		33	27	93,6	48
safe-50	50	0,5	50	9	50	29,4	50
safe-70	70	1,4	70	41	70	109,9	70
safe-100	100	6	100	$> 2.1GB$		1252,4	100
sortnet-07	256	2,9	28	480	25	SNH	
sortnet-08	512	9,8	36	$> 2h$		SNH	
sortnet-09	1k	77,7	45	$> 2h$		SNH	
sortnet-10	2k	$> 2.1GB$		$> 2h$		SNH	
uts-k-08	16	0,6	46	24	47	4,4	46
uts-k-10	20	1,2	58	2219	67	16,5	58

Table 6.1: K_{S0} translation fed into FF planner compared with POND and Conformant FF (CFF) along both times and reported plan lengths. $\#S_0$ stands for number of initial states, 'SNH' means goal syntax not handled (by CFF). Times reported in seconds and rounded to the closest decimal.

that expresses that under the assumption $at(l_1)$ in the initial situation, $hold$ and $\neg at(l_2)$ are not known to be true after the action $pick(l_2)$, if under the same assumption, $\neg hold$ was not known to be true before the action.

6.3 A Complete Translation: K_{S0}

A *complete* instance of the translation scheme $K_{T,M}$ can be obtained in a simple manner by setting the tags to the possible initial states of the problem P and by having a merge for each precondition and goal literal L that includes all these tags. We call the resulting 'exhaustive' translation K_{S0}:

Definition 6.5 (Translation K_{S0}). *For a conformant problem P, the translation $K_{S0}(P)$ is an instance of the translation $K_{T,M}(P)$ where*

- *T is set to the union of the empty tag and the set S_0 of all possible initial states of P (understood as the maximal sets of literals that are consistent with I), and*
- *M is set to contain a single merge $m = S_0$ for each precondition and goal literal L in P.*

The translation K_{S0} is valid and hence sound, and it is complete due the correspondence between tags and possible initial states:

Theorem 6.6 (Completeness of K_{S0}). *If π is a conformant plan for P, then there is a classical plan π' for $K_{S0}(P)$ such that π is the result of dropping the merge actions from π'.*

For problems P whose actions have no preconditions, the argument is simple: if π is a conformant plan for P then π must be a classical plan for P/s for each possible initial state s, but then if π achieves the (goal) literal G_i in P/s for each s, π must achieve the literal KG_i/s in $K_{S0}(P)$ for each s as well, so that π followed by the merge action for G_i, must achieve the literal KG_i. In the presence of action preconditions, this argument must be applied inductively on the plan length, but the idea remains the same (see the proof in the appendix for details): a correspondence can be established between the evolution of the fluents L in each problem P/s and the evolution of the fluents KL/s in the problem $K_{S0}(P)$.

The significance of the exhaustive K_{S0} translation is not only theoretical. There are plenty of conformant problems that are quite hard for current planners even if they involve a handful of possible initial states. An example of this is the Square-Center-n task (Cimatti and Roveri, 2000), where an agent has to reach the center of an empty square grid with certainty, not knowing its initial location. There are four actions that move the agent one unit in each direction, except when in the border of the grid, where they have no effects. In the standard version of the problem, the initial position is fully unknown resulting in n^2 possible initial states, yet the problem remains difficult, and actually beyond the reach of most planners, for small values of n, even when the uncertainty is reduced to *a pair of possible initial states.* The reason is that the agent must locate itself before heading for the goal. The domain Corners-Square-n in Table 6.1 is a variation of Square-Center-n where the possible initial states are the four corners of the grid.

Table 6.1 shows results for a conformant planner based on the $K_{S0}(P)$ translation that uses FF (Hoffmann and Nebel, 2001) for solving the resulting classical problem, and compares it with two of the planners that entered the Conformant track of the IPC-2006 (Bonet and Givan, 2006): POND (Bryce et al., 2006) and Conformant FF (Hoffmann and Brafman, 2006) (the other two planners in the competition were translation-based: T_0, based on the formulation developed in this chapter, and $K(P)$, based on an earlier and more restricted formulation and presented in section 8.2 (Palacios and Geffner, 2006a)). Clearly, the approach based on the $K_{S0}(P)$ translation does not scale up to problems with many possible initial states, yet when the number of such states is small, it does quite well.

6.4 Complete Translations that May be Compact Too

In order to have complete translations that are polynomial, certain assumptions about the formulas in the initial situation I need to be made. Otherwise, just checking whether a goal is true in I is intractable by itself, and therefore a polynomial but complete translation would be impossible (unless P = NP). We will thus assume that I is in *prime implicate (PI) form* (Marquis, 2000), meaning that I includes only the inclusion-minimal clauses that it entails but no tautologies. It is known that checking whether a clause follows logically from a formula I in PI form reduces to checking whether the clause is subsumed by a clause in I or is a tautology, and hence is a polynomial operation. The initial situations I in most benchmarks is in PI form or can easily be cast into PI form as they are normally specified by means of a set of non-overlapping $oneof(X_1, \ldots, X_n)$ expressions that translate into clauses $X_1 \vee \cdots \vee X_n$ and binary clauses $\neg X_i \vee \neg X_j$ for $i \neq j$ where any resolvent is a tautology.

Conformant Relevance

The translation $K_{S0}(P)$ is complete but introduces a number of literals KL/t that is exponential in the worst case: one for each possible initial state s_0. This raises the question: is it possible to have complete translations that are not exhaustive in this sense? The answer is yes and in this section we provide a simple condition that ensures that a translation $K_{T,M}(P)$ is complete. It makes use of the notion of relevance:[4]

Definition 6.7 (Relevance)**.** *The conformant relevance relation $L \longrightarrow L'$ in P, read L is relevant to L', is defined inductively as*

1. $L \longrightarrow L$
2. $L \longrightarrow L'$ if $a : C \rightarrow L'$ is in P with $L \in C$ for some action a in P
3. $L \longrightarrow L'$ if $L \longrightarrow L''$ and $L'' \longrightarrow L'$
4. $L \longrightarrow L'$ if $L \longrightarrow \neg L''$ and $L'' \longrightarrow \neg L'$.

The first clause stands for reflexivity, the third for transitivity, the second captures conditions that are relevant to the effect, and the fourth, the conditions under which L preempts conditional effects that may delete L'. If we replace 4 by

4' $L \longrightarrow L'$ if $\neg L \rightarrow \neg L'$

which is equivalent to 4 in the context of 1–3, the resulting definition is the one by Son and Tu (2006), where the notion of relevance is used to generate a limited set of possible 'partial' initial states over which the 0-approximation is complete (see Section 6.5 for a discussion on the relation between tags and partial initial states).

[4]While we follow an earlier account (Palacios and Geffner, 2007), many of the definitions and theorems differ in a number of details (for example, the notion of relevance depends on the rules in P but not on the clauses in the initial situation). The changes are aimed at making the resulting formulation simpler and cleaner.

Notice that according to the definition, a precondition p of an action a is not taken to be 'relevant' to an effect q. The reason is that we want the relation $L \longrightarrow L'$ to capture the conditions under which *uncertainty about* L is relevant *to the uncertainty about* L'. This is why we say this is a relation of *conformant relevance.* Preconditions must be known to be true in order for an action to be applied, so they do not introduce nor propagate uncertainty into the effects of an action.

If we let C_I stand for the set of clauses representing uncertainty about the initial situation, namely, the non-unit clauses in I along with the tautologies $L \vee \neg L$ for complementary literals L and $\neg L$ not appearing as unit clauses in I, the notion of (conformant) relevance can be extended to clauses as follows:

Definition 6.8 (Relevant Clauses)**.** *A clause* $c \in C_I$ *is relevant to a literal* L *in* P *if all literals* $L' \in c$ *are relevant to* L*. The set of clauses in* C_I *relevant to* L *is denoted as* $C_I(L)$*.*

Having a representation of the uncertainty in the initial situation that is relevant to a literal L, it is possible to analyze the completeness of a translation $K_{T,M}$ in terms of the relation between the merges m for the literals L, on one hand, and the sets of clauses $C_I(L)$ that are relevant to L on the other.

Covering Translations

It may appear that a translation $K_{T,M}$ would be complete when the merges m for precondition and goal literals L, understood as the DNF formulas $\bigvee_{t \in m} t$, contain as much information, and thus are equivalent to the CNF formula $C_I(L)$ that captures the fragment of the initial situation I that is relevant to L. This intuition is partially correct, but misses one important point; namely that not every DNF formula equivalent to $C_I(L)$ will do: the DNF representation captured by the merges must be 'vivid' enough. For example, if $C_I(L)$ is the single clause $x \vee \neg x$, completeness requires a tag for x, a tag for $\neg x$, and a merge $m = \{x, \neg x\}$ for L containing the two tags, even if the clause $x \vee \neg x$ is a tautology and is thus equivalent to the DNF formula *true*.

For defining the types of tags and merges that are required for completeness then, let us first define the *closure* S^* of a set of literals S, relative to a conformant problem $P = \langle F, I, O, G \rangle$, as the set of literals that follow from S and I:

$$S^* \; = \; \{L \mid I, S \models L\} \;\; .$$

Let us also say that S is *consistent* if S^* does not contain a pair of complementary literals.

The type of merges m required for precondition and goal literals L are then those that do not only imply $C_I(L)$ but that *satisfy* it as well. The notion of satisfaction associates a consistent set of literals S with the *partial truth assignment* that is implicit in the closure S^* of S, and is extended to account for the conditions under which a DNF formula (e.g., a merge for L) satisfies a CNF formula (e.g., $C_I(L)$).

Definition 6.9 (Satisfaction)**.** 1. *A consistent set of literals* S satisfies a clause $L_1 \vee L_2 \vee \cdots \vee L_m$ *if* S^* *contains one of the literals* L_i*,* $i = 1, \ldots, m$*.*

2. *A consistent set of literals S* satisfies a collection of clauses $\mathcal{C}$ *if S satisfies each clause in $\mathcal{C}$.*

3. *A collection $\mathcal{S}$ of consistent sets of literals* satisfies *a collection of clauses $\mathcal{C}$ if each set S in $\mathcal{S}$ satisfies $\mathcal{C}$.*

The type of merges required for completeness are then simply the valid merges m that satisfy the set of clauses $C_I(L)$. We call them *covering merges*:

Definition 6.10 (Covering Merges)**.** *A valid merge m in a translation $K_{T,M}(P)$* covers *a literal L if m satisfies $C_I(L)$.*

For example, if $C_I(L)$ is given by the clauses that result from a $oneof(x_1, \ldots, x_n)$ expression, i.e. $x_1 \vee x_2 \vee \cdots \vee x_n$ and $\neg x_i \vee \neg x_j$ for all i and j, $1 \leq i, j \leq n$, $i \neq j$, then the merge $m = \{x_1, \ldots, x_n\}$ covers the literal L, as each x_i^* not only includes x_i but also $\neg x_j$ for all $j \neq i$, and thus x_i^* satisfies $C_I(L)$.

If for a merge $m = \{t_1, \ldots, t_n\}$, we denote by m^* the DNF formula $\bigvee_{t_i \in m} t_i^*$, where each tag t_i is replaced by its closure t_i^*, then it is simple to prove that if m covers the literal L, m^* entails $C_I(L)$. A merge m that covers L is thus a DNF formula that is strong enough to imply the CNF formula $C_I(L)$ (through the closure), weak enough to be entailed by I, and vivid enough to satisfy $C_I(L)$.

As a further illustration, if $C_I(L)$ is given by the tautologies $p \vee \neg p$ and $q \vee \neg q$, and $I = C_I(L)$, the merge $m_1 = \{p, \neg p\}$ implies $C_I(L)$ but does not satisfy $C_I(L)$. Likewise, the merge $m_2 = \{\{p, q\}, \{\neg p, \neg q\}\}$ satisfies $C_I(L)$ but is not entailed by I. Finally, the merge $m_3 = \{\{p, q\}, \{p, \neg q\}, \{\neg p, q\}, \{\neg p, \neg q\}\}$ satisfies $C_I(L)$ and is entailed by I, and thus is a valid merge that covers L.

If a valid translation $K_{T,M}(P)$ contains a merge m that covers L for each precondition and goal literal L in P, we say that the translation *covers P* or just that it is a *covering translation*:

Definition 6.11 (Covering Translation)**.** *A covering translation is a valid translation $K_{T,M}(P)$ that includes one merge that covers L, for each precondition and goal literal L in P.*

A central result of the chapter is that covering translations are complete:

Theorem 6.12 (Completeness)**.** *Covering translations $K_{T,M}(P)$ are complete; i.e., if π is a conformant plan for P, then there is a classical plan π' for $K_{T,M}(P)$ such that π is π' with the merge actions removed.*

In other words, complete translations $K_{T,M}(P)$ result when the tags and merges in T and M capture the information in the initial situation that is relevant to each precondition and goal literal in a suitable manner.

Theorem 6.12 can be used in two ways: for proving the completeness of a translation, by checking that the covering condition holds, and for constructing complete translations, by enforcing the covering condition. In addition, while our interest in this chapter is on conformant planning with no optimality guarantees, the theorem is useful for *optimal conformant planning* as well, whether the cost of plans is defined as their length (action costs equal to 1) or as the sum of non-uniform action costs.

In both cases, the theorem ensures that the problem of optimal conformant planning gets mapped into a problem of optimal classical planning provided that the cost of the merge actions in $K_{T,M}(P)$ is made sufficiently small.

As an illustration of Theorem 6.12, consider the conformant problem P with initial situation $I = \{x_1 \vee \cdots \vee x_m\}$, goal $G = L$, and actions $a_i, i = 1, \ldots, m$, each with effect $x_i \rightarrow L$. The number of possible initial states for this problem is exponential in m, as the disjunction among the x_i's is not exclusive. So, the translation $K_{S0}(P)$ is complete but exponential in size. On the other hand, consider the translation $K_{T,M}(P)$ where $T = \{x_1, \ldots, x_m\}$ and M contains the single valid merge $m = \{x_1, \ldots, x_m\}$ for L. It is simple to verify that this merge covers the goal L (satisfies $C_I(L) = I$), and hence that the translation $K_{T,M}(P)$ is covering, and by Theorem 6.12, complete, while being polynomial in m.

Notice that testing whether a valid translation $K_{T,M}(P)$ is a covering translation can be done in polynomial time, as in particular, computing the set of literals t^* from every tag t in T is a tractable operation provided that I is in PI form; indeed, $I, t \models L'$ iff $I \models t \supset L'$ iff $\neg t \vee L'$ is a tautology or is subsumed by a clause in I.

Translation $K_{\mathbf{models}}$

It is straightforward to show that the exponential translation K_{S0} considered in Section 6.3, where (non-empty) tags stand for the possible initial states, is covering and hence complete according to Theorem 6.12. It is possible, however, to take further advantage of Theorem 6.12 for devising a complete translation that is usually more compact. We call it K_{models}.

Definition 6.13. *The translation $K_{models}(P)$ is obtained from the general scheme $K_{T,M}(P)$ by defining*

- *M to contain one merge m for each precondition and goal literal L given by the models of $C_I(L)$ that are consistent with I,*[5] *and*
- *T to contain the tags in all such merges along with the empty tag.*

The translation K_{models} is equivalent to K_{S0} when for all the precondition and goal literals L, $C_I(L) = I$; i.e., when all the clauses in I are relevant to L. Yet, in other cases, the first translation is exponential in the number of variables appearing in one such $C_I(L)$ set (the one with the largest number of such variables), while the second is exponential in the number of unknown variables in I. For example, if there are n precondition and goal literals L_i, $i = 1, \ldots, n$ in P such that for each one, $C_I(L_i)$ is a unique $oneof(x_1^i, \ldots, x_m^i)$ expression, the merge for the literal L_i in $K_{S0}(P)$ will contain the m^n models of the n one-of expressions in I, while the merge for L_i in $K_{\text{models}}(P)$ will just contain the m models of the single $oneof(x_1^i, \ldots, x_m^i)$ expression in $C_I(L_i)$. The translation K_{models}can thus be exponentially more compact than the exhaustive K_{S0} translation while remaining sound and complete:

Theorem 6.14. *The translation $K_{models}(P)$ is sound and complete.*

In the worst case, however, K_{models} is also an exponential translation. We thus consider next *polynomial* translations and the conditions under which they are complete.

[5]The models of $C_I(L)$ are to be understood as conjuntions of literals.

Conformant Width

We address now the conditions under which a compact, covering translation can be constructed in polynomial time. For this, we define a structural parameter that we call the *conformant width* of a problem P, that in analogy to the notion of width used in graphical models (Dechter, 2003), will provide an upper bound on the time and space complexity required for generating a covering translation. More precisely, the complexity of this construction will be exponential in the conformant width of the problem P that cannot exceed the number of fluents in P but can be much lower.

In principle, we would like to define the width $w(P)$ as the maximum tag size required in a translation $K_{T,M}(P)$ to be a covering translation. Such a definition, however, would not give us the complexity bounds that we want, as just checking the validity of a merge with tags of bounded size is an intractable operation, whether the initial situation I is in prime implicate form or not.[6] So we need to define width in a different way. First, let the *cover* of a set of clauses be defined as follows:

Definition 6.15 (Cover)**.** *The cover $c(\mathcal{C})$ of a set of clauses $\mathcal{C}$, relative to a conformant problem P with initial situation I, is the collection of all minimal sets of literals S consistent with I such that S contains a literal of each clause in $\mathcal{C}$.*

Two important properties of the cover $c(\mathcal{C})$ of a set of clauses $\mathcal{C}$ are that $c(\mathcal{C})$ stands for a DNF formula that is logically equivalent to the CNF formula $\mathcal{C}$ given I, and that $c(\mathcal{C})$ can be computed in polynomial time if the size of $\mathcal{C}$ is bounded by a constant. Moreover, $c(\mathcal{C})$ not only implies $\mathcal{C}$ but *satisfies* $\mathcal{C}$ as well. Thus in particular, if $\mathcal{C}$ is the collection of clauses $C_I(L)$ that are relevant to the literal L, the cover $c(C_I(L))$ of $C_I(L)$ is a valid merge that covers L. From this and the completeness of covering translations, it follows that a complete translation $K_{T,M}(P)$ can be constructed in polynomial time if the size $|C_I(L)|$ of the sets of clauses $C_I(L)$ for all precondition and goal literals L in P is bounded. Unfortunately, this condition rarely seems to hold, yet there is a weaker sufficient condition that does: namely, it is often possible to find a subset $\mathcal{C}$ of clauses that are either in $C_I(L)$ or are tautologies such that $c(\mathcal{C})$ satisfies $C_I(L)$ and thus covers the literal L. We thus define the *width of the literal* L as the size of the smallest such set (cardinality-wise). For this, we denote by $C^*_I(L)$ the set of clauses $C_I(L)$ extended with tautologies of the form $p \vee \neg p$ for fluents p such that either p or $\neg p$ appears in $C_I(L)$ (if both appear in $C_I(L)$ then $p \vee \neg p$ is in $C_I(L)$ from its definition).

Definition 6.16 (Width of Literal)**.** *The conformant width of a literal L in P, written $w(L)$, is the size of the smallest (cardinality-wise) set of clauses $\mathcal{C}$ in $C^*_I(L)$ such that $c(\mathcal{C})$ satisfies $C_I(L)$.*

A consequence of this definition is that the width of a literal must lie in the interval $0 \leq w(L) \leq n$, where n is the number of fluents in P whose status in the initial situation is not known. Indeed, if $C_I(L)$ is empty, $w(L) = 0$, while for any set of

[6] The problem of checking whether I entails a DNF formula whose terms may have more than 2 literals is coNP-hard even if I is equivalent to true. Indeed, if Φ is a 3-CNF formula; Φ is contradictory iff its negation $\neg\Phi$ (which is in 3-DNF) is valid, which in turn is true iff $\neg\Phi$ is implied by I. Actually, for a general I in prime implicate form, the problem remains coNP-hard even if the terms of the DNF formula contain at most 2 literals. We thank Pierre Marquis for pointing these results to us.

clauses $C_I(L)$, the cover $c(\mathcal{C})$ of the set $\mathcal{C}$ of tautologies in $C_I^*(L)$ must satisfy $C_I(L)$, and thus $w(L) \leq |\mathcal{C}| \leq n$. Similarly, if $C_I(L)$ contains a single clause $x_1 \vee \cdots \vee x_m$ or the clauses $x_1 \vee \cdots \vee x_m$ and $\neg x_i \vee \neg x_j$ that correspond to the $oneof(x_1, \ldots, x_m)$ expression, it is simple to prove that $w(L) = 1$ with the singleton $\mathcal{C} = \{x_1 \vee \cdots \vee x_m\}$ generating the cover $c(\mathcal{C}) = \{\{x_1\}, \ldots, \{x_n\}\}$ that satisfies $C_I(L)$. Finally, if $C_I(L)$ contains the two tautologies $p \vee \neg p$ and $q \vee \neg q$, $w(L) = 2$ as the smallest $\mathcal{C}$ in $C_I^*(L)$ whose cover satisfies $C_I(L)$ is $C_I(L)$ itself.

The width of a problem is the width of the precondition or goal literal with maximum width:

Definition 6.17 (Width of Problem)**.** *The conformant width of a problem P, written as $w(P)$, is $w(P) = \max_L w(L)$, where L ranges over the precondition and goal literals in P.*

We show below that for problems with bounded width, complete translations can be constructed in polynomial time, and moreover, that almost all existing conformant benchmarks have bounded width, and more precisely, width equal to 1. In such a case, the resulting translations will use tags that are never greater in size than $w(P)$, so that for problems with width 1, tags will be single literals.

Like for the (tree)width of graphical models, computing the width of a problem P is exponential in $w(P)$, so the recognition of problems with small width can be carried out quite efficiently:

Proposition 6.18 (Determining Width)**.** *The width $w(P)$ of P can be determined in time that is exponential in $w(P)$.*

In particular, we can test if $w(P) = 1$ by considering one by one each of the sets $\mathcal{C}$ that includes a single clause from $C_I^*(L)$, verifying whether $c(\mathcal{C})$ satisfies $C_I(L)$ or not. If $w(P) \not\leq 1$, then the same verification must be carried out by setting $\mathcal{C}$ to each set of i clauses in $C_I^*(L)$ for increasing values of i. For a fixed value of i, there is a polynomial number of such clause sets $\mathcal{C}$ and the verification of each one can be done in polynomial time. Moreover, from the arguments above regarding $w(L)$, $w(P)$ can never exceed the number of unknown fluents in the problem:

Proposition 6.19 (Bounds on Width)**.** *The width of P is such that $0 \leq w(P) \leq n$, where n is the number of fluents whose value in the initial situation is not known.*

Polynomial Translation K_i

The translation K_i, where the parameter i is a non-negative integer, is an instance of the general $K_{T,M}$ scheme designed to be sound, polynomial for a fixed i, and complete for problems with width $w(P) \leq i$. Thus, for example, the translation K_1 is sound, polynomial, and complete for problems with width 1.

Definition 6.20 (Translation K_i)**.** *The translation $K_i(P)$ is obtained from the general scheme $K_{T,M}(P)$ where*

- *M is set to contain one merge $m = c(\mathcal{C})$ for each precondition and goal literal L in P if there is a set $\mathcal{C}$ of at most i clauses in $C_I^*(L)$ such that m covers L.*

If no such set exists, one merge $m = c(\mathcal{C})$ for L is created for each set $\mathcal{C}$ of i clauses in $C^*_I(L)$, and no merges are created for L if $C^*_I(L)$ is empty;

- *T is the collection of tags appearing in those merges and the empty tag.*

The translation $K_i(P)$ applies to problems P of any width, remaining in all cases exponential in i but polynomial in the number of fluents, actions, and clauses in P. In addition, the translation $K_i(P)$ is sound, and for problems with width bounded by i, complete.

Theorem 6.21 (Properties K_i)**.** *For a fixed i, the translation $K_i(P)$ is sound, polynomial, and if $w(P) \leq i$, covering and complete.*

Soundness is the result of the merges being all valid by construction, as the covers $c(\mathcal{C})$ for any $\mathcal{C}$ in $C^*_I(L)$ are entailed by $\mathcal{C}$ and hence by I. The complexity is polynomial for a fixed i, because there is a polynomial number of clause sets $\mathcal{C}$ of size i in $C^*_I(L)$, and constructing the cover $c(\mathcal{C})$ for each one of them, is a polynomial operation. Finally, completeness follows from the definition of width: if $w(P) \leq i$, then there is a set of clauses $\mathcal{C}$ in $C^*_I(L)$ with size $|\mathcal{C}|$ no greater than i whose cover satisfies $C_I(L)$, and thus M in $K_i(P)$ must contain a merge $m = c(\mathcal{C})$ for L that covers L.

Notice that for $i = 0$, the translation $K_i(P)$ reduces to the basic $K_0(P)$ translation introduced in Section 6.3 that has no tags (other than the empty tag) and no merges. Before, we assessed the completeness of this translation in terms of the 0-approximation semantics. Theorem 6.21 provides an alternative interpretation: the translation $K_0(P)$ is complete for problems P with zero width. These are the problems for which the set of clauses $C_I(L)$ relevant to a precondition or goal literal L is empty. This makes precise the intuition mentioned above that the $K_0(P)$ translation is complete for problems where the uncertain information in I is not relevant. In such cases, none of the clauses in the initial situation I make it into the sets of relevant clauses $C_I(L)$ for preconditions and goal literals L.

As an illustration of Theorem 6.21, consider again the conformant problem P with initial situation $I = \{x_1 \vee \cdots \vee x_m\}$, goal $G = \{L\}$, and actions $a_i, i = 1, \ldots, m$, each with effect $x_i \rightarrow L$. For this problem, the singleton set of clauses $\mathcal{C} = C_I(L) = I$ is such that $c(\mathcal{C}) = \{\{x_1\}, \ldots, \{x_m\}\}$ covers $C_I(L)$. Then, since there is no other precondition or goal literal, $K_1(P)$ includes the single merge $m = c(\mathcal{C})$ for L with the singleton tags $t_i = \{x_i\}$, that we write simply as $m = \{x_1, \ldots, x_m\}$. The translation $K_1(P)$ is polynomial in m, and since $w(P) = 1$, by Theorem 6.21 it is complete. Notice that for this same example, the translations $K_{S0}(P)$ and $K_{\text{models}}(P)$ are identical and exponential in m (the number of models of I and $C_I(L)$).

Width of Conformant Benchmarks

The practical value of the notion of width becomes apparent when the width of existing benchmarks is considered. Table 6.2 summarizes the width of many of the existing benchmark domains for conformant planning. The domains all depend on certain parameters n or m that capture the size of the instances (e.g., size of a grid,

	Domain-Parameter	# Unknown Fluents	Width
1	Safe-n combinations	n	1
2	UTS-n locs	n	1
3	Ring-n rooms	$4n$	1
4	Bomb-in-the-toilet-n bombs	n	1
5	Comm-n signals	n	1
6	Square-Center-$n \times n$ grid	$2n$	1
7	Cube-Center-$n \times n \times n$ cube	$3n$	1
8	Grid-n shapes of n keys	$n \times m$	1
9	Logistics n pack m locs	$n \times m$	1
10	Coins-n coins m locs	$n \times m$	1
11	Block-Tower-n Blocks	$n \times (n-1) + 3n + 1$	*max*
12	Sortnet-n bits	n	*max*
13	Adder n pairs of bits	$2n$	*max*
14	Look-and-Grab m objs from $n \times n$ locs	$n \times n \times m$	m
15	1-dispose m objs from $n \times n$ locs	$n \times n \times m$	m

Table 6.2: Width of parameterized domains. *max* means that the width is the number of unknown fluents

number of objects, etc).[7] A *domain* has a bounded width when its width does not grow with the size of its instances, and has width equal to i when all of its instances have width i regardless of the parameter values.

As it can be seen from the table, the width of most existing benchmarks is 1. In all these cases, this means that the sets $C_I(L)$ of clauses that are relevant to a precondition or goal literal L contain a single clause (often a tautology $p \vee \neg p$ or a disjunction $x_1 \vee \ldots \vee x_m$) or a single $oneof(x_1, \ldots, x_m)$ expression (that translates into the disjunction $x_1 \vee \cdots \vee x_m$ and clauses $\neg x_i \vee \neg x_k$). As shown above, $w(L)$, and therefore, $w(P)$, is equal to 1 in theses cases.

On the other extreme are domains such as Blocks, Sortnet, and Adder, all of which have maximal widths; i.e., widths that are equivalent to the number of fluents whose status in the initial situation is not known. This is because all fluents interact through the action conditions (not the preconditions). The numbers for Blocks in Table 6.2, thus follow from the number of fluents involved; namely, the fluents $on(x, y)$, $clear(x)$, $ontable(x)$, and $holding(x)$.

Finally, the domains 1-dispose and Look-and-Grab (Palacios and Geffner, 2006a, 2007) where m objects with unknown locations in a grid of n by n must be collected by a robot whose gripper can hold one object at a time, have width equal to m, meaning that the width of these domains grows with the number of objects but not with the size of the grid. This is because in this case, the clauses about the possible locations of the m objects are all relevant to the condition 'hand empty' of the pick up actions.

Let us point out that the completeness of the translation $K_i(P)$ for problems P with width $w(P)$ bounded by i, establishes a correspondence between the conformant plans for P and the classical plans for $K_{T,M}(P)$. For solving P, however, this correspondence is not needed; it suffices for $K_i(P)$ to be *solvable*; a plan for $K_i(P)$ will then encode a conformant plan for P, even if $K_i(P)$ does not capture *all* conformant

[7] The names of the parameterized domains in the table do not coincide with the names of the instances as currently used. E.g. Comm-n in IPC-2006 refers to a Communication instance but not necessarily to an instance with n signals.

plans for P. From this perspective, it makes sense to refer to the smallest value of the i parameter for which the classical problem $K_i(P)$ is solvable, as the *effective width* of P, denoted $w_e(P)$. It turns out that while $w_e(P)$ cannot be larger than $w(P)$, it may be much smaller.

An interesting example of this comes from the Sortnet-n domain (Bonet and Geffner, 2000). Sortnet-n is considered a challenging domain in conformant planning with very few planners able to scale up to even small values of n (the number of entries to be sorted in a sorting network). The domain has width n, and in the compact encoding used in IPC-2006, the input vector is represented by a set of bits, exploiting the fact that sorting vectors of numbers reduces to sorting vector of bits (0's and 1's). The domain cannot be solved by the K_1 translation that FF reports correctly as unsolvable after a brief unsuccessful search. On the other hand, it is possible to reformulate the domain, replacing the unary $high(i)$ and $low(i)$ predicates by binary predicates $less(i, j)$ that compare two vector entries. We call this reformulation Sort-2-n. While the encoding Sort-n is linear in n, the encoding Sort-2-n is quadratic in n, and in both cases, the problem width is maximum, given by the number of fluents whose status in the initial situation is unknown. Yet, while the more compact Sort-n encoding is not solvable by the K_1 translation, K_1 suffices to solve the problem over the expanded Sort-2-n encoding that actually can also be solved by K_0. Thus the effective width of Sort-2-n is 0. Interestingly, provided the K_0 translation of Sort-2-n, instances can be solved with up to 20 entries. On the other hand, conformant planners such as Conformant-FF and POND can solve Sort-2-n instances for n no greater than 3.

Before explaining the formulation of Sort-2-n more in detail, let us start with Sortnet-n. The initial situation is form by the clauses $high(i) \vee \neg high(i)$ for all $1 \leq i \leq n$. For all $i < j$ there is an action `cmp-n-swap(`i,j`)` without preconditions, with conditional effects $high(i) \rightarrow high(j)$ and $\neg high(j) \rightarrow high(i)$. The goal is a set of implications, ensuring the ordering of the bits: for all $1 \leq i < n$, $high(i) \supset high(i + 1)$.

In Sort-2-n, instead of $high(i)$ we use predicates $less(i, j)$ for explicitly encoding the relation between those two vector entries. So, the initial situation is $less(i, j) \vee \neg less(i, j)$ for all $1 \leq i, j \leq n$. For $i < j$ we have actions `cmp-n-swap(`i,j`)` without preconditions but with effect $less(i, j) \wedge \neg less(j, i)$. This action also have conditional effects, for any k, $less(k, i) \rightarrow less(k, j) \wedge \neg less(j, k)$ and $less(j, k) \rightarrow less(i, k) \wedge \neg less(k, i)$. The goal is $less(i, i + 1)$ for all $1 \leq i < n$.[8]

The intuition of Sort-2-n is to represent explicitly in a predicate $less(i, j)$ what is represented in Sortnet-n by the implication $high(i) \supset high(j)$. After setting up $less(i, j)$, we need to propagate this fact to other predicates $less(k, i)$ and $less(j, k)$, to enforce transitivity of $less()$ on the numbers we have interchanged so far. Sort-2-n can be solved by $K_0(P)$ because after each action `cmp-n-swap(`i,j`)` we can be sure that at least $less(i, j)$ will be true, and everything that this can propagate. This process can continue without really having to keep track of a more complex belief state.[9]

[8]A PDDL encoding of this problem can be found in appendix C.1 on page 169

[9]Note that the initial situation of Sort-2-n is a relaxation of the one of Sortnet-n. As was described, the initial belief state of Sort-2-n is consistent with $less(i, j) \wedge less(j, i)$. If binary clauses are used for enforcing these constraints, the translation that use the possible initial states as tags, K_{S0}, will be different but the translation K_0 will lead to the same classical problem.

It may also be the case that *all* the solutions to a problem P can be found by $K_i(P)$ for i smaller than $w(P)$. Consider a problem P_k that for each j such that $1 \leq j \leq k$, has:

- a clause $p_j \vee \neg p_j$ in the initial situation I,
- and action rule $a_j : \neg p_j \rightarrow p_j$ in O,
- and a literal p_j in goal G.

Such problem has width 1 as every clause $p_j \vee \neg p_j$ is only relevant to a different goal p_j. The plans are the execution in any order of all the actions a_j. Consider now the following modification to P_k called P'_k, that have the literal r true in the initial situation I', and for each j such that $1 \leq j \leq k$:

- a clause $p_j \vee \neg p_j$ in the initial situation I',
- an action rule $a_j : \neg p_j \wedge r \rightarrow p_j$ in O',
- an action rule $b_j : p_j \wedge r$ in O',
- and a literal p_j in goal G'.

Observe that in P'_k, the literal r is relevant to any p_j and $\neg p_j$, and that any p_j and $\neg p_j$ is relevant to r. This way, all the clauses get relevant to any p_j through the literal r and hence the problem P'_k has width k. Notice that actions b do nothing as r is always true. Actually, r can be removed automatically, however the width is again k if b_j were changed to $b_j : p_j \wedge \not r$. In such case, executing any action b before all the actions a may turn the problem unsolvable. Again, doing *landmarks analysis* b actions can also be removed by showing the they do no lead to any solution (Richter et al., 2008). In any case, there may be situations difficult to detect that lead to a width k where a translation K_i for $i < k$ may allow to get all the solutions to a problem.

6.5 Tags and Initial States

A deeper understanding of the results above can be obtained by relating tags with possible initial states. By looking more closely at this relation in the context of covering translations, we will be able to answer the question of how a polynomial number of contexts (tags) can play the role of an exponential number of possible initial states in problems with bounded width.

