SMT TECHNIQUES FOR PLANNING PROBLEMS

Joan Espasa Arxer

Universitat de Girona

DOCTORAL THESIS

SMT Techniques for Planning Problems

Author:
Joan Espasa Arxer

2018

Universitat de Girona

DOCTORAL THESIS

SMT Techniques for Planning Problems

Author:
Joan Espasa Arxer

2018

PhD Program in Technology

Supervisors:
Dr. Miquel Bofill Arasa
Dr. Mateu Villaret Auselle

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy by the University of Girona

Publications

Publications derived from this thesis:

- Journals
  - Miquel Bofill, Joan Espasa, and Mateu Villaret, *The RANTANPLAN Planner: System Description.* Knowledge Engineering Review (KER) vol. 31, no. 5, 2016, 452-464.

    JCR: Q3, position 79/133, *Computer Science, Artificial Intelligence*
- Conference Proceedings
  - Miquel Bofill, Joan Espasa and Mateu Villaret, *A Semantic Notion of Interference for Planning Modulo Theories.* International Conference on Automated Planning and Scheduling (ICAPS), 2016.

    CORE: A*, GSS Conference Rating: A
  - Miquel Bofill, Joan Espasa and Mateu Villaret, *Relaxed Exists-Step Plans in Planning as SMT.* International Joint Conference on Artificial Intelligence (IJCAI), 2017.

    CORE: A*, GSS Conference Rating: A++
- Workshops
  - Miquel Bofill, Joan Espasa and Mateu Villaret, *Efficient SMT Encodings for the Petrobras Domain.* Proceedings of the 13th International Workshop on Constraint Modelling and Reformulation (ModRef), 2014.
  - Miquel Bofill, Joan Espasa and Mateu Villaret, *The RANTANPLAN Planner: System Description.* Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS), 2015.

List of Figures

List of Tables

List of Algorithms

Acknowledgments

First at all I would like to express my gratitude to the $L \wedge P$ Research Group for accepting me as a PhD student. Mateu Villaret, Miquel Bofill, Jordi Coll, Pep Suy, and Miquel Palahí have not only become colleagues, but also and more importantly, friends. During these years next to them, I can confidently say that I have grown academically and personally speaking.

This thesis has only been possible thanks to the many thoughtful discussions, reviews and collaboration that we had together, specially with my supervisors Mateu Villaret and Miquel Bofill. I hope that in the future we will be able to continue working together while we share another cup of tea.

I would also like to express my gratitude to my family, friends and specially Yolanda, for their near-infinite reserve of patience during these years. Oh, and last but not least, my two cats: Leia and Tama, because without their unconditional support and patience I would have not been able to stand where I am.

The work presented in this thesis has been partially supported by the Spanish Ministry of Science and Innovation through projects HeLo (ref. TIN2012-33042) and LOCOS (ref. TIN2015-66293-R), and by the Universitat de Girona under a BR2013 grant.

Contents

Abstracts

Abstract

Automated planning is a discipline in the field of Artificial Intelligence that can be described as the process of finding a course of action that achieves a specified task. In other words, it focuses on reasoning about causal structures and identifying the necessary actions for achieving a given goal.

Although classical planning approaches have been widely successful, the needs of real-world applications go way beyond its potential. In the area of automated planning many formalisms exist in order to express all the needs these problems encompass.

This huge variety of problems range from classical planning to reasoning about partially observable Markov decision processes, multi-agent planning, real-time perceiving and acting or temporal and numeric reasoning. There exist a wide range of techniques to confront each of the aforementioned formalisms, each one having its own advantages and weaknesses. In this thesis we restrict ourselves to the setting of hybrid planning. That is, the combination of the propositional planning with extensions to be able to reason about different theories, such as integer or real arithmetic.

This thesis presents a set of techniques to efficiently encode planning problems that involve reasoning at propositional level as well as to deal with background theories. To address reasoning about the different theories, we use SAT Modulo Theories (SMT), an extension to SAT that allows the solver to, in a modular way, reason about non-propositional symbols belonging to background theories. This framework is interesting because it is expressive enough to translate many real-world planning problems.

The main objective of the thesis is to push forward the state of the art of planning as SMT, by devising encodings of planning problems to SMT. The focus is especially on numeric planning, combining classical planning with the ability to reason about integer or floating point numbers. In this setting, many real-world resource-based problems can be encoded.

Our implementation of the encodings resulted in a new planner called Rantanplan, which preprocesses and translates numeric planning problems into SMT formulas, to solve them using a SMT solver of choice.

We also provide detailed experimental results on new and well-known domains, to show that our approach is competitive with the existing exact numeric planners.