For this, let us first recall a notation introduced in Section 2.5 on page 19, where for a state s, we wrote $I(s)$ to refer to the set of atoms encoding s (i.e, $p \in I(s)$ iff p is true in s) and P/s to refer to the *classical* planning problem $P/s = \langle F, I(s), O, G\rangle$ that is like the conformant problem $P = \langle F, I, O, G\rangle$ but with the initial state fixed to s.

Let us now extend this notation and say that an action sequence π *conforms* with a set of states S given the conformant problem P iff π is a plan for the classical problem P/s for each $s \in S$. Clearly, a conformant plan for P is nothing else but an action sequence that conforms with the set S_0 of possible initial states of P, yet the notion of 'conforms' allows us to abstract away the initial situation I and make precise the notion of a *basis*:

Definition 6.22 (Basis for P)**.** *A set of states S' is a* basis *for a conformant problem $P = \langle F, I, O, G\rangle$ if S' is a subset of the set S_0 of possible initial states of P and every plan that conforms with S' conforms with the set of possible initial states S_0.*

In words, if S' is a basis for P, it is not necessary to consider all the states in S_0 for computing the conformant plans for P; it suffices to consider just the states in S'. We aim to show that if the width of P is bounded, then P has a polynomial basis S' even if S_0 has exponential size. Moreover, the states s in such a basis are in close correspondence with the tags appearing in a covering translation.

As an illustration, consider a problem P with actions a_i, $i = 1, \ldots, n$, and effects $a_i : x_i \rightarrow L$. Let $G = \{L\}$ be the goal and $I = \{x_1 \vee \cdots \vee x_n\}$ the initial situation. The set S_0 of all possible initial states are the truth valuations over the x_i atoms where *at least* one of these atoms is true. There are $2^n - 1$ such states. On the other hand, one can show that the set S'_0 of n valuations in which *exactly* one of these atoms is true provides a basis for P; i.e., the plans that conform with these n possible initial states, are exactly the plans that conform with the complete set of $2^n - 1$ possible initial states in S_0.

The reduction in the number of possible initial states that must be considered for computing conformant plans results from two *monotonicity properties* that we formulate using the notation $rel(s, L)$ to refer to the set of literals L' that are true in the state s and are relevant to the literal L:

$$rel(s, L) = \{L' \mid L' \in s \text{ and } L' \text{ is relevant to } L\} \; .$$

Proposition 6.23 (Monotonicity 1)**.** *Let s and s' be two states and let π be an action sequence applicable in the classical problems P/s and P/s'. Then if π achieves a literal L in P/s' and $rel(s', L) \subseteq rel(s, L)$, π achieves the literal L in P/s.*

Proposition 6.24 (Monotonicity 2)**.** *If S and S' are two collections of states such that for every state s in S and every precondition and goal literal L in P, there is a state s' in S' such that $rel(s', L) \subseteq rel(s, L)$, then if π is a plan for P that conforms with S', π is a plan for P that conforms with S.*

From these properties, it follows that

Proposition 6.25. *S' is a basis for P if for every possible initial state s of P and every precondition and goal literal L in P, S' contains a state s' such that $rel(s', L) \subseteq rel(s, L)$.*

This proposition allows us to verify the claim made in the example above that the set S'_0, that contains a number of states that is linear in n, is a basis for P that has an exponential number of possible initial states. Indeed, such a problem has no precondition and a single goal literal L, and for every state s that makes more than one atom x_i true (these are the literals relevant to L), there is a state s' in S'_0 that makes only one of those atoms true, and hence for which the relation $rel(s', L) \subseteq rel(s, L)$ holds.

The question that we address now is how to build a basis that complies with the condition in Proposition 6.25 given a covering translation $K_{T,M}(P)$. For this, let $m = \{t_1, \ldots, t_n\}$ be a merge in M that covers a precondition or goal literal L, and let $S[t_i, L]$ denote the set of possible initial states s of P such that $rel(s, L) \subseteq t_i^*$; i.e., $S[t_i, L]$ contains the possible initial states of P that make all the literals L' that are relevant to L false, except for those in the closure t_i^* of t_i. We show first that if I is in prime implicate form, $S[t_i, L]$ is a non-empty set:[10]

Proposition 6.26. *If the initial situation I is in prime implicate form and $m = \{t_1, \ldots, t_n\}$ is a valid merge that covers a literal L in P, then the set $S[t_i, L]$ of possible initial states s of P such that $rel(s, L) \subseteq t_i^*$ is non-empty.*

Let then $s[t_i, L]$ stand for an arbitrary state in $S[t_i, L]$. We obtain the following result:

Theorem 6.27. *Let $K_{T,M}(P)$ be a covering translation for a problem P with an initial situation in PI form, and let S' stand for the collection of states $s[t_i, L]$ where L is a precondition or goal literal of P and t_i is a tag in a merge that covers L. Then S' is a basis for P.*

This is an important result for three reasons. First, it tells us how to build a basis for P given the tags t_i in a covering translation $K_{T,M}(P)$. Second, it tells us that the size of the resulting basis is linear in the number of precondition and goal literals L and tags t_i. And third, it makes the role of the tags t_i in the covering translation $K_{T,M}(P)$ explicit, providing an intuition for why it works: each tag t_i in a merge that covers a literal L represents one possible initial state; namely, a state $s[t_i, L]$ that makes false all the literals L' that are relevant to L except those in t_i^*. If a plan conforms with those *critical states*, then it will conform with all the possible initial states by monotonicity (Proposition 6.24). It follows then in particular that:

Theorem 6.28. *If P is a conformant planning problem with bounded width, then P admits a basis of polynomial size.*

Namely, conformant problems P with width bounded by a non-negative integer i admit polynomial translations that are complete, because the plans that conform with the possibly exponential number of initial states of P correspond with the plans that conform with a subset of *critical initial states* that are polynomial in number (namely, those in the polynomial basis). Thus, one complete polynomial translation for such problems is the K_i translation; another one, is the K_{S0} translation but with the tags associated with those critical initial states *only* rather than with all the initial states.

As an illustration, for the problem P above with actions a_i and effects $a_i : x_i \to L$, goal $G = \{L\}$, and initial situation $I = \{x_1 \vee \cdots \vee x_n\}$, the $K_1(P)$ translation with tags x_i, $i = 1, \ldots, n$, and the merge $m = \{x_1, \ldots, x_n\}$ for the goal literal L, is a covering translation. Theorem 6.27 then states that a basis S' for P results from the collection of states s_i that make each tag x_i true, and all the literals that are relevant to L that are not in x_i^* false (i.e., all x_k atoms for $k \neq i$). This is precisely the basis for P that we had above that includes the states that make a single atom x_i true for $i = 1, \ldots, n$: the plans that conform with this basis are then exactly the plans that conform with the whole collection of possible initial states of P. This basis has a size that is polynomial in m though, while the number of possible initial states of P is exponential in m.

[10]Recall that we are assuming throughout that the initial situation I is logically consistent and that the tags t are consistent with I.

Chapter 7

The Conformant Planner T_0

No he de proferir adornada falsedad ni poner
tinta dudosa ni añadir brillos a lo que es.
Esto me obliga a oírme.
Pero estamos aquí para decir verdad.
Seamos reales.
Quiero exactitudes aterradoras.

I shall not utter adorned falsehood nor pour
doubtful ink nor add gloss to what it is.
This forces me to hear myself.
But we are here to tell the truth.
Let us be real.
I want terrifying accuracies.

Ars Poética. Poem by Rafael Cadenas[1]

This chapter describes the T_0 planner, based on the translation scheme presented in the previous chapter (Palacios and Geffner, 2009). A preliminary version of the T_0 planner was the winner of conformant track of the 2006 *International Planning Competitions* (IPC-2006 Bonet and Givan, 2006) and was a runner-up in the conformant track of IPC-2008 (Bryce and Buffet, 2008). The planner T_0 is based on two instances of the general translation $K_{T,M}(P)$ from conformant into classical planning: the instance $K_1(P)$, the polynomial translation that is complete for problems of conformant width one, and the instance $K_{\text{models}}(P)$ that is complete for any problem but may have exponential size.

7.1 Implementation

The current version of the conformant planner T_0 is based on two instances of the general translation scheme $K_{T,M}(P)$ whose outputs are fed into the classical planner

[1]Collaborative translation by friends on `facebook.com`

FF v2.3.[2] One instance is polynomial but not necessarily complete; the other is complete but not necessarily polynomial. For the incomplete translation, T_0 uses K_1 that is complete for problems with width no greater than 1, and as argued above, can result in solvable instances for problems of larger widths. For the complete translation, the K_{models} translation is used instead with a simple optimization: if the K_1 translation produces a single merge m that covers L, then this merge m is used for L instead of the potentially more complex one determined by K_{models}. This is a mere optimization as the resulting translation remains complete. The other merges in K_{models}, that result from the models of the set of clauses $C_I(L)$ that are consistent with I, are computed using the SAT solver `relsat` v2.20 (Bayardo and Schrag, 1997). In the current default mode in T_0, which is the one used in the experiments below, the two translations K_1 and K_{models} are used in sequence: FF is called first upon the output of K_1 and if this fails, it is called upon the output of K_{models}. In the experiments below, we indicate the cases when K_{models} was invoked.

The translations used in T_0 accommodate certain simplifications and two additional actions that capture other types of deductions. The simplifications have to do with the fact that the translations considered are all *uniform* in the sense that all literals L in P and all rules $C \rightarrow L$ are 'conditioned' by each of the tags t in T. From a practical point of view, however, this is not needed. The simplifications address this source of inefficiency. In particular:

- literals KL/t are not created when the closure t^* contains no literal relevant to L. In such a case, the invariance $KL/t \supset KL$ holds, and thus, every occurrence of the literal KL/t in $K_{T,M}(P)$ is replaced by KL.

- support rules $a : KC/t \rightarrow KL/t$ for non-empty tags t are not created when L is not relevant to a literal L' with a merge that contains t, as in such a case, the literal KL/t cannot contribute to establish a precondition or goal. Similarly, cancellation rules $a : \neg K\neg C/t \rightarrow \neg K\neg L/t$ for non-empty tags t are not created when $\neg L$ is not relevant to a literal L' with a merge that contains t.

- support and cancellation rules $a : KC/t \rightarrow KL/t$ and $a : \neg K\neg C/t \rightarrow \neg K\neg L/t$ are grouped as $a : KC/t \rightarrow KL/t \wedge \neg K\neg L/t$ when for every fluent L' relevant to L, either L' or $\neg L'$ is entailed by I and t. In such a case, there is no incomplete information about L given t in the initial situation, and thus the invariant KL/t or $K\neg L/t$ holds, and $\neg K\neg C/t$ is equivalent to KC/t.

- When a state contains the atom KL/t if and only if that state contains KL, then the atom KL/t can be replaced by KL. This invariant is satisfied if for all atoms L' relevant to L, it holds that $KL'/t \in I' \supset KL' \in I'$.

Two other types of sound deductive rules are included in the translations:

- a rule $a : KC \rightarrow KL$ is added if $a : C, \neg L \rightarrow L$ is a rule in P for an action a, and no rule in P has the form $a : C' \rightarrow \neg L$,

[2]The conformant planner T_0 along with all the benchmarks considered in this chapter are available at `http://www.ldc.usb.ve/~hlp/software`, where the source code is also available. The classical planner FF was modified to allow larger instances as the generated by T_0

- rules $K\neg L_1, \ldots, K\neg L_{i-1}, K\neg L_{i+1}, \ldots, K\neg L_n \rightarrow KL_i$ for $i = 1, \ldots, n$ are added to a new unique action with no precondition, when $L_1 \vee \cdots \vee L_n$ is a static clause in P (a clause in P is static if true in the initial situation and provably true after any action).

These rules are versions of the *action compilation* and *static disjunctions* rules (Palacios and Geffner, 2006a, 2007), and they appear to help in certain domains without hurting in others. They are presented in more detail in Section 8.2 on page 119 where we explain the work presented in 2006a.

In the second case, we also 'ramify' the heads KL_i into all the existing literals of the form KL_i/t for maintaining the invariant $KL_i \supset KL_i/t$. On the other hand, for speed, the 'ramifications' in the merge actions $a_{m,L}$ in $K_{T,M}$ that enforce the invariant $KL \supset K\neg L'$ for each literal L' mutex with L in P are eliminated in T_0, as the mutex computation may be expensive, and these ramifications are not strictly needed (see XL on Section 6.2, page 72). From a theoretical point of view, they are needed to ensure that the classical problem $K_{T,M}(P)$ is consistent if the conformant problem P is consistent (see appendix B on page 163). From a practical point of view, however, the consistency of $K_{T,M}(P)$ is not needed; for a classical plan π for $K_{T,M}(P)$ to be sound, it suffices that none of the actions a in π trigger inconsistent effects KL/t and $\neg KL/t$, a condition that is easy to verify.[3]

The version of T_0 reported below does not assume that the initial situation I of P is in prime implicate form but it rather renders it in PI form by running a version of Tison's algorithm 1967, a computation that in none of the benchmarks solved took more than 48 seconds.

The translators in T_0 are written in OCaml while the code for parsing the PDDL files is written in C++. An sketch of the algorithm for generating a classical problem $K_{T,M}(P)$ is describe in function `GetKtm()` on the next page, and the whole algorithm of the planner T_0 is on page 93. Other complete or incomplete planners can be construct using instances of $K_{T,M}(P)$, variating ways of getting merges and tags, and in which order try them out.

7.2 Experiments

We considered instances from three sources: the Conformant-FF distribution, the Conformant track of the IPC-2006, and relevant publications (Palacios and Geffner, 2006a, 2007; Cimatti et al., 2004; Hoffmann and Brafman, 2006). The instances were run on a cluster of Linux boxes at 2.33 GHz with 8GB. Each experiment had a cutoff of 2h or 2.1GB of memory. Times for T_0 include all the steps, in particular, computation of prime implicates, translation, and search (done by FF). We also include results from the Conformant Track of the recent IPC-2008.

Goals that are not sets of literals but sets of clauses are transformed in T_0 in a standard way: each goal clause $C : L_1 \vee \cdots \vee L_m$ is modeled by a new goal atom

[3]All plans below have been verified in this way, and as a double check, the resulting conformant plans have been checked also with the conformant verifier used at IPC-2006 due to Blai Bonet, available at `http://www.ldc.usb.ve/~bonet/ipc5/softw/verifier.tar.gz`.

Figure 7.1: Function GetKtm(Conformant Problem P, Merges M). Returns a Classical Problem $K_{T,M}(P)$

```
Input: Conformant Problem $P = \langle F, O, I, G \rangle$
Input: Merge $M(X)$ for each $X$ goal or precondition in $P$
Output: Classical Problem $K_{T,M}(P)$

Tags $T \longleftarrow$ any tag mentioned in merges $M$

function PossibleTag($L$,$t$):                (* $t$ is a possible tag for $L$ *)
    return $t == \{\}$ or
             $t^*$ relevant to $L \wedge L$ relevant to $X \wedge t$ in $M(X) = \{t_1, \ldots, t_n\}$
               if $X$ is goal or precondition

(* Actions of $K_{T,M}(P)$ *)
foreach $X$ goal or precondition in $P$, and $M(X) = \{t_1, \ldots, t_n\}$ do
    Add merge action $m_X$ to $O'$ with rule
        $m_X : KX/t_1, \ldots, KX/t_n \rightarrow KX$
foreach action $a$ in $O$ of $P$ do
    Add action $a$ to $O'$ with
        preconditions $KL$ for each $L$ in preconditions of $a$
    foreach $a : C_1, \ldots, C_n \rightarrow L$ rule in $O$ of $P$ do
        foreach $t$ in $T$ do
            begin
                if PossibleTag($L$,$t$) then
                    Add to $a$ the rule
                        $a : KC_1/t, \ldots, KC_n/t \rightarrow KL/t$
                if PossibleTag($\neg L$,$t$) then
                    Add to $a$ the rule
                        $a : \neg K\neg C_1/t, \ldots, \neg K\neg C_n/t \rightarrow \neg K\neg L/t$
                if $t^*$ is not relevant to $x$ then
                    Replace $Kx/t$ by $Kx$ in the condition of an added rule
            end
(* Init situation of $K_{T,M}(P)$ *)
foreach tag $t == \{\}$ or $t$ in $M(X)$ for $X$ goal or precondition in $P$ do
    ConsequenceOfTag $\longleftarrow$ literals in UnitPropagation($I \cup \{t\}$)
    foreach $L$ in ConsequenceOfTag do              (* $\{L \mid I \models t \supset L\}$ *)
        Add $KL/t$ to $I'$

(* Goal of $K_{T,M}(P)$   *)
foreach $X$ in $G$ do
    Add $KX$ to $G'$

(* Fluents of $K_{T,M}(P)$   *)
$F' \longleftarrow$ any atom $KL/t$ mentioned in $O'$, $I'$ or $G'$

return $K_{T,M}(P) = \langle F', I', O', G' \rangle$
```

Figure 7.2: Algorithm for the Conformant Planner T_0

```
Input: Conformant Problem $P = \langle F, O, I, G \rangle$
Output: Conformant plan $\pi$ for $P$

function Merge-$K_1$(Conformant $P = \langle F, O, I, G \rangle$):
begin
    $M \longleftarrow$ function from literals $L$ to a set merges $m$
    foreach goal or precondition literal $L$ in $P$ do
        $C_I(L) \longleftarrow$ clauses in $I$ and tautologies relevant to $L$
        foreach $C$ in $C_I(L)$ do
            if for all $t$ in $C$, $t^*$ satisfies $C_I(L)$ then
                Add $\{t_i \mid t_i \in C\}$ to set $M(L)$
                continue
        if $M(L)$ is empty then
            foreach $C$ in $C_I(L)$ do
                Add $\{t_i \mid t_i \in C\}$ to set $M(L)$
    return $M$
end

function Merge-$Kmodels$(Conformant $P = \langle F, O, I, G \rangle$):
begin
    $M \longleftarrow$ function from literals $L$ to merges $m$
    foreach goal or precondition literal $L$ in $P$ do
        $C_I(L) \longleftarrow$ clauses in $I$ and tautologies relevant to $L$
        $M(L) \longleftarrow \{t_i \mid t_i \in Models(C_I(L))\}$      (* Calls a SAT solver *)
    return $M$
end

function Solve(Conformant $P$, GetMerge()):
begin
    Relevance($P$)                      (* to be used also in GetKtm() *)
    $K_{T,M} \longleftarrow$ GetKtm($P$,GetMerge($P$))      (* on the preceding page *)
    $K_{T,M} \longleftarrow$ SimplifyFurther($K_{T,M}$)
    $K_{T,M} \longleftarrow K_{T,M}$ + Action Compilation($P$)
    $K_{T,M} \longleftarrow K_{T,M}$ + Static Disjunctions($P$)
    $\pi' \longleftarrow$ Classical Planner($K_{T,M}$)
    return $\pi'$ without the merge actions
end

try
    return Solve($P$,Merge-$K_1$)
else
    return Solve($P$,Merge-$Kmodels$)
```

	P				$K_1(P)$			PDDL
Problem	#Acts	#Atoms	#Effs	Time	#Acts	#Atoms	#Effs	Size
bomb-100-100	10100	404	40200	2	10201	1595	50500	2,9
square-center-96	4	196	760	35,1	7	37248	75054	3,8
sortnet-09	46	68	109	8,3	56	29707	154913	5,1
blocks-03	32	30	152	4	37	11370	35232	0,7
dispose-16-1	1217	1479	2434	163,6	1218	133122	3458	0,3
look-and-grab-8-1-1	352	358	2220	6,9	353	8708	118497	7,8
sgripper-30	487	239	1456	21,5	860	1127	12769	1

Table 7.1: Translation data for selected instances: #Acts, #Atoms, and #Effs stand for the number of actions, fluents, and conditional effects. Time is the translation time in seconds rounded to the closest decimal, and PDDL Size is the size of the PDDL file in Megabytes.

G_C, and a new action that can be executed once is added with rules $L_i \rightarrow G_C$, $i = 1, \ldots, m$.[4]

Table 7.1 shows data concerning the translation of a group of selected instances. As it can be seen, the number of conditional effects grows considerably in all cases, and sometimes the translation may take several seconds.

Tables 7.2, 7.3, 7.4, and 7.5, show the plan times and lengths obtained on a number of benchmarks by T_0, POND 2.2 (Bryce et al., 2006), Conformant FF (Hoffmann and Brafman, 2006), MBP (Cimatti and Roveri, 2000) and KACMBP (Cimatti et al., 2004). These last two planners do not accept problems in the standard syntax (based on PDDL), so only a limited number of experiments were performed on them. The general picture is that T_0 scales up well in most domains, the exceptions being Square-Center and Cube-Center in Table 7.3, where KACMBP scales up better.

The problems in Table 7.2 are encodings from the Conformant-FF repository: Bomb-x-y refers to the Bomb-in-the-toilet problem with x packages, y toilets, and clogging; Logistics-i-j-k is a variation of the classical version with uncertainty about initial location of packages; Ring-n is about closing and locking windows in a ring of n rooms without knowing the current room; and Safe-n is about opening a safe with n possible combinations. All these problems have width 1. T_0 does clearly best on the last two domains, while in the first two domains, Conformant-FF does well too.

Table 7.3 reports experiments on four grid domains: Cube-Center-n refers to the problem of reaching the center of a cube of size n^3 from a completely unknown location; Square-Center-n is similar but involves square with n^2 possible locations; Corners-Cube-n and Corners-Square-n are variations of these problems where the set of possible initial locations is restricted to the Cube and Square corners respectively. MBP and KACMBP appear to be effective in these domains, although KACMBP doesn't scale up well in the corner versions. T_0 solves most of the problems, but in the corner versions, the quality of the plans is poor. These problems have also width 1.

The problems reported in Table 7.4 and Table 7.5 are variations of a family of grid problems (Palacios and Geffner, 2006a, 2007), described as follows.

[4] An alternative way to represent such CNF goals is by converting them into DNF first and having an action *End* map each of its non-mutex terms into a dummy goal L_G. This alternative encoding pays off in some cases, such as in the Adder-01 instance that does not get solved in the default CNF goal encoding (see below).

	T_0		POND		CFF		MBP		KACMBP	
Problem	time	len	time	len	time	len	time	len	time	len
bomb-20-1	0,1	49	4139	39	0	39	$> 2h$		0	40
bomb-20-5	0,1	35	$> 2h$		0	35	$> 2h$		0,2	40
bomb-20-10	0,1	30	$> 2h$		0	30	$> 2h$		0,5	40
bomb-20-20	0,1	20	$> 2h$		0	20	$> 2h$		2	40
bomb-100-1	0,5	199	–		56,7	199	–		1,9	200
bomb-100-5	0,7	195	–		52,9	195	–		4,3	200
bomb-100-10	1,1	190	–		46,8	190	–		16,4	200
bomb-100-60	4,25	140	–		9,4	140	–		$> 2h$	
bomb-100-100	9,4	100	–		1	100	–		$> 2h$	
logistics-4-3-3	0,1	35	56	40	0	37	$> 2h$		$> 2.1GB$	
logistics-2-10-10	1	84	$> 2h$		1,6	83	$> 2h$		$> 2.1GB$	
logistics-3-10-10	1,5	108	$> 2h$		4,7	108	$> 2h$		$> 2.1GB$	
logistics-4-10-10	2,5	125	$> 2h$		4,4	121	$> 2h$		$> 2.1GB$	
ring-4	0,1	13	1	18	0,4	18	0	11	0	26
ring-5	0,1	17	6	20	4,3	31	0,1	14	0,1	58
ring-6	0,1	20	33	27	93,6	48	0,6	17	0,2	99
ring-7	0,1	30	444	33	837	71	3,8	20	0,5	204
ring-8	0,1	39	$> 2h$		$> 2h$		40	23	2	432
ring-30	13,4	121	–		–		$> 2h$		$> 2.1GB$	
safe-10	0,1	10	0	10	0	10	0,1	10	0	10
safe-30	0,1	30	2	30	1,4	30	$> 2h$		0,2	30
safe-50	0,4	50	9	50	29,4	50	$> 2h$		0,7	50
safe-70	1,12	70	41	70	109,9	70	$> 2h$		2,4	70
safe-100	2,5	100	$> 2.1GB$		1252,4	100	$> 2h$		8,6	100

Table 7.2: Experiments over well known benchmarks. Times reported in seconds and rounded to the closest decimal. '–' means time or memory out for smaller instances.

- Dispose-n-m is about retrieving m objects whose initial location is unknown in a $n \times n$ grid , and placing them in a trash can at a given, known location.
- Push-to-n-m is a variation where m objects can be picked up only at two designated positions in the $n \times n$ grid to which all objects have to be pushed to: pushing an object from a cell into a contiguous cell moves the object if it is in the cell.
- 1-Dispose-n-m is a variation of Dispose where the robot hand being empty is a condition for the pick up actions to work. As a result, a plan for 1-Dispose has to scan the grid, performing pick ups in every cell, followed by excursions to the trash can, and so on. The plans can get very long (a plan is reported with 1316 actions).
- Look-and-Grab-n-m-r has an action that picks up the any of the m objects that are sufficiently close if any in the $n \times n$ grid, and after each pick-up must dump the objects it collected into the trash before continuing. The parameter r is the radius of the action: 1 means that the hand picks up all the objects in the 8 surrounding cells, 2 that that the hand picks up all the objects in the 15 surrounding cells, and so on. An illustration of a solution of Look-and-Grab-8-1-1 found by T_0 is in figure 2.3 on page 17.

PDDL examples encoding instances of this domains can be found in appendix C on page 169.

	T_0		POND		CFF		MBP		KACMBP	
Problem	time	len	time	len	time	len	time	len	time	len
square-center-8	0,2	21	2	41	70,6	50	0	24	0	28
square-center-12	0,2	33	12	52	$> 2h$		0	36	0	42
square-center-16	0,3	44	1322	61	$> 2h$		0	48	0	56
square-center-24	0,8	69	$> 2h$		–		0	72	0	84
square-center-92	45,3	273	$> 2h$		–		0,9	276	0,3	322
square-center-96	50,2	285	–		–		0,9	288	0,3	336
square-center-100	$> 2.1GB$		–		–		1,1	300	0,3	350
square-center-120	$> 2.1GB$		–		–		1,9	360	0,4	420
cube-center-5	0,1	18	1	22	8,2	45	0	28	0	25
cube-center-7	0,1	27	2	43	$> 2h$		0	33	0	35
cube-center-9	0,2	33	3	47	$> 2h$		0,1	54	0	45
cube-center-11	0,3	45	29	87	–		0,2	59	0	55
cube-center-15	0,5	63	880	109	–		0,2	69	0	75
cube-center-19	0,8	81	$> 2h$		–		1,6	111	0,1	95
cube-center-63	28,5	279	$> 2h$		–		28	285	0,5	315
cube-center-67	41,6	297	–		–		$> 2.1GB$		0,7	335
cube-center-87	137,5	387	–		–		$> 2.1GB$		1,2	435
cube-center-91	$> 2.1GB$		–		–		–		1,2	455
cube-center-119	$> 2.1GB$		–		–		–		2,1	595
corners-square-12	0,1	64	11	44	1,7	82	0	36	0,2	106
corners-square-16	0,2	102	1131	67	13,1	140	0	48	0,6	158
corners-square-20	0,3	148	$> 2h$		73,7	214	0,3	60	3	268
corners-square-24	0,5	202	$> 2h$		321	304	0,6	72	7,5	346
corners-square-28	0,7	264	–		MPL		1,1	84	20,7	502
corners-square-36	1,7	412	–		–		1,5	108	3308,8	808
corners-square-40	2,5	498	–		–		7,8	120	$> 2h$	
corners-square-72	26,1	1474	–		–		118,8	216	$> 2h$	
corners-square-76	30,5	1632	–		–		371	228	–	
corners-square-80	38,2	1798	–		–		649,6	240	–	
corners-square-120	223,6	3898	–		–		$> 2.1GB$		–	
corners-cube-15	0,8	147	907	105	134,5	284	3,7	69	174,1	391
corners-cube-16	0,9	174	3168	115	439,4	214	12,5	72	270,5	316
corners-cube-19	2,5	225	$> 2h$		868,4	456	549,5	111	1503,1	488
corners-cube-20	2,7	258	$> 2h$		3975,6	332	1061,9	90	2759	625
corners-cube-23	6,3	319	–		MPL		$> 2h$		6265,9	899
corners-cube-24	6,7	358	–		–		$> 2h$		$> 2h$	
corners-cube-27	14,6	429	–		–		–		$> 2h$	
corners-cube-52	448	1506	–		–		–		–	
corners-cube-55	$> 2.1GB$		–		–		–		–	

Table 7.3: Experiments over grid problems. Times reported in seconds and rounded to the closest decimal. 'MPL' for CFF means that plan exceeds maximal plan length (500 actions). '–' means time or memory out for smaller instances.

	T_0		POND		CFF		MBP		KACMBP	
Problem	time	len	time	len	time	len	time	len	time	len
dispose-4-1	0,1	59	9	55	0,1	39	$> 2h$		17,1	81
dispose-4-2	0,1	110	36	70	0,2	56	$> 2h$		$> 2h$	
dispose-4-3	0,3	122	308	102	0,6	73	–		$> 2h$	
dispose-8-1	2,7	426	$> 2.1GB$		339,1	227	–		–	
dispose-8-2	18,4	639	$> 2.1GB$		2592,1	338	–		–	
dispose-8-3	197,1	761	–		$> 2h$		–		–	
dispose-12-1	78	1274	–		ME		–		–	
dispose-12-2	2555	1437	–		$> 2.1GB$		–		–	
dispose-12-3	$> 2.1GB$		–		–		–		–	
dispose-16-1	382	1702	–		–		–		–	
dispose-16-2	$> 2.1GB$		–		–		–		–	
look-and-grab-4-1-1	0,3	30	3098	16	$> 2h$		$> 2h$		0,6	54
look-and-grab-4-1-2	0,5	4	$> 2h$		Mcl		0,02	5	0,0	6
look-and-grab-4-1-3	0,61	4	$> 2h$		Mcl		0,01	5	0,0	6
look-and-grab-4-2-1	35	12	$> 2.1GB$		$> 2h$		$> 2h$		0,63	40
look-and-grab-4-2-2	49,41	4	$> 2h$		Mcl		0,02	5	0,01	6
look-and-grab-4-2-3	60,02	4	$> 2h$		Mcl		0,02	5	0,01	6
look-and-grab-4-3-1	$> 2.1GB$		$> 2.1GB$		$> 2h$		$> 2h$		0,98	60
look-and-grab-4-3-2	213,3	4	–		$> 2h$		0,02	5	0,02	6
look-and-grab-4-3-3	$> 2.1GB$		–		$> 2h$		0,02	5	0,01	6
look-and-grab-8-1-1	58,2	242	–		–		$> 2h$		$> 2h$	
look-and-grab-8-1-2	75,3	90	–		–		$> 2h$		$> 2h$	
look-and-grab-8-1-3	55,89	58	–		–		$> 2h$		$> 2h$	
look-and-grab-8-2-1	$> 2h$		–		–		$> 2h$		$> 2h$	
look-and-grab-8-2-2	$> 2h$		–		–		$> 2h$		$> 2h$	
look-and-grab-8-2-3	$> 2h$		–		–		$> 2h$		1195	178
look-and-grab-8-3-1	$> 2h$		–		–		$> 2h$		$> 2h$	
look-and-grab-8-3-2	$> 2h$		–		–		$> 2h$		$> 2h$	
look-and-grab-8-3-3	$> 2h$		–		–		$> 2h$		17,9	58

Table 7.4: Problems from Palacios and Geffner (2006, 2007): Times reported in seconds and rounded to the closest decimal. '–' means time or memory out for smaller instances. 'ME' and 'Mcl' mean too many edges and too many clauses respectively.

	T_0		POND		CFF	
Problem	time	len	time	len	time	len
push-to-4-1	0,2	78	5	50	0,3	46
push-to-4-2	0,3	85	171	58	0,7	47
push-to-4-3	0,6	87	–		1,6	48
push-to-8-1	81,8	464	$> 2h$		$> 2.1GB$	
push-to-8-2	457,9	423	$> 2h$		$> 2.1GB$	
push-to-8-3	1293,1	597	$> 2h$		$> 2.1GB$	
push-to-12-1	$> 2h$		–		–	
push-to-12-2	$> 2h$		–		–	
push-to-12-3	$> 2.1GB$		–		–	
1-dispose-8-1	82,2	1316	$> 2.1GB$		$> 2h$	
1-dispose-8-2	$> 2.1GB$		$> 2.1GB$		$> 2h$	
1-dispose-8-3	$> 2.1GB$		–		–	

Table 7.5: Other problems from Palacios and Geffner (2006, 2007). MBP and KACMBP were not tried on these problems as they use a different syntax. Times reported in seconds and rounded to the closest decimal. '–' means time or memory out for smaller instances.

The domains in Tables 7.4 and 7.5 have width 1 except 1-Dispose and Look-n-Grab. This is because, the hand being empty is a fluent that is relevant to the goal, and clauses about the location of objects are all relevant to 'hand empty'. In all these domains T_0 appears to do better than the other planners. The K_{models} translation was triggered only in the instances Look-and-Grab-n-m-r for $m > 1$ (the width of these instances, as mentioned in Section 6.4, is m, independent of grid size).

7.3 Results of the Conformant track of the International Planning Competition 2006

We now consider the instances used in the Conformant track of the IPC-2006, where a previous version of T_0 end up as the winner. The other participants were CFF and POND. We also compare with MBP and KACMBP, and show the results in Table 7.6. In summary, T_0 scales up well in most of the used benchmarks, except in Sortnet where MBP and KACMBP scale up better, and Adder where POND is the only planner able to solve one instance.

Table 7.6 reports experiments over problems from the conformant track of the IPC-2006 (Bonet and Givan, 2006). The domains Coins, Comm and UTS have all width 1. The others have max width given by the number of unknown fluents in the initial situation. T_0 dominates in all these domains except in Adder where POND is the only planner able to solve an instance, and Sortnet, where MBP and KACMBP do very well, possibly due to use of the cardinality heuristic and OBDD representations. T_0 fails on Adder because FF gets lost in the search. Looking at this problem more closely, we found that FF could solve the (translation of the) first instance in less than a minute provided that the CNF goal for this problem is encoded in DNF as explained in footnote 4, page 94. The domains Adder, Blocks, and Sortnet in the table, along with the domain Look-and-Grab in the next table, are the only domains considered where FF run on the K_1 translation reports no solution after a brief search, triggering then the use of the complete K_{models} translation. In all the other cases where K_{models} was used, the K_1 translation had an unreachable goal literal, detected in the translated problem, and there was no need to try FF on it.

7.4 Results of the Conformant track of the International Planning Competition 2008

Tables 7.7 and 7.8 provide details on the results of the Conformant Track of the IPC-2008 (Bryce and Buffet, 2008), The version of T_0 in IPC-2008 was different from the version of T_0 used in IPC-2006, and different also from the upgraded version used in Chapter 7 (Palacios and Geffner, 2009). In relation, to the former, T_0 IPC-2008 was a cleaner but complete reimplementation; in relation to the latter, T_0 IPC-2008 handled problems with width greater than 1 in a different way. As explained in Chapter 7, the current version of T_0 uses K_1 as the basic translation regardless of the width of the problem, switching to K_{models} when the search over K_1 fails. In the version of T_0 at IPC-2008, the basic translation was a combination of K_0 and K_1. More precisely, merges for literals L with width $w(L) = 1$, were generated according to K_1, but merges for literals L with width $w(L) \neq 1$ were not generated at all. The

	T_0		POND		CFF		MBP		KACMBP	
Problem	time	len	time	len	time	len	time	len	time	len
adder-01	$> 2h$		1591	5	SNH		NR		NR	
adder-02	$> 2h$		$> 2h$		SNH		NR		NR	
blocks-01	0,1	5	0,1	4	0	6	NR		NR	
blocks-02	0,3	23	0,4	26	$> 2h$		NR		NR	
blocks-03	82,6	80	126,8	129	$> 2h$		NR		NR	
coins-10	0,1	26	5	28	0,1	38	$> 2h$		4,2	106
coins-12	0,1	67	$> 2h$		0,8	72	$> 2h$		3654,7	674
coins-15	0,1	79	$> 2h$		3	89	–		$> 2h$	
coins-16	0,3	113	–		33,3	145	–		$> 2h$	
coins-17	0,2	96	–		1,4	94	–		–	
coins-18	0,2	97	–		6,2	118	–		–	
coins-19	0,2	105	–		16,5	128	–		–	
coins-20	0,2	107	–		20,6	143	–		–	
coins-21	$> 2h$		–		$> 2h$		–		–	
comm-07	0,1	54	0	47	0	47	0,2	55	63,6	53
comm-08	0,1	61	1	53	0	53	0,2	71	1966,8	53
comm-09	0,1	68	1	59	0	59	0,2	77	$> 2h$	
comm-10	0,1	75	1	65	0	65	0,3	85	$> 2h$	
comm-15	0,1	110	6	95	0,2	95	0,9	115	–	
comm-16	0,2	138	$> 2h$		0,4	119	1,6	151	–	
comm-20	0,8	278	$> 2.1GB$		6,4	239	50,9	340	–	
comm-25	2,3	453	–		56,1	389	$> 2h$		–	
sortnet-06	0,6	21	18	20	SNH		0	17	0	21
sortnet-07	2,5	28	480	25	SNH		0	20	0	28
sortnet-08	9,6	36	$> 2h$		SNH		0	28	0	36
sortnet-09	76,8	45	$> 2h$		SNH		0	36	0	45
sortnet-10	$> 2.1GB$		–		SNH		0,1	37	0,1	55
sortnet-11	$> 2.1GB$		–		SNH		0,1	47	0,1	66
uts-k-04	0,1	23	2	22	0,1	22	5,4	32	1,5	30
uts-k-05	0,1	29	4	28	0,3	28	1247,3	38	195,4	42
uts-k-06	0,2	35	10	34	0,8	34	1704,8	50	$> 2h$	
uts-k-07	0,4	41	13	40	1,9	40	$> 2h$		$> 2h$	
uts-k-08	0,6	47	24	47	4,4	46	$> 2h$		–	
uts-k-09	0,9	53	$> 2h$		8,6	52	–		–	
uts-k-10	1,3	59	2219	67	16,5	58	–		–	
uts-l-07	0,2	70	201	58	0,2	41	10,5	89	$> 2h$	
uts-l-08	0,3	80	937	67	0,4	47	41,1	106	$> 2h$	
uts-l-09	0,6	93	$> 2h$		0,8	53	1176	137	–	
uts-l-10	0,7	97	$> 2h$		1,6	59	$> 2h$		–	

Table 7.6: Experiments over problems from IPC-2006. Times reported in seconds and rounded to the closest decimal. 'SNH' for CFF means that goal syntax not handled, while 'NR' for MBP and KACMBP that these planners were not run due to lack of translations from PDDL. '–' means time or memory out for smaller instances.

Domain	# Instances	`CpA(H)`	`CpA(C)`	T_0 IPC-2008
Blocks	4	**4**	3	3
Adder	4	1	**1**	1
UTS Cycle	27	2	2	**3**
Forest	9	1	1	**8**
Rao's keys	29	**2**	2	1
Dispose	90	**76**	59	20

Table 7.7: Data from the Conformant Track of the recent IPC-2008 Competition: Number of problems solved by each of the conformant planners, with time out of 20 minutes. In bold, entry for planner that performed best in each domain. The data is by Bryce and Buffet (2008)

result was that the basic translation in T_0 in IPC-2008 was lighter than the basic translation of the current version of T_0 but could fail on problems with width higher than 1 that the latter can solve. Retrospectively, this was not a good choice, but it didn't have much of an impact on the results. There was however a bug in the program that prevented two width-1 domains, Forest and Dispose, to be recognized as such, and thus resulted in the use of the K_{models} translation, that is complete for all widths, but does not scale up that well.

The other two conformant planners entered into IPC-2008 where `CpA(H)` and `CpA(C)`; these are belief-space planners that represent beliefs as DNF formulas, and use simple belief-state heuristics for guiding the search (Tran et al., 2009). The belief progression in these planners is done quite effectively, by progressing each term in turn, according to the 0-approximation semantics. More about this in the Section 9.5 on page 141, in the Related Work chapter. The heuristics used by `CpA(H)` and `CpA(C)` are combinations of the cardinality heuristic, that measures the number of states in a belief state, the total sum heuristic, that adds the heuristic distances to the goal from each possible state, and the number of satisfied goals, that counts the number of top goals achieved. These heuristics are all very simple, yet they work well on some benchmarks.

Tables 7.7 and 7.8 show data obtained from the IPC-2008 organizers from the planner logs. The first table appears in the IPC-2008 report (Bryce and Buffet, 2008), where the new domains Forest and Rao's keys are explained, and shows the number of problems solved by each planner, displaying in bold the planner that did best in each domain. The planner `CpA(H)`, was declared the winner, as it was declared best in three domains (Blocks, Rao's keys, Dispose), with T_0 doing best in two domains (UTS Cycle and Forest), and `CpA(C)` doing best in one (Adder).

Table 7.8 shows additional details on some of the instances; in particular, the total time taken to solve the instance and the length of the plans for each of the three planners.

In terms of domain coverage, the planners do similarly on most domains, except in Forest, where T_0 solved most of the instances and CPA(H) solved few (8/9 vs. 1/9), and Dispose, where CPA(H) solved most of the instances and T_0 solved few (76/90 vs. 20/90).

In terms of time and plan quality, `CpA(H)` and `CpA(C)` appear to be slightly faster than T_0 on Blocks, but produce much longer plans. In Dispose, T_0 scales up better than `CpA(H)` and `CpA(C)` over the size of the grids, and worse on the number of objects.

Indeed, only T_0 manages to solve the largest grid but for a single object (Dispose-10-01), and only `CpA(H)` and `CpA(C)` solve instances with more than 2 objects in the largest grids. As in most cases, plan lengths produced by T_0 are shorter; e.g., the plan for Dispose-04-03 contains 125 actions for T_0, 314 for `CpA(H)`, and 320 for `CpA(C)`.

Dispose is actually a domain where the cardinality heuristic does very well in the generation of plans, even if the plans tend to be rather long. In this domain, an agent has to scan a grid collecting a set of objects at unknown locations, and each time the action of picking up an object from a cell that may contain the object is made (except for the first time), the cardinality of the belief state is reduced. Indeed, if initially an object may be at positions p_1, p_2, ..., p_n, after a pick up at p_1, the object can be in positions $p_2, \ldots, p_n$ or in the gripper, after a pick up at p_2, the object can be in positions $p_3, \ldots, p_n$ or in the gripper, and so on, each pick up action decreasing the cardinality of the belief state, until becoming a singleton belief where the object must be held by the gripper with certainty.

The problem with the version of T_0 used in IPC-2008 in the Dispose domain, was not only that FF explores too many states in the search, but as explained above, that it used the expensive K_{models} translation instead of the lighter K_1 translation that is complete for this domain that has width 1. With this bug fixed, T_0 solves 60 rather than 20 of the 90 Dispose instances, still failing on some of the larger grids with many objects, but producing much shorter plans. For example, Dispose-06-8 is solved with a plan with 470 actions, while `CpA(H)` and `CpA(C)` solve it with plans with 2881 and 3693 actions respectively. The same bug surfaced in the Forest domain, but it just prevented the solution of one instance only. Forest, Dispose, and UTS Cycle have all conformant widths equal to 1, while the other domains have all larger widths (see Table 6.2 on page 84 for the widths of Blocks and Adder).

The second domain in IPC-2008 where FF got lost in the search was Adder, where indeed, T_0 did not solve any instance. The instance that is shown to be solved by T_0 in the competition report, appears to be a mistake. Similarly, the fourth instance of blocks, that is reported as solved by CPA(H), may be a mistake too; indeed, no plan for such an instance can be found in the logs, and T_0 reports that the goal is unreachable in the K_{models} translation that is complete. According to T_0, instance four of Rao's key is unsolvable too. On the other hand, T_0 failed on the larger UTS Cycle and Rao's key instances during the *translation*. In the the first, the resulting PDDL's are too large and can't be loaded into FF; in the second, the number of init clauses turns out to be quite large (above 300), giving rise to a still larger set of prime implicates (above 5000) that caused the translator to run out of memory. The second instance of Rao's keys, however, is rather small and T_0 didn't solve it due to a different bug. With this bug fixed, T_0 solves it in 0.3 seconds, producing a plan with 53 actions, which compares well with the solutions produced by `CpA(H)` and `CpA(C)` in 0.7 and 1.9 seconds, with 85 and 99 steps, respectively.

7.5 Discussion

We have developed a conformant planner T_0 based on two instances of the $K_{T,M}(P)$ translation. One based on the instance $K_1(P)$, complete for problems with conformant width 1; the other based on K_{models}, that is complete for any problem P. We

Problem	Instance	CpA(H)		CpA(C)		T_0 IPC-2008	
		time	len	time	len	time	len
Blocks	1	0	4	0	7	0,1	5
	2	0,1	28	0,1	35	0,1	23
	3	5,9	411	6,3	157	17,8	83
	4	143,9	257				
Adder	1	8,5	3	8,3	3		
UTS Cycle	1	0,8	3	0,6	3	0,1	3
	2	25,3	6	24,7	6	0,7	7
	3					5,4	10
Forest	1	3,6	24	11,6	18	0,2	16
	2					1,3	45
	3					2,2	78
	4					12,1	129
	5					14,4	115
	6					69,7	200
	7					355,1	256
	8						
Rao's keys	1	0,1	28	0	29	0	16
	2	0,7	85	1,9	99		
Dispose	4,1	0,3	80	0,4	88	0,1	77
	4,2	0,7	197	0,9	206	3,6	110
	4,3	1,3	314	1,8	320	528,3	125
	4,4	2	431	2,8	434		
	6,1	4,7	270	4,5	187	0,9	204
	6,2	10,4	643	42,2	735	217,7	329
	6,3	17,7	1016	97,9	1228		
	6,4	27,6	1389	172,5	1721		
	8,1	40,1	753	40,3	518	7,4	326
	8,2	86,7	1851	524,6	1962		
	8,3	86,7	1851				
	10,1					45	683
	10,2						

Table 7.8: Running time and plan length from IPC-2008 logs. Time in seconds. Blanks stand for time or memory out. Only 13 of the 90 Dispose-n-m instances shown, At IPC-2008, size n of grid ranged from 2 to 10, while number m of objects, from 1 to 10. T_0 scales up best on n and worst on m.

have shown that T_0 exhibits a good performance in comparison with existing conformant planners, specially because many current benchmarks have conformant width 1, and problems with width > 1 tend to be hard for both T_0 and other conformant planners of the state of the art.

The classical planner FF was used for solving the instances of $K_{T,M}(P)$. Even though there are many good classical planners, we faced troubles on making use of them for our instances. Most of them claim to support conditional effects, and actually work on benchmarks used in the planning competitions, but failed in simple small instances of $K_{T,M}(P)$ (Helmert, 2006; Chen et al., 2006; Wah and Chen, 2006). Further improvement in the support of conditional effects of classical planners, as well as their ability to accept grounded PDDL of moderate size, may boost the performance of T_0. In the future we would like classical planners to support better automated-generated instances, similarly to what is called *industrial* instances in the SAT solver competitions (Berre and Roussel, 2009). It might also be interesting to consider using a compact binarized form of STRIPS which can be fast and efficiently read into memory.

Using different instances of $K_{T,M}(P)$ allows to tradeoff efficiency for completeness and more combinations than the current one are possible. We tried out to do a first trial using $K_0(P)$ or only using $K_1(P)$ when the problem has conformant width 1. This issue should be explored further.

We would like to improve the current version of the T_0 conformant planner, based on instances of the $K_{T,M}(P)$ translation. So far, the planner uses the instance $K_1(P)$, which is complete for problems having conformant width 1, or the instance K_{models} that is complete for any problem, but might be exponentially larger to what is needed. We would like to develop an efficient implementations of $K_i(P)$ for $i > 1$, even though the current benchmarks has not needed it. Even in the conformant track of the IPC-2008, where the organizers introduced new benchmarks with conformant width greater than 1, we did not observe that using K_{models} was a limitation, meaning that the tags and merges generated by K_{models} were the merge that would have generated the appropriated instances of $K_i(P)$ for the width of those benchmarks.

Problems with conformant width i can be solved by the instance $K_i(P)$, whose tags have size i. Some problems may be solvable using smaller tags for some literals. Besides, observe that if $\{t_1, \ldots, t_n\}$ is a valid merge for a problem P, the following merge rule is sound

$$KL/\{t_1, t\} \wedge \ldots \wedge KL/\{t_n, t\} \rightarrow KL/t.$$

We would like to obtain instances of $K_{T,M}(P)$ smaller than $K_i(P)$, but also complete for problems with conformant width i.

Classical planning seems to be very restricted, but it has been used recently to tackle other flavors of planning. In probabilistic planning, simplified versions of the problems are solved with classical planners. The resulting sequences are tried, hoping the agent can arrive to a goal, or planning from scratch if they fail (Yoon et al., 2007). In planning with complex preferences along the execution of actions, tasks are first transformed that preferences should be achieve during actions execution until the goal state, and then a modified versions of classical planning heuristics are used to guide the search (Baier et al., 2009).

Part IV

Conclusions

Chapter **8**

Extensions and Variations

…man can and does select the variations given to him by nature, and thus accumulate them in any desired manner.

Origin of Species by Charles Darwin

In this chapter we consider some extensions or variations of the ideas and algorithms presented in parts II and III.

We discuss first an extension of the model-counting formulation presented in Chapter 3 for solving conformant probabilistic planning problems; *i.e.*, problems with a probabilistic distribution over the possible initial states and probabilistic effects of actions. An extension to our scheme, together with important ideas for achieving good performance was proposed by Huang (2006).

In Section 8.2, we extend the basic translation $K_0(P)$ in a way different from the general extension $K_{T,M}(P)$. The new extension $K(P)$, first published by Palacios and Geffner (2006a), is incomplete but quite effective, and was the base of the planner KP that perform well in the conformant track of IPC-2006 (Bonet and Givan, 2006).

We then present, in Section 8.3, an extension to the translation $K_{T,M}(P)$ for supporting non-deterministic effects, based on the well known intuition of introducing uncertain literals for representing the possible effects of actions (Smith and Weld, 1998). We incorporate such extension in the conformant planner T_0 and refine it further by using properties of the $K_{T,M}(P)$ translation.

Finally, in Section 8.4, we show how to use the translation $K_{T,M}(P)$ for *optimal* conformant planning.

8.1 Logic-based Conformant Probabilistic Planning

Probabilistic conformant planning is planning where the initial situation is a probabilistic distribution over possible initial states, the effects of actions are also probabilistic, and the goal must be achieve with maximal probability by a plan of N time steps (Kushmerick et al., 1995; Majercik and Littman, 1998; Hyafil and Bacchus, 2003; Huang, 2006). In this section we relate the work of Huang (2006) on conformant probabilistic planning with our model-counting-based algorithm for conformant non-probabilistic planning presented in Chapter 3.

In this section we explore the use of our approaches to non-probabilistic conformant planning to the probabilistic case. A natural candidate for such exploration is our logic-based model-counting formulation because of two reasons. On one hand, a propositional formulation for conformant probabilistic planning was used by MAX-PLAN (Majercik and Littman, 1998) for conformant probabilistic planning.[1] On the other, d-DNNF has been applied to probabilistic reasoning, specially with Bayesian Networks (Darwiche, 2003; Chavira and Darwiche, 2008; Darwiche, 2009).

We start the section defining the probabilistic conformant planning model and then extend the non-probabilistic CNF encoding presented for the model-counting formulation (Chapter 3), following ideas of Majercik and Littman (1998). We show that Huang's algorithm can be understood as a refinement and extension of our algorithm to the probabilistic case. We also evaluate the difference in performance of Huang's algorithm and our extension, and comment on the possible impact of Huang's ideas on non-probabilistic benchmarks.

Conformant Probabilistic Planning

A conformant probabilistic planning problems is a tuple of the form $P = \langle F, I, O, G, N\rangle$ where F stands for the fluent symbols in the problem, I expresses the probability distribution over the possible initial situations, O stands for a set of (ground) operators or actions a, G is a set of literals over F defining the goal, and N is a number. A solution to a problem P is a plan of length N with maximal probability of success. The initial situation I is given by set containing fluents literals or expressions $p_1 t_1 \mid \ldots \mid p_n t_n$ such that t_i are set of fluent literals and $\sum_{i=1}^{n} p_i = 1$. Every action a in O has an effect $\mathit{Eff}(a)$ that is a set containing fluents literals or probabilistic effects $p_1 E_1 \mid \ldots \mid p_n E_n$, where $\sum_{i=1}^{n} p_i = 1$ and each E_i is a set of rules. Each rule is of the form $C \to E$, where C and E are set of fluent literals, and if C is empty, the rule is interpreted as $\mathit{true} \to E$.[2] For simplifying the presentation of this section, we assume that actions are always executable, and thus have no preconditions.

Given S, the set of the possible states of P, the initial situation I encodes a probabilistic density $p_0 : S \to [0, 1]$. If there are m elements of the form $p_1 t_1 \mid \ldots \mid p_n t_n$ in I, then the cross-product of them generates terms of the form $(p^1 \cdots p^m \;\; t^1 \wedge \ldots \wedge t^m)$. The set of possible initial states are the terms $t^1 \wedge \ldots \wedge t^m$ consistent with the literals of I not appearing in expressions $p_1 t_1 \mid \ldots \mid p_n t_n$. The probability of each such state is $p^1 \cdots p^m$. We assume that the probabilities of each possible initial states calculated in this way adds up one, making p_0 a probability density. This always holds if the atoms in $p_1 t_1 \mid \ldots \mid p_n t_n$ do not appear somewhere else in I.

The effect of applying an action a in an state s is defined as a conditional probabilistic

[1]They call it probabilistic planning, but wanted to obtain a plan with maximal probability of success.

[2]For probabilistics effects we follow the notation of Rintanen (2003). This simplified language is used for the sake of clarity of presentation and also for the preliminary implementation of the algorithms of this section.

density of the possible resulting states s' defined as

$$p(s'|s,a) = \sum\{p_i \mid s' = f(E_i, s)\},$$

where $f(E_i, s)$ is the result of applying the set of rules E on the state s, returning a new state.

Given a sequence of n actions $\pi = \{a_1, \ldots, a_n\}$, the probability density of obtaining a sequence of states $\{s_0, \ldots, s_n\}$ after the execution of π is

$$p_\pi(\{s_0, \ldots, s_n\}) = p_0(s_0) \prod_{i=1}^{n} p(s_i|s_{i-1}, a_i) \ ,$$

where p_0 is the probabilistic density of the possible initial states.

Observe that for a sequence of actions π that makes impossible to reach a state s_n, the probability of reaching it through any intermediary states $s_1, \ldots, s_{n-1}$ is zero. Thus, for calculating the probability of a sequence of actions π reaching the goal G, we add up all the possible sequence of length N such that their last states s_n satisfy the goal G, *i.e.*

$$Pr(\pi, G) = \sum_{\{s_0,\ldots,s_n\} \ \models \ G\in s_n} p_\pi(\{s_0, \ldots, s_n\})$$

that can be summarized as

$$Pr(\pi, G) = \sum_{\{s_0,\ldots,s_n\} \ \models \ G\in s_n} p_0(s_0) \prod_{i=1}^{n} p(s_i|s_{i-1}, a_i) \ . \tag{8.1}$$

Given a problem $P = \langle F, I, O, G, N \rangle$, a conformant probabilistic planning task is to obtain a sequence of N actions π^* with maximal probability of success $Pr(\pi^*, G)$.

We would like to express the task of obtaining the N-step plan π^* with maximal probability of success in the problem P in terms of of its propositional encoding $T_N(P)$ presented in Section 3.2 on page 29. It would allow us to calculate $Pr(\pi, G)$ using a new propositional encoding $T'_N(P)$ and a representation of the actions sequence π using propositional literals.

The MAXPLAN algorithm (Majercik and Littman, 1998) uses propositional variables, called *chance variables*, for encoding the probabilities appearing in the initial situation and action effects. For a problem P, MAXPLAN creates a formula whose models correspond to all the possible N-step plans for P, and all the possible results of the probabilistics effects.

We now show how to enrich the PDDL language for expressing conformant probabilistic problems and how to obtain the new encoding $T'_N(P)$ for them.

Probabilistic PDDL and its CNF encoding

We describe PROBABILISTIC PDDL, a simple extension of PDDL for expressing probabilistic uncertainty in the initial state and in the actions effects, through examples (Younes and Littman, 2004; Bonet and Givan, 2006). The initial situation can contain expressions of the form

```
(probabilistic 0.2 (a) 0.8 (and (b) (c)))
```

meaning `(a)` is true with probability 0.2 and `(b)`$\wedge$`(c)` is true with probability 0.8. The CNF encoding $T'_N(P)$ uses two new propositional variables, called *chance variables*, p^1 and p^2, one for each probability in the expression. We associate each chance variable with its corresponding term, labeled with time step 0, using clauses for expressing $p^1 \leftrightarrow a_0$ and $p^2 \leftrightarrow b_0 \wedge c_0$. Finally, clauses are added for encoding that at least one p^i is true, and not two different p_i can be true at the same time.

Actions may have one or more probabilistics effects, for example, the expression

```
(:action o
    :effect (and
        (probabilistic 0.3 (a) 0.7 (when (b) (c)))
        ...
        (d)
    )
)
```

means that after applying action o, `(d)` will be true for sure, with probability 0.3, `(a)` will be true, and with probability 0.7, if `(b)` is true in the state where the actions is applied, then `(c)` will be true afterwards.

The propositional theory $T'_N(P)$ has literals corresponding to actions and fluents for each time step t until the horizon N. The introduction of chance variables introduces small changes respect to the encoding $T_N(P)$ and we describe them using time-stamped literals and actions, abusing slightly of the PDDL syntax, as follows

```
(:action o_t
    :effect (and
        (probabilistic 0.3 (a_{t+1}) 0.7 (when (b_t) (c_{t+1})))
        ...
        (d_{t+1})
     )
).
```

The probabilistics effects of an action o at time step t can be understood as the effects of o_t depending on chance variables p_t^i, which probability distribution is described like in the initial situation as

$$\texttt{(probabilistic 0.3 (}p_t^1\texttt{) 0.7 (}p_t^2\texttt{)).}$$

The new chance variables should depend on t, as different executions of a probabilistic effect behave differently. Now the action o at time step t can be expressed in terms of these chance variables as

```
(:action o_t
    :effect (and
        (when (p_t^1) (a_{t+1}))
        (when (p_t^2) (when (b_t) (c_{t+1})))
        ...
        (d_{t+1})
    )
)
```

In summary, for encoding the probabilistic effects of an action o at time step t in CNF, its effects are conditioned on chance variables p_t^i. We also add clauses expressing that the p_t^i vars are exclusive of each other, and that the action o_t is executed if and only if at least one p_t^i is true. This guarantees that when the action o is not executed all its chance variables are false.

The CNF theory $T'_N(P)$ constructed in this way guarantees that any assignment of actions and chance variables corresponds to a unique model encoding one possible execution. The probability of reaching the goal in such execution is the product of the probabilities associated to the positive p^i appearing in the model.[3] The probability of success of a plan π is the sum of the probabilities of each possible execution consistent with π. If π is not a plan for a possible execution, then it will be inconsistent with the CNF theory and will have, thus, zero probability of success. We encode action sequences π using the set of action literals $T_A(\pi)$, also called an *action set*, where $a_t \in T_A$ iff a is $\pi[t]$, and $\neg a_t \in T_A$ iff a is not $\pi[t]$.

The *weighted model counting* (WMC) of a propositional theory ϕ and a weight function w returns the sum of the weight of each model of ϕ, that is the product of the weight of each literal $w(l)$ in such model. Thus, the probability of success of a plan can be calculated using weighted model counting over the theory $T'_N(P)$ as

$$Pr(\pi, G) = \mathsf{WMC}(\ T'_N(P), w_\pi\ ), \tag{8.2}$$

where the weight function is

$$w_\pi(l) = \begin{cases} 0 & : \text{ if l is inconsistent with the action set } T_A(\pi) \\ \psi & : \text{ if l is a positive chance variable } p_t^i \text{ with probability } \psi \\ 1 & : \text{ otherwise.} \end{cases}$$

A naive algorithm for obtaining a plan of length N with maximal probability of success would generate all possible plans π, compute their probability using $Pr(\pi, G)$, and return a plan with maximal probability. In practice, a branch-and-bound algorithm is more effective, using a criterion for pruning branches that will necessary lead to plans with lower probability of success that the best found so far. For this we need an algorithm $val(\Delta, T_A)$ that, given a propositional encoding abbreviated as Δ, returns the probability of the best plan *consistent* with T_A, a *partial* action set of literals.[4]

Huang (2006) algorithm for computing $val(\Delta, T_A)$ is:

[3]It is possible to use less chance variables by assigning probabilities to literals $\neg p^i$ (Majercik and Littman, 1998; Huang, 2006), but we presented this way for simplicity of presentation, and used it for the experiments reported below as they are very preliminary.

[4]Recall that an action set T_A is partial when there are action variables that do not appear in T_A. See page 31.

- $val(\Delta, T_A) = max(val(\Delta \,|\, a, T_A),\ val(\Delta \,|\, \neg a, T_A))$ if Δ mentions some action variable a;
- $val(\Delta, T_A) = val(\Delta \,|\, p, T_A) \cdot Pr(p) + val(\Delta \,|\, \neg p, T_A)$ if Δ mentions no action variables, but some chance variable p;
- $val(\Delta, T_A) = 1$ if Δ mentions no action or chance variables, and is consistent with T_A;
- $val(\Delta, T_A) = 0$ if Δ mentions no action or chance variables, and is inconsistent with T_A.

Following Huang, the idea behind this algorithm saying is that the first case of the definition says that the actions should be chosen to give the maximum probability of success; the second case says that for a complete sequence of actions chosen, the success probability is the weighted average between the success probabilities under the two complementary scenarios p and $\neg p$, for each chance variable p; the other two cases ensure that all and only the scenarios satisfying the goal are counted. Actually, for the second case Huang uses instead $val(\Delta \,|\, p, T_A) \cdot Pr(p) + val(\Delta \,|\, \neg p, T_A) \cdot (1 - Pr(p))$ because in his encoding negative chance variables are also associated to probabilities.

As weighted model counting, the evaluation of *val* can be very expensive, as it is enumerating all the models of a formula. Weighted model counting is tractable for d-DNNF (Darwiche and Marquis, 2004), thus we would like to get a a similar algorithm for calculating *val* on d-DNNF theories. As Huang points out, evaluating $val(\cdot)$ involves a sequence of maximizations followed by a sequence of summations. If we wanted to evaluate $val(.)$ in linear time in a d-DNNF we should compile it with all actions on top, and then all the chance variables. Such restriction make the theories very difficult to compile, and for the non-probabilistic case allows to obtain a solution without any search. Huang (2006) explain that such restriction increased the *constrained treewidth*, that is usually much higher than the normal treewidth (Park and Darwiche, 2004). Given that compiling the theory using such order is not feasible, we consider an approximation to *val* that provides enough pruning for developing an effective branch-and-bound algorithm for conformant probabilistic planning.

Extending the model-counting-based algorithm

The weighted model counting of a propositional theory can be calculated in linear time if the theory is in d-DNNF (Darwiche, 2001a). Recall that a d-DNNF is a propositional formula represented as a graph, where the leaves are literals and the inner nodes can be labelled with $\vee$ or $\wedge$ (see fig. 3.1 on page 37). The $\vee$ nodes in the d-DNNF are deterministic, *i.e.* $\vee$-children are inconsistent each other and it is safe to sum them; and that the $\wedge$ nodes are decomposable, *i.e.* $\wedge$-children do not share literals and it is safe to multiply them. Both determinism and decomposability are crucial properties for computing the number of models or the weighted model counting of a d-DNNF, by replacing labels $\vee$ with $+$ and labels $\wedge$ by $*$, and evaluating bottom-up the resulting *arithmetic circuit* (Darwiche, 2003; Chavira and Darwiche, 2008; Darwiche, 2009).

Using the WMC algorithm for d-DNNF theories, the probability of a plan π reaching the goal given the propositional theory $T'_N(P)$ can be calculated as

$$\mathsf{WMC}(\ T'_N(P), w_\pi\ ) = \mathit{prob}(\Delta, T_A)$$

where Δ is $T'_N(P)$ converted to d-DNNF, T_A is the action set corresponding to the plan π, and

$$\mathit{prob}(\Delta, T_A) = \begin{cases} 0 & : \text{if } \Delta \text{ is literal inconsistent with the action set } T_A \\ \psi & : \text{if } \Delta \text{ is a positive chance var } p_t^i \text{ with probability } \psi \\ 1 & : \text{for any other literal} \\ \prod_i \mathit{prod}(\Delta_i, T_A) & : \text{if } \Delta \text{ root is the operator } * \text{ and } \Delta_i \text{ are the children of } \Delta \\ \sum_i \mathit{prod}(\Delta_i, T_A) & : \text{if } \Delta \text{ root is the operator } + \text{ and } \Delta_i \text{ are the children of } \Delta\ . \end{cases}$$

The algorithm $\mathit{prob}(\Delta, T_A)$ does not work when the action set T_A is partial, [5] and thus cannot be used for a branch and bound algorithm. Indeed, we need an algorithm $\mathit{prob}'(\Delta, T_A)$ that returns the probability of the best plan *consistent* with T_A.

For obtaining the algorithm *prob'*, let us recall the model-counting formulation for conformant non-probabilistic planning of Chapter 3 that computes the number of initial states consistent with the action set T_A (see page 33)

$$\mathsf{MC}(\mathsf{project}[\ T_N(P) + T_A\ ;\ F_0\ ]) \tag{8.3}$$

where F_0 is the set of variables appearing in the propositional encoding of the initial situation I of P. If a for partial plan T_A such number is lower than the number of possible initial states, T_A could not be extended to encoded a conformant plan, and thus can be discarded from the search tree.

Let us recall the algorithm for model counting and projection `MC-and-P` (Figure 3.4 on page 42) for computing the equation (8.3) for theories compiled into d-DNNF. The version presented here, called *MCP*, is written at a higher level and leave out the possibility of doing conditioning that `MC-and-P` detailed, even though it is also necessary by the algorithms of this section.

$$\mathit{MCP}(\Delta, T_A) = \begin{cases} 0 & : \text{if } \Delta \text{ is a literal inconsistent with the action set } T_A \\ 1 & : \text{for any other literal} \\ \prod_i \mathit{MCP}(\Delta_i, T_A) & : \text{if } \Delta \text{ root is the operator } * \text{ and } \Delta_i \text{ are the children of } \Delta \\ \sum_i \mathit{MCP}(\Delta_i, T_A) & : \text{if } \Delta \text{ root is the operator } +,\ \Delta \text{ root is a decision node over a variable of the initial situation, and } \Delta_i \text{ are the children of } \Delta \\ \max_i \mathit{MCP}(\Delta_i, T_A) & : \text{if } \Delta \text{ was not in the previous case but } \Delta \text{ root is the operator } + \text{ and } \Delta_i \text{ are the children of } \Delta \end{cases}$$

[5] Recall that an action set T_A is partial when there are action variables that do not appear in T_A. See page 31

$$UB(\Delta, T_A) = \begin{cases} 0 & : \text{if } \Delta \text{ is inconsistent with partial plan } T_A \\ \psi & : \text{if } \Delta \text{ is a positive chance var } p_t^i \text{ with probability } \psi \\ 1 & : \text{for any other literal} \\ \prod_i UB(\Delta_i, T_A) & : \text{if } \Delta \text{ root is the operator } * \text{ and } \Delta_i \text{ are the children of } \Delta \\ \sum_i UB(\Delta_i, T_A) & : \text{if } \Delta \text{ root is the operator } +, \Delta \text{ root is a decision node over a variable of the initial situation, and } \Delta_i \text{ are the children of } \Delta \\ \max_i UB(\Delta_i, T_A) & : \text{if } \Delta \text{ was not in the previous case but } \Delta \text{ root is the operator } + \text{ and } \Delta_i \text{ are the children of } \Delta \end{cases}$$

Figure 8.1: Extension of the model-counting algorithm for conformant planning to the probabilistic case

A *decision node* is an $\wedge$ node with two children, one corresponding to a positive literal A, the other to the negative literal $\neg A$, following the *Shannon expansion* $(\Delta \mid A \wedge A) \vee (\Delta \mid \neg A \wedge \neg A)$. Where $\Delta \mid A$ is the *conditioning* of Δ on A, equivalent to replace all the literals A ($\neg A$, respectively) by true (false, respectively) in Δ.

The compilation step to d-DNNF for the algorithms presented in the CNF-based part, both the model-counting formulation of Chapter 3 and the SAT-formulation of Chapter 4, was guaranteed to keep the variables encoding the initial situation to be on top of the d-DNNF and to be decision nodes. This is needed for the algorithm *MCP* to be sound on computing the Equation 8.3. More details in Section 3.5 on page 38.

We can construct an upper bound to $val(.)$ by modifying the *MCP* algorithm on the preceding page. The algorithm UB in figure 8.1 differs from `MCP` only in the treatment of chance literals. Huang propose a similar algorithm called $val'(.)$ to the one in figure 8.1, defined as

$$val'(\Delta, T_A) = \begin{cases} 0 & : \text{if } \Delta \text{ is inconsistent with partial plan } T_A \\ \psi & : \text{if } \Delta \text{ is a positive chance var } p_t^i \text{ with probability } \psi \\ 1 & : \text{for any other literal} \\ \prod_i val'(\Delta_i, T_A) & : \text{if } \Delta \text{ root is the operator } * \text{ and } \Delta_i \text{ are the children of } \Delta \\ \max_i val'(\Delta_i, T_A) & : \textit{if } \Delta \textit{ root is the operator } +, \Delta \textit{ root is a decision node over an action variable, and } \Delta_i \textit{ are the children of } \Delta \\ \sum_i val'(\Delta_i, T_A) & : \textit{if } \Delta \textit{ was not in the previous case but } \Delta \textit{ root is the operator } + \textit{ and } \Delta_i \textit{ are the children of } \Delta \end{cases}$$

where we *emphasize* the two cases where *UB* and *val'* differ. Actually, it can be proved that both algorithms *UB* and *val'* compute the same number for the propositional encoding $T'_N(P)$ if 1) all the variables but chance and actions vars are forget away during compilation to d-DNNF, and 2) the same strategy used by us in Chapter 3 and by Huang (2006) is use for compiling to d-DNNF; *i.e.* to split initial chance variable first, then actions at time 0, then chance effects at time 0, actions at time 1, and so on.