Resum

La planificació automàtica és una disciplina dins de la intel·ligència artificial que pot ser descrita com el procés de trobar un seguit d'accions que assoleixen una tasca específica. En altres paraules, es focalitza en raonar sobre estructures causals i identificar les accions necessàries per assolir un objectiu donat. Encara que les aproximacions a la planificació automàtica clàssica han tingut un gran èxit, les necessitats que tenen moltes aplicacions al món real estan per sobre de les seves possibilitats.

Existeixen molts formalismes a l'àrea de la planificació automàtica que poden expressar totes les necessitats que tenen aquest tipus de problemes. Aquesta enorme varietat de problemes van des de la planificació clàssica, passant per problemes expressats amb processos de decisió de Markov parcialment observables, problemes de percepció i decisió en temps real o problemes que incorporen raonament temporal i numèric. Existeixen un ampli ventall de tècniques per a afrontar cada un dels formalismes esmentats, cada una amb els seus avantatges i inconvenients. En aquesta tesi ens restringim en el marc de la planificació híbrida. Exactament, la combinació de la planificació proposicional amb extensions per a poder raonar sobre diferents teories, tals com l'aritmètica real o entera.

Aquesta tesi presenta un conjunt de tècniques per a codificar de manera eficient problemes de planificació que involucren raonament a nivell proposicional així com raonament amb teories de fons. Per abordar el raonament sobre les diferents teories, farem anar SAT Modulo Teories (SMT), una extensió de SAT que permet al resoledor, de manera modular, raonar sobre símbols no proposicionals pertanyents a teories de fons. Aquest marc és interessant perquè és prou expressiu per a poder traduir molts problemes provinents del món real.

L'objectiu principal és millorar l'estat de l'art de la planificació automàtica mitjançant SMT, a través de la codificació dels problemes de planificació a SMT. El focus de la tesi és especialment en la planificació numèrica, on es combina la planificació clàssica amb l'habilitat de raonar sobre nombres enters o reals. En aquest context es poden codificar molts problemes reals amb restriccions sobre recursos.

La nostra implementació de les codificacions ha donat fruit a un planificador anomenat RANTANPLAN, el qual preprocessa i tradueix problemes de planificació numèrics cap a fórmules SMT, finalment resolent-los amb el resoledor SMT que l'usuari triï.

També s'inclouen resultats detallats d'alguns dominis ben coneguts i alguns de nous, per a demostrar que el nostre enfocament és competitiu amb els planificadors numèrics exactes existents.

Resumen

La planificación automática es una disciplina dentro de la inteligencia artificial que puede ser descrita como el proceso de encontrar una serie de acciones que logren un objetivo. En otras palabras, se focaliza en razonar sobre estructuras causales e identificar las acciones necesarias para alcanzar un fin. Aunque las aproximaciones a la planificación automática clásica son muy potentes, las necesidades que tienen muchas aplicaciones del mundo real están por encima de sus posibilidades.

Existen muchos formalismos en el área de la planificación automática que pueden expresar todas las necesidades que tienen este tipo de problemas. Esta enorme variedad de problemas va des de la planificación clásica, pasando por problemas de decisión parcialmente observables, problemas de percepción y decisión en tiempo real o problemas que incorporan razonamiento temporal y numérico. Existe una amplia selección de técnicas para afrontar cada uno de los formalismos mencionados, cada una con sus ventajas e inconvenientes. En esta tesis nos restringimos al marco de la planificación híbrida. Exactamente, en la combinación de la planificación proposicional con extensiones para poder razonar sobre diferentes teorías, tales como la aritmética real o entera.

Esta tesis presenta un conjunto de técnicas para codificar de manera eficiente problemas de planificación que involucren tanto razonamiento a nivel proposicional como razonamiento con teorías de fondo. Para abordar el razonamiento sobre las diferentes teorías, usaremos SAT Modulo Teorías (SMT), una extensión de SAT que permite al solucionador, de manera modular, razonar sobre símbolos no proposicionales pertenecientes a teorías de fondo. Este marco es interesante porqué tiene suficiente poder expresivo para traducir muchos problemas procedentes del mundo real.

El objetivo principal es el de mejorar el estado del arte de la planificación automática mediante SMT, a través de la codificación de los problemas de planificación a SMT. El foco de la tesis est puesto especialmente en la planificación numérica, donde se combina la planificación clásica con la posibilidad de razonar sobre números enteros o reales. En este contexto se pueden expresar muchos problemas reales con restricciones de recursos.

Nuestra implementación de las codificaciones ha dado fruto a un planificador llamado Rantanplan, el cual procesa y traduce problemas de planificación numéricos a fórmulas SMT, finalmente resolviéndolos con un solucionador SMT que el usuario escoja.

También incluimos resultados detallados de algunos dominios bien conocidos y de algunos de nuevos, para demostrar que nuestro enfoque es competitivo con los planificadores numéricos exactos existentes.