Branch-and-bound algorithm

We have seen an upper bound UB of the probability of success of a plan and saw that Huang's val' is equivalent. We now discuss the difference in variable and value ordering, and some other optimizations in both the algorithm in Chapter 3, called `vplan`, and Huang's called `ComPlan`.

For `vplan` we considered the ordering for selecting the propositional variables, preferring action literals l that appears in most models. We use the same strategy for both serial and parallel planning. In contrast, `ComPlan` is tailored for serial plans, and consider the variable to be the time step k to be assigned, and the value the action at k. `ComPlan` selects k such that setting an action literal will produce the tightest bound. This is a most-critical heuristics that is usually very useful in other branch-and-bound algorithms (Marriot and Stuckey, 1999). However, for calculating such bound, `ComPlan` requires to calculate $val(\Delta, T_A + a)$ for each possible action a, that cannot be done simultaneously for all actions in one pass over the d-DNNF graph, in contrast to the criterion used by `vplan`. However, their experiments support paying such cost, specially because they take additional advantage by pruning actions such that $val(\Delta, T_A, a) \leq lb$, where lb is the probability of success of the best plan found so far. We call this pruning *strong pruning.*

Experiments

For studying the differences between `ComPlan` and the straightforward extension of our model-counting for the probabilistic case, we made experiments comparing the impact of both algorithms. Based on the source-code of `vplan` we extended it to support probabilistic planning using the lower bound calculated by UB.[6] We also built over `vplan` a preliminary implementation of Huang's `ComPlan` algorithm, as it have some syntactical restriction and do not implement an optimization of `ComPlan` called *variable sharing* for reducing the size of the propositional theory. We report the performance of `vplan`-strong, that is similar to our `vplan` with the UB bound, but uses the stronger pruning of Huang's `ComPlan`. In all cases, given an horizon N, a CNF is generated, then is compiled into d-DNNF, and the problem with maximum probability of success is found. Only the search time is reported. We choose to not show the success probability of the plans as they are the same for all planners. Obviously, the success probability increases with the horizon N.

In Table 8.1 we report the number of nodes and time spent by `vplan`, `vplan`-strong and `ComPlan` in the benchmarks by Huang (2006), running the experiments on a Intel/Linux machine running at 2.80GHz and 2GB of memory. The number of nodes found for `ComPlan` is similar to the reported but Huang, validating our preliminary

[6]The algorithms presented in this section can be run using the options 'mc' and '-prob' of the conformant planner 'Translator' available at `http://www.ldc.usb.ve/~hlp/software`.

		vplan		vplan-strong		ComPlan	
prob	horiz	nodes	time	nodes	time	nodes	time
castel	20	40.030	6,51	9.134	7,7	1.326	1,73
castel	21	68.100	18,29	15.472	14,13	2.052	2,66
castel	22	98.130	18,88	21.842	20,98	2.760	3,96
castel	23	157.362	32,28	35.394	38,67	3.878	5,75
castel	24	245.042	51,62	53.576	61,64	6.240	17,28
castel	25	318.118	98,77	66.142	86,29	8.224	14,95
castel	26	531.764	134,11	113.728	148,47	13.394	23,8
castel	27	745.646	282,54	166.978	266,78	19.350	38,33
castel	28	1.124.610	452,17	267.986	417,48	29.678	65,91
castel	29	1.470.022	549,51	367.024	625,64	36.604	81,03
castel	30	2.475.358	927,19	652.238	1132,12	58.052	134,43
castel	31	2.641.562	957,94	735.040	1340,41	70.590	181,93
castel	32	6.619.584	2660,28	1.919.562	3471,54	115.338	305,81
castel	33					199.770	551,06
castel	34					190.246	535,12
castel	35					704.966	2006,91
castel	36					959.268	2855,6
castel	37						
gripper	10	20.292	7,77	2.702	3,71	1.792	3,09
gripper	11	74.164	33,63	11.404	16,63	6.916	12,79
gripper	12	196.782	105,16	21.866	37,6	18.806	37,89
gripper	13	537.378	341,6	61.872	130,49	105.592	227,29
gripper	14	1.595.740	1332,28	127.748	291,38	362.090	1046,93
gripper	15			442.192	1459,8		
gripper	16						
grid	18	146.096	85,33	25.832	65,8	25.784	67,47
grid	19	21.454	36,73	4.884	29,91	4.736	42,86
grid	20	48.918	152,49	66.798	529,7	13.164	127,78
grid	21	31.464	139,85	178.760	2203,89	6.422	107,36
grid	22	103.690	624,22			21.906	427,11
grid	23	74.256	763,34			12.726	342,63
grid	24	267.314	1988,36			47.916	1878,93
grid	25	183.006	2474,98			28.526	1529,94
grid	26					74.040	3046,82
grid	27						

Table 8.1: Comparison of vplan, vplan-strong and ComPlan over domains taken from Huang (2006). Times are in seconds. Empty spots means more than 3600 seconds. 'Castle' is about building a sand castle, 'Gripper' about picking balls, and 'Grid' about moving in a grid with actions that may end in other cell.

problem	vplan nodes/time/len	vplan-strong nodes/time/len	ComPlan nodes/time/len	SAT-based time/len
sqr-center-4	352/0,65/8	61/0,63/8	165/0,68/8	0,64/8
sqr-center-8	676k/to	150k/to	350k/to	**412,5/20**
sqr-center-16	140k/to	9k/to	11k/to	mo
sortnet-3	7/0,17/3	7/0,18/3	9/0,19/3	0,17/3
sortnet-4	489/0,44/5	197/0,42/5	480/0,49/5	0,36/5
sortnet-5	145k/to	337k/to	551k/to	**28,34/9**
cube-3	6/1,25/6	5/0,50/6	5/0,50/6	0,51/6
cube-5	52k/to	**71k/2766/15**	9k/to	**711,1/15**
cube-7	27k/to	4k/to	2k/to	mo
safe-5	307/0,30/5	123/0,55/5	277/0,43/5	0,29/5
safe-10	2171k/to	444k/to	799k/to	**27,09/10**
safe-30	44k/to	4k/to	6k/to	mo
blocks-1	4/0,34/4	1/0,33/4	1/0,31/4	0,35/4
blocks-2	43k/to	**22k/2526/13**	2k/to	**10,19/13**
blocks-3	640/to	122/to	130/to	mo
coins-1	125/4,05/9	7/3,51/9	8/3,27/9	4,54/9
coins-2	125/4,27/9	7/3,83/9	8/3,22/9	2,82/9
coins-3	148/5,15/10	8/4,05/10	7/4,02/10	3,46/10
coins-4	125/4,06/9	7/5,77/9	8/3,34/9	2,90/9
coins-5	148/5,33/10	8/4,25/10	7/4,24/10	6,79/10
comm-1	17/2,09/11	7/2,07/11	3/2,05/11	2,06/11
comm-2	31/33,36/17	9/54,35/17	5/31,59/17	31,48/17
comm-3	45/1038/23	19/1293/23	10/1063/23	1010/23
uts-k1	10/0,19/4	5/0,22/4	7/0,23/4	0,21/4
uts-k2	488k/2019/10	**14k/313,2/10**	**11k/263,3/10**	4,61/10
uts-k3	2k/to	1k/to	1k/to	**890,5/16**
uts-k4	4k/to	34/to	34/to	to
uts-l1	10/0,34/4	5/0,21/4	7/0,22/4	0,24/4
uts-l2	4k/3,34/10	227/2,35/10	308/2,88/10	1,37/10
uts-l3	482k/to	**16k/1773/16**	17k/to	15,66/16
uts-l4	107k/to	3k/to	6k/to	**303,2/22**

Table 8.2: Comparison of vplan, vplan-strong, ComPlan and satconf over non-probabilistic conformant domains. 'to' means time out of 3600 seconds. 'mo' means memory out of the 2GB available. A suffix k means that number has been divided by 1000 and rounded up. We put in **bold** face when an algorithm outperform the rest of the based on branch-and-bound.

implementation. Fixing the precision of the probabilities to six decimals was critical to approximate the number of nodes reported in Huang (2006). We observe that ComPlan definitively dominates in castle, but in gripper and grid vplan and ComPlan solve almost the same number of problems. The pruning in vplan-strong makes a difference in gripper-15 but is very expensive in the grid domain. The planner ComPlan has an slower node-generation rate than vplan. For the largest instances solved by ComPlan, it generates 337,15, 345,85 and 18,64 nodes per second, for castle-32, gripper-14 and grid-25, respectively. In contrast, vplan generates 2488, 1197,75 and 73,94 nodes per second, respectively. The node-generation rate of vplan-strong is faster than ComPlan but slower than vplan.

Even though the ideas in `ComPlan` were developed for the non-probabilistic setting, we also tried `ComPlan`, `vplan`-strong in pure conformant planning problems, for compare them with our `vplan` algorithm. The results are reported in Table 8.2. In non-probabilistic problems, the algorithm of `ComPlan` is modified to return the first plan that achieve the goal for all the initial states, which is to achieve them with probability one (Huang, 2006). We tested over square-center and sortnet, two problems used for the experiments in Chapters 3 and 4, and also used the problems cube and safe from the Conformant FF benchmarks (Hoffmann and Brafman, 2006), and blocks, coins, comm and uts as used in the conformant track of the IPC-2006 (Gerevini et al., 2009). For these experiments we added up the search time for all horizons until the optimal one, and included also the time used for CNF generation, d-DNNF compilation, that is the same for the three algorithms. As a reference we also compare with the planner developed in Chapter 4, that also uses d-DNNF but for generating for each horizon N an new CNF, obtaining a plan in a single SAT call if there is one.

In Table 8.2, the algorithms `vplan` and `ComPlan` are able to solve the same problems, almost in the same time, except for the instance uts-k2. `ComPlan` generates less nodes but the slower generation rate compensate the advantage. `vplan`-strong is able to solve problems that `ComPlan` cannot solve like cube-5, blocks-2 and uts-l3. The SAT-based planner solves an extra instance than `vplan`-strong in the problems square-center, sortnet, safe, uts-k and uts-l.

The algorithm of `ComPlan` was developed for the probabilistic case and it is not expected to perform well in non-probabilistics instances. In particular, their pruning and variable selection heuristics are designed for probabilities that allow strong pruning when a better lower bound is found. Anyway, it is interesting to observe its performance on pure conformant problems.

Huang (2006) argues that *"Having variable orders of bounded width implies that the size of the d-DNNF compilation will grow at most linearly with the horizon, which will allow us to ultimately scale the proposed approach to large horizons."* However, sometimes being able to compile into d-DNNF is not enough for achieving good performance. For example, in the instance blocks-2 in table 8.2, `ComPlan` and `vplan` are reported to generate 0,55 and 11,94 nodes per second. Even though d-DNNF are linear in the size of the compiled formulas, and that the compilation strategy used by `vplan` and `ComPlan` leads to bounded treewidth, the resulting d-DNNFs can be quite big. We would like to observe the performance of `ComPlan` on conformant probabilistic problems with more fluents and actions.

The same idea used for extending the `vplan` algorithm to the probabilistic case, could be used for conformant non-probabilistic non-deterministic planning. That can be implemented translating non-deterministic effects `(oneof (a) (b))` into `(probabilistic 0.5 (a) 0.5 (b))`, and increase the horizon until founding a plan with a success probability of 1. The issue of non-deterministic effects is explored in Section 8.3, presenting an extension to the translation $K_{T,M}(P)$ from conformant into classical planner introduced in Chapter 6.

8.2 An alternative extension of K_0

While translation $K_{T,M}(P)$ presented in Chapter 6 is based on conditionals KL/t that represent that L *must be true if* t *is initially true*, this alternative translation $K(P)$ presented earlier by us (Palacios and Geffner, 2006a), uses conditionals L/t that represent that *if* t *then* L. The main difference is that t in KL/t literals refers to the initial situation, but for L/t they are associated with the current situation. As shown in (Palacios and Geffner, 2006a) and summarized here, this alternative translation provides also effective solutions to many of the existing conformant benchmarks, even if is not complete. Such translation, called $K(P)$, is made of the basic translation $K_0(P)$, new literals L/t, and a set of extra actions capturing deductions and characterized as rules.

In this section we discuss about the kind of reasoning we want to capture with $K(P)$, then introduce the corresponding extension, we prove its correctness and evaluate its performance through experiments. Later, we conclude on a final discussion.

Motivation

We want to extend the translation $K_0(P)$ presented in Chapter 5 to more expressive forms of reasoning, but without obtaining exponentially larger classical problems. Even though this retain us from obtaining completeness, we can still obtain interesting results.

Some conformant planning problems are solvable through simple rules. For example, a robot that systematically scans a grid, collecting the objects in each cell, will pick up all the objects in the grid, regardless of their original locations. Or similarly, a robot that moves n times to the right in an empty grid of size n, will necessarily end up in the rightmost column.

This raises the question of whether it is possible to identify and use such forms of inference for developing an efficient but incomplete conformant planner capable of solving non-trivial problems quickly. In this section, we show that this is possible by formulating a suitable translation of conformant problems into classical problems which are then solved by an off-the-shelf classical planner. The translation is sound as the classical plans are all conformant, but it is incomplete as the converse relation does not always hold. The translation scheme accommodates 'reasoning by cases' by means of a 'split-protect-and-merge' strategy; namely, atoms L/X_i that represent conditional beliefs 'if X_i then L' are introduced in the classical encoding that are then combined by suitable actions to yield the literal L when the disjunction $X_1 \vee \cdots \vee X_n$ holds and certain invariants in the plan are verified.

As an illustration, assuming that there is a *pickup*(l) action with precondition $at(l)$ and effect 'if $at(o,l)$ then $hold(o)$' with $at(o,l)$ unknown and $l \in L$, where L is a set of locations. In this case, the translation $K(P)$ introduces effects of the form 'if **true** then $hold(o)/at(o,l)$' for each $l \in L$. This way, the consequence of *pickup*(l) is the atom $hold(o)/at(o,l)$ that stands for the conditional belief 'if $at(o,l)$ is true, then $hold(o)$ is true'. Then any classical plan that achieves the atoms $hold(o)/at(o,l)$ for each one of the possible locations $l \in L$ of o, and which preserves certain invariants (like that the 'hidden' locations do not change), can be shown to be a valid conformant plan for achieving $hold(o)$.

By accounting for this type of simple disjunctive reasoning in a translation scheme that has a clear semantics, we will see that many other patterns of inference fall into place. For example, if in the robot domain above 'push' actions that move objects from one cell to the next are added (for each one of the possible directions), and at the same time, the pick up actions are restricted to particular cells (like corners or centers), then the classical encoding would produce valid conformant plans where enough pushes are done so that all objects are forced into such cells *regardless of their original location*, from which they can be collected. Several effective but incomplete formulations of conformant planning have been formulated before (some of which handle sensing as well; see Baral and Son, 1997; Petrick and Bacchus, 2002), none, as far as we know, can represent these types of plans, except those that are complete and thus exponential in the worst case, like our translation $K_{T,M}(P)$ in Chapter 6.

We call the new translation $K(P)$, and will make it stronger by accounting for certain disjunctive inferences in the translation. This will result into more actions and conditional effects added to $K(P)$ which is initially set to $K_0(P)$.

Case Analysis over Single Actions

Consider an action a that in a given context C' can force a literal L to make the transition from false to true, while preventing the opposite transition. In such a context C', even if L is unknown, a can be used to make L true. This type of inference is captured in the translation as follows:

Rule 8.1 (Action Compilation)**.** *If P contains a rule $a : C \land \neg L \rightarrow L$, and the rules for the same action a with $\neg L$ in the head are $C_i \rightarrow \neg L$, $i = 1, \ldots, n$ for $n \geq 0$, then add to $K(P)$ the rules $KC \land K\neg L_1 \land \cdots \land K\neg L_n \rightarrow KL$ where L_i is a literal in C_i.*

This is a modular translation rule in which the context C' above is the formula $C \land \neg L_1 \land \cdots \land \neg L_n$, for any combination of literals L_i chosen as to preempt the rules $C_i \rightarrow \neg L$ associated with the same action a that can clobber L. All the literals in C' are preceded by K's as they refer to literals in $K(P)$ that ensure that the condition holds with certainty.

It is not difficult to show that this translation rule preserves soundness. A key characteristic of the rule and others to be introduced below is that they make use of the conditional effects $a : C \land X \rightarrow L$ in the problem P for deriving L with certainty when the body of the rule $C \land X$ is not fully known.

In an example like 'square center', where a robot moves in an empty square grid and literals X_i are used to represent the column location of the robot, this translation ensures that literal $K\neg X_1$ is obtained right after a single 'move right' action (namely, that the robot cannot be in the leftmost column then), and similarly, that $K\neg X_2$ is obtained after two consecutive right moves, etc. If the grid is nxn, the resulting classical theory yields $K\neg X_i$ for all $i < n$ after $n - 1$ steps, although it does not yield KX_n (being in the rightmost column). For this, the disjunction expressing the possible column positions, namely $X_1 \lor X_2 \lor \cdots \lor X_n$, needs to be taken into account as well. We address this next.

Case Analysis over Action Sequences

We extend the translation further so that the disjunctions in P are taken into account in a form that is similar to the Disjunction Elimination inference rule used in Logic (Barwise and Etchemendy, 1991):

$$\text{If } X_1 \vee \cdots \vee X_n,\ \ X_1 \supset L, \ldots, \text{ and } X_n \supset L \ \text{ then } \ L \tag{8.4}$$

For this, we create new atoms in $K(P)$, written L/X_i, that aim to capture the conditional beliefs $X_i \supset L$. Then, the resulting classical encoding will be such that when these atoms are 'achieved' for each $i = 1, \ldots, n$, and they are suitably 'protected', the literal L will be rendered 'achievable' by means of an extra 'dummy' action with conditional effect similar to (8.4).

As already mentioned, the atoms L/X_i will stand for the conditional belief 'if X_i then L'. In principle, any rule $a : C \wedge X_i \rightarrow L$ in P with X_i uncertain can be used to produce a rule $a : KC \rightarrow L/X_i$ in $K(P)$, meaning that if KC is known and a is applied, then if X_i was true, L will become true. However, we want L/X_i to mean exactly that '*right after the action* a, if X_i is true, then L is true', and for this, some additional care is needed. Indeed, if a contains also rules $a : C_k \rightarrow X_i$ that can make X_i true, it may be possible that L and X_i are false at time t when a is applied, and that L remains false but X_i becomes true, and then that 'if X_i at t, then L at $t+1$' is true, but 'if X_i at $t+1$, then L at $t+1$' is false. In order to rule out this situation we define the corresponding translation rule as follows:

Rule 8.2 (Split). *For each rule $a : C \wedge X_i \rightarrow L$ in P where X_i is a literal that appears in a disjunction $X : X_1 \vee X_2 \vee \cdots \vee X_n$, if $a : C_k \rightarrow X_i$, $k = 1, \ldots, m$ for $m \geq 0$ are the rules in P for the same action a with X_i in the head, then add to $K(P)$ the atoms L/X_j, $j = 1, \ldots, n$, all initialized to **false**, and the rules $a : KC \wedge K\neg L_1 \wedge \cdots \wedge K\neg L_m \rightarrow L/X_i$ where L_k is a literal in C_k.*

The combination of the conditional beliefs represented by the atoms L/X_i is achieved by means of extra actions added to the classical encoding $K(P)$ that generalize (8.4) slightly, allowing some of the cases X_i to be disproved:[7]

Rule 8.3 (Merge). *For each disjunction $X : X_1 \vee \cdots X_n$ and atom L in P such that L/X_i is an atom in $K(P)$, add to $K(P)$ a new action $a_{X,L}$ with conditional effect*

$$(L/X_1 \vee K\neg X_1) \wedge \cdots \wedge (L/X_n \vee K\neg X_n) \wedge FLAG_{X,L} \rightarrow KL$$

*where $FLAG_{X,L}$ is a boolean initialized to **true**. If $L = X_i$ for some $i \in [1, n]$, remove the conjunct $(L/X_i \vee K\neg X_i)$ from the rule body.*

A key distinction from Logic is that the disjunction $X_1 \vee \cdots \vee X_n$ and the conditional beliefs 'if X_i then L' represented by the atoms L/X_i need all be **preserved** until they are combined together to yield L. This is the purpose of the boolean $FLAG_{X,L}$ that is initially set to true, but which is deleted when an action is done in a context where it is not possible to prove that 1) L is preserved (if true), 2) the disjunction $X \vee L$ is preserved (the disjunction X is initially true but it is actually sufficient to

[7]When using the classical plans obtained from $K(P)$ as conformant plans for P, such 'dummy' actions must be removed, as was done for merge actions in the translations introduced in Chapter 6.

preserve the weaker disjunction $X \vee L$), and 3) the conditional beliefs represented by the atoms L/X_i achieved are preserved. This is accomplished by extending $K(P)$ with the following cancellation rules:

Rule 8.4 (Protect)**.** *If there is a boolean flag $FLAG_{X,L}$ in $K(P)$ for $X : X_1 \vee \cdots \vee X_n$, then for each action a: 1) if $a : C \rightarrow \neg L$ in P, add to $K(P)$ the rule $a : \neg K \neg C \rightarrow \neg FLAG_{X,L}$, 2) if $a : C \rightarrow \neg X_i$ in P and neither $a : C \rightarrow X_k$ nor $a : C \rightarrow L$ in P for X_i and X_k in X, add to $K(P)$ the rule $a : \neg K \neg C \rightarrow \neg FLAG_{X,L}$, and 3) if $a : C \rightarrow X_k$ for X_k in X, then add to $K(P)$ the rule $a : \neg K \neg C \wedge L/X_k \rightarrow \neg FLAG_{X,L}$.*

These rules, as we will see, yield expressivity without sacrificing efficiency, as they manage to *accommodate non-trivial forms of disjunctive inference in a classical theory without having to carry disjunctive information explicitly in the belief state:* disjunctive information is represented implicitly in $K(P)$ in terms of the the conditional atoms L/X_i, the 'merge' actions, and the *invariants* enforced by the 'flags'.

Theorem 8.1 (Soundness $K(P)$)**.** *Any plan that achieves the literal KL in $K(P)$ is a plan that achieves L in the conformant problem P.*[8]

The key element in the proof is the following lemma that captures the meaning of the L/X_i atoms:

Lemma 8.2 (L/X_i Atoms)**.** *Any plan that yields L/X_i while preserving $FLAG_{X,L}$ in $K(P)$ is a plan that achieves the conditional $X_i \supset L$ in P.*

Proof. Let us assume that L/X_i, which is initially false, is made true at time t by an action a in the plan. We need to prove that if $FLAG_{X,L}$ remains true in $K(P)$ until time $t' \geq t$, then the conditional $X_i \supset L$ remains true until t' in P, which we write as $X_i(t') \supset L(t')$. From the argument above, if L/X_i became true in $K(P)$ at time t, so does the conditional $X_i(t) \supset L(t)$ in P. From this, $X_i(t') \supset L(t')$ follows if we can show both $X_i(t') \supset X_i(t)$ and $L(t) \supset L(t')$. The latter is true because rule 8.4.1 in $K(P)$ ensure that if a rule $a' : C' \rightarrow \neg L$ gets triggered by the plan in P, the rule $a' : \neg K \neg C' \rightarrow \neg FLAG_{X,L}$ will be triggered by the plan in $K(P)$. Similarly, the former is true because rule 8.4.3 in $K(P)$ guarantee that if a rule $a' : C' \rightarrow X_i$ is triggered by the plan in P when L/X_i is true in $K(P)$, then the rule $a' : \neg K \neg C' \wedge L/X_i \rightarrow FLAG_{X,L}$ will be triggered in $K(P)$. In either case, $FLAG_{X,L}$ would be deleted, so if it is not, $X_i(t') \supset X_i(t)$ and $L(t) \supset L(t')$ must hold, and since $X_i(t) \supset L(t)$ holds, so must $X_i(t') \supset L(t')$. □

Proof of theorem 8.1. We proceed by induction on the length of the plan π. If π is empty, then π achieving KL in $K(P)$ means that L is true with certainty in the initial I of the conformant problem P, and thus π achieve L.

Otherwise, π is a sequence of actions π' and an action a. π achieve KL is either because (1) a cause KL directly or because (2) π' achieve it and a does not cause $\neg KL$.

[8] For this result to hold, we assume that for every pair of conflicting rules $a : C \rightarrow \alpha$ and $a : C' \rightarrow \neg\alpha$ associated with the same action a in P, the bodies C and C' are such that they contain a mutex pair L, L'. This mutex relation is enforced on the corresponding K-literals by adding to every effect $C'' \rightarrow KL$ associated with the 'dummy' merge action $a_{X,L}$ in $K(P)$, the effects $C'' \rightarrow K\neg L'$ and $C'' \rightarrow \neg KL'$.

For case (2), let us assume that a is not a merge but a normal action. We will show by contrapositive that if a does not achieve $\neg KL$ then a does not achieve $\neg L$, preserving L achieved by π'. Let us also assume that a cause $\neg L$ in P using the rule $a : C \rightarrow \neg L$ and that π' achieve some $K\neg C_i$ in $K(P)$, avoiding the execution of rule $a : \neg K\neg C \rightarrow \neg KL$ for C_i in C. In such case, by inductive hypothesis, π' achieve $\neg C_i$ in P and the rule $a : C \rightarrow \neg L$ would not achieve $\neg L$. Thus, π' does not achieve any $K\neg C_i$ and the rule $a : \neg K\neg C \rightarrow \neg KL$ produces $\neg KL$.

On the other hand, if a is a merge action, the only way to achieve $\neg KL$ is because a achieve KL' where L and L' are mutex. But if L and L' are mutex, then KL and KL' are also mutex because in the problem $K(P)$, without merge actions, each rule $a : KC \rightarrow KL''$ corresponds to a rule $a : C \rightarrow L''$, producing the same mutexes. Even considering merge actions, merging an atom KL also produces $K\neg L'$ (see footnote 8 on the preceding page). Thus, it is impossible to achieve KL' after π' achieving KL.

For case (1), *i.e.* if π achieve KL because a cause KL directly, it maybe through a rule $a : KC \rightarrow KL$, but then π' achieve KC and by inductive hypothesis achieve C, and $a : C \rightarrow L$ achieve L.

The other case to consider is that KL is achieved by a rule added by action compilation (rule) of the form $KC \wedge K\neg L_1 \wedge \cdots \wedge K\neg L_n \rightarrow KL$. Again, by inductive hypothesis π' achieve C and $L_1, \ldots, L_n$. Each L_i guarantee that no rule $C_i \rightarrow \neg L$ is applied. The rule $a : C \wedge \neg L \rightarrow L$ is applied when $\neg L$ was true after π'. Thus, in any case L is true afterwards.

A third option is that the action a is a merge achieving KL.

$$a : (L/X_1 \vee K\neg X_1) \wedge \cdots \wedge (L/X_n \vee K\neg X_n) \wedge FLAG_{X,L} \rightarrow KL$$

For any X_i if $K\neg X_i$ is achieved, then KL or another KX_j was made true. Otherwise $FLAG_{X,L}$ would have been deleted because of rule rule 8.4.2. Let us consider the set of preserved literals $X_P = \{X_i\}$ such that $K\neg X_i$ was not achieved. If X_P is empty, as $FLAG_{X,L}$ is still true, then KL is achieved by π' and thus L is achieved and the merge action does not delete it. If X_P is not empty, then atoms L/X_i were achieved for X_i not in X_P and by lemma 8.2 the conditional $X_i \supset L$ is achieved in P. Given that $X_1 \vee \ldots \vee X_n \vee L$ was preserved because $FLAG_{X,L}$ remains true, some X_i is true for a literal L/X_i, and hence L should holds after the merge.

□

Example

As an illustration, given an object O to be collected from an unknown location in a grid with two cells A and B using the actions $pick(X)$, $push(X, Y)$, and $go(X, Y)$, where X and Y are cells and the three actions have as a precondition that the agent is at X, it follows that if the agent is initially at A, the plans

$$\pi_1 = \{pick(A), go(A, B), pick(B)\}, \text{ and}$$
$$\pi_2 = \{push(A, B), go(A, B), pick(B)\},$$

achieve $hold(O)$ in $K(P)$, but the following plan does not

$$\pi_3 = \{pick(A), go(A, B), push(B, A)\} .$$

	P			Trans.	$K(P)$		
Problem	#Act	#Atoms	#Eff	time	#Act	#Atoms	#Eff
bomb-100-60	6060	320	24120	1.35	6260	1041	79560
cube-11	6	33	120	0.036	72	226	1152
cube-75	6	225	888	1.08	456	1789	8448
square-center-64	4	128	504	0.31	260	893	4796
square-center-240	4	480	1912	6.11	964	3833	18172
grid-4-5	174	155	444	5.65	183	351	1244
safe-100	100	101	100	0.11	101	304	804
logistics-4-10-10	3320	610	6640	3.52	3370	1321	13880

Table 8.3: Data concerning the translation of some conformant problems P into classical encodings $K(P)$. The sizes refer to the grounded versions, and all times are in seconds and they include grounding time.

If $at(O,A) \vee at(O,B)$ is the disjunction X in P and L is $hold(O)$, then using the split rule (8.2) π_1 achieves $hold(O)/at(O,A)$ and $hold(O)/at(O,B)$, π_2 achieves $K\neg at(O,A)$ and $hold(B)/at(O,B)$, while π_3 achieves both $hold(O)/at(O,A)$ and $K\neg at(O,B)$ but clobbers $FLAG_{X,L}$, preempting the merge action $a_{X,L}$ from achieving $hold(O)$. This happens because the rule

$$push(B,A) : at(O,B) \rightarrow at(O,A)$$

in P yields the rule

$$push(B,A) : \neg K\neg at(O,B) \wedge L/at(O,A) \rightarrow \neg FLAG_{X,L}$$

which gets triggered in $K(P)$ by the action sequence π_3.

Experimental Results

We implemented and studied the performance of the KP conformant planner, that takes a conformant problem in PDDL, translate into a classical planning problem using the $K(P)$ translation, solve it using FF classical planner, and report the resulting plan without the merge actions, or reports *fail* if it cannot find a plan. Table 8.3 shows data concerning the translation of a number of problems from various sources, used and explained by Brafman and Hoffmann (2004). Bomb-x-y refers to the Bomb-in-the-toilet problem with x packages, y toilets, and clogging. Cube-n to the problem of reaching the center of a cube of size n^3 from a completely unknown location. Square-Center-n is similar but involves only n^2 possible locations. Logistics-i-j-k, Grid-n and Safe-n are from (Brafman and Hoffmann, 2004). The table provides information about the size of the original (ground) conformant problems P, the resulting classical problems $K(P)$, as well as the time taken in the translation. This last figure is less than a second in most problems, but grows up to a few seconds in some.

In comparison with the performance of the translation step of T_0 reported in table 7.1, the increasing on number of conditional effects is much larger for T_0. For example, for the problem bomb-100-100, T_0 increase the number of rules by 300% but KP only increased it in 10%. For square-center, T_0 increased the number of rules by near 10000% but $K(P)$ increased them by 1000%.

The empirical results of tables 8.4 and 8.5 are over instances taken from the Conformant-FF distribution, from the IPC-2006 (Bonet and Givan, 2006), and some presented in

	T_0		KP		Conformant-FF	
problem	time	len	time	len	time	len
bomb-100-60	5,6	140	4,54	140	9,38	140
bomb-50-50	1,11	50	0,96	50	0,1	50
square-center-4	0,05	8	0,05	8	0,01	12
square-center-8	0,07	26	0,05	0	70,63	50
square-center-12	0,1	32	0,07	32	$> 2h$	
square-center-64	10,68	188	1,66	188	$> 2h$	
square-center-120	$> 1.8GB$		13,23	356	$> 1.8GB$	
corners-square-4-16	0,2	86	fail		13,13	140
corners-square-4-20	0,51	128	fail		73,73	214
corners-square-4-24	1,13	178	fail		320,9	304
corners-square-4-64	267,3	1118	fail		$> 2h$	
log-3-10-10	3,42	109	2,67	109	4,67	108
log-4-10-10	6,52	125	3,07	125	4,36	121
ring-4	0,09	13	fail		1,37	26
ring-5	0,1	17	fail		27,35	45
safe-100	0,18	100	0,26	100	1252,3	100
safe-50	0,09	50	0,09	50	29,37	50
safe-70	0,11	70	0,14	70	109,92	70
uts-k10	1,09	58	2,11	59	16,53	58
uts-l10	0,33	88	$> 2h$		1,64	59
comm-21	0,39	313	fail		10,39	269
comm-22	0,51	348	fail		17,31	299
comm-23	0,61	383	fail		27,04	329
comm-24	0,7	418	fail		37,52	359
comm-25	0,84	453	fail		56,13	389

Table 8.4: Evaluation of the planners T_0, $K(P)$ and Conformant-FF. Plan times in seconds and lengths over standard domains. 'fail' means that KP problem reported unsolvable by FF

.

Chapter 7. The experiments were run on a Linux machine running at 2.33 GHz with 8GB of RAM, with a cutoff of 2h or 1.8GB of memory. The version of T_0 used for this experiments was the reported in Palacios and Geffner (2007), and hence not the same reported in Chapter 7, as a full reimplementation of T_0 was done before the experiments of that chapter.[9]

Table 8.4 shows the plan times and lengths obtained by three conformant planners on several standard domains: T_0, $K(P)$ (Palacios and Geffner, 2006a), and Conformant FF (Hoffmann and Brafman, 2006). In all these domains, T_0 scale up very well with the exception of the Square-center-n family, where the task is to get to the middle of a grid of size $n \times n$ without having any information about the initial location. Here KP does best. Surprisingly, though, when the set of possible initial locations is restricted to the four corners as in the Corners-Square-n family, KP produces encodings without solutions. Conformant-FF does not appears better than KP in any domain that KP is able to deal with, except in UTS-L10. The problems reported in Table 8.5 are the family of grid problems introduced in Section 7.1. KP fails to

[9] During the reimplementation we found out that FF action selection was sensible to the appearing order of actions in the PDDL file. We crafted the new implementation of T_0 to imitate the ordering of actions generated by the old implementation. The exact match was not possible, and new simplifications and optimizations dominate in most of those difference in favor of the new T_0.

	T_0		KP	
Problem	len	len	len	len
dispose-12-1	17,77	709	14,02	683
dispose-12-2	39,62	811	317,42	826
dispose-12-3	$> 1.8GB$		2434,04	985
dispose-16-1	60,34	1357	454,91	1361
dispose-16-2	818,2	1748	4764,13	1680
dispose-16-3	$> 1.8GB$		$> 1.8GB$	
dispose-20-1	174,71	1926	$> 1.8GB$	
dispose-20-2	$> 1.8GB$		$> 1.8GB$	
push-to-4-1	0,16	64	$> 1.8GB$	
push-to-4-2	0,3	67	0,16	69
push-to-4-3	0,48	83	0,22	71
push-to-8-1	63,23	369	$> 1.8GB$	
push-to-8-2	928,63	452	5,7	289
push-to-8-3	1153,16	395	10,12	291
push-to-12-1	$> 2h$		$> 1.8GB$	
1-dispose-4-1	0,21	140	fail	
1-dispose-4-2	0,68	140	fail	
1-dispose-4-3	1,82	140	fail	
1-dispose-8-1	124,5	1268	fail	
1-dispose-8-2	699,11	1268	fail	
1-dispose-8-3	1296,02	1268	fail	
1-dispose-12-1	$> 2h$		fail	
look-n-grab-4-1-1	0,31	26	fail	
look-n-grab-4-2-1	1,49	26	fail	
look-n-grab-4-3-1	5,12	26	fail	
look-n-grab-8-1-1	45,27	212	fail	
look-n-grab-8-1-2	84,04	88	fail	
look-n-grab-8-2-1	$> 1.8GB$		fail	

Table 8.5: Challenging Problems over a Grid: plan times in seconds and lengths shown. 'fail' means that FF found $K(P)$ problem unsolvable. Figures for Conformant-FF not included, as it solves only 3 instances: Push-to-4-1/3.

solve two of them and perform worst in one of them. In summary, when KP is able to solve a domain, it may outperform T_0.

Discussion

In this section we proposed an alternative translation for mapping conformant problems P into classical problems $K(P)$ that can then be solved by a classical planner. The translation uses literals L/t that represent that *if t then* L, in contrast with the literals KL/t used by the translation $K_{T,M}(P)$, presented in Chapter 6, where t referred to the initial situation. The translation $K(P)$ is different from the translation $K_{T,M}(P)$, but also adds to the basic translation $K_0(P)$ a set of new literals, new actions rules, and new deductions actions, that allow to solve a wide range of problems even being an incomplete translation. This is apparent from the results of the conformant track of the IPC-2008 (Bonet and Givan, 2006), where the translation $K(P)$ was fed to the classical FF planner (Hoffmann and Nebel, 2001) producing a planner, KP, that was dominated only by T_0, the planner presented in Chapter 7.

The translation $K(P)$ ensures that KL is true when L is known to be true, but also uses the new literals L/X for literals L and X in the problem P, ensuring that L/X is true when the conditional 'if X then L' is known to be true. The tagged literals L/X are 'produced' by conditional effects of the form $a : C \wedge X \rightarrow L$, translated into $a : KC \rightarrow L/X$, and are 'consumed' by merge actions of the form $d : L/X_1, L/X_2 \rightarrow KL$ provided that $X_1 \vee X_2$ is known to hold, accounting thus for a simple form of reasoning by cases.

The key departures from $K(P)$ respect to $K_{T,M}(P)$ are in the syntax and semantics of the conditionals represented by the literals L/t: syntactically, t has to be a single literal, and semantically, the truth of L/t means that L is true if t is true at the same time, instead of KL/t meaning that L is true if t was true in the initial situation. Also, the translation $K_{T,M}(P)$ distinguish between the atoms KL/t and $K\neg L/t$ for guaranteeing soundness, but in this section we used instead the literals $FLAG$.

The simplicity of the translation and the semantics captured by the theorems of this section not only allow us to prove the soundness of the approach, but as importantly, to delimit its scope. In relation to natural deduction systems in the style of Ficht (Barwise and Etchemendy, 1991), the type of disjunctive reasoning accounted for in the translation is limited in two ways. First, while disjunctions $X_1 \vee \cdots \vee X_n$ in P are used to create sub-derivations by making assumptions of the form X_i, these sub-derivations are not nested, and therefore, disjunctions are not combined. This implies, for example, that four action rules like $a_{i,j} : X_i \wedge Y_j \rightarrow L$ for $i = 1, 2$ and $j = 1, 2$ cannot be used to produce a plan for L given the disjunctions $X_1 \vee X_2$ and $Y_1 \vee Y_2$ in the initial situation. Second, the sub-derivations that arise when making the assumptions X_i are very limited, and in particular the atoms L/X_i can only be used for proving L but no other literal. Thus, as a result, four action rules like $a_1 : X_1 \rightarrow L_1$, $a_2 : X_2 \rightarrow L_2$, $b_1 : L_1 \rightarrow L$, and $b_2 : L_2 \rightarrow L$ cannot be used to generate a plan that achieves L given the single disjunction $X_1 \vee X_2$. These are the two sources of incompleteness in the translation that is aimed at capturing conformant plans that can be verified easily, by reasoning with 'one disjunction' at a time. These two limitations are overcome by the translation $K_{T,M}(P)$ presented in Chapter 6. The example illustrating the first limitation will have conformant width 2 and be solvable by $K_2(P)$, an instance of the translation $K_{T,M}(P)$. For the second limitation, the rules $KC/t \rightarrow KL/t$ allows to 'carry' assumptions from the initial situation.

The translation $K(P)$ may produce bad performance in problems where the literals $FLAG_{X,L}$ are easily deleted, as the heuristic of the classical planner FF do not detected some dead-ends. Any translation-based approach based on classical planning, SAT or CSP, should be aware of the weakness of the underlying tools.

8.3 Non-Deterministic Actions for $K_{T,M}(P)$

The translation schemes considered in the classical-planning-based part (III) are all limited to problems with deterministic actions only. Nonetheless, as we illustrate in this section, those schemes can be applied to non-deterministic actions as well provided suitable transformations are included. We cover these transformations briefly as a matter of illustration only.

Consider a conformant problem P with *non-deterministic* action effects $a : C \rightarrow oneof(S_1, S_2, \ldots, S_m)$, where each S_i is a set (conjunction) of literals, and the transformed problem P', where these effects are mapped into *deterministic* rules of the form $a : C, h_i \rightarrow S_i$, with the expression $oneof(h_1, \ldots, h_m)$ added to the initial situation of P'. In P', the 'hidden' h_i variables are used for encoding the uncertainty on the possible outcomes S_i of the action a.

It is easy to show that the non-deterministic conformant problem P and the deterministic conformant problem P' are equivalent provided that only plans for P and P' are considered where the non-deterministic action a from P are executed at most once. Namely, a correspondence exists between the conformant plans for P that use such actions at most once with the conformant plans for P' that use the same actions at most once too. On the other hand, a conformant plan for P' where these actions are done many times will not necessarily represent a conformant plan for P. Indeed, if a non-deterministically moves an agent up or right in a square grid $n \times n$, starting in the bottom left corner, n actions a in a row would leave the agent at either the top left corner or the bottom right corner in P', and anywhere at Manhattan distance n from the origin in P. The divergence between P and P', however, does not arise if non-deterministic actions are executed at most once.

Building on this idea, a non-deterministic conformant planner can be obtained from a deterministic conformant planner in the following way. For the non-deterministic problem P, let P_1 be the problem P' above, with the additional constraint that the actions a in P_1 arising from the non-deterministic actions in P can be executed at most once. This is easily achieved by adding a precondition $enabled(a)$ to a that is true initially and that a sets to false. Let then P_2 represent the deterministic conformant problem where each non-deterministic action a in P is mapped into 2 deterministic actions, each executable only once, and each having its own 'hidden fluents' $h_1, \ldots, h_m$ with the $oneof(h_1, \ldots, h_m)$ expression in the initial situation. Similarly, let P_i be the deterministic problem that results from encoding each non-deterministic action in P with i deterministic 'copies'.

From this encoding, a simple iterative conformant planner for non-deterministic problems P can be defined in terms of a conformant planner for deterministic problems by invoking the latter upon P_1, P_2, P_3, and so on, until a solution is reported. The reported solution uses each copy of a 'non-deterministic action' at most once, and thus encodes a solution to the original problem.

We have implemented this strategy on top of T_0 with an additional refinement that takes advantage of the nature of the $K_{T,M}$ translation, where assumptions about the initial situation are maintained explicitly in tags.[10] Basically, 'non-deterministic' actions a in P_i are allowed to be executed more than once provided that all the literals KL/h_i that depend on a particular outcome of these actions (S_i) are erased. This is implemented by means of an additional $reset(a)$ action in P_i whose unconditional effect is $enabled(a)$ (i.e., the action a can then be done again) and whose conditional effects are $\neg KL \rightarrow \neg KL/h_i$ and $KL \rightarrow KL/h_i$ for $i = 1, \ldots, m$. Namely, literals KL/h_i where the truth of L depends on a particular non-deterministic outcome (S_i) are erased, except when L is true with no assumptions; i.e. when KL is true. Then non-deterministic actions a can be executed more than once in a plan provided that each occurrence of a, except for the first one, is preceded by a $reset(a)$ action.

Table 8.6 compares the resulting non-deterministic planner with MBP and KACMBP on

[10]The non-dermististic extension is integrated into T_0, part of the conformant planner 'Translator' available at `http://www.ldc.usb.ve/~hlp/software`.

	T_0		MBP		kacmbp	
Problem	time	len	time	len	time	len
sgripper-10	1,4	48	$> 2h$		0,6	68
sgripper-20	16,7	93	$> 2h$		5,4	148
sgripper-30	90	138	–		23,3	228
btuc-100	2,9	200	$> 2h$		2	200
btuc-150	9,2	300	$> 2h$		7,9	300
btuc-200	23	400	–		16,9	400
btuc-250	44,6	500	–		33,2	500
btuc-300	82	600	–		62,1	600
bmtuc-10-10	0,1	20	65,9	29	0,2	20
bmtuc-20-10	0,1	40	$> 2h$		0,6	40
bmtuc-20-20	0,3	40	$> 2h$		2,2	40
bmtuc-50-10	0,9	100	–		3,6	100
bmtuc-50-50	3,3	100	–		2722,4	100
bmtuc-100-10	4,9	200	–		25,1	200
bmtuc-100-50	14,9	200	–		$> 2h$	
bmtuc-100-100	30,2	200	–		$> 2h$	
nondet-ring-5	18,3	19	0	18	0,1	32
nondet-ring-10	$> 2h$		2,1	38	0,5	112
nondet-ring-15	$> 2h$		1298,9	58	2,4	242
nondet-ring-20	–		$> 2h$		7,3	422
nondet-ring-50	–		–		603,1	2552
nondet-ring-1key-5	$> 2h$		0,1	33	0,2	42
nondet-ring-1key-10	$> 2.1GB$		11,2	122	4	197
nondet-ring-1key-15	–		5164,4	87	33,7	375
nondet-ring-1key-20	–		$> 2.1GB$		246,5	1104
nondet-ring-1key-25	–		–		1417,5	2043
nondet-ring-1key-30	–		–		$> 2h$	

Table 8.6: Non-deterministic problems. All problems except sgripper are from MBP and KACMBP. These problems were modified to render a simple translation into PDDL; in particular, complex preconditions were moved in as conditions. Times reported in seconds and rounded to the closest decimal. '–' means time or memory out for smaller instances.

a number of non-deterministic problems considered in the MBP and KACMBP papers. We have just added an additional domain, Slippery Gripper (sgripper), that is similar to classical Gripper where a number of balls have to be moved from room A to B, except that the robot cannot move from A to B directly, but has a non-deterministic move action $move(A, C, D)$ that moves the robot from A to either C or D. A typical plan for moving two balls from A to B is to pick them at A, move to C or D, move from C to B, and from D to B, finally dropping the balls at B.[11]

For the deterministic conformant planner (T_0) used in the non-deterministic setting we added the following modification: merges are not introduced only for precondition and goal literals but for all literals. The reason is that in this setting it pays to remove the uncertainty of all literals when the reset mechanism is used. Indeed, provided with this simple change and the reset mechanism, in none of the problems we had to move beyond P_1 (a single copy of each non-deterministic action) even if in all the domains non-deterministic actions are required many times in the plans (e.g., if there are more than 2 balls in room A).

As it can be seen from the table, T_0 does better than MBP on these collection of

[11] A PDDL encoding of the Slippery Gripper problem can be found in appendix C.6 on page 180.

non-deterministic domains, although not as well as KACMBP, in particular, in the NonDet-Ring and Non-Det-Ring-1Key domains. In any case, the results obtained with T_0 on these domains are quite meaningful. In all cases where T_0 failed to solved a problem, the reason was that the classical planner (FF) got lost in the search for plans, something that may improve with further advances in classical planning technology.

8.4 Optimal conformant planning based on $K_{T,M}(P)$

In Chapters 3 and 4, we presented algorithms for optimal conformant planning, for both the serial and parallel setting, using a CNF-based translation. In this section we show that the translation into classical planning presented in Chapter 6 can be used for obtaining serial optimal conformant plans.

We start by considering classical planning with costs, extending the definition of Chapter 1 for having a cost function $c(a)$ that assign to any action a in the actions of the problem O a positive integer. The cost of a classical plan π is $c(\pi) = \sum_{a \in \pi} c(a)$. A plan π has *minimal cost* if there is no plan π' such that $c(\pi') < c(\pi)$.

For obtaining optimal conformant plans for P given a translation $K_{T,M}(P)$ we use the cost function

$$c(a) = \begin{cases} 1 & : \text{ if } a \text{ is a merge action} \\ \kappa & : \text{ otherwise} \end{cases}$$

where κ is a sufficiently large positive constant. We will assume that the integer κ is larger than $|F| \cdot |\pi^*|$ where $|\pi^*|$ is the *longest optimal conformant plan* of the problem P, and $|F|$ denotes the number of fluents of P. Remember that merge actions allow to obtain KL literals given a set of literals KL/t where $\bigvee t$ is a valid disjunction in the initial situation. By removing the merges of a minimal cost plan π for $K_{T,M}(P)$ we obtain a plan π' for P, and we will prove that π' is an optimal conformant plan for P. However, notice that there may be many optimal conformant plans. Consider two of them π' and π'_2, both with the same number of actions, but observe that the cost of the smaller corresponding classical plans π and π_2 may be different as they may require different number of merge actions. Thus, an optimal conformant plan may not produce a minimal cost classical plan, but an optimal classical plan does translate into an optimal conformant plan.

Theorem 8.3. *If π is a minimal cost classical plan of a covering translation $K_{T,M}(P)$ of the conformant problem P, then the plan π' that results from π by removing the merge actions is an optimal conformant plan for P.*

Proof. For a classical plan π we denote as π' the plan obtained by dropping the merge actions. We denote by $|\pi'|$ the number of actions of the conformant plan π' and observe that the cost of a classical plan π is $c(\pi) = \kappa \cdot |\pi'| + N$ where N is the number of merge actions in π. Observe that any optimal plan for $K_{T,M}(P)$ has at most $|F|$ merges after each action or before achieving the goal. Thus, $c(\pi)$ is bounded by $\kappa \cdot |\pi'| + |F| \cdot |\pi'|$.

Suppose that π is a minimal cost plan of $K_{T,M}(P)$ and π' is not an optimal conformant plan of P. Thus, there is another conformant plan π'_2 such that $|\pi'_2| < |\pi'|$. If π_2 is π'_2 with the necessary merges for being a classical plan of $K_{T,M}(P)$, we get a contradiction by proving that $c(\pi_2) < c(\pi)$ because π was supposed to be an minimal cost plan for $K_{T,M}(P)$.

We now prove that $c(\pi_2) < c(\pi)$, giving justification between { *braces* }.

$$\begin{array}{ll}
 & c(\pi_2) \\
\leq & \quad \left\{ \begin{array}{l} \pi_2 \textit{ has at most one merge for each precondition of each action, and a} \\ \textit{merge for each goal. Having more is not optimal.} \end{array} \right\} \\
 & \kappa \cdot |\pi'_2| \ + \ |F| \cdot |\pi'_2| \\
< & \quad \{ \ \kappa \textit{ is larger than } |F| \cdot |\pi| \textit{ for any optimal plan } \pi. \ \} \\
 & \kappa \cdot |\pi'_2| \ + \ \kappa \\
= & \quad \{ \ \textit{distributivity} \ \} \\
 & \kappa \cdot (|\pi'_2| \ + \ 1) \\
\leq & \quad \{ \ |\pi'_2| \textit{ is strictly smaller than } |\pi'| \textit{ and both are positive integers } \} \\
 & \kappa \cdot |\pi'| \\
< & \quad \{ \ \textit{previous observation} \ \} \\
 & c(\pi)
\end{array}$$

□

The evaluation of the effectiveness of this approach for optimal conformant planning is left as future work.

Chapter 9

Related Work

Hablan poco los árboles, se sabe.
Pasan la vida entera meditando
y moviendo sus ramas.
Basta mirarlos en otoño
cuando se juntan en los parques:
sólo conversan los más viejos,
. . . muy poco nos llega, casi nada.

The trees speak so little, you know.
They spend their entire life meditating
and moving their branches.
Just look at them closely in autumn
as they seek each other out in public places:
only the oldest attempt some conversation,
. . . so little filters down to us, nothing really.

The Trees. Poem by Eugenio Montejo[1]

In this chapter we present related work. We first comment on the use of *Quantified Boolean Formulas* (QBF) for solving conformant planning problems and its connection with our methods that use CNF encodings. Then, we comment on approaches for obtaining conformant plans by incrementally building plans in one initial state and reject those that do not work for all of them. In the third section, we look at belief-space-based conformant planners and their relation with the approaches presented in this dissertation, where the belief states are not represented explicitly. We review two approaches that share some similarities with our translations to classical planning: one is a knowledge-based algorithm, commented in Section 9.4, the other is based on the 0-approximation semantics, commented in Section 9.5 (Baral and Son, 1997). Also in Section 9.5, we comment about the relation of our classical-based planner with `CpA`, winner of the conformant track of the 2008 *International Planning Competition* (IPC-2008), and is based on an extension of the 0-approximation semantics. In Section 9.6, we discuss existing relations between our work and some

[1]English translation from "The trees: selected poems, 1967-2004". Op. Cit.

approaches used for conformant probabilistic planning. Finally, Section 9.7 relates our notion of conformant width with other width definitions in AI.

9.1 QBF-based Conformant Planning and QBF solving

In this section, we relate our CNF-based approach to conformant planning with the problem of satisfying a *Quantified Boolean Formula* (QBF), a quantified version of propositional formulas. We introduce QBF, current approaches for QBF satisfiability, and relate them to our work. We review a well known mapping from conformant planning into QBF and describe a possible use of our methods for solving a specific type of QBF.

Quantified Boolean Formula

A *Quantified Boolean Formula* (QBF) is a quantified version of a propositional formula. For a propositional formula ϕ, a quantified boolean formula with kernel ϕ is

$$Q_0 v_0 Q_1 v_1 \ldots Q_n v_n \phi \tag{9.1}$$

where each v_i is a list of distinct variables, and each Q_i is a quantifier $\forall$ or $\exists$, such that no adjacent quantifiers Q_j and Q_{j+1} are the same. The QBF in Equation 9.1 is said to have *n alternations.* We assume that all the variables in ϕ are quantified, *i.e.* all of them appear in some v_i.

For a set of variables v, a truth-assignment of them is denote by $\hat{v}$, and recall that we denote $\phi \,|\, \hat{v}$ as the conditioning of ϕ on $\hat{v}$, *i.e.* replacing in ϕ all variables v_i by *true* (resp. $false$) if v_i (resp. $\neg v_i$) appears in $\hat{v}$.

QBF satisfiability is defined recursively as follows. A QBF formula $\forall v_i \phi$ is satisfiable when for all the possible truth-value assignment $\hat{v}_i$ for the variables v_i, the QBF $\phi \,|\, \hat{v}_i$ is satisfiable. A QBF formula $\exists v_i \phi$ is satisfiable when there exists a truth-value assignment $\hat{v}_i$ of the variables v_i such that the QBF $\phi \,|\, \hat{v}_i$ is satisfiable. Note that after evaluation of the inner-most quantifier, the formula is *true* or $false$, and no variable remains.

There are two main approaches for QBF satisfiability. One uses variant of DPLL (Davis et al., 1962), a procedure for SAT that recursively selects a variable and tries (splits) both possible values, *true* and $false$, until a solution is obtained or an inconsistency is detected through *unit resolution.* An inconsistency leads to backtracking and to a search of the alternative value. However, QBF satisfiability requires *all* the branches (search paths) for a variable quantified by a $\forall$ quantifier to be satisfiable, and one satisfying branch for a variable quantified by a $\exists$ quantifier. In addition, the ordering in which variables are considered must be compatible with the order in which they appear in the QBF quantifier. Both constraints are necessary for such algorithm to be sound. However, since the performance of current implementations of DPLL strongly depend on the freedom to select the next variable to split, the two previous constraints may lead to a dramatically decreased performance.

A second approach to QBF satisfiability is to use a variant of variable elimination, called *eliminate and expand* (Biere, 2005; Dechter, 2003). It is possible to eliminate a quantifier of a QBF using *resolution* between pairs of clauses, an expensive process that motivates the use of the simpler *unit resolution* in DPLL. Applying variable elimination typically increases the size of the formula and may harm the scalability of this approach.

QBF-based Conformant Planning

Rintanen (1999) solves the problem of finding a N-step conformant plan in terms of a QBF formula of the form $\exists actions \, \forall s_0 \, \exists f_i \; \phi$, where f_i stands for the propositional variables that corresponds to the fluents of the problem. The solution for this formula is a truth assignment to *actions* variables such that, for all variables in the initial state s_0, the formula ϕ is satisfiable. For deterministic actions, the case considered in this dissertation, once actions and initial variables have been set to a truth-value, there is at most one model describing the execution of such a plan for each initial state.

We are interested in evaluating the performance of current solvers over our QBF formulas constructed using the CNF encoding for conformant planning presented in Chapters 3 and 4. We add the prefix $\exists actions \, \forall s_0 \, \exists f_i$ to our CNF and the resulting QBF formula can be feed into a QBF solver for deciding whether there exists a N-step conformant plan or not. This approach represents an alternative to the compilation of the propositional theory into d-DNNF.

Encoding conformant problems into QBF requires a special treatment concerning the variables modeling the initial states. Otherwise, there may be assignments to initial state variables that do not correspond to possible initial states, leading to an easily falsified QBF formula (Ansótegui et al., 2005). Following Rintanen (1999), we introduce dummy vars for the $\forall s_0$ part of the QBF. Some of the 2^m combinations of dummy literals are associated to possible initial states, but there might be combinations that are not.

$$
\begin{aligned}
\exists a_1, \ldots, a_n \; \forall x_1, \ldots, x_m \; \exists f_1, \ldots, f_o(\;\; & T_N(P) \wedge \\
& (x_1 \wedge x_2 \wedge \ldots x_m \supset s_1) \wedge \\
& (\neg x_1 \wedge x_2 \wedge \ldots x_m \supset s_2) \wedge \\
& (x_1 \wedge \neg x_2 \wedge \ldots x_m \supset s_3) \;\wedge\; \ldots)
\end{aligned}
\tag{9.2}
$$

where $T_N(P)$ is the propositional theory of the conformant problem P for horizon N, $a_1, \ldots, a_n$ are the action variables, $x_1, \ldots, x_m$ are the dummy vars, $s_1, \ldots$ are conjunctions of literals corresponding to the possible initial states, and $f_1, \ldots, f_o$ are the rest of the variables in the theory $T_N(P)$.

The QBF of Equation 9.2 is satisfiable iff there is a truth value assignment for action variables such that for all possible initial states, such assignment encodes a plan. For a given truth assignment a upon action literals, any combination of dummy literals l_i associated to an initial state through a formula of the form $l_1 \wedge l_2 \wedge \ldots \wedge l_m \supset s$, encodes that if such set of dummy literals is satisfied, the set of literals s will be satisfied too. Thus, as the truth-assignment also satisfies $T_N(P)$, the only possible model is the execution of the sequence of actions encoded in a starting at the initial state encoded in s. For any other combination of dummy variables not associated to an initial state, the kernel of the QBF will be satisfied if there is at least a plan for some possible initial state. Thus, the QBF of Equation (9.2) ensures that for a given sequence of actions, the theory will be satisfiable iff the sequence conforms with all the possible initial states.

As a useful evaluation and as a contribution to the QBF community, we generated QBF instances using the Equation (9.2), built over the CNF formulas generated for the model-counting and SAT-based algorithms presented in Chapters 3 and 4, respectively.

We submitted some of those instances to the the QBF-solver evaluation (Palacios, 2007,2008). The best performer participant on our instances in the 2008 *QBFEVAL* was `quantor3.0`, based on the *eliminate and expand* method (Biere, 2008).[2] The *variable elimination* performed by `quantor3.0` might be related to the *projection* used by our SAT-based algorithm. [3]

In Chapter 4 we proposed an algorithm for verifying whether there is a N-time-step conformant plan using a single call to a SAT solver. The algorithm converts the theory $T_N(P)$ into d-DNNF and performs logical operations to obtain a new CNF whose models are in correspondence to the conformant plans of N time steps. In principle, the satisfiability of QBFs with prefix $\exists\forall\exists$ can be assessed by using the same transformation to a new CNF and using a SAT solver on it. This is not direct, as the performance of our d-DNNF-based algorithms depends strongly on the strategy used for compilation of the CNF theories, and on taking profit of the fact that the inner variables f_i of the QBF in Equation (9.2) are determined by the rest of the variables, which is not true for general QBFs with prefix $\exists\forall\exists$.

Conformant planning without search

In principle, it is also possible to look for N-time-step plans using another strategy while compiling the theory $T_N(P)$ into d-DNNF. If we instruct the compiler to split first over the actions variables, as we did in Section 3.5, it is possible to eliminate easily the remaining ones, fluents and auxiliary literals. The resulting formula, $\mathsf{project}[\ \Delta\ ;\ \mathit{Actions}\ ]$, encodes all the possible conformant plans and, as such formula remains in d-DNNF, a model can be obtained in linear time. We are, then, deciding whether there is a N-step conformant plan with a SAT call, but SAT is tractable for d-DNNF. This approach, however, does not scale up, as the compiling process with actions on top of the d-DNNF tends to be very difficult. This maybe related with two issues. First, the order we used for the algorithm presented in this chapter coincides with the layer structure of the propositional planning theories (Huang, 2006). Fixing fluents to be true, preconditions and conditions get narrowed, and once actions literals are set, their consequences get determined, making the compilation to d-DNNF be very efficient. In fact, as far as we know, the CNF formulas in the first part of this dissertation are the largest that has been transformed into d-DNNF. Second, by compiling with all the actions variables on top, the resulting formula not only allows us to obtain a plan in linear time, but also allows us to enumerate all possible conformant plans of N time steps. An algorithm having such a huge side product is likely to be an overkill.

[2]The 'Translator' planner has an option for exporting QBF of conformant problems for a given horizon. The software is available at `http://www.ldc.usb.ve/~hlp/software`.

[3]See for a definition of projection and how to perform it on d-DNNF formulas, starting at Section 3.4 on page 33.

Indeed, the satisfiability of QBF has been shown to be polynomial for QBF with a OBDD formula as kernel, provided the OBDD respects the ordering of the variables in the QBF prefix (Coste-Marquis et al., 2005, Proposition 4). As it was mentioned in Section 3.5, we instruct the compiler to do strict case reasoning over the initial-state atoms. If we did the same for all the actions variables following an strict order when compiling and forgetting the rest of the variables, the resulting formula will be in OBDD form, consistent with the fact that all d-DNNF can be converted into OBDD in polynomial time, but not the other way around (Huang and Darwiche, 2004, 2005; Darwiche and Marquis, 2002). Thus, the proposed algorithm for extracting conformant problems without searching suggests that the tractability result for QBF extends to some other d-DNNF formulas.

We also look for an algorithm that trades off by compiling with some actions on top, and search on the remaining actions for a conformant plan. With all the algorithms we tried, compiling with as many actions on top as possible leads to a huge d-DNNF, making it very expensive to run the linear time operations and queries. A conclusion is that even though d-DNNF operations can be quite cheap, it stops paying off as the size of the formula increases. This means that finding good strategies for compiling formulas, by getting good *decomposition trees (dtree)*, is critical for effective applications.

9.2 Using plans for a single initial state

Some conformant planning algorithms find a plan for a specific initial state and then test if such plan conforms with the rest of the initial states. We consider now the relation of such approaches with the algorithms of this dissertation.

C-*Plan* finds conformant plans with N time-steps. It does this using a CNF encoding similar to the one used in the CNF-based part of this dissertation (Ferraris and Giunchiglia, 2000). Given an initial state, C-*Plan* uses a DPLL algorithm to look for a plan. That is, it recursively attempts assign propositional variables to values until a total consistent assignment is found. As such assignment encodes a plan that conforms with an initial state, C-*Plan* uses a SAT solver call to verify whether such plan conforms also with the rest of the possible initial states. If the plan conforms, a solution is returned. Otherwise, the DPLL search continues.

The C-*Plan* algorithm can be related to both the branch-and-prune procedure in the model-counting algorithm of Chapter 3, and to the SAT-based algorithm of Chapter 4. However, in both cases, partial assignments of the action literals are discarded before they are completed. The *validity* criterion of the model-counting-based algorithm discards partial plans that cannot be extended to conformant plans. Modern SAT solvers, as the used in our SAT-based scheme, are able to detect inconsistencies before assigning values to all the variables of a propositional formula.

The planner *DLVk* also generates a candidate plan and checks whether it is conformant, in a similar way to C-*Plan*, but uses answer set programming instead of propositional satisfiability (Eiter et al., 2003; Lifschitz, 2008). The algorithm *Fragplan* generates a plan for an initial state, and then generates new plans by forcing some actions to be applied that were found to be part of the solution for other initial states. By accumulating these fragments, *Fragplan* generates a plan conforming with all the possible initial states (Kurien et al., 2002). In contrast, CGP generates a

Graphplan for each possible initial state and attempts to extract a conformant plan by considering all such planning graphs at the same time (Smith and Weld, 1998). *Fragplan* and CGP work better when the set of possible initial states is small.

Let us consider the role of the plans for a possible initial state in our algorithms. Our SAT-based algorithm presented in Chapter 4 starts by producing a CNF theory $T_N(P)$ that encodes all the possible executions of plans for each specific initial state and horizon N. The algorithm then creates a CNF theory $T_{\text{cf}}(P)$ that is fed into a SAT solver and obtains a conformant plan for P if there is any. The theory $T_{\text{cf}}(P)$ is created by logical operations that are made tractable by compiling the theory $T_N(P)$ into d-DNNF. However, another way of getting such theory is indeed possible. If the number m of possible initial states s_0 remains bounded and actions are deterministic, the problem of obtaining and N-step conformant plan of P can be mapped into the SAT problem over a formula very similar to $T_{\text{cf}}(P) = \bigwedge_{s_0 \in Init} \mathsf{project}[\, T(P) \mid s_0 \; ; \; Actions \,]$ (Equation 4.5 on page 54 Palacios and Geffner, 2006b):

$$T_{\text{cf}}{}'(P) = \bigwedge_{s_0 \in Init} T_N^{s0}(P), \tag{9.3}$$

where $T_N(P)$ is the propositional theory that encodes the problem P with horizon N, and $T_N^{s0}(P)$ is $T(P)$ with two modifications: first, fluent literals L_0 (L at time 0) are replaced by $true/false$ iff L is $true/false$ in the (complete) state s_0, and second, fluent literals L_i, $i > 0$, are replaced by 'fresh' literals L_i^{s0}, one for each $s_0 \in Init$. A model of $T_{\text{cf}}{}'(P)$ encodes a truth-assignment of the action variables, and a truth-assignment of the execution of such actions upon each possible initial states, each of them achieving the goal. Equation (9.3) can be thought as expressing m "classical planning problems", one for each possible initial state $s_0 \in Init$, that are *coupled* in the sense that they all share the same set of actions; namely, the action variables are the only variables shared across the different sub-theories $T_N^{s0}(P)$ for $s_0 \in Init$. As we comment in Section 4.5), page 58; this approach did not scale up in the limited test we performed. Nevertheless, the formulation is still interesting and has an interesting relation with the rest of the work done in this dissertation.

For bounded m, the resulting class of conformant planning problems with a fixed horizon can be mapped polynomially into SAT using the encoding $T_{\text{cf}}{}'(P)$, generalizing the SAT encoding of classical planning problems which corresponds to $m = 1$. Also, for a sufficiently large horizon, this formulation is *complete.*

If the translation from conformant into classical $K_{T,M}(P)$ of Chapter 6 is used upon a classical plan P, the result is the same classical problem.[4] $K_{S0}(P)$, a complete instance of the general translation $K_{T,M}(P)$, is related with Equation (9.3). For each literal L in the conformant problem P and each possible initial states s_0, there are two literals KL/s_0 and $K\neg L/s_0$ in $K_{S0}(P)$. When an action is applied in $K_{S0}(P)$, all the literals KL/s_0 are updated accordingly, and for any initial state s_0', the set of literals KL/s_0' represents the current state assuming that the initial state was s_0'. Indeed, it can be proved than, given a sequence of actions $\pi = \{a_0, \ldots, a_n\}$, the action variable L_t^{s0} is true for the formula $T_{\text{cf}}{}'(P) \cup T_A(\pi)$ iff KL/s_0 is true after applying the first t actions of π in $K_{S0}(P)$, where $T_A(\pi)$ is the conjunction of literals representing the plan π.

[4]Actually, for problems P with no clauses in its initial situation I, after some simplifications done by the planner T_0, the classical $K_{T,M}(P)$ is equivalent to compile away negations of conditions, by replacing rules of the form $a : L_1, \neg L_2 \to L$ by $a : KL_1, K\neg L_2 \to KL \wedge \neg K\neg L$, equivalent to use two atoms KL and $K\neg L$ for representing the literals L and $\neg L$ (Nebel, 2000).

However, it is possible to have more compact instances that are also complete, as we discussed in Section 6.4, leading to exponential savings with respect to the instances $K_{S0}(P)$ and the Equation 9.3. Smith and Weld (1998) point to the need of such compact representation when they comment about the limitations of the CGP algorithm: *"Although the possible worlds mechanism is conceptually clear, it is also cumbersome. As the amount of uncertainty grows, the number of possible worlds grows exponentially and performance deteriorates. To fix this, we would like to confine the representation of uncertainty to only those propositions that we are uncertain about."*

Indeed, the planner T_0, presented on Chapter 7 and based on the $K_{T,M}(P)$ translation of Chapter 6, allows us to obtain polynomial translations for problems with bounded width even if the number of possible initial states is exponential.

9.3 Belief State Conformant Planners

The most common approach to conformant planning is to search in *belief-space*, where each belief-state is a set of possible plain states. The search starts at the initial belief state consisting of the possible initial states. Applying an action on a belief state leads to the set of possible consequences of each possible state in the current belief state. As we discussed in Section 2.7, the two main issues for this approach are the representation and update of the belief states, and obtaining appropriate heuristics for guiding the search.