Chapter 1

Introduction

1.1 Motivation

The satisfiability problem (SAT) can be defined as the decision problem for propositional formulas. That is, to be able to say if there exists an assignment to the variables of the formula that satisfies it.

During the last 20 years, there has been a dramatic improvement on the efficiency of SAT solvers. SAT solvers went from being able to comfortably solve problems with tens of variables and hundreds of constraints in the early 90's, to nowadays solve problems with a million of variables and some millions of constraints. Although Moore's Law helped in this regard, a twice as fast computer does not mean that it can solve a twice as large instance, because the search difficulty does not necessarily scale with the problem size. The constant improvement of SAT solvers, driven by the regular SAT competitions, has been the key to the success of many SAT-based applications. Therefore, SAT solvers transitioned from being only an academically interesting problem, to a well-known technology used in the industry. Some of the uses of SAT solvers are, for example, doing model checking for critical systems [BCCZ99], combinatorial design [Zha09] or solving scheduling problems [BEG^{+}14].

Satisfiability Modulo Theories (SMT) can be defined as the decision problem for logical formulas with respect to combination of background theories, including equality. Solvers for this problem are continuously improving and nowadays are very efficient. This is in part due to the improvement of SAT solvers, which are the basis of most of the SMT solvers.

Automated planning is an area of Artificial Intelligence (AI) that aims to automate the decision of what actions an agent should perform to achieve a given objective. This area is not only important for autonomous systems, which need deliberation capabilities to be truly autonomous; but also to help humans to plan complex problems.

The first motivation for research in automated planning is theoretical: planning is an important component of rational behaviour. In our everyday activities we continuously act, anticipating the outcomes of our actions, even if we are not fully aware of it. The prefrontal cortex of our brain is responsible, amongst other things, for these areas of planning and decision-making. So, if the purpose of AI is to represent some aspects of intelligence, then certainly planning would be a key component of that purpose.

The other motivation is entirely practical: Many different fields need tools that can give the ability to the users to face complex and changing tasks in an efficient way. Satellites and other kind of spacecraft are an example, where these autonomous agents need to reason about their limited resources and their current and future goals. Some other recent uses of planning research in the industry are, to name a few: reconfigurable manufacturing systems [PLF$^+$17], planning for power grid operations [BCC$^+$09], automated personalized group tours [LCLK16] or optimizing warehouse operations [Hüt16].

Some areas in the field of automated planning are extensively explored and raise a lot of interest, giving many mature techniques to confront interesting problems. However, there are other areas, like automated planning with numeric reasoning, where further efforts are needed if the planning community wants to provide useful approaches to interesting problems. As a matter of fact, it is especially noticeable the lack of domains that require numeric reasoning in the international planning competition (IPC). This competition is held in the context of the International Conference on Planning and Scheduling Conference (ICAPS). It empirically evaluates state-of-the-art planning systems on a number of benchmark problems.

Classical planning as propositional satisfiability has shown to be, by many authors, a competitive approach to solve classical planning problems [BF97, KSH06, Rin12b]. An important improvement that made planning as propositional satisfiability competitive with state of the art planners was the introduction of parallel plans. Sequential plans have one action per step , while parallel plans can have multiple actions per step. Either way, the formula always grow linearly with time. Although called parallel, the main motivation of these parallel plans was that they could compactly represent intermediate states of sequential plans. This reduction of explicitly represented states lead to smaller formulas with less variables, and sometimes to more easily solvable problems. Therefore, the focus on this thesis is to give a response to the following question: Can Satisfiability Modulo Theories be a good technique to translate and solve complex planning problems, as SAT solvers has proven to be for classical planning problems?

1.2 Objectives

- The main objective of the thesis is to develop a non-heuristic planner that can solve numeric planning problems. The main approach will be using a SMT solver as a black box.
- SMT has many theories available, where each one has its own expressivity and strengths. For translating planning problems to SMT, intelligent and compact encodings will have to be developed in one or many theories.
- Classical planning as SAT flourished partly thanks to the relaxed parallel semantics, enabling many actions to be taken in parallel. These semantics may not be enough for numeric planning, as classical planning as SAT only deals with Boolean variables. Therefore, any notion on these semantics that involves reasoning with numerical variables should be revisited. The objective is to attain finer approaches, to be able to deal with numeric variables more effectively.
- Our final objective is to be competitive with the state of the art non-heuristic numeric planners.

1.3 Outline of the Thesis

Chapter 2 presents an overview of the automated planning area, reviewing classical planning, its representations, basic definitions and some of the approaches about how to attack the problem.

In Chapters 3 and 4 the approach of planning as satisfiability is explained, together with the ideas of how numeric reasoning is integrated with classical planning and the background needed to understand how numeric planning as satisfiability can be solved.