Most recent conformant planners such as Conformant-FF, POND, MBP, and CpA cast conformant planning as an heuristic search problem in belief space (Bonet and Geffner, 2000). In contrast, all algorithms presented in this dissertation do not search explicitly on belief space. Indeed, our CNF algorithms based on CNF search on the space of the possible plans, not representing the state explicitly, similarly to the SAT-based approach to classical planning (Kautz and Selman, 1996). In contrast, the translations to classical planning have a clear relation to belief-state-based planners.

The general translation $K_{T,M}(P)$ presented in Chapter 6 represents the current belief state by a plain classical state with literals KL/t representing conditionals *if t is true in the initial situation, L must be true.* Such translations can be far more compact by restricting the context t to the uncertainty relevant to different goals or preconditions. The instances $K_{T,M}(P)$ can be complete even if they do not track explicitly all interactions in belief space. The conformant relevance, as presented in Section 6.4, works as criterion for independency, showing that some interactions can be ignored. Such independency allows to keep track of actions effect using a cheap representation of the possible current states. Therefore, the representation of the problem depends not only on the initial states and the effect of actions, but also in the precondition of actions and goals.

For example, in problems with conformant width one are easily solvable by the planner T_0 using the instance $K_1(P)$. Belief-state planners, on the other hand, may face symmetries that need to be overcome in order for them to be effective. Compact representation of belief states using OBDDs probably help with this issue (Cimatti and Roveri, 2000), but are not aware of the semantic of the planning problem that would lead to a even cheaper representation. The planner CpA uses relevance analysis for simplifying the initial situation and decomposing the goal of conformant problems.

	CFF			FF in T_0		
Problem	Nodes	Time	Nodes/sec	Nodes	Time	Nodes/sec
bomb-100-1	5149	32,9	156,5	5250	0,4	12804,9
bomb-100-100	100	0,8	125	201	7,5	26,7
Safe-100	100	1747,4	0,1	102	<0,1	25500
logistics-4-10-10	356	4,4	80,5	774	0,5	1646,8
square-center-8	4634	59,3	78,1	**46**	<0,1	920
square-center-12	39000	>5602,5	7	**72**	<0,1	2400
cube-center-5	2211	8,2	269,6	**74**	<0,1	7400
cube-center-7	81600	>5602,5	14,6	**105**	<0,1	5250
blocks-01	46	<0,1	4600	47	<0,1	11750
blocks-02	1420	>5602,5	0,3	**86**	<0,1	4300
coins-20	1235	20,6	60	783	0,1	19575
comm-25	517	56,1	9,2	1777	0,4	4132,6
uts-k-10	58	16,5	3,5	62	0,3	182,4
dispose-8-1	**1107**	339,1	3,3	11713	0,8	15016,7
dispose-8-2	**1797**	2592,1	0,7	87030	14,3	6077,5
dispose-8-3	2494	>5602,5	0,4	580896	190,2	3054,1
look-and-grab-4-1-1	4955	>5602,5	0,9	**79**	0,1	790

Table 9.1: Conformant-FF over Conformant Problems vs. FF over Translations: Nodes stand for number of nodes evaluated, Time is expressed in seconds, and Nodes/sec stands for average number of nodes per second. Numbers shown in bold when either Conformant-FF or FF evaluate significantly less nodes (an order-of-magnitude reduction or more). Times preceded by '>' are time outs.

Tran et al. (2009) showed that this technique has some impact on the performance of Conformant-FF and POND.

In general, the heuristics used in belief-state conformant planning are not as effective as the ones in classical planning. Belief-state planning may fail to generalize over a wide range of problems. We showed in Chapter 7 that the planner T_0 was quite effective in comparison with most of such planners. It may be possible to extract useful heuristics from a relaxation of $K_{T,M}(P)$, even though it was not suitable for obtaining a solution using a classical planner on the original $K_{T,M}(P)$. This could be either because of the size of $K_{T,M}(P)$ or because of the behavior of the classical planning heuristics. Additionally, it is possible that problems P with high conformant width can be relaxed to problems P' with lower conformant width, obtaining a useful heuristics for solving P.

As the planner T_0 uses FF as underlying classical planning, it is important to consider its relative performance with respect to Conformant-FF. We report experiments in Table 9.1, comparing the search that results from the use of the FF planner over the classical translations in T_0, to the search carried out by Conformant-FF over the original conformant problems. The table illustrates the problems faced by belief-space planners mentioned in Section 2.8 on page 21. It also illustrates how the translation to classical planning handles them: representation of the states, and heuristics for guiding the search. The belief representation and update problem appears in the *overhead* of maintaining and evaluating the beliefs. This is shown in the number of nodes that are evaluated per second: while CFF evaluates a few hundred nodes per second; FF evaluates several thousands. At the same time, the *heuristic* used in CFF in the conformant setting, appears to be less informed than the heuristic used by FF over the classical translations. In domains like Square-Center-n, Cube-Center-n, Blocks, and Look-and-Grab, FF needs orders-of-magnitude less nodes than CFF to find a plan. The opposite is true in Dispose-n-m, where FF evaluates many more nodes than CFF. Nonetheless, even then, due to the overhead involved in carrying the beliefs, FF manages to solve problems that CFF cannot solve. For example, the instance Dispose-8-3 is solved by T_0 after evaluating more than half a million nodes, but times out in CFF after evaluating less than three thousand nodes.

Hoffmann and Brafman (2006) emphasized that Conformant-FF handles better conformant problems that are closer to classical planning: *"our approach demonstrates the potential to combine the strengths of FF with conformant abilities in domains that combine classical and conformant aspects."* In Table 9.1 the problem logistics-4-10-10 is taken from Conformant-FF distribution an reported as an enriched classical benchmarks (Hoffmann and Brafman, 2006). Both T_0 and Conformant-FF solved it in a few hundred nodes, but T_0 generated them 21 times faster than Conformant-FF.

9.4 Knowledge-based planners

A sound but incomplete approach to planning with incomplete information is introduced by Petrick and Bacchus (2002), where belief states are represented as formulas. In contrast with approaches that rely on complete representations, which can make the search unfeasible, the authors propose to model explicitly the knowledge of the agent, using modal logic, but restricting the language in order to ensure that the required inference can be accomplished. The language allows to represent, for example, when a conjunction of literals is known to be true or known to be false after applying an action, but not in the case of a disjunction of literals. Also they support domain depending knowledge to be added.

To summarize, in order to make belief updates efficient, several approximations are introduced. In particular, while existing disjunctions can be carried from one belief to the next, no new disjunctions are added. This imposes a limitation on the types of problems that can be handled. The two other limitations of this approach are that domains must be crafted by hand, and that no control information is derived from the domains so that the search for plans is blind, using iterative deepening. Our translations to classical planning in part III provide a solution to these two problems. First, the problem is moved to the 'knowledge-level' automatically. Second, once moved, the problem is solved by classical planners, which are able to search with control information derived automatically from the new representation.

9.5 0-approximation Semantics

The 0-approximation semantics, introduced by Baral and Son (1997), represents belief states b not by sets of states but by a single 3-valued state where fluents can be true, false, or unknown.[5] In this representation, checking whether an action a

[5] Actually, belief states b are represented using two sets: the set of literals that are true in b, and the set of literals that are false in b. Variables which do not appear in either set are unknown. This representation is equivalent to a 3-value representation, that we found more convenient.

is applicable in a belief state becomes tractable. Son and Tu (2006) introduced a complete algorithm based on 0-approximation. The idea is to create a *set of partial states*, such that a plan that conforms with all of them, would be conformant with the original problem. The size of the set of partial states could be exponentially smaller than the corresponding belief state. Conformant planners based on the 0-approximation semantics use the complete extension for searching on belief space, and need a way to guide the search for plans in the simplified belief space.

The 0-approximation semantics is very related to our translations to classical planning in chapters 5 and 6. In Proposition 5.3 on page 66, a correspondence was established between the plans for P that are conformant according to the 0-approximation semantics and the classical plans for the translation $K_0(P)$. Furthermore, the latter is an instance of the more general translation $K_i(P)$ that is complete for problems with width $i = 0$ (see Section 6.4). Thus, the semantics of the translation K_0 are related to the 0-approximation semantics. And yet, the K_0 translation delivers something more: a computational method for obtaining conformant plans that comply with the 0-approximation semantics using a classical planner.

The 0-approximation and the basic K_0 translation are too weak for dealing with the existing benchmarks. The translations K_i extend K_0 for problems of higher width by enriching the set of fluents KL by fluents KL/t where the tags t encode assumptions about the initial situation. The extensions of the 0-approximation semantics in the context of conformant planning have taken a different form: switching from a single 3-valued state for representing beliefs to *sets* of 3-valued states, each 3-valued state progressed efficiently and independently of the others (Son et al., 2005b). The initial set of 3-valued states is obtained by forcing states to assign a boolean truth-value (true or false) to a number of fluents. Crucial for this approach to work is the number of such fluents; belief representation and update are exponential in it. The conditions that ensure the completeness of this extension of the 0-approximation semantics can be expressed in terms of a relevance analysis similar to the one underlying our analysis of width in Section 6.4 on page 77 (Son and Tu, 2006): the fluents that must be set to true or false in each initial 3-valued state are those appearing in a clause in $C_I(L)$ for a precondition or goal literal L. In particular, if in the initial situation there are n tautologies $p_i \vee \neg p_i$, each relevant to a precondition or goal literal L, then the number of initial 3-valued states required for completeness is exponential in n, as each has to make each fluent p_i true or false. The difference with our approach can be seen when each of the tautologies $p_i \vee \neg p_i$ is relevant to a *unique* precondition or goal literal L_i. In such a case, the number of 3-valued or 'partial' states required for completeness remains exponential in n, while the resulting problem has width 1 and thus can be solved with the K_1 translation that involves tags with a single literal. In other words, while the tags used in our translation scheme encode the *local contexts* required by the different literals in the problem, the initial 3-valued states (Son and Tu, 2006) encode their possible combinations in the form of *global contexts*. These global contexts correspond to the consistent combinations of such local contexts, which may thus be exponential in number even if the problem has bounded width.

Another difference with the 3-valued approach (Son et al., 2005b; Son and Tu, 2006) is that the translation to classical planning not only addresses the representation of beliefs but also the computation of conformant plans. Once a conformant problem P is translated into a problem $K_{T,M}(P)$, it can be solved by a classical planner.

This is in contrast with the 0-approximation semantics that needs an explicit search algorithm and convenient heuristics.

The planner CpA is based on the extensions of the 0-approximation semantics to be complete (Son and Tu, 2006), and participated with two versions in the conformant track of the IPC-2008, as we commented about then in Section 7.4 on page 98. Consistent with this extension, CpA is a belief-space planner that represent beliefs as DNF formulas, and use simple belief-state heuristics for guiding the search (Tran et al., 2009). Its main weakness is the potential blow up coming from the number of terms in the DNF formula encoding the initial belief state.

In order to reduce further the number of terms in this initial DNF formula, 'independent' one-of expressions are combined by CpA. For example, two independent one-of clauses $oneof(x_1, x_2)$ and $oneof(y_1, y_2)$, which would give rise to 4 possible initial states and DNF terms, are combined into the single one-of expression $oneof(x_1 \wedge y_1, x_2 \wedge y_2)$, that results into 2 possible initial states and terms. These one-of expressions are independent when they can be shown not to interact in the problem. The technique appears to be related to the notion of 'critical initial states' considered in Section 6.5, where it was shown that plans that conform with all critical initial states must conform also with all possible initial states. However, such one-of combination is still weak in comparison with our approach. For a given n, consider a conformant problem as follows

Init $oneof(a_1, \ldots, a_n), oneof(b_1, \ldots, b_n)$

Goal a_0, b_0

Actions with no preconditions, but with conditional effects

- $do_{a1} : a_1 \rightarrow a_0$
- $\cdots$
- $do_{an} : a_n \rightarrow a_0$
- $confuse : a_1 \wedge b_1 \rightarrow a_0 \wedge b_0$
- $do_{b1} : b_1 \rightarrow b_0$
- $\cdots$
- $do_{bn} : b_n \rightarrow b_0$

The problem has $2(n+1)$ atoms and $2n+1$ actions. For any n, the one-of combinations reported by Tran et al. (2009) cannot transform the initial situation into $oneof(a_1 \wedge b_1, \ldots, a_n \wedge b_n)$ because of the action *confuse*. The conformant width of this problem is, however, one and the instance $K_1(P)$ is able to solve it. Indeed, CpA reports $n \times n$ partial states for this problem, while T_0 creates a linear number of atoms[6], that after simplifications drops to $2(n+1)$, making the performance of CpA degrade much faster than T_0's, as n grows. The same happens when the *oneof* in the initial situation are replaced by clauses $a_1 \vee \ldots \vee a_n, b_1 \vee \ldots \vee b_n$.

9.6 Probabilistic Planning

There are two tasks related with our work that are extensions of models used in planning that use probabilities. One is probabilistic conformant planning, which

[6]For each literal l_i of an atom a_i or b_i, for $0 < i \leq n$, the following atoms are generated: Kl_i, $K\neg l_i$, Kl_i/l_i, Kl_0/l_i and $K\neg l_0/l_i$, adding so far $2 \cdot 5 \cdot n$ atoms. There are also the atoms Ka_0, $K\neg a_0$, Kb_0 and $K\neg b_0$. To summarize, T_0 generates before simplifications $10n + 4$ atoms for this problem.

is an extension of classical planning for the case where the initial situation is a probabilistic distribution over possible initial states, and the effects of actions are also probabilistic (Kushmerick et al., 1995; Majercik and Littman, 1998; Hyafil and Bacchus, 2003; Huang, 2006). A probabilistic conformant planning task is to find a sequence of N actions with maximal probability of achieving the goal. Another related case is probabilistic planning, where there is also a probabilistic distribution of the possible initial states, the actions also have probabilistic effects, but the resulting state after applying an action is totally observable (Hansen and Zilberstein, 2001; Bonet and Geffner, 2001b). In this case the solution is represented as a policy that maps states into actions.

In Section 8.1 we introduced an extension of our model-counting-based algorithm for conformant planning to the probabilistic case, and relate it with a similar algorithm by Huang (2006). Other algorithms for probabilistic conformant planning uses propositional logic or constraints (Majercik and Littman, 1998; Hyafil and Bacchus, 2003). Some others are based on partial-order planning (POP) (Onder et al., 2006). Some probabilistic planners were also able to obtain plans also for the case without observations, as Buridan (Kushmerick et al., 1995).

A recently proposed strategy for probabilistic planning is called *replanning* (Yoon et al., 2007; Little and Thiébaux, 2007). Given a simplification of the problem for eliminating uncertainty, a classical planner is called upon it. The resulting plan is applied as far as the real effect coincides with what is expected by the classical plan. In case of failure, a new plan is obtained from a classical problem with the initial situation reflecting the current one. This idea is related to our translation from conformant into classical, but our approach obtains a solution with only one call to a classical planner, and the guarantee of completeness.

9.7 Width and Tractability

Many AI models, as CSP or SAT, can be depicted as graphs, whose nodes are the variables and whose edges represent there appearance together in a constraint or a clause. The complexity of solving those models can be bounded by properties of such graphs. For example, the satisfiability of a CNF formula can be decided by a systematic scanning on its variables, and applying resolution to clauses mentioning each variable. This algorithm, DP (Davis and Putnam, 1960), has the drawback of potentially generating an exponential number of new clauses, but the benefit that for some problems it may outperform the most used algorithm for SAT solving: DPLL (Davis et al., 1962). Dechter and Rish (1994) proved that DP's runtime and space complexity depends on a property called the *induced-width*, that bound the size of the largest intermediate result. Such width can be calculated from a graph representation of the CNF formula, and depends on both the theory and the order in which the variables are considered. Even being exponential in the worst case, DP may solve instances of some classes of problems in polynomial time, for example for CNF formulas consisting only of *horn* clauses.

The notion of width is important in the CNF-based part of this dissertation. A key step of the proposed algorithms is to compile a propositional formula encoding N-time-step plans into d-DNNF. Such compilation is exponential on the *treewidth* of the CNF formula given a *decomposition tree* (dtree) that guides the compilation

(Darwiche, 2001b). Using an argument from Huang (2006), the compilation of our propositional theories scales up because we use an ordering that induces a dtree that leads to a low *treewidth*.

The conformant width measure defined for bounding the complexity of the translations from conformant into classical planning (Section 6.4 on page 77)is different to the treewidth notion because bounded conformant width does not translate into a polynomial time algorithm for solving the problem. Instead, having bounded conformant width means that the problem can be mapped into a polynomially larger classical problem. But classical planning is still intractable, although of a lower complexity than general conformant planning (Turner, 2002).

Chapter 10

Conclusions

Those who plant in tears
will harvest with shouts of joy.
They weep as they go
to plant their seed,
but they sing
as they return with the harvest.

Psalm 125 (126). The Bible[1]

In this chapter we summarize the work presented in the dissertation and enumerate the contributions, indicating the chapters and publications where they appeared. At the end of the chapter we comment on current and future work.

10.1 Introduction

In this dissertation we have introduced and investigated translation-based approaches to conformant planning. Most of the previous recent work on this topic has been based on the *search on belief space* paradigm where nodes are *belief states*; *i.e.* sets of possible states. Our work, in contrast, introduces a formulation that addresses the problem by translating the conformant problem into other models used in artificial intelligence: propositional logic and classical planning. These translations are exponential in the worst case but are not necessarily so, and allow us to use state-of-the-art SAT solvers, d-DNNF compilers, and classical planning tools and algorithms.

In the first part of this dissertation, we considered CNF encodings and used search and operations over logical formulas for obtaining conformant plans. Our algorithms used available tools for SAT solving and d-DNNF compilation.

In the second part, we mapped conformant planning problems into classical problems whose solutions are the conformant plans. We have presented a variety of translations and a structural criterion, the conformant width, and showed that a

[1]Taken from the *Holy Bible, New Living Translation*, copyright 1996, 2004. Used by permission of Tyndale House Publishers, Inc., Wheaton, Illinois 60189. All rights reserved.

polynomial translation called K_i is complete for problems whose width is bounded by i. This translation allows us to solve conformant problems with bounded width using a polynomial size transformation and *off-the-shelf* classical planners, even if the number of possible initial states is exponential. The translation K_1 is the base of our conformant planner T_0 that was the winner of the Conformant track of the 2006 *International Planning Competition* (IPC-2006).

10.2 Contributions

In this section we outline the main contributions of the dissertation, referring to the appropriate chapters and publications where they originally appeared:

1. An algorithm for finding conformant plans of N time steps based on an encoding the problem in CNF, a compilation to d-DNNF, and a DPLL-like search in the space of possible plans. To prune partial plans that cannot be extended to conform with all possible initial states, the algorithm uses *model counting* and *projection* operations, that are rendered efficient by the d-DNNF compilation. (in Chapter 3, and Palacios, Bonet, Darwiche, and Geffner, 2005).

2. An algorithm for finding conformant plans of N time steps based on an encoding the problem in CNF and a compilation into d-DNNF, but to produce a new CNF formula whose satisfying assignments correspond to conformant plans. Thus, for obtaining a conformant plan, a SAT solver is called once upon the resulting CNF formula. The *projection* required to obtain such formula runs in linear time in the size of the d-DNNF representation. (in Chapter 4, and Palacios and Geffner, 2006b).

   An appealing feature of this algorithm is that it is based on *two off-the-shelf components:* a d-DNNF compiler and a SAT solver (Palacios and Geffner, 2006b).

3. A sound but incomplete mapping from conformant into classical planning, called $K_0(P)$, which allows us to solve problems using a classical planner when the missing information is not relevant for obtaining a solution (in Chapter 5, and Palacios and Geffner, 2006a, 2009).

4. A general sound translation scheme from conformant problems P into classical planning problems $K_{T,M}(P)$ and the conditions under which this translation is complete, meaning that all the conformant plans of P can be obtained from the classical planning problem $K_{T,M}(P)$ (in Chapter 6, and Palacios and Geffner, 2007, 2009).

5. A characterization of the complexity of the complete $K_{T,M}(P)$ translation in terms of a structural parameter of the problem P that we call the *conformant width.* The complexity of the complete translation is exponential on the conformant width, which for most benchmark domains turns out to be bounded. For these domains, the complete translation is polynomial in the number of variables of the problem (in Chapter 6, and Palacios and Geffner, 2009).

6. A polynomial translation called K_i, instance of $K_{T,M}(P)$, which is complete for problems with conformant width no greater than i (in Chapter 6, and Palacios and Geffner, 2007, 2009).

7. A conformant planner T_0, based on the translations K_1 and K_{models}, that uses the classical planner FF and was the winner of the Conformant track of the IPC-2006 (in Chapter 7, and Palacios and Geffner, 2009).

We have also discussed the relation between the tags t used in the literals KL/t of the $K_{T,M}(P)$ translation and the possible initial states of a conformant problem P, providing a novel perspective on how the incomplete information can be compiled away for obtaining problems with no uncertainty (See Section 6.5, and Palacios and Geffner, 2009).

10.3 The Model-based approach to AI

We have introduced novel approaches for solving conformant planning problems by translating them into well known problems: propositional satisfiability, d-DNNF compilation, and classical planning. These mappings have resulted in a variety of successful conformant planning algorithms, in many cases outperforming state-of-the-art planners.

These translations and algorithms exploit a number of well-defined models and solvers. The *model-based* approach to artificial intelligence has produced a number of models and powerful solvers, and has received increasing attention in the literature. To assess the impact of the work on models and algorithms in current research, we show in Figure 10.1 on the following page a *word cloud* using the abstracts of the *Journal of Artificial Intelligence Research* (JAIR) from April 2004 to April 2009. Without regard to any formal statistical meaning, larger words in Figure 10.1 like *problem*, *model*, and *algorithm* appear larger than others, as they appear more frequently in JAIR abstracts.

The algorithms and systems introduced in this dissertation can take immediate advantage of future improvements in the state of the art for SAT solving, d-DNNF compilation and classical planning algorithms, without having to modify the code at all. Tools performing well in future editions of the SAT *Competition*[2] or in the IPC are likely thus to improve the performance of our planners.

10.4 Current and Future Work

We outline current and future research lines based on the work presented in the dissertation.

Conformant planning is a particular case of *contingent planning* that also features uncertainty but allows for feedback and thus leads to a different solution form. While solutions for conformant problems are sequences of actions, solutions of contingent problems are trees or graphs, as it is useful and in many cases necessary, to apply different actions for different outcomes of an observation.

We have collaborated with Albore and Geffner in creating a contingent planner called CLG, that builds upon our $K_{T,M}(P)$ translation. CLG is an effective *action selection* mechanism that enables the solution of contingent problems on-line or off-line.

[2]http://www.satcompetition.org

Figure 10.1: Cloud of words appearing in abstracts of the *Journal of Artificial Intelligence Research* (JAIR) from April 2004 to April 2009. Some plurals were collapsed with their respective singular versions. Articles and other less meaningful words were removed by the software. Elaborated at www.wordle.net. See it online at `http://www.wordle.net/show/wrdl/948576/jair.org_abstract_from_2004_to_june_2009`

Reaching the goal on-line avoids generating a solution for all the possible observations, dealing only with those coming from the environment, allowing to scale to problems whose complete solution would be too big. The CLG planner takes a contingent planning problem as input, removes the observations, and feeds the resulting conformant problem into our translation $K_1(P)$ as used by the T_0 conformant planner. The resulting classical problem is modified so that it takes into account observations and the possibility that such observations rule out a possible initial state. CLG is complete for problems with *contingent width* 1. Such width is a measure for bounding the size of the translation required for completeness (Albore, Palacios, and Geffner, 2009).

We have also collaborated with Bonet and Geffner on the derivation of finite state controllers for a class of contingent planning problems (Bonet, Palacios, and Geffner, 2009). Instead of having solutions that are trees or graphs, the form was restricted to apply the same action when the agent is in presence of the same observation and in the same *controller state* of the agent. When the agent applies an action, it possibly changes its controller state to a new one, so that it can apply a different action in the presence of the same observation. The solutions obtained with this restriction are like *Mealy* finite-state automaton (Mealy, 1955). Automatons are very robust controllers frequently used in robotics, but typically written by hand. Many of the obtained controllers are able to solve problems with different size or number of objects.

Neither of these two works on contingent planning are part of this dissertation, but represent an interesting line of work that is being explored.

The problems generated using the translation $K_{T,M}(P)$ are different to most benchmarks normally used for evaluating the performance of classical planners. The planner T_0 currently uses the classical planner FF (Hoffmann and Nebel, 2001), because it performed better than other options we tried. In the future, we plan to use more recent and powerful planners such as LAMA (Richter et al., 2008), but this requires further support for handling conditional effects. Another limitation of current classical planners is that the size of the $K_{T,M}(P)$ translation may be beyond the limit of most planner parsers. We considered the possibility of not generating grounded PDDLs but using predicates to have a smaller theory. The drawback of this approach is that many simplifications done by the planner T_0 cannot be applied.

Part V

Appendix

Appendix **A**

Proofs of the $K_0(P)$ and $K_{T,M}(P)$ translations

In this appendix we proof formal results of Chapter 6 and also of basic translation $K_0(P)$ in Section 5.2. We proof assuming consistency of the resulting classical problems, and in Chapter B we prove that given a consistency conformant problem P, the resulting classical translation $K_{T,M}(P)$ is also consistent.

P below stands for a conformant planning problem $P = \langle F, I, O, G\rangle$ and $K_{T,M}(P) = \langle F', I', O', G'\rangle$ for its translation. Propositions and theorems in chapters 5 and 6 appear in this appendix with the same numbers; while new lemmas and propositions have numbers preceded by the letters A and B (for Appendix A and B). The conformant problem P and the classical problems P/s and $K_{T,M}(P)$ that arise from P are all assumed to be *consistent*. Consistency issues are important, and they are addressed in more detail in the second part of this appendix where it is shown that if P is consistent, $K_{T,M}(P)$ is consistent too (Appendix B). For a consistent *classical* problem P', the standard progression lemma applies; namely, a literal L is achieved by an applicable action sequence $\pi_{+1} = \pi, a$, where π is an action sequence and a is an action iff A) π achieves C for a rule $a : C \rightarrow L$ in P', or B) π achieves L and the negation $\neg L'$ of a literal L' in the body C' of each rule in P' of the form $a : C' \rightarrow \neg L$ (see Theorem B.2 below).

Lemma A.1. *Let π be an action sequence applicable in both P and $K_0(P)$. Then if π achieves KL in $K_0(P)$, π achieves L in P.*

Proof. By induction on the length of π. If π is empty and π achieves KL in $K_0(P)$, then KL must be in I', and hence L must be in I, so that π achieves L in P.

Likewise, if $\pi_{+1} = \pi, a$ achieves KL in $K_0(P)$ then *A)* there is rule $a : KC \rightarrow KL$ in $K_0(P)$, such that π achieves KC in $K_0(P)$; or *B)* π achieves KL in $K_0(P)$ and for each rule $a : \neg K\neg C' \rightarrow \neg KL$ in $K_0(P)$, π achieves $K\neg L'$ in $K_0(P)$ for some L' in C'.

If *A)* is true, then P must contain a rule $a : C \rightarrow L$, and by inductive hypothesis, π must achieve C in P, and therefore, $\pi_{+1} = \pi, a$ must achieve L in P. If *B)* is true, by inductive hypothesis, π must achieve L in P along with $\neg L'$ for some literal L' in the body C' of each rule $a : C' \rightarrow \neg L$, and thus $\pi_{+1} = \pi, a$ must achieve L in P too. □

Lemma A.2. *If an action sequence π is applicable in $K_0(P)$, then π is applicable in P.*

Proof. If π is empty, this is trivial. Likewise, if $\pi_{+1} = \pi, a$ is applicable in $K_0(P)$, π is applicable in $K_0(P)$, and thus by inductive hypothesis, π is applicable in P. Also since, π, a is applicable in $K_0(P)$, π must achieve the literals KL in $K_0(P)$ for each precondition L of a, but then from Lemma A.1, π must achieve the literals L for the same preconditions in P, and thus, the sequence $\pi_{+1} = \pi, a$ is applicable in P. □

Proposition 5.2 *(p. 65) If π is a classical plan for $K_0(P)$, then π is a conformant plan for P.*

Proof. Direct from Lemma A.2 once we consider a problem P' similar to P but with a new dummy action a_G whose preconditions are the goals G of P. Then if π is a plan for $K_0(P)$, π, a_G is applicable in $K_0(P')$, and by Lemma A.2, π, a_G is applicable in P', which implies that π is applicable in P and achieves G, and thus, that π is a plan for P. □

Proposition 5.3 *(p. 66) An action sequence π is a classical plan for $K_0(P)$ iff π is a conformant plan for P according to the 0-approximation semantics.*

Proof. Let us say that an action sequence $\pi = a_0, \ldots, a_n$ is 0-applicable in P and 0-achieves a literal L in P if the belief sequence $b_0, \ldots, b_{n+1}$ generated according to the 0-approximation semantics is such that the preconditions of the actions a_i in π are true in b_i, and the goals are true in b_{n+1} respectively. From the definition of the 0-approximation semantics (and the consistency of P), an applicable action sequence π thus 0-achieves a literal L in P iff π is empty and $L \in I$, or $\pi = \pi', a$ and A) $a : C \rightarrow L$ is an effect of P and π' 0-achieves each literal L' in C, or B) π' 0-achieves L and for all effects $a : C' \rightarrow \neg L$ in P, π' 0-achieves $\neg L'$ for some $L' \in C'$. These, however, are the conditions under which π achieves the literal KL in $K_0(P)$ once 'a sequence 0-achieving a literal L in P' is replaced by 'a sequence achieving the literal KL in $K_0(P)$'. Thus, an action sequence π that is applicable in $K_0(P)$ and 0-applicable in P achieves a literal KL in $K_0(P)$ iff π 0-achieves the literal L in P, while π is applicable to $K_0(P)$ iff it is 0-applicable to P, with the last part following from the first using induction on the plan length. □

Definition A.3. *For an action a in P, define a^* to be the action sequence where a is followed by all merges in $K_{T,M}(P)$ in arbitrary order. Similarly, if $\pi = a_0, \ldots, a_i$ is an action sequence in P, define π^* to be the action sequence $\pi^* = a_0^*, \ldots, a_n^*$ in $K_{T,M}(P)$.*

Lemma A.4. *Let π be an action sequence such that π is applicable in P and π^* is applicable in a valid translation $K_{T,M}(P)$. If π^* achieves KL/t in $K_{T,M}(P)$, then π achieves L in P/s for all possible initial states s that satisfy t.*

Proof. For an empty π, if π^* achieves KL/t, from the definition of $K_{T,M}(P)$ and since $I \models t \supset L$, L must be in any such s, and thus π must achieve L in P/s.

Likewise, if $\pi_{+1} = \pi, a$ and t is *not* the empty tag, $\pi^*_{+1} = \pi^*, a^*$ achieves KL/t in $K_{T,M}(P)$ iff A) π^* achieves KC/t in $K_{T,M}(P)$ for a rule $a : KC/t \rightarrow KL/t$ in $K_{T,M}(P)$, or B) π^* achieves KL/t, and for any rule $a : \neg K\neg C'/t \rightarrow \neg KL/t$, π^* achieves $K\neg L'/t$ in $K_{T,M}(P)$ for some L' in C' (merge actions do not delete positive literals KL/t).

If A, by inductive hypothesis, π achieves C in P/s for each possible initial state s that satisfies t, and hence $\pi_{+1} = \pi, a$ achieves L in P/s from the rule $a : C \rightarrow L$ that must be in P. If B, by inductive hypothesis, π achieves L and $\neg L'$ in P/s, for some L' in the body of each rule $a : C' \rightarrow \neg L$ in P, and thus $\pi_{+1} = \pi, a$ achieves L in P/s.

For the empty tag $t = \emptyset$, a third case must be considered: a merge action $\bigwedge_{t' \in m} KL/t' \rightarrow KL$ in a^* may be the cause for the action sequence $\pi^*_{+1} = \pi^*, a^*$ achieving KL in $K_{T,M}(P)$. In such a case, the sequence π^*, a, and hence π^*, a^*, must achieve KL/t' for each (non-empty) $t' \in m$ in $K_{T,M}(P)$, and hence from the inductive hypothesis and the two cases above, the sequence π, a must achieve L in P/s for each possible initial state s that satisfies any such t'. Yet, since the merge m is valid, all possible initial states s must satisfy one such t', and thus π must achieve L in P/s for all possible initial states s, that are the initial states that satisfy $t = \emptyset$. □

Lemma A.5. *If π^* is applicable in a valid translation $K_{T,M}(P)$, then π is applicable in P.*

Proof. If π is empty, this is direct. For $\pi_{+1} = \pi, a$, if $\pi^*_{+1} = \pi^*, a^*$ is applicable in $K_{T,M}(P)$, then π^* is applicable in $K_{T,M}(P)$, achieving KL for each precondition L of a, and hence from the inductive hypothesis, π is applicable in P, and from Lemma A.4, π must achieve L for each precondition L of a, and thus $\pi_{+1} = \pi, a$ is applicable in P. □

Theorem 6.4 *(p. 72) The translation $K_{T,M}(P)$ is sound provided that all merges in M are valid and all tags in T are consistent.*

Proof. Consider the problem P' that is similar to P but with a new dummy action a_G whose preconditions are the goals G of P. We have then that π^* is a plan for $K_{T,M}(P)$ iff π^*_1, a^*_G is applicable in $K_{T,M}(P')$, which from Lemma A.5 implies that π, a_G is applicable in P', which means that π is a plan for P. □

Lemma A.6. *Let π be an action sequence such that π is applicable in P and π^* is applicable in $K_{S0}(P)$. If π achieves L in P/s for some possible initial state s, π^* achieves KL/s in $K_{S0}(P)$.*

Proof. If π is empty and π achieves L in P/s, then $L \in s$, and since $I \models s \supset L$, KL/s must be in I' and thus π^* achieves KL/s in $K_{S0}(P)$.

Likewise, if $\pi_{+1} = \pi, a$ achieves L in P/s then A) there is rule $a : C \rightarrow L$ such that π achieves C in P/s; or B) π achieves L and for any rule $a : C' \rightarrow \neg L$, π achieves $\neg L'$ in $K_{S0}(P)$ for some $L' \in C'$.

If A), by inductive hypothesis, π^* achieves KC/s in $K_{S0}(P)$ and, from rule $a : KC/s \to KL/s$, π^*, a must achieve KL/s, and thus, $\pi^*_{+1} = \pi^*, a^*$ achieves KL/s (merges in a^* do not delete positive literals KL/t).

If B), by inductive hypothesis, π^* achieves KL/s and $K\neg L'/s$ in $K_{S0}(P)$ for some L' in the body of each rule $a : C' \to \neg L$ in P, and therefore π^*, a achieves KL/s, and so does $\pi^*_{+1} = \pi^*, a^*$. □

Lemma A.7. *If π is applicable in P, π^* is applicable in $K_{S0}(P)$.*

Proof. If π is empty, this is trivial. If $\pi_{+1} = \pi, a$ is applicable in P, then π must be applicable in P and must achieve each precondition L of a in P/s for every possible initial state s, $s \in S_0$. From the inductive hypothesis, π^* must then be applicable in $K_{S0}(P)$, and from Lemma A.6, it must achieve the literals KL/s for all $s \in S_0$, and then, the last merge action with effect $\bigwedge_{s \in S_0} KL/s \to KL$ in π^* must achieve KL, and so does π^*, and therefore, π^*, a^* is applicable in $K_{S0}(P)$. □

Theorem 6.6 *(p. 76) If π is a conformant plan for P, then there is a classical plan π' for $K_{S0}(P)$ such that π is the result of dropping the merge actions from π'.*

Proof. Direct from Lemma A.7 if we consider a problem P' similar to P but with a new action a_G whose preconditions are the goals G of P. If π is a plan for P, the sequence π, a_G is applicable in P', and from Lemma A.7, π^*, a^*_G is applicable in $K_{S0}(P')$, and thus π^* is a plan for $K_{S0}(P)$. □

Definition A.8. *$rel(s, L)$ stands for the set of literals L' in s that are relevant to L in P:*

$$rel(s, L) = \{L' \mid L' \in s \text{ and } L' \text{ is relevant to } L\} \ .$$

Definition A.9. *t^* stands for the deductive closure of t under I:*

$$t^* = \{\ L \mid I, t \models L\} \ .$$

Theorem A.10. *Let $m = \{t_1, \ldots, t_n\}$ be a covering merge for a literal L in a valid translation $K_{T,M}(P)$ for a problem P whose initial situation is in prime implicate form. Then for each tag t_i in m there must be a possible initial state s of P such that $rel(s, L) \subseteq t^*_i$.*

Proof. Assume otherwise that each state s satisfying I makes true a literal L_s relevant to L such that $L_s \notin t^*_i$. If we then take c to be the disjunction of such literals L_s over all the states s that satisfy I, we obtain that I entails c, which since I is in prime implicate form, means that c contains a tautology c' or is subsumed by a clause c'' in I. But, in either case, this is a contradiction, as all the literals in c' or c'' are relevant to L, and hence t^*_i, where t_i is part of the covering merge m, must contain a literal in either c' or c'', and hence in c. □

Lemma A.11. *Let π be an action sequence such that π is applicable in P and π^* is applicable in a covering translation $K_{T,M}(P)$. Then, if π achieves L in P/s for some possible initial state s and there is a tag t in T such that $rel(s, L) \subseteq t^*$, π^* achieves KL/t in $K_{T,M}(P)$.*

Proof. If π is empty and π achieves L in P/s, then L is in s and thus, in $rel(s, L)$. Since $rel(s, L) \subseteq t^*$, then $L \in t^*$, and thus KL/t is in the initial situation I' of $K_{T,M}(P)$, and π^* achieves KL/t in $K_{T,M}(P)$. Likewise, if $\pi_{+1} = \pi, a$ achieves L in P/s, then A) there is a rule $a : C \rightarrow L$ in P such that π achieves C in P/s, or B) π achieves L in P/s and for each rule $a : C' \rightarrow \neg L$, π achieves $\neg L'$ in P/s for some L' in C'. If A, by inductive hypothesis, π^* achieves KC/t, and from the support rule $a : KC/t \rightarrow KL/t$ in $K_{T,M}(P)$, π^*, a must achieve KL/t in $K_{T,M}(P)$, and so must $\pi^*_{+1} = \pi^*, a^*$, as the merges in a^* cannot delete a positive literal KL/t. If B, by inductive hypothesis, π^* achieves KL/t, and for each cancellation rule $a : \neg K\neg C'/t \rightarrow \neg KL/t$ arising from the rule $a : C' \rightarrow \neg L$ in P, π^* must achieve $K\neg L'/t$ for some literal $L' \in C'$. This means that π^*, a, and therefore, $\pi^*_{+1} = \pi^*, a^*$, must achieve KL/t. □

Lemma A.12. *Let $K_{T,M}(P)$ be a covering translation of P. Then if π is applicable in P, π^* is applicable in $K_{T,M}(P)$.*

Proof. If π is empty, this is direct. Else, if $\pi_{+1} = \pi, a$ is applicable in P, then π must be applicable in P where it must achieve each literal L in $Pre(a)$, and therefore, by inductive hypothesis π^* must be applicable in $K_{T,M}(P)$. Then, let $m = \{t_1, \ldots, t_n\}$ be a covering merge for $L \in Pre(a)$ in $K_{T,M}(P)$. From Theorem A.10, for each $t_i \in m$ there must be a possible initial state s such that $rel(s, L) \subseteq t_i^*$, and then from Lemma A.11, π achieving L in P/s implies π^* achieving KL/t_i in $K_{T,M}(P)$. Since this is true for all $t_i \in m$ and π achieves $L \in Pre(a)$ in P/s for all possible initial states s, then it follows that π^* achieves KL/t_i for all $t_i \in m$ in $K_{T,M}(P)$, and therefore that π^* achieves KL in $K_{T,M}(P)$ as π^* ends with a sequence of merges that include the action merge $a_{m,L}$ with effect $\bigwedge_{t_i \in m} KL/t_i \rightarrow KL$. As a result, $\pi^*_{+1} = \pi^*, a^*$ is applicable in $K_{T,M}(P)$. □

Theorem 6.12 *(p. 79) Covering translations $K_{T,M}(P)$ are complete; i.e., if π is a conformant plan for P, then there is a classical plan π' for $K_{T,M}(P)$ such that π is π' with the merge actions removed.*

Proof. The theorem follows trivially from Lemma A.12 by having a problem P' that is like P but with an additional, dummy action a_G such that the goals G of P are the preconditions of a_G. The action sequence π is a plan for P iff the action sequence π, a_G is applicable in P', which due to Lemma A.12 implies that the action sequence π^*, a^*_G is applicable in $K_{T,M}(P')$ which in turn is true iff the action sequence π^* is a plan for $K_{T,M}(P)$. The sequence π, in turn, is the sequence π^* with all the merge actions removed. □

Theorem 6.14 *(p. 80) The translation $K_{models}(P)$ is sound and complete.*

Proof. Direct from the merges m generated by K_{models} for each precondition and goal literals L. Clearly these merges are all valid, their tags are consistent with I, and they cover L (the models of $C_I(L)$ all satisfy $C_I(L)$). Thus the result follows from Theorems 6.4 and 6.12. □

Proposition 6.18 *(p. 82) The width $w(P)$ of P can be determined in time that is exponential in $w(P)$.*

Proof. If m is the number of clauses in $C^*_I(L)$, then there are at most m^i sets of clauses $\mathcal{C}$ in $C^*_I(L)$ such that $|\mathcal{C}| = i$. Each clause in one such set must have at most n literals, where n is the number of fluents in P, and hence, if one literal from each clause in $\mathcal{C}$ is collected, we end up with at most n^i sets of literals of size no greater than i, some of which are inconsistent with I and some of which are consistent and minimal (no other consistent set in the collection is properly included); both tests being polynomial given that I is in prime implicate form. Thus constructing the cover $c(\mathcal{C})$ for a set of clauses $\mathcal{C}$ with $|\mathcal{C}| = i$ is exponential in i, while checking whether one such cover satisfies $C_I(L)$ is a polynomial operation provided that I is in prime implicate form. Indeed, if $c(\mathcal{C}) = \{t_1, \ldots, t_n\}$, computing the closures t^*_i for each $t_i \in c(\mathcal{C})$, when I is in PI, and testing whether each t^*_i intersects each clause in $C_I(L)$ are polynomial operations (the former reducing to checking for each literal L' whether $I \models \neg t^*_i \vee L'$). Thus for computing $width(L)$, we generate all sets $\mathcal{C}$ of clauses in $C^*_I(L)$ with $|\mathcal{C}| = i$, starting with $i = 0$, increasing i one by one until for one such set, $c(\mathcal{C})$ satisfies $C_I(L)$. This computation is exponential in $w(L)$, and the computation over all preconditions and goal literals in P is exponential in $w(P)$. □

Proposition 6.19 *(p. 82) The width of P is such that $0 \leq w(P) \leq n$, where n is the number of fluents whose value in the initial situation is not known.*

Proof. The inequality $0 \leq w(P)$ is direct as $w(L)$ is defined as the size $|\mathcal{C}|$ of the minimal set of clauses $\mathcal{C}$ in $C^*_I(L)$ such that $c(\mathcal{C})$ satisfies $C_I(L)$, and $w(P) = w(L)$ for some precondition and goal literal L. The inequality $w(P) \leq n$ follows by noticing that for the set $\mathcal{C}$ of clauses given by the tautologies $L' \vee \neg L'$ in $C^*_I(L)$, $c(\mathcal{C})$ must satisfy each clause c in $C_I(L)$, as each $t \in c(\mathcal{C})$ must assign a truth value to each literal in c, and if inconsistent with c, it will be inconsistent with I and thus pruned from $c(\mathcal{C})$. Finally, the max number of such tautologies in $C^*_I(L)$ is the number of fluents L' such that neither L' nor $\neg L'$ are unit clauses in I. □

Theorem 6.21 *(p. 83) For a fixed i, the translation $K_i(P)$ is sound, polynomial, and if $w(P) \leq i$, covering and complete.*

Proof. For soundness, we just need to prove that all merges m in $K_i(P)$ are valid and that all tags t in $K_i(P)$ are consistent. The soundness follows from Theorem 6.4. The merges m for a literal L in $K_i(P)$ are given by the covers $c(\mathcal{C})$ of collections $\mathcal{C}$ of i or less clauses in $C^*_i(L)$ and clearly since each model $\mathcal{M}$ of I must satisfy $C^*_I(L)$, it must satisfy some $t \in c(\mathcal{C})$ so that $I \models \bigvee_{t \in m} t$ for $m = c(\mathcal{C})$. At the same time, from the definition of the cover $c(\mathcal{C})$, each of these tags t must be consistent with I.

For proving that K_i is polynomial for a fixed i, we follow ideas similar to the ones used in the proof of Proposition 6.18 above, where we have shown that the width of P can be determined in time that is exponential in $w(P)$ and polynomial in the number of clauses and fluents in P. For a fixed i, the number of sets of clauses $\mathcal{C}$ in $C^*_I(L)$ with size $|\mathcal{C}| \leq i$ is polynomial, and the complexity of computing the covers $c(\mathcal{C})$ for such sets, and hence, the merges m for L in $K_i(P)$ is polynomial too. Thus, the whole translation $K_i(P)$ for a fixed i is polynomial in the number of clauses, fluents, and rules in P.

Finally, for proving completeness, if $w(P) \leq i$, then $w(L) \leq i$ for each precondition and goal literal L in P. Therefore, for each such literal L, there is a set $\mathcal{C}$ of clauses in $C_I^*(L)$ such that $c(\mathcal{C})$ satisfies $C_I(L)$. The translation $K_i(P)$ will then generate a unique merge for L that covers L. Since $K_i(P)$ is a valid translation, this means that $K_i(P)$ is a covering translation, that is then complete, by virtue of Theorem 6.12. □

Lemma A.13. *If L' is relevant to L and $rel(s, L) \subseteq rel(s', L)$, then $rel(s, L') \subseteq rel(s', L')$.*

Proof. If L'' is in $rel(s, L')$, then L'' is relevant to L', and since L' is relevant to L and the relevance relation is transitive, L'' is relevant to L. Thus, L'' is in $rel(s, L)$ and therefore, since $rel(s, L) \subseteq rel(s', L)$, L'' is in $rel(s', L)$. But then L'' is in s' and since it is relevant to L', L'' is in $rel(s', L')$. □

Proposition 6.23 *(p. 87) Let s and s' be two states and let π be an action sequence applicable in the classical problems P/s and P/s'. Then if π achieves a literal L in P/s' and $rel(s', L) \subseteq rel(s, L)$, π achieves the literal L in P/s.*

Proof. By induction on the length of π. If π is empty, and π achieves a literal L in P/s', L must be in s', and since L is relevant to itself, $L \in rel(s', L)$. Then as $rel(s', L) \subseteq rel(s, L)$, L must be in s, and thus π achieves L in P/s.

Likewise, if $\pi_{+1} = \pi, a$ achieves L in P/s' then A) there is rule $a : C \to L$ such that π achieves C in P/s'; or B) π achieves L in P/s' and for any rule $a : C' \to \neg L$, π achieves $\neg L'$ in P/s' for some $L' \in C'$.

If A, π must achieve each literal $L_i \in C$ in P/s'. Since L_i is relevant to L and $rel(s', L) \subseteq rel(s, L)$, by Lemma A.13, $rel(s', L_i) \subseteq rel(s, L_i)$. Then, by inductive hypothesis, the plan π must achieve L_i in P/s for each $L_i \in C$, and thus $\pi_{+1} = \pi, a$ must achieve L in P/s

If B, since each such $\neg L'$ is relevant to L (as L' is relevant to $\neg L$), and $rel(s', L) \subseteq rel(s, L)$, by Lemma A.13, $rel(s', \neg L') \subseteq rel(s, \neg L')$, and thus by inductive hypothesis, π must achieve $\neg L'$ in P/s and also L, so that $\pi_{+1} = \pi, a$ must achieve L in P/s. □

Lemma A.14. *If S and S' are two collection of states such that for every state s in S and every precondition and goal literal L in P, there is a state s' in S' such that $rel(s', L) \subseteq rel(s, L)$, then if π is applicable in P/S', π is applicable in P/S.*

Proof. By induction on the length of π. If π is empty, it is obvious. If $\pi_{+1} = \pi, a$ is applicable in P/S', then π is applicable in P/S' and, by inductive hypothesis, π is applicable in P/S. We need to prove that π achieves the preconditions of action a in P/S.

For any $L \in Prec(a)$ and any $s \in S$, from the hypothesis, there is a state $s' \in S'$ such that $rel(s', L) \subseteq rel(s, L)$. From Proposition 6.23, and since π achieves L in P/s', π must achieve L in P/s. Since the argument applies to any $s \in S$, π achieves L in P/S, and thus $\pi_{+1} = \pi, a$ must be applicable in P/S. □

Proposition 6.24 *(p. 87) If S and S' are two collections of states such that for every state s in S and every precondition and goal literal L in P, there is a state s' in S' such that $rel(s', L) \subseteq rel(s, L)$, then if π is a plan for P that conforms with S', π is a plan for P that conforms with S.*

Proof. From Lemma A.14, we consider a problem P' similar to P but with a new action a_G whose preconditions are the goals G of P. If π is a plan for P that conforms with S', then the action sequence π, a_G is applicable in P'/S', and then from the lemma, π, a_G is applicable in P'/S, and thus π must be a plan for P/S □

Proposition 6.25 *(p. 87) S' is a basis for P if for every possible initial state s of P and every precondition and goal literal L in P, S' contains a state s' such that $rel(s', L) \subseteq rel(s, L)$.*

Proof. Direct from Proposition 6.24, by considering S to be the set of possible initial states of P. □

Proposition 6.26 *(p. 88) If the initial situation I is in prime implicate form and $m = \{t_1, \ldots, t_n\}$ is a merge that covers a literal L in P, then the set $S[t_i, L]$ of possible initial states s of P such that $rel(s, L) \subseteq t_i^*$ is non-empty.*

Proof. Direct from Theorem A.10. □

Theorem 6.27 *(p. 88) Let $K_{T,M}(P)$ be a covering translation and let S' stand for the collection of states $s[t_i, L]$ where L is a precondition or goal literal of P and t_i is a tag in a merge m that covers L. Then S' is a basis for P.*

Proof. We show that for every possible initial state s and any precondition and goal literal L, S' in the theorem contains a state s' such that $rel(s', L) \subseteq rel(s, L)$. The result then follows from Proposition 6.25. Indeed, any such state s must satisfy a tag t_i in a covering merge $m = \{t_1, \ldots, t_n\}$ for L, as these merges are valid. But from Theorem A.10, there must be a possible initial state s' such that $rel(s', L) \subseteq t_i^*$, and therefore, $rel(s', L) \subseteq rel(s, L)$ as s must satisfy t_i^* and possibly other literals L' that are relevant to L. □

Theorem 6.28 *(p. 88) If P is a conformant planning problem with bounded width, then P admits a basis of polynomial size.*

Proof. If $w(P) \leq i$ for a fixed i, $K_i(P)$ is a covering translation with a polynomial number of merges and tags, and in such case, the basis S' for P defined by Theorem 6.27 contains a polynomial number of states, regardless of the number of possible initial states. □

Appendix **B**

Consistency of the $K_{T,M}(P)$ translation

We have been assuming throughout chapters 5 and 6 appear that the conformant planning problems P and their translations $K_{T,M}(P)$ are consistent. In this section we make this notion precise, explain why it is needed, and prove that $K_{T,M}(P)$ is consistent if P is. For the proof, we take into account that the heads KL of the merge actions $a_{m,L}$ in $K_{T,M}(P)$, are extended with the literals $K\neg L'$ for the literals L' that are mutex with L in P (see Definition 6.1).

We start at the beginning assuming that *states are not truth-assignments but sets of literals over the fluents of the language.* A state is *complete* if for every literal L, L or $\neg L$ is in s, and *consistent* if for no literal both L and $\neg L$ are in s. Complete and consistent states represent truth-assignments over the fluents F and the consistency of P and of the translation $K_{T,M}(P)$ ensures that all applicable action sequences π map complete and consistent states s into complete and consistent states s'. Once this is guaranteed, complete and consistent states can be referred to simply as states which is what we have done in chapters 5 and 6 and in the proofs in appendix A.

Given a complete state s and an action a applicable in s, the next state s_a is

$$s_a = (s \setminus Del(a,s)) \cup Add(a,s)$$

where

$$Add(a,s) = \{L \mid a : C \rightarrow L \text{ in } P \text{ and } C \subseteq s\}$$

and

$$Del(a,s) = \{\neg L \mid L \in Add(a,s)\} \ .$$

It follows from this that s_a is a complete state if s is a complete state, as the action a only 'deletes' a literal L in s if $\neg L$ is added by a in s. On the other hand, s may be consistent and s_a inconsistent, as for example, when there are rules $a : C \rightarrow L$ and $a : C' \rightarrow \neg L$ such that both C and C' are in s. In order to exclude this possibility, ensuring that all reachable states are complete and consistent, and thus represent genuine *truth assignments* over the fluents in F, a consistency condition on P is needed:

Definition B.1 (Consistency)**.** *A classical or conformant problem $P = \langle F, I, O, G\rangle$ is* consistent *if the initial situation I is logically consistent and every pair of complementary literals L and $\neg L$ is* mutex *in P.*

In a *consistent* classical problem P, all the reachable states are complete and consistent, and the standard progression lemma used in the preceding proofs holds:

Theorem B.2 (Progression)**.** *An action sequence $\pi_{+1} = \pi, a$ applicable in the complete and consistent state s achieves a literal L in a* consistent *classical problem P iff A) π achieves the body C of a rule $a : C \to L$ in P, or B) π achieves L and for every rule $a : C' \to \neg L$, π achieves $\neg L'$ for a literal L' in C'.*

We will see below that if a conformant problem P is consistent in this sense, so will be any valid translation $K_{T,M}(P)$. We have tested all the benchmarks considered in chapters 5 and 6 for consistency and found all of them to be consistent except for two domains:1-Dispose and Look-and-Grab. In these cases, since the consistency of the classical problem $K_{T,M}(P)$ cannot be inferred from the consistency of P, it can be checked explicitly using Definition B.1, or similarly, the plans that are obtained from $K_{T,M}(P)$ can be checked for consistency as indicated in Section 7.1 on page 89: the soundness of these plans is ensured provided that they never trigger conflicting effects KL/t and $\neg KL/t$.[1]

Proof. The proof of Theorem B.2 does not rest on a particular definition of mutexes, just that mutex atoms are not both true in a reachable state. In a consistent problem P, an applicable action sequence π maps s into a complete and consistent state s' that represents a truth assignment. Then, the action sequence $\pi_{+1} = \pi, a$ achieves L iff C) $L \in \mathit{Add}(a, s')$ or D) $L \in s'$ and $\neg L \notin \mathit{Del}(a, s')$. Condition A in the theorem, however, is equivalent to C, and Condition B in the theorem, is equivalent to D. Indeed, $L \notin \mathit{Del}(a, s')$ iff for each rule $a : C' \to \neg L$ there is a literal $L' \in C'$ such that $L' \notin s'$, which, given that s' is complete and consistent, is true iff $\neg L' \in s'$ (this is precisely where consistency is needed; else $\neg L' \in s'$ would not imply $L' \notin s'$). □

The notion of mutex used in the definition of consistency expresses a guarantee that a pair of literals is not true in a reachable state. Sufficient and polynomial conditions for mutual exclusivity and other type of invariants have been defined in various papers, here we follow the definition by Bonet and Geffner (1999).

Definition B.3 (Mutex Set)**.** *A mutex set is a collection R of unordered literals pairs (L, L') over a* classical *or* conformant *problem P such that:*

1. *for no pair (L, L') in R, both L and L' are in a possible initial state s,*
2. *if $a : C \to L$ and $a : C' \to L'$ are two rules for the same action where (L, L') is a pair in R, then $\mathit{Pre}(a) \cup C \cup C'$ is mutex in R, and*
3. *if $a : C \to L$ is a rule in P for a literal L in a pair (L, L') in R, then either a) $L' = \neg L$, b) $\mathit{Pre}(a) \cup C$ is mutex with L' in R, or c) $\mathit{Pre}(a) \cup C$ implies C' in R for a rule $a : C' \to \neg L'$ in P;*

[1]The consistency of the two domains, 1-Dispose and Look-and-Grab, can be established however if a definition of mutexes slightly stronger than the one below is used. It actually suffices to change the expression $\mathit{Pre}(a) \cup C$ in clause 3c) of the definition of mutex sets below by $\mathit{Pre}(a) \cup C \cup \{L'\}$.

In this definition, a pair is said to be mutex in R if it belongs to R, a set of literals S is said to be mutex in R if S contains a pair in R, and a set of literals S is said to imply a set of literals S' in R when S is mutex in R with the complement $\neg L$ of each literal L in $S' \setminus S$.

It easy to verify that if R_1 and R_2 are mutex sets, their union $R_1 \cup R_2$ is a mutex set, and thus that there is a maximal mutex set for P that we denote as R^*. The pairs in R^* are just called *mutexes.*

For simplicity and without loss of generality, we will assume that preconditions $Pre(a)$ are empty. Indeed, it is simple to show that the mutexes of a problem P remain the same if preconditions are pushed in as conditions. We also assume that no condition C in a rule $C \rightarrow L$ in P is mutex, as these rules can be simply pruned. In addition, we assume that no literal L is mutex with a pair of complementary literals L' and $\neg L'$, as then L cannot be true in a reachable state, and thus, can be pruned as well.

The definition of mutexes is sound, meaning that no pair in a mutex set can be true in a reachable state:

Theorem B.4. *If (L, L') is a pair in a mutex set R of a classical or conformant problem P, then for no reachable state s in P, $\{L, L'\} \subseteq s$.*

Proof. We proceed inductively. Clearly, L and L' cannot be part of a possible initial state, as this is ruled out by the definition of mutex sets. Thus, let us assume as inductive hypothesis that L and L' are not part of any state s reachable in less than i steps, and let us prove that the same is true for the states $s' = s_a$ that are reachable from s in one step. Clearly if L and L' belong to s', then either A) both L and L' belong to $Add(a, s)$, or B) L belongs to $Add(a, s)$ and L' belongs to s but not to $Del(a, s)$. We show that this is not possible. For A, P must comprise rules $a : C \rightarrow L$ and $a : C' \rightarrow L'$ such that $C \cup C' \subseteq s$, yet from the definition of mutex sets, $C \cup C'$ must be mutex, and from the inductive hypothesis then $C \cup C' \not\subseteq s$. For B, there must be a rule $a : C \rightarrow L$ with $C \subseteq s$, but then from $L' \in s$ and the inductive hypothesis, it follows that L' is not mutex with C in R, and thus, from the mutex set definition, that either $L' = \neg L$ or C implies C' for a rule $a : C' \rightarrow \neg L'$. In the first case, however, due to the rule $a : C \rightarrow L$ and $C \subseteq s$, $L' \in Del(a, s)$, while in the second case, from the completeness of all reachable states, we must have $C' \subseteq s$, and hence $L' \in Del(a, s)$, contradicting B in both cases. □

Provided that the initial situation I of a conformant planning problem P is in prime implicate form, computing the largest mutex set R^* and testing the consistency of P are polynomial time operations. For the former, one starts with the set of literal pairs and then iteratively drops from this set the pairs that do not comply with the definition until reaching a fixed point (Bonet and Geffner, 1999).

We move on now to prove that if a conformant problem P is consistent, so is a valid translation $K_{T,M}(P)$. The consistency of the classical problems P/s for possible initial states s is direct, as the set of mutexes in P is a subset of the set of mutexes in P/s where the initial situation is more constrained.

Proposition B.5 (Mutex Set R_T)**.** *For a valid translation $K_{T,M}(P)$ of a consistent conformant problem P, define R_T to be the set of (unordered) literals pairs $(KL/t, KL'/t')$ and $(KL/t, \neg K\neg L'/t)$ where (L, L') is a mutex in P, and t and t'*

are two tags jointly satisfiable with I ($I \not\models \neg(t \cup t')$). Then R_T is a mutex set in $K_{T,M}(P)$.

It follows from this that $K_{T,M}(P)$ is consistent if P is consistent, as then $L' = \neg L$ is mutex with L in P, and so $(KL/t, \neg KL/t)$ must be a mutex in R_T.

Theorem B.6 (Consistency $K_{T,M}(P)$)**.** *A valid translation $K_{T,M}(P)$ is consistent if P is consistent.*

The consistency of the translation $K_0(P)$ follows as a special case, as $K_0(P)$ is $K_{T,M}(P)$ with an empty set of merges M and a set of tags T containing only the empty tag. We are left to prove Proposition B.5.

Proof of Proposition B.5. We must show that the set R_T comprised of the pairs $(KL/t, KL'/t')$ and $(KL/t, \neg K\neg L'/t)$ for L' mutex with L in P, and tags t and t' jointly satisfiable with I, is a set that complies with clauses 1, 2, and 3 of Definition B.3. We go one clause at a time.

1. No pair in R_T can be true initially in $K_{T,M}(P) = \langle F', I', O', G' \rangle$ for jointly satisfiable I, t, and t'. Indeed, if both KL/t and KL'/t' are in I' there must be a possible initial state satisfying t and t' where L and L' are true in contradiction with L and L' being mutex in P. Similarly, if KL/t is in I' but $K\neg L'/t$ not, it must be the case that $I \models t \supset L$ and $I \not\models t \supset \neg L'$, so that there must be some possible initial state of P where t, L, and L' hold, a contradiction with L and L' being mutex in P too.

2. If there is an action a with rules for KL/t and KL'/t' then the rules must be support rules of the form $a : KC/t \to KL/t$ and $a : KC'/t' \to KL'/t'$ arising from rules $a : C \to L$ and $a : C' \to L'$ in P.[2] Then since L and L' are mutex in P, C and C' must contain literals $L_1 \in C$ and $L_2 \in C'$ such that (L_1, L_2) is a mutex in P, and hence $(KL_1/t, KL_2/t')$ belongs to R_T, so that KC/t and KC'/t' are mutex in R_T as well.

   Similarly, if there is an action with rules for KL/t and $\neg K\neg L'/t$ for a literal L' mutex with L in P, the rules must be support and cancellation rules of the form $a : KC/t \to KL/t$ $a : \neg K\neg C'/t \to \neg K\neg L'/t$, arising from rules $a : C \to L$ and $a : C' \to L'$ in P. Since L and L' are mutex in P, C and C' must contain literals $L_1 \in C$ and $L_2 \in C'$ that are mutex in P, and hence R_T must contain the pair $(KL_1/t, \neg K\neg L_2/t)$, so that KC/t and $\neg K\neg C'/t$ must be mutex in R_T.

3. We are left to show that the set R_T given by the pairs $(KL/t, KL'/t')$ and $(KL/t, \neg K\neg L'/t)$ complies with clause 3 in the definition of mutex sets as well. Consider the first class of pairs $(KL/t, KL'/t')$ and a rule $a : KC/t \to KL/t$ for KL/t arising from a rule $a : C \to L$ in P. Since L is mutex with L' in P, then one of the conditions 3a, 3b, or 3c must hold for the rule $a : C \to L$ and L'. If 3a, then $L' = \neg L$, and KC/t must imply the body $\neg K\neg C/t'$ of the cancellation

[2]The action a cannot be a merge for a literal L'' mutex with both $\neg L$ and $\neg L'$, as in such case, L'' implies that L and L' that are mutex. Similarly, a cannot be a merge for L as in such a case, L will be mutex with both L' and $\neg L'$. For the same reason, a cannot be a merge for L' either. Thus, the action a above cannot be a merge and must be an action from P.

rule $a : \neg K\neg C/t' \rightarrow \neg K\neg L/t'$, as for each literal L_1 in C, R_T must contain the pair $(KL_1/t, K\neg L_1/t')$ so that KL_1/t implies $\neg K\neg L_1/t'$, and KC/t implies $\neg K\neg C/t'$ (case 3c). If 3b, then C and L' are mutex in P, and thus C contains a literal L_1 mutex with L' in P. This means that the pair $(KL_1/t, KL'/t')$ is in R_T and hence that KC/t is mutex with KL'/t' in R_T (case 3b). Last, if 3c, C implies C' in P for a rule $a : C' \rightarrow \neg L'$, but then KC/t must imply the body $\neg K\neg C'/t'$ of the cancellation rule $a : \neg K\neg C'/t' \rightarrow \neg KL'/t'$. Indeed, for each literal L_1 in both C and C', we had above that KL_1/t implies $\neg K\neg L_1/t'$, while if L_2 is a literal in C' but not in C, then some literal $L_3 \in C$ must be mutex with $\neg L_2$ in P, and hence the pair $(KL_3/t, K\neg L_2/t')$ must be in R_T and KL_3/t implies then $\neg K\neg L_2/t'$ (case 3c)

Consider now the same pair $(KL/t, KL'/t')$ along with a merge action $a_{m,L}$ with a rule $\bigwedge_{t_i \in m} KL/t_i \rightarrow KL$ for $KL/t = KL$ (thus t is the empty tag). In this case, since the merge m is valid and t' is consistent, there must be some $t_i \in m$ such that t_i and t' are jointly consistent with I. It follows then that $(KL/t_i, KL'/t')$ is a pair in R_T and thus that the body of the merge is mutex with KL'/t' in R_T (case 3b).

There is no need to consider the pair $(KL/t, KL'/t')$ along with the rules for KL'/t', as the literals KL/t and KL'/t' have the same structure, and thus the same argument above applies, replacing t with t' and L with L'.

We switch now to the second class of pairs $(KL/t, \neg K/\neg L'/t)$ and the rules $a : KC/t \rightarrow KL/t$ for KL/t. Since L and L' are mutex in P, then conditions 3a, 3b, or 3c must hold. If a, then $L' = \neg L$, and in such a case, condition 3c holds in $K_{T,M}(P)$ as KC/t implies the body KC/t of the rule $a : KC/t \rightarrow K\neg L'$ $(\neg L' = L)$. If b, C is mutex with L', and thus there is a literal L_1 in C such that L_1 and L' are mutex in P, and therefore KC/t and KL'/t are mutex in R_T (case 3b). Finally, if c, C implies C' for a rule $a : C' \rightarrow \neg L'$ in P, then KC/t must imply KC'/t in R_T for a rule $a : KC'/t \rightarrow K\neg L'/t$ (case 3c).

For the empty tag t, the rule for KL/t may also be a merge, but then due to the extra effects $K\neg L'$ in the merge action for L, the merge for KL is also a merge for $K\neg L'$, and then case 3c holds.

Last, for the same class of pairs, the only rules for $\neg K\neg L'/t$ are cancellation rules of the form $a : \neg K\neg C''/t \rightarrow \neg K\neg L'/t$ for a rule $a : C'' \rightarrow L'$ in P. Since L' is mutex with L in P, then conditions 3a, 3b, or 3c must hold for the rule $a : C'' \rightarrow L'$ and L' in P. If a, then $L = \neg L'$, and the cancellation rule is then $a : \neg K\neg C''/t \rightarrow \neg KL$ (case 3c). If b, C'' is mutex with L, and thus there is a literal L_2 in C'' such that (L_2, L) is a mutex in P, and therefore KL/t implies $K\neg L_2/t$ in R_T, and hence $\neg K\neg L_2/t$ and $\neg K\neg C''/t$ imply $\neg KL/t$ in R_T (case 3b). Finally, if c, C'' implies C' for a rule $a : C' \rightarrow \neg L$ in P, and then $\neg K\neg C''/t$ must imply $\neg K\neg C'/t$ for a rule $a : \neg K\neg C'/t \rightarrow \neg KL/t$ in R_T. Indeed, if L_A implies L_B in P, $\neg L_B$ implies $\neg L_A$ in P, and $K\neg L_B/t$ implies $K\neg L_A/t$ in R_T, and $\neg K\neg L_A/t$ implies $\neg K\neg L_B/t$.