In Chapter 5, the definition of planning modulo theories is given, together with an encoding for translating planning problems to SMT formulas, with support for various parallelism semantics. Experimental results are presented for this encoding, showing its usefulness. These contributions correspond to various of the aforementioned articles: [BEV14, BEV15, BEV16b]

Chapter 6 presents the formalization of a permissive notion of interference between actions, its correctness for two parallelism semantics and how this interference can be easily checked using SMT. Finally, experiments show the increased parallelism and efficiency of using this new notion. The contributions of this Chapter are published in [BEV16a].

Chapter 7 presents a different approach to the notion of parallelism in the setting of planning as SMT. A new encoding that does not use the classical notion of mutexes between actions is presented, using an even more relaxed parallelism semantics. Also, an approach to prune unnecessary actions from a plan is explained. Experiments support the claim that this encoding is even more parallel, together with the usefulness of the unnecessary action removal technique. This new encoding and experiments are presented in [BEV17].

In Chapter 8, the final conclusions of the thesis are discussed, together with pointers to what interesting works could be derived from this point.

Finally, the Appendices include a description of the Rantanplan planner, where all the encodings and techniques have been implemented.

Chapter 2

Preliminaries

Regardless of which theories an automated planner can reason about, all of them can be broadly divided in two categories: domain independent and domain specific planners.

Domain specific planners are designed for specific problems. They generally cannot work on other problems. The advantage they have is that they can use some problem-specific techniques that are difficult to generalize to other planning domains. Some examples of problems with specialized planners can be the mars exploration rovers activities [BJMR05] or bridge playing [SNT98].

Domain independent planners aim to solve a planning problem specified in a given general language via some appropriate general planning algorithm. The output of the planner is usually a totally or partially ordered plan, that is, a sequence of actions, where some of which may be executed in parallel sometimes. The focus in this thesis is into domain independent planners, and therefore from now on we will be focusing on them.

2.1 State-Variable Representation

Our formalization of the classical planning problem is inspired in the ideas in [GNT16]. A *planning domain* is the description of a problem to be solved by a planning system. These domains can be seen as simplified representations of the real-world problem they are trying to solve. A classical planning domain can be seen as a finite-state automaton:

Definition 2.1.1 (State transition representation)**.** *A state transition system (also called a classical planning domain) is a tuple* $\prod = (S, A, \gamma)$*, where:*

- *S is a finite set of states*
- *A is is a finite set of actions that can be performed*
- *$\gamma : S \times A \to S$ is a partial function that maps pairs of states and actions to states. If $\gamma(s, a)$ s.t. $s \in S, a \in A$ is defined, then a is* applicable *in s and $\gamma(s, a)$ is the state resulting from applying a to s. If $\gamma(s, a)$ is not defined then a is not applicable in s*

Example 1. *Consider a scenario where an airline that has to transport some passengers to their destinations. The airline operates in three major cities: Barcelona, Berlin and London. The airline has only one aircraft, stationed in Barcelona, and two passengers, one in Barcelona and the other in Berlin. The objects of the problem are:*

$$\begin{aligned} \texttt{Cities} &= \{\texttt{Barcelona}, \texttt{London}, \texttt{Berlin}\} \\ \texttt{Aircrafts} &= \{\texttt{Aircraft1}\} \\ \texttt{Passengers} &= \{\texttt{Person1}, \texttt{Person2}\} \end{aligned}$$

Barcelona	**Berlin**	**London**
`Person1`		`Person2`
`Aircraft1`		

Figure 2.1: A representation of the initial state

The problem actions are: fly*, where an aircraft moves from one city to another,* embark*, where a passenger embarks an aircraft and* disembark*, where a passenger disembarks from the aircraft.*

To make problem representation and solving easier, some simplifications are commonly made. Those are called the *classical planning assumptions*:

- There is no explicit model of time. Only discrete sequences of actions are considered.
- The set of states is finite, observable and static. The change in the current state is only due to executed actions. The initial state is unique and known, and the set of actions is finite.

- All actions are deterministic. This excludes the possibility of interference by external events, or random effects in actions.

Since the initial state is known and actions are deterministic, the state of the world after any given sequence of actions can be determined unambiguously.

The states in this transition system are configurations of the objects considered in the problem. To represent them, we will use sets B and X.

- B is a set of names for all the objects, and all the constants (values) needed to represent properties of these objects. Typically, constants true and false ($\top$ and $\bot$ respectively) are added to B.

- X is a set of state variables, which will be used to represent the relevant properties of the problem. The value of each $x \in X$ will depend solely on the states $s \in S$.

State variables are defined as follows:

Definition 2.1.2 (State Variable)**.** *A state variable over B is expressed as*

$$x = sv(b_1, \dots, b_n)$$

where sv is a symbol representing the state variable name and $b_1, \dots, b_n$ are members of B. Each state variable x has a domain $Domain(x) \subseteq B$, which is the set of all possible values for x.

These variables associate a value, which changes over time, with a relevant attribute of the world. For example, consider a logistics domain that involves transporting packages using trucks. State variables would describe the location of a truck or a package, and how it changes over time. Handy values to include to B could be the Booleans or a *nil*.

Definition 2.1.3 (State)**.** *A* state *is a variable-assignment (or valuation) function over state variables X, which maps each $x_i \in X$ into a value $z_i \in Domain(x_i)$. This function is a set of ordered pairs*

$$s = \{(x_1, z_1), (x_2, z_2), \dots, (x_n, z_n)\}$$

which we will usually represent as a set of assertions:

$$s = \{x_1 = z_1, x_2 = z_2, \dots, x_n = z_n\}$$

Example 2. *In Example 1, if we consider a state variable* at *saying where a plane or a passenger is and a predicate* in *saying if a passenger is inside a plane, the initial state s_0 could be represented as follows:*

$$s_0 = \{\mathtt{at}(\mathtt{Aircraft1}) = \mathtt{Barcelona},\ \mathtt{at}(\mathtt{Person1}) = \mathtt{Barcelona},\\ \mathtt{at}(\mathtt{Person2}) = \mathtt{Berlin},\ \mathtt{in}(\mathtt{Person1}) = \mathtt{nil},\ \mathtt{in}(\mathtt{Person2}) = \mathtt{nil}\}$$

where the nil value in the in predicate represents that the passenger is not inside any plane.

Note that as X and B are finite, so is the number of variable-assignment functions. This definition of state is the same as the one in the SAS+ formalism [Bäc92].

Definition 2.1.4 (State-variable state space)**.** *The state-variable state space S is a set of states over state variables X.*

Note that the purpose of a state is to represent a feasible configuration of the problem elements. Therefore, typically not all variable-assignment functions can be considered meaningful states.

To express a way to write actions, we introduce some terminology loosely-borrowed from first-order logic with equality.

Definition 2.1.5 (Literal)**.** *A positive literal, or atom, is an expression of the form:*

$$sv(z_1, \ldots, z_n) = z_0$$

sv is a state variable name and each z_i is either a variable (an ordinary mathematical variable, not a state variable) or an element of B. The left-hand side of the literal ($sv(z_1, \ldots, z_n)$) is said to be the target.

A negative literal *is an expression of the form:*

$$sv(z_1, \ldots, z_n) \neq z_0$$

A literal is ground *if it contains no variables, and* unground *otherwise.*

Definition 2.1.6 (Instance of a Literal)**.** *Let l be an unground literal, and Z a subset of the variables in l. An instance of l is any expression l' produced by replacing each $z \in Z$ with a term z' such that $z' \in Domain(z)$ or z' is a variable with $Domain(z') \subseteq Domain(z)$.*

Definition 2.1.6 generalizes to any syntactic expression that contains literals. We will say that the expression is *ground* if it does not contain any (ordinary) variable, or *unground* otherwise. Now we can introduce actions.

Definition 2.1.7 (Action). *An* ungrounded action*, or* action template*, is a tuple* $\alpha = (head(\alpha), Pre(\alpha), Eff(\alpha))$ *such that:*

- *head*(α) *is an expression of the form:*

$$act(z_1, \ldots, z_n)$$

  such that act *is the* action name *and* $z_1, \ldots, z_n$ *are variables (ordinary variables, not state variables) also called* parameters*. These parameters must include all variables that appear in* $Pre(\alpha)$ *and* $Eff(\alpha)$*.*

- $Pre(\alpha) = \{p_1, \ldots, p_n\}$ *is a set of* preconditions*, each of it being a literal.*

- $Eff(\alpha) = \{e_1, \ldots, e_n\}$ *is a set of* effects*, each of it being a positive literal. No target can appear in Eff more than once.*

A ground action *a is a ground instance of an action template* α *where all state variables in* $Pre(a)$ *and* $Eff(a)$ *are ground, its parameters contain no variables and no target can appear in Eff more than once. For the action to be* applicable *in a state s, s must satisfy* $Pre(a)$*. The outcome after the application of the action will be the state:*

$$\begin{aligned}\gamma(s,a) = \{&(x,w) \mid x = w \in Eff(a)\} \cup \\ &\{(x,w) \in s \mid x \textit{ is not a target of any effect in } Eff(a)\}\end{aligned}$$

Note that if a is not *applicable*, $\gamma(s,a)$ is undefined.

Example 3 (Action representation). *If we continue with Example 1, the action named* `fly` *could be represented as follows:*

$$\alpha_1 = (\texttt{fly(plane, from, to)},\ \{\texttt{at(plane)} = \texttt{from}\},\ \{\texttt{at(plane)} = \texttt{to}\})$$

And a grounded instantiation of this action could be:

$$\begin{aligned}a_1 = &(\texttt{fly(Aircraft1, Barcelona, London)}, \\ &\{\texttt{at(Aircraft1)} = \texttt{Barcelona}\}, \{\texttt{at(Aircraft1)} = \texttt{London}\})\end{aligned}$$

This action would be applicable in any state satisfying `at(Aircraft1) = Barcelona`*. The effects after applying* a_1 *would satisfy* `at(Aircraft1) = London`.

Definition 2.1.8 (Plan)**.** *A* plan *is represented as a finite sequence of ground actions* $\pi = \langle a_1, \dots, a_n \rangle$*. A plan is* applicable *in a state* s_0 *if there are states* $s_1, \dots, s_n$ *such that* s_i *results from applying action* a_i *to state* s_{i-1} *for* $i \in 1, \dots, n$*, i.e.* $\gamma(s_{i-1}, a_i) = s_i$*. We call the* resulting final state *the state* s_n*.*

As a special case, the $\langle\rangle$ is the empty plan, which contains no actions and its length is 0.

Definition 2.1.9 (Classical Planning Problem)**.** *A classical planning problem is a tuple* $\prod = (S, A, I, G)$*, where*

- S *is the set of possible states,*
- A *is a set of action templates,*
- I *is the initial state and*
- G *is a set of positive ground literals, the goal.*

A plan $\pi = \langle a_1, \dots, a_n \rangle$ *is also called a* solution *of* $\prod$ *when all* $a_1, \dots, a_n$ *are grounded actions of* A*,* π *is applicable in* I *and the resulting final state satisfies the goal.*

A state is said to be a goal state *when it satisfies the goal.*

A solution (or plan) can be described as a sequence of actions taken one at a time that brings the problem from the initial state to a goal state. Due to the classical planning assumptions, the initial state is unique and known (a total function).

Example 4. *To quantify the state space size, we recover Example 1. The size of the domains of the state variables are:*

$$\begin{aligned}
\texttt{size(Domain(at(Person1)))} &= |\{\texttt{Barcelona}, \texttt{London}, \texttt{Berlin}\}| = 3 \\
\texttt{size(Domain(at(Person2)))} &= |\{\texttt{Barcelona}, \texttt{London}, \texttt{Berlin}\}| = 3 \\
\texttt{size(Domain(in(Person1)))} &= |\{\texttt{Aircraft1}, \texttt{nil}\}| = 2 \\
\texttt{size(Domain(in(Person2)))} &= |\{\texttt{Aircraft1}, \texttt{nil}\}| = 2 \\
\texttt{size(Domain(at(Aircraft1)))} &= |\{\texttt{Barcelona}, \texttt{London}, \texttt{Berlin}\}| = 3
\end{aligned}$$

which gives $(1+2)^3 \times 2^2 = 36$ *as the number of possible states of the problem with this representation. This is a small number of states, but as the problem grows in size, the number of possible states grows quickly. With the same representation, a fleet of 20 planes, 30 cities and 100 passengers* *yields a search space of approximately* $(20 + 100)^{30} \times 100^{21} = 2.37 \times 10^{104}$. *Given that the estimated number of atoms in the universe is about* 10^{80}, *it can be said that the search space grows quickly enough.*

However, note that few of those values are valid states in S. *For example, a passenger cannot be at two different cities at the same time. A consistent representation of the problem would only permit a transition from one state to another if the next state would be a valid state.*

2.2 The STRIPS formalism

The situation calculus [McC69] was one of the first approaches to represent and solve a planning problem based on state variables. The STRIPS [FN71] planning system followed, together with some other planners [GNT04, RN10] that used a slightly different problem representation. STRIPS was also the planner used in Shakey [Int70], one of the first robots built using Artificial Intelligence techniques.

The STRIPS representation used a similar representation than the one introduced in the previous section, but with only propositional state variables. Therefore the domain of all state variables is restricted to $\{\top, \bot\}$.

Example 5 (Classical state representation)**.** *Following Example 2, now the representation of the initial state would be:*

$$\begin{aligned} s_0 = \{&\texttt{at}(\texttt{Aircraft1}, \texttt{Barcelona}) = \top, \texttt{at}(\texttt{Aircraft1}, \texttt{London}) = \bot, \\ &\texttt{at}(\texttt{Aircraft1}, \texttt{Berlin}) = \bot, \texttt{at}(\texttt{Person1}, \texttt{Barcelona}) = \top, \\ &\texttt{at}(\texttt{Person1}, \texttt{London}) = \bot, \texttt{at}(\texttt{Person1}, \texttt{Berlin}) = \bot, \\ &\texttt{at}(\texttt{Person2}, \texttt{Barcelona}) = \bot, \texttt{at}(\texttt{Person2}, \texttt{London}) = \top, \\ &\texttt{at}(\texttt{Person2}, \texttt{Berlin}) = \bot, \texttt{in}(\texttt{Person1}, \texttt{Aircraft1}) = \bot, \\ &\texttt{in}(\texttt{Person2}, \texttt{Aircraft1}) = \bot\} \end{aligned}$$

While an action that brings an aircraft from one city to another would be:

$$\begin{aligned} \alpha_1 = (&\texttt{fly}(\texttt{plane}, \texttt{from}, \texttt{to}), \{\texttt{at}(\texttt{plane}, \texttt{from}) = \top\}, \\ &\qquad\qquad \{\texttt{at}(\texttt{plane}, \texttt{to}) = \top, \texttt{at}(\texttt{plane}, \texttt{from}) = \bot\}) \end{aligned}$$

and one of the many possible ground instantiations of it:

$$\begin{aligned} a_1 = (&\texttt{fly}(\texttt{Aircraft1}, \texttt{Barcelona}, \texttt{London}), \\ &\{\texttt{at}(\texttt{Aircraft1}, \texttt{Barcelona}) = \top\}, \\ &\{\texttt{at}(\texttt{Aircraft1}, \texttt{London}) = \top, \texttt{at}(\texttt{Aircraft}, \texttt{Barcelona}) = \bot\}) \end{aligned}$$

Again, this action would be applicable in any state that the action precondition `at(Aircraft1,Barcelona)` *has the value* $\top$. *The effects after applying this action would make ground state variable* $\texttt{at}(\texttt{Aircraft1}, \texttt{London}) = \top$ *and* $\texttt{at}(\texttt{Aircraft1}, \texttt{Barcelona}) = \bot$.

There are other formalisms to express planning problems, such as SAS+, TWEAK or variants of STRIPS. Although these may seem to exhibit different degrees of expressive power, it is proven that they are, in fact, expressively equivalent [Bäc95]. This means that, for example, neither negative goals, partial initial states nor multi-valued state variables increase the expressiveness of propositional STRIPS.

2.3 State-Space Planning

The main approach to planning is searching forward from the initial state and try to build a sequential plan that can reach one of the goal states.

Algorithm 1 Schematic Forward Search

```
Input: $\prod = (S, A, I, G)$
Output: A valid sequential plan $\pi$ or $UNSAT$
1: $Frontier \leftarrow \{(\langle\rangle, I)\}$
2: $Visited \leftarrow \emptyset$
3: while $Frontier \neq \emptyset$ do
4:   Select a node $n = \langle \pi, s \rangle \in Frontier$
5:   Remove $n$ from $Frontier$ and add it to $Visited$
6:   if $s$ satisfies $G$ then
7:     return $\pi$
8:   end if
9:   $NewNodes \leftarrow$ generate nodes that can be reached from $s$
10:  Update $Frontier$ with $NewNodes$ with unseen states
11: end while
12: return $UNSAT$
```

A schema that implements a *forward search* approach could be Algorithm 1. Note that, as the domains of the state variables are finite, termination is guaranteed. Each node is a pair $\langle \pi, s \rangle$ where π is a plan that represents the actions already selected and s is the state resulting of the application of the plan π to I. *Frontier* is a set of nodes waiting to be visited by the search algorithm. The plan and states of nodes are being generated on line 9, while in line 10 the reachable nodes that generate an unseen state are added to the $Frontier$. *Visited* is the set of nodes already visited by the algorithm. The algorithm has two ways of finishing: If the state being considered a goal state, it returns the plan. Otherwise, returns *UNSAT*. Many forward-search algorithms can be represented as algorithm 1 by slight modifications and specifying how they select the next node.

Most forward-search algorithms try to find a solution without having to explore all the search space, as it can get exponentially large. The key is the operation *select* on line 4, where the algorithm deterministically selects the *best* node out of all the nodes in the frontier. This *select* function is often called a *heuristic function* $h : S \to \mathbb{R}_{\geq 0}$ that maps a state to an estimate of the cost of reaching a goal state from that state.

2.3.1 Heuristic Functions

Many well known search algorithms can be used, for example the A^* algorithm [HNR72], IDA^* [Kor85a], Hill climbing [RN10] or the Greedy Best-First search (GBFS) [RN10]. Choosing the right algorithm depend on factors like the search space size, optimality of the solution or the characteristics of the heuristic function, among many other possible factors.

A heuristic function for a planning problem is a function h that, given a state s returns an estimate $h(s)$ of the minimum cost $h^*(s)$ of getting from s to a goal state. The heuristic function is said to be *admissible* if $h(s) \leq h^*(s)$ for every state s. The main way of computing a heuristic is doing a *relaxation* of the planning problem: given a planning problem $\prod$, produce an easier problem $\prod'$. This is done by weakening some of the constraints that restrict when an action is applicable or what it achieves, by restricting what the problem states are, by redefining what the problem actions are, ...This new problem $\prod'$ has the property that for every valid plan π for $\prod$, $\prod'$ has a solution π' such that $cost(\pi') \leq cost(\pi)$.

Given an algorithm that can find a valid plan in $\prod'$, it can be used to make a heuristic function for $\prod$ that works as follows: given a state s, solve $\prod' = (S', A', I, G)$ and return the cost of the solution. Heuristics can be *domain-independent*. In the following subsections some of these approaches are described.

A *delete relaxation heuristic* tries to estimate the cost to a goal state by making $\prod'$ a copy of $\prod$ where now state variables can have not one but many values at the same time. Moreover, once a state variable “obtains” a certain value, that value can always be used to satisfy further preconditions or goals. For example, in the setting of classical planning, if the initial state valuates a set of state variables X all to $\perp$ and the goal needs all of them to $\top$, the relaxation will remove all positive literals of the form $x = \bot$ for all $x \in X$ from the effects of the actions.

It can be expressed more generally as $\prod'$ is a planning problem that never removes literals from the state, it only adds new ones. For example, if the positive literal `at(Aircraft) = Barcelona` is true, and an action that has `at(Aircraft) = Berlin` as an effect is executed, the state becomes {`at(Aircraft) = { Barcelona, Berlin }`}. The additive heuristic h^{add} [BG99], the Fast Forward h^{ff} heuristic [HN01] and the h^{+} heuristic are examples of delete relaxation heuristics.

Let $\phi = \phi_1 \vee \cdots \vee \phi_n$ be a disjunction of atoms. ϕ is a disjunctive *landmark* if all valid plans for a problem produce a state during its execution where ϕ is true. Fact landmarks consider atoms in ϕ as ground state variables, while action landmarks consider atoms in ϕ as ground actions that must exist in any plan.

Landmarks can be used to help heuristics be more precise in their cost estimations. As computing landmarks has proven to be very complex (in fact, PSPACE-complete [HPS04] in the worst case) many heuristics in this category, like the LAMA heuristic [RHW08] work not on the original, but on a relaxed problem. Research on landmark generation has also focused on the development of polynomial-time criteria that are sufficient (but not necessary) to guarantee that a fact is a landmark.

Other work on landmarks includes, for example, using them to find optimal solutions to planning problems [KD09], improving the efficiency of planning by splitting planning problems into subproblems [VIV13].

Critical path heuristics estimate the cost of achieving a goal by examining the critical path length (the makespan) of a concurrent plan for a simplified problem. Some examples of this family of heuristics are h^m [HG00], or the *additive* h^m [HBG05] heuristics.

Another approximation are *abstraction heuristics*, where the idea is to estimate the cost by projecting the state space to a smaller space applying a graph homomorphism. Pattern databases [HBH$^+$07] are an example of an abstraction heuristic. A *pattern* is considered a subset of the state variables, and the simplified problem has all literals with state variables not in this subset removed. This approach has the problem of the lack of informedness of variables outside this set, and thus techniques that try to merge various abstractions have been developed [KD08].

The basic idea of the *network flow heuristic* is that, in all plans, the number of times each fact is asserted versus the number of times it is negated must be "balanced". This heuristic is normally calculated by solving a linear program encoding this idea. The flow heuristic [vdBBKV07] or its enhanced version [BvdB14] are examples of this family.

2.4 Plan-Space Planning

One of the problems with forward state-space planning is that given a plan, the search algorithm has to try all the possible combinations of orders between the actions in the plan to conclude that a goal cannot be reached.

A *partial order plan* is a plan in which the actions are partially ordered, along with a guarantee that every total ordering that is compatible with this partial ordering will be a solution plan. This idea gives the planner flexibility to postpone some of the ordering decisions until they are really needed. This concept is described as the least-commitment strategy: do not commit to orderings or instantiations until necessary.

The idea is to keep doing refinements to a partial plan until a solution is found. A partial plan can be defined as a set of totally or partially grounded actions, together with a set of constraints. This constraints can be of two types: causal links or value constraints. A causal link is a relation between two actions, involving a state variable. For example, the action `disembark(Passenger,Aircraft)` needs the passenger to be inside the plane, so to fulfill the condition `in(Passenger,Aircraft)` the action `embark(Passenger,Aircraft)` is