□

Appendix C

PDDLs of some Conformant Problems

C.1 Sort-2-n

This is the PDDL encoding for $n = 3$ of the reformulation of the Sortnet-n problem called Sort-2-n, explained in Section 6.4 on page 83.

```
(define (problem sort-2-3-p)
    (:domain sort-2-3)
    (:init (and
              (or (less n1 n2) (not (less n1 n2)))
              (or (less n1 n3) (not (less n1 n3)))
              (or (less n2 n1) (not (less n2 n1)))
              (or (less n2 n3) (not (less n2 n3)))
              (or (less n3 n1) (not (less n3 n1)))
              (or (less n3 n2) (not (less n3 n2)))
    ))
    (:goal (and
             (less n1 n2)
             (less n2 n3)
    ))
)
```

```
(define (domain sort-2-3)
    (:requirements :typing :conditional-effects)
    (:types num)
    (:constants  n1 n2 n3 - num)
    (:predicates (less ?n1 ?n2 - num))

    (:action cmpswap-1-2
     :effect (and (less n1 n2) (not (less n2 n1))
                  (when (less n3 n1)
                        (and (less n3 n2) (not (less n2 n3))))
                  (when (and (less n3 n1) (not (less n3 n2)))
                        (not (less n3 n1)))
                  (when (less n2 n3)
                        (and (less n1 n3) (not (less n3 n1))))
                  (when (and (less n2 n3) (not (less n1 n3)))
                        (not (less n2 n3)))
    ))

    (:action cmpswap-1-3
     :effect (and (less n1 n3) (not (less n3 n1))
                  (when (less n2 n1)
                        (and (less n2 n3) (not (less n3 n2))))
                  (when (and (less n2 n1) (not (less n2 n3)))
                        (not (less n2 n1)))
                  (when (less n3 n2)
                        (and (less n1 n2) (not (less n2 n1))))
                  (when (and (less n3 n2) (not (less n1 n2)))
                        (not (less n3 n2)))
    ))

    (:action cmpswap-2-3
     :effect (and (less n2 n3) (not (less n3 n2))
                  (when (less n1 n2)
                        (and (less n1 n3) (not (less n3 n1))))
                  (when (and (less n1 n2) (not (less n1 n3)))
                        (not (less n1 n2)))
                  (when (less n3 n1)
                        (and (less n2 n1) (not (less n1 n2))))
                  (when (and (less n3 n1) (not (less n2 n1)))
                        (not (less n3 n1)))
    ))
)
```

C.2 Dispose

Excerpt of the PDDL encoding of the Dispose problem presented in Section 7.2 on page 91. This PDDL corresponds to a 3×3 grid with a object in an unknown initial position, that should be disposed at a trash.

```
(define (problem dispose-3-1)
 (:domain dispose)

 (:objects o1 - obj
          p1-1 p1-2 p1-3 p2-1 p2-2 p2-3 p3-1 p3-2 p3-3 - pos)
 (:init
    (and
     (at p2-2)
     (trash-at p1-1)

     (adj p1-1 p2-1)
     (adj p2-1 p1-1)

     (adj p2-1 p3-1)
     (adj p3-1 p2-1)

     ...

     (adj p1-1 p1-2)
     (adj p1-2 p1-1)

     (adj p1-2 p1-3)
     (adj p1-3 p1-2)

     ....

     (oneof
        (obj-at o1 p1-1)
        (obj-at o1 p1-2)
        (obj-at o1 p1-3)
        (obj-at o1 p2-1)
        (obj-at o1 p2-2)
        (obj-at o1 p2-3)
        (obj-at o1 p3-1)
        (obj-at o1 p3-2)
        (obj-at o1 p3-3)
     )

    )
   )
 (:goal (disposed o1)))
```

```
(define (domain dispose)
   (:requirements :typing :conditional-effects)
   (:types pos obj)
   (:predicates (adj ?i ?j) (at ?i) (holding ?o) (obj-at ?o ?i)
                (trash-at ?x) (disposed ?o))
   (:action move
      :parameters (?i - pos ?j - pos)
      :precondition (and (adj ?i ?j) (at ?i))
      :effect (and (not (at ?i)) (at ?j)))
   (:action pickup
      :parameters (?o - obj ?i - pos)
      :precondition (at ?i)
      :effect (when (obj-at ?o ?i)
                    (and (holding ?o) (not (obj-at ?o ?i)))))
   (:action drop
      :parameters (?o - obj ?i - pos)
      :precondition (and (at ?i) (trash-at ?i))
      :effect (when (holding ?o)
                    (and (not (holding ?o)) (disposed ?o)))))
```

C.3 Push To

Excerpt of the PDDL encoding of the Push-to problem presented in Section 7.2 on page 91. This PDDL corresponds to a 3×3 grid with a object in an unknown initial position, that should be grab, but there are only two position where the pickup action can be applied. Instead, it is possible to push an object from one cell to other.

```
(define (problem push-to-3-1)
  (:domain push-to)

  (:objects p1-1 p1-2 p1-3 p2-1 p2-2 p2-3 p3-1 p3-2 p3-3 - pos)
  (:init
    (and
     (at p2-2)
     (pick-loc p1-1)  (pick-loc p3-3)

     (adj p1-1 p2-1)
     (adj p2-1 p1-1)

     (adj p2-1 p3-1)
     (adj p3-1 p2-1)

     ...

     (adj p1-1 p1-2)
     (adj p1-2 p1-1)

     (adj p1-2 p1-3)
     (adj p1-3 p1-2)

     ....

     (oneof
        (obj-at o1 p1-1)
        (obj-at o1 p1-2)
        (obj-at o1 p1-3)
        (obj-at o1 p2-1)
        (obj-at o1 p2-2)
        (obj-at o1 p2-3)
        (obj-at o1 p3-1)
        (obj-at o1 p3-2)
        (obj-at o1 p3-3)
     )

    )
  )
  (:goal (holding o1)))
```

```
(define (domain push-to)
   (:requirements :typing :conditional-effects)
   (:types pos obj)
   (:constants  o1 - obj)
   (:predicates (adj ?i ?j) (at ?i) (holding ?o) (obj-at ?o ?i)
                (pick-loc ?i))
   (:action move
      :parameters (?i -pos ?j - pos)
      :precondition (and (adj ?i ?j) (at ?i))
      :effect (and (not (at ?i)) (at ?j)))
   (:action pickup
      :parameters (?o - obj ?i - pos)
      :precondition (and (at ?i) (pick-loc ?i))
      :effect (when (obj-at ?o ?i)
                    (and (holding ?o) (not (obj-at ?o ?i)))))
   (:action push
      :parameters (?i - pos  ?j - pos)
      :precondition (and (adj ?i ?j) (at ?i))
      :effect (when (obj-at o1 ?i)
                    (and (obj-at o1 ?j) (not (obj-at o1 ?i))))
      ))
```

C.4 1-Dispose

Excerpt of the PDDL encoding of the 1-Dispose problem presented in Section 7.2 on page 91. This PDDL corresponds to a 3×3 grid with a object in an unknown initial position, that should be put at position $(1, 1)$. If the agent is holding an object, and tries to pickup, then the object get lost. Thus, a solution requires to visit each cell of the grid, pick a possible object in it, and go to the position $(1, 1)$ to release such object.

```
(define (problem one-dispose-3-1)
(:domain one-dispose)
   (:objects p1-1 p1-2 p1-3 p2-1 p2-2 p2-3 p3-1 p3-2 p3-3 - pos)
   (:init
    (and
     (handempty)
     (at p2-2)

     (adj p1-1 p2-1)
     (adj p2-1 p1-1)

     (adj p2-1 p3-1)
     (adj p3-1 p2-1)

     ...

     (adj p1-1 p1-2)
     (adj p1-2 p1-1)

     (adj p1-2 p1-3)
     (adj p1-3 p1-2)

     ....

     (oneof
        (obj-at o1 p1-1)
        (obj-at o1 p1-2)
        (obj-at o1 p1-3)
        (obj-at o1 p2-1)
        (obj-at o1 p2-2)
        (obj-at o1 p2-3)
        (obj-at o1 p3-1)
        (obj-at o1 p3-2)
        (obj-at o1 p3-3)
     )

    )
   )
   (:goal (obj-at o1 p1-1))
)
```

```
(define (domain one-dispose)
   (:requirements :typing :conditional-effects)
   (:types pos obj)
   (:constants  o1 - obj)
   (:predicates (adj ?i ?j) (at ?i) (holding ?o)
                (obj-at ?o ?i) (handempty))
   (:action move
      :parameters (?i -pos ?j - pos)
      :precondition (and (adj ?i ?j) (at ?i))
      :effect (and (not (at ?i)) (at ?j)))
   (:action pickup
      :parameters (?p - pos)
      :precondition (at ?p)
      :effect
         (and
            (when (and (handempty) (obj-at o1 ?p))
                  (and (not (handempty)) (holding o1)
                       (not (obj-at o1 ?p))))
            (when (holding o1)
                  (and (handempty) (not (holding o1))))
         ))
   (:action putdown
      :parameters (?p - pos)
      :precondition (at ?p)
      :effect (when (holding o1)
                    (and (handempty) (not (holding o1))
                         (obj-at o1 ?p)))))
```

C.5 Look and Grab

Excerpt of the PDDL encoding of the Look-and-Grab problem presented in Section 7.2 on page 91. This PDDL corresponds to a 4×4 grid with one object in a unknown initial position and where grab actions can pick an object in radios of one cell around the current position of the agent. This domain can be understood as a generalization of the 1-Dispose domain.

Figure 2.3 on page 17 depicts a solution to a 8×8 instance, with one object, found using the conformant planner T_0, presented in Chapter 7.

```
(define (problem look-and-grab-4-1-1-p)
   (:domain look-and-grab-4-1-1)
   (:init
     (and
        (handempty)
        (at p2-2)

        (adj p1-1 p2-1)
        (adj p2-1 p1-1)
        ....
        (adj p1-4 p2-4)
        (adj p2-4 p1-4)

        (adj p2-1 p3-1)
        (adj p3-1 p2-1)
        ....
        (adj p3-4 p4-4)
        (adj p4-4 p3-4)

        (adj p1-1 p1-2)
        (adj p1-2 p1-1)
        ....
        (adj p4-3 p4-4)
        (adj p4-4 p4-3)

        (oneof
           (obj-at o1 p1-1)
           ...
           (obj-at o1 p1-4)
           (obj-at o1 p2-1)
           ...
           (obj-at o1 p4-4)
        )
     )
   )
   (:goal (obj-at o1 p1-1))
)
```

```
(define (domain look-and-grab-4-1-1)
  (:requirements :typing :conditional-effects)
  (:types pos obj)
  (:constants o1 - obj
     p1-1 p1-2 p1-3 p1-4 p2-1 p2-2 p2-3 p2-4 p3-1
     p3-2 p3-3 p3-4 p4-1 p4-2 p4-3 p4-4 - pos)
  (:predicates (adj ?i ?j) (at ?i)
               (holding ?o) (obj-at ?o ?i) (handempty))

  (:action move
    :parameters (?i -pos ?j - pos)
    :precondition (and (adj ?i ?j) (at ?i))
    :effect (and (not (at ?i)) (at ?j)))

  (:action putdown
    :parameters (?p - pos)
    :precondition (at ?p)
    :effect (when (holding o1)
        (and (handempty) (not (holding o1)) (obj-at o1 ?p))))

  (:action pickup-1-1-look-1
    :precondition (at p1-1)
    :effect (and
      (when (holding o1)
          (and (handempty) (not (holding o1))))
       ; X = 1
          ; Y = 1
      (when (and (handempty) (obj-at o1 p1-1))
          (and (not (handempty)) (holding o1)
               (not (obj-at o1 p1-1))))
          ; Y = 2
      (when (and (handempty) (obj-at o1 p1-2))
          (and (not (handempty)) (holding o1)
               (not (obj-at o1 p1-2))))
       ; X = 2
          ; Y = 1
      (when (and (handempty) (obj-at o1 p2-1))
          (and (not (handempty)) (holding o1)
               (not (obj-at o1 p2-1))))
          ; Y = 2
      (when (and (handempty) (obj-at o1 p2-2))
          (and (not (handempty)) (holding o1)
               (not (obj-at o1 p2-2))))
   ))
   ...
```

```
(:action pickup-2-2-look-1
  :precondition (at p2-2)
  :effect (and
     (when (holding o1)
         (and (handempty) (not (holding o1))))
      ; X = 1
         ; Y = 1
     (when (and (handempty) (obj-at o1 p1-1))
         (and (not (handempty)) (holding o1)
              (not (obj-at o1 p1-1))))
         ; Y = 2
     (when (and (handempty) (obj-at o1 p1-2))
         (and (not (handempty)) (holding o1)
              (not (obj-at o1 p1-2))))
         ; Y = 3
     (when (and (handempty) (obj-at o1 p1-3))
         (and (not (handempty)) (holding o1)
              (not (obj-at o1 p1-3))))
      ; X = 2
         ; Y = 1
     (when (and (handempty) (obj-at o1 p2-1))
         (and (not (handempty)) (holding o1)
              (not (obj-at o1 p2-1))))
         ; Y = 2
     (when (and (handempty) (obj-at o1 p2-2))
         (and (not (handempty)) (holding o1)
              (not (obj-at o1 p2-2))))
         ; Y = 3
     (when (and (handempty) (obj-at o1 p2-3))
         (and (not (handempty)) (holding o1)
              (not (obj-at o1 p2-3))))
      ; X = 3
         ; Y = 1
     (when (and (handempty) (obj-at o1 p3-1))
         (and (not (handempty)) (holding o1)
              (not (obj-at o1 p3-1))))
         ; Y = 2
     (when (and (handempty) (obj-at o1 p3-2))
         (and (not (handempty)) (holding o1)
              (not (obj-at o1 p3-2))))
         ; Y = 3
     (when (and (handempty) (obj-at o1 p3-3))
         (and (not (handempty)) (holding o1)
              (not (obj-at o1 p3-3))))
 ))
 ...
```

C.6 Slippery Gripper

PDDL encoding of the non-deterministic domain Slippery Gripper presented in version 8.3 on page 127. This instance is for three balls.

```
(define (problem gripper-3)
   (:domain gripper)
   (:objects rooma roomb1 roomb2 roomc - room
             ball1  ball2  ball3   - ball)
   (:init (and
             (at-robby rooma)
             (free left)
             (free right)
             (det roomb1 roomc)
             (det roomc roomb1)

             (det roomb2 roomc)
             (det roomc roomb2)

             (non-det rooma roomb1 roomb2)

             (det roomb1 rooma)
             (det roomb2 rooma)

             (at ball1 rooma)
             (at ball2 rooma)
             (at ball3 rooma)
   ))
   (:goal (and
             (at ball1 roomc)
             (at ball2 roomc)
             (at ball3 roomc)
   ))
)
```

```
(define (domain gripper)
   (:requirements :typing :conditional-effects)
   (:types room ball gripper)
   (:constants left right - gripper)
   (:predicates (at-robby ?r - room)
                (det ?r1 ?r2 - room)
                (non-det ?from ?r1 ?r2 - room)
                (at ?b - ball ?r - room)
                (free ?g - gripper)
                (carry ?o - ball ?g - gripper))

   (:action move-d
       :parameters   (?from ?to - room)
       :precondition (det ?from ?to)
       :effect (when (at-robby ?from) (and (at-robby ?to)
                     (not (at-robby ?from)))))

   (:action move-nd
       :parameters  (?from ?to1 ?to2 - room)
       :precondition (non-det ?from ?to1 ?to2)
       :effect (when (at-robby ?from)
                     (and (oneof (at-robby ?to1) (at-robby ?to2))
                          (not (at-robby ?from)))))



   (:action pick
       :parameters (?obj - ball ?room - room ?gripper - gripper)
       :effect (when (and (at ?obj ?room) (at-robby ?room)
                          (free ?gripper))
                     (and (carry ?obj ?gripper)
                          (not (at ?obj ?room))
                          (not (free ?gripper)))))

   (:action drop
       :parameters  (?obj - ball ?room - room ?gripper - gripper)
       :effect (when (and (carry ?obj ?gripper) (at-robby ?room))
                     (and (at ?obj ?room)
                          (free ?gripper)
                          (not (carry ?obj ?gripper)))))
)
```

Bibliography

Each reference indicates the pages where it appears.

Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA, 1998. AAAI, AAAI Press / The MIT Press. AAAI-98. 186, 189

Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA, 2000. AAAI, AAAI Press / The MIT Press. AAAI-2000. 185, 186

AAAI-97. *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97, July 27-31, 1997, Providence, Rhode Island, USA*, 1997. AAAI Press / The MIT Press. 183, 184

Alexandre Albore, Héctor Palacios, and Héctor Geffner. A translation-based approach to contingent planning. In *Proc. 21st Int. Conference on AI – IJCAI*, pages 1623–1628, 2009. ix, 14, 150

Carlos Ansótegui, Carla P. Gomes, and Bart Selman. The achilles' heel of QBF. In Veloso and Kambhampati (2005), pages 275–281. AAAI-2005. 135

Fahiem Bacchus. The 2000 AI Planning Systems Competition. *Artificial Intelligence Magazine*, 22(3), 2001. 7

Jorge A. Baier, Fahiem Bacchus, and Sheila A. McIlraith. A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence*, 173(5-6): 593–618, 2009. 103

Chitta Baral and Tran Cao Son. Approximate reasoning about actions in presence of sensing and incomplete information. In *Proc. ILPS 1997*, pages 387–401, 1997. 23, 64, 66, 120, 133, 141

Jon Barwise, editor. *Handbook of Mathematical Logic*. North-Holland, 1977. 36

Jon Barwise and John Etchemendy. *The Language of First-Order Logic*. CSLI, Stanford, 1991. 121, 127

Roberto J. Bayardo and Robert C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In AAAI-97 AAAI-97, pages 203–208. 90

Daniel Le Berre and Olivier Roussel. SAT competition. `http://www.satcompetition.org`, 2009. 103

Piergiorgio Bertoli, Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Heuristic search + symbolic model checking = efficient conformant planning. In *Proc. IJCAI-01*, pages 467–472, 2001. 21, 22

Piergiorgio Bertoli, Marco Pistore, and Paolo Traverso. Automated web service composition by on-the-fly belief space search. In *Proc. ICAPS-2006*, pages 358–361, 2006. 14

Armin Biere. Resolve and expand. In *Proc. 7th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'04)*, volume LNCS 3542. Springer, 2005. 134

Armin Biere. Quantor home site. `http://fmv.jku.at/quantor`, 2008. 136

Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. In *Proceedings of IJCAI-95*, pages 1636–1642. Morgan Kaufmann, 1995. 4

Blai Bonet and Hector Geffner. Planning as heuristic search. *Artificial Intelligence*, 129(1–2):5–33, 2001a. 4, 10, 12, 21

Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. In *Proc. of AIPS-2000*, pages 52–61. AAAI Press, 2000. 19, 20, 21, 28, 48, 63, 64, 85, 139

Blai Bonet and Hector Geffner. Gpt: A tool for planning with uncertainty and partial information. In *Proc. IJCAI Workshop on Planning with Uncertainty and Partial Information*, 2001b. 144

Blai Bonet and Hector Geffner. Planning as heuristic search: New results. In *Proceedings of ECP-99*, pages 359–371. Springer, 1999. 10, 164, 165

Blai Bonet and Bob Givan. Results of the conformant track of the 5th int. planning competition. `http://www.ldc.usb.ve/~bonet/ipc5/docs/results-conformant.pdf`, 2006. 76, 89, 98, 107, 109, 124, 126

Blai Bonet, Gabor Loerincs, and Hector Geffner. A robust and fast action selection mechanism for planning. In AAAI-97 AAAI-97, pages 714–719. 10

Blai Bonet, Hector Palacios, and Hector Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proc. ICAPS-09*, page to appear, 2009. ix, 150

Ronen Brafman and Jörg Hoffmann. Conformant planning via heuristic forward search: A new approach. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04)*, 2004. 48, 59, 124

Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992. 21, 22, 28, 36, 52

Daniel Bryce and Oliver Buffet. International planning competition uncertainty part: Benchmarks and results. `http://ippc--2008.loria.-fr/wiki-/images/0/03-/Results.pdf`, 2008. 23, 89, 98, 100

Daniel Bryce and Seungchan Kim. Planning for gene regulatory network intervention. In *Proc. IJCAI-2007*, pages 1834–1839, 2007. 14

Daniel Bryce, Subbarao Kambhampati, and David E. Smith. Planning graph heuristics for belief space search. *Journal of Artificial Intelligence Research*, 26:35–99, 2006. 20, 22, 28, 76, 94

Tom Bylander. The computational complexity of STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994. 7

Marco Cadoli and Francesco Donini. A survey on knowledge compilation. *AI Communications*, 10(3-4):137–150, 1997. 36

Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008. ISSN 0004-3702. 108, 112

Yixin Chen, Benjamin W. Wah, and Chih-Wei Hsu. Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of Artificial Intelligence Research*, 26:323–369, 2006. 10, 103

Alessandro Cimatti and Marco Roveri. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000. 20, 22, 28, 76, 94, 139

Alessandro Cimatti, Marco Roveri, and Piergiorgio Bertoli. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence*, 159:127–206, 2004. 15, 20, 22, 28, 45, 48, 91, 94

Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 2000. 36

Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms*. MIT Press and McGraw-Hill, 1990. 10

Sylvie Coste-Marquis, Daniel Le Berre, Florian Letombe, and Pierre Marquis. Propositional fragments for knowledge compilation and quantified boolean formulae. In Veloso and Kambhampati (2005), pages 288–293. AAAI-2005. 137

Adnan Darwiche. On the tractable counting of theory models and its application to truth maintenance and belief revision. *Journal of Applied Non-Classical Logics*, 11(1-2):11–34, 2001a. x, 23, 29, 33, 37, 38, 40, 43, 44, 112

Adnan Darwiche. A differential approach to inference in bayesian networks. *Journal of the ACM*, 50(3):280–305, 2003. ISSN 0004-5411. 108, 112

Adnan Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4): 608–647, 2001b. 36, 38, 39, 145

Adnan Darwiche. New advances in compiling cnf into decomposable negation normal form. In *Proc. ECAI 2004*, pages 328–332, 2004. 39, 40, 57

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002. 28, 33, 34, 37, 39, 40, 51, 52, 55, 137

Adnan Darwiche and Pierre Marquis. Compiling propositional weighted bases. *Artificial Intelligence*, 157(1-2):81–113, 2004. ISSN 0004-3702. 112

Professor Adnan Darwiche. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press, New York, NY, USA, 2009. 108, 112

Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960. 144

Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962. 11, 134, 144

Rina Dechter. *Constraint Processing*. Morgan Kaufmann, 2003. 81, 134

Rina Dechter and Robert Mateescu. And/or search spaces for graphical models. *Artificial Intelligence*, 171(2-3):73–106, 2007. ISSN 0004-3702. 40

Rina Dechter and Irina Rish. Directional resolution: The Davis-Putnam procedure, revisited. In Jon Doyle, Erik Sandewall, and Pietro Torasso, editors, *KR'94: Principles of Knowledge Representation and Reasoning*, pages 134–145, San Francisco, California, 1994. Morgan Kaufmann. 144

Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. A logic programming approach to knowledge-state planning, ii: the dlvk system. *Artificial Intelligence*, 144(1-2):157–211, 2003. ISSN 0004-3702. 23, 137

Paolo Ferraris and Enrico Giunchiglia. Planning as satisfiability in nondeterministic domains. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA* AAA (2000), pages 748–753. AAAI-2000. 52, 137

Richard Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1:27–120, 1971. 5, 7, 12

Alberto Finzi, Fiora Pirri, and Ray Reiter. Open world planning in the situation calculus. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA* AAA (2000), pages 754–760. AAAI-2000. 23

Dieter Fox and Carla P. Gomes, editors. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, 2008. AAAI Press. AAAI-2008. 187, 189

Maria Fox and Derek Long. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20:1–59, 2003. 7

Alfonso E. Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners. *Artificial Intelligence*, 173 (5-6):619–668, 2009. ISSN 0004-3702. 7, 118

Enrico Giunchiglia, Alessandro Massarotto, and Roberto Sebastiani. Act, and the rest will follow: Exploiting determinism in planning as satisfiability. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA* AAA (1998), pages 948–953. AAAI-98. 29

Fausto Giunchiglia and Paolo Traverso. Planning as model checking. In *Proceedings of ECP-99*. Springer, 1999. 36

Robert P. Goldman and Mark S. Boddy. Expressive planning and explicit knowledge. In *Proc. AIPS-1996*, 1996. 13, 28

Georg Gottlob, Reinhard Pichler, and Fang Wei, editors. *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, 2007. AAAI Press. AAAI-2007. 187, 188

Eric Hansen and Shlomo Zilberstein. Lao*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence*, 129:35–62, 2001. 144

Patrik Haslum and Peter Jonsson. Some results on the complexity of planning with incomplete information. In *Proc. 5th European Conf. on Planning (ECP-99), Lect. Notes in AI VOl 1809*, pages 308–318. Springer, 1999. 20, 23, 28

Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006. 10, 103

Malte Helmert, Minh B. Do, and Ioannis Refanidis. Sixth international planning competition. `http://ipc.informatik.uni-freiburg.de`, 2008. 7

Joerg Hoffmann and Héctor Geffner. Branching matters: Alternative branching in graphplan. In *13th Int. Conf. on Automated Planning and Scheduling (ICAPS-2003)*, 2003. 21

Jörg Hoffmann and Ronen Brafman. Conformant planning via heuristic forward search: A new approach. *Artificial Intelligence*, 170(6-7):507–541, 2006. 20, 22, 23, 76, 91, 94, 118, 125, 141

Jörg Hoffmann and Ronen Brafman. Contingent planning via heuristic forward search with implicit belief states. In *Proc. 15th Int. Conf. on Automated Planning and Scheduling (ICAPS 2005)*, pages 71–80. AAAI, 2005. 14

Jörg Hoffmann and Stefan Edelkamp. The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research*, 24:519–579, 2005. 7

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001. 10, 21, 22, 76, 126, 150

Jörg Hoffmann, Piergiorgio Bertoli, and Marco Pistore. Web service composition as planning, revisited: In between background theories and initial state uncertainty. In Gottlob et al. (2007), pages 1013–1018. AAAI-2007. 14

Jörg Hoffmann, Piergiorgio Bertoli, Malte Helmert, and Marco Pistore. Message-based web service composition, integrity constraints, and planning under uncertainty: A new connection. *Journal of Artificial Intelligence Research*, 35:49–117, 2009. 14

Jinbo Huang. Combining knowledge compilation and search for conformant probabilistic planning. In *Proc. of ICAPS-2006*, pages 253–262, 2006. 23, 107, 108, 111, 112, 114, 115, 116, 117, 118, 136, 144, 145

Jinbo Huang and Adnan Darwiche. DPLL with a trace: From SAT to knowledge compilation. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30-August 5, 2005*, pages 156–162. Professional Book Center, 2005. IJCAI-2005. 137

Jinbo Huang and Adnan Darwiche. Using DPLL for efficient OBDD construction. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing*, pages 127–136, 2004. 39, 137

Nathanael Hyafil and Fahiem Bacchus. Conformant probabilistic planning via csps. In Enrico Giunchiglia, Nicola Muscettola, and Dana S. Nau, editors, *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), June 9-13, 2003, Trento, Italy*, pages 205–214. AAAI, 2003. AIPS-2003. 23, 107, 144

Henry Kautz and Bart Selman. Planning as satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363, 1992. 10

Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, August 4-8, 1996, Portland, Oregon, USA*, pages 1194–1201. AAAI, AAAI Press / The MIT Press, 1996. AAAI-96. 4, 10, 11, 20, 21, 29, 139

Donald Knuth. *The Art of Computer Programming, Vol. III: Sorting and Searching.* Addison-Wesley, 1973. 45

James Kurien, P. Pandurang Nayak, and David E. Smith. Fragment-based conformant planning. In *Proc. AIPS-2002*, pages 153–162. AAAI Press, 2002. 137

Nicholas Kushmerick, Steve Hanks, and Daniel Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 76:239–286, 1995. 107, 144

Jérôme Lang, Paolo Liberatore, and Pierre Marquis. Propositional independence: Formula-variable independence and forgetting. *Journal of Artificial Intelligence Research*, 18:391–443, 2003. 34

Vladimir Lifschitz. What is answer set programming? In Fox and Gomes (2008), pages 1594–1597. AAAI-2008. 137

Fangzhen Lin and Ray Reiter. Forget it! In *Working Notes, AAAI Fall Symposium on Relevance*, pages 154–159. American Association for Artificial Intelligence, 1994. 34, 38, 54

Iain Little and Silvie Thiébaux. Probabilistic planning vs replanning. In *Workshop on International Planning Competition: Past, Present and Future (ICAPS)*, 2007. 144

Michael L. Littman, Judy Goldsmith, and Martin Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9:1–36, 1998. 20

Stephen M. Majercik and Michael L. Littman. Maxplan: A new approach to probabilistic planning. In *Proc. AIPS-98*, pages 86–93, 1998. 23, 107, 108, 109, 111, 144

Pierre Marquis. Consequence finding algorithms. In D. Gabbay and Ph. Smets, editors, *Handbook on Defeasible Reasoning and Uncertainty Management Systems*, volume 5, pages 41–145. Kluwer, 2000. 77

Kim Marriot and Peter Stuckey. *Programming with Constraints.* MIT Press, 1999. 115

Drew McDermott. Estimated-regression planning for interactions with web services. In Gottlob et al. (2007), pages 204–211. AAAI-2007. 14

Drew McDermott. The 1998 AI Planning Systems Competition. *Artificial Intelligence Magazine*, 21(2):35–56, 2000. 7

Drew McDermott. A heuristic estimator for means-ends analysis in planning. In *Proc. Third Int. Conf. on AI Planning Systems (AIPS-96)*, 1996. 10

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL – the planning domain definition language. Technical report, Yale Center for Computational Vision and Control, New Haven, Connecticut, 1998. Technical Report CVC TR-98-003. `http://ftp.cs.yale.edu/pub/mcdermott`. 7

Kenneth L. McMillan. Hiearchical representation of discrete functions with applications to model checking. In *Lecture Notes In Computer Science*, volume 818, pages 41–54, 1994. Proc. 6th Int. Conf. on Computer Aided Verification. 40

George H Mealy. A method for synthesizing sequential circuits. *Bell Systems Technical Journal*, 34:1045–1079, 1955. 150

A. Ricardo Morales, Phan Huy Tu, and Tran Cao Son. An extension to conformant planning using logic programming. In *Proc. IJCAI-2007*, pages 1991–1996, 2007. 23

Bernhard Nebel. On the compilability and expressive power of propositional planning. *Journal of Artificial Intelligence Research*, 12:271–315, 2000. 5, 138

Nilufer Onder, Garrett C. Whelan, and Li Li. Engineering a conformant probabilistic planner. *Journal of Artificial Intelligence Research*, 25:1–15, 2006. 144

Hector Palacios. Instances submitted to the QBF evaluation. `http://www.qbflib.org/family_detail.php?idFamily=707`, 2007,2008. 136

Héctor Palacios and Hector Geffner. Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes). In *AAAI*, pages 900–905. AAAI, AAAI Press, 2006a. AAAI-2006. 63, 67, 71, 76, 84, 91, 94, 107, 119, 125, 148

Hector Palacios and Hector Geffner. Mapping conformant planning to SAT through compilation and projection. In *Current Topics in Artificial Intelligence*, volume 4177, pages 311–320, Berlin, Germany, 2006b. Springer Berlin / Heidelberg. Selected Papers from the 11th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2005). 52, 138, 148

Hector Palacios and Hector Geffner. From conformant into classical planning: Efficient translations that may be complete too. In *Proc. ICAPS-07*, pages 264–271, 2007. 67, 69, 77, 84, 91, 94, 125, 148

Hector Palacios and Hector Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research*, 35:623–675, 2009. 67, 69, 89, 98, 148, 149

Hector Palacios, Blai Bonet, Adnan Darwiche, and Hector Geffner. Pruning conformant plans by counting models on compiled d-DNNF representations. In *Proc. of the 15th Int. Conf. on Planning and Scheduling (ICAPS-05)*, pages 141–150. AAAI Press, 2005. 28, 46, 48, 52, 148

Christos Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994. 20

James D. Park and Adnan Darwiche. Complexity results and approximation strategies for map explanations. *Journal of Artificial Intelligence Research*, 21:101–133, 2004. 112

Judea Pearl. *Heuristics.* Addison Wesley, 1983. 10

Ronald Petrick and Fahiem Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. AIPS'02*, pages 212–221, 2002. 22, 63, 64, 70, 120, 141

Marco Pistore, Fabio Barbon, Piergiorgio Bertoli, Dmitry Shaparau, and Paolo Traverso. Planning and monitoring web service composition. In *Planning and monitoring web service composition*, pages 106–115, 2004. 14

Ray Reiter and Johan de Kleer. Foundations of assumption-based truth maintenance systems: a preliminary report. In *Proceedings of the 6th National Conference on Artificial Intelligence. July 1987, Seattle, WA, USA*, pages 183–188. Morgan Kaufmann, 1987. AAAI-87. 36

Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks revisited. In Fox and Gomes (2008), pages 975–982. AAAI-2008. 10, 86, 150

Jussi Rintanen. Distance estimates for planning in the discrete belief space. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pages 525–530. AAAI Press / The MIT Press, 2004a. AAAI-2004. 28, 45, 46, 48, 52, 57, 58

Jussi Rintanen. Complexity of planning with partial observability. In *Proc. ICAPS-2004*, pages 345–354, 2004b. 20, 23, 28

Jussi Rintanen. Expressive equivalence of formalisms for planning with sensing. In

Proc. ICAPS-2003, pages 185–194, 2003. 12, 108

Jussi Rintanen. Constructing conditional plans by a theorem-prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999. 44, 135

Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as satisfiability: Parallel plans and algorithms for plan search. *Artificial Intelligence*, 180:2006, 2005. 12

Bart Selman and Henry Kautz. Knowledge compilation and theory approximation. *Journal of the ACM*, 43(2):193–224, 1996. 36

Carsten Sinz and Himanshu Jain. SAT-race 2008 – AIG special track. `http://baldur.iti.uka.de/sat-race-2008`, 2008. 60

David E. Smith and Daniel Weld. Conformant graphplan. In *AAAI-98* AAA (1998), pages 889–896. AAAI-98. 13, 28, 107, 138, 139

Tran Cao Son and Phan Huy Tu. On the completeness of approximation based reasoning and planning in action theories with incomplete information. In *Proc. 10th Int. Conf. on Principles of KR and Reasoning (KR-06)*, pages 481–491, 2006. 23, 77, 142, 143

Tran Cao Son, Phan Huy Tu, Michael Gelfond, and A. Ricardo Morales. An approximation of action theories of and its application to conformant planning. In Chitta Baral, Gianluigi Greco, Nicola Leone, and Giorgio Terracina, editors, *Logic Programming and Nonmonotonic Reasoning, 8th International Conference, LPNMR 2005, Diamante, Italy, September 5-8, 2005, Proceedings*, volume 3662 of *Lecture Notes in Computer Science*, pages 172–184. Springer, 2005a. LPNMR-2005. 23

Tran Cao Son, Phan Huy Tu, and A. Ricardo Morales Michael Gelfond. Conformant planning for domains with constraints-a new approach. In Veloso and Kambhampati (2005), pages 1211–1216. AAAI-2005. 64, 142

Christian Thiffault, Fahiem Bacchus, and Toby Walsh. Solving non-clausal formulas with DPLL search. In *Proc. of SAT*, 2004. `http://www.satisfiability.org/SAT04/programme/63.pdf`. 60

Pierre Tison. Generalized consensus theory and applications to the minimization of boolean circuits. *IEEE Transactions on Computers*, EC-16(4):446–456, 1967. 91

Dang-Vien Tran, Hoang-Khoi Nguyen, Enrico Pontelli, and Tran Cao Son. Improving performance of conformant planners: Static analysis of declarative planning domain specifications. In *Proc. PADL-2009*, pages 239–253, 2009. 20, 23, 64, 100, 140, 143

Hudson Turner. Polynomial-length planning spans the polynomial hierarchy. In *JELIA '02: Proc. of the European Conference on Logics in AI*, pages 111–124. Springer-Verlag, 2002. 20, 145

Manuela M. Veloso and Subbarao Kambhampati, editors. *Proceedings of The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, 2005. AAAI Press / The MIT Press. AAAI-2005. 183, 185, 190

Benjamin W. Wah and Yixin Chen. Constraint partitioning in penalty formulations for solving temporal planning problems. *Artificial Intelligence*, 170(3):187–231, 2006. ISSN 0004-3702. 10, 103

Sung Wook Yoon, Alan Fern, and Robert Givan. Ff-replan: A baseline for probabilistic planning. In Mark S. Boddy, Maria Fox, and Sylvie Thiébaux, editors, *Proceedings of the Seventeenth International Conference on Automated Planning* *and Scheduling, ICAPS 2007, Providence, Rhode Island, USA, September 22-26, 2007*, pages 352–. AAAI, 2007. AIPS-2007. 103, 144

Håkan L. S. Younes and Michael L. Littman. PPDDL1.0: The language for the probabilistic part of IPC-4. In *In Proc. International Planning Competition*, 2004. 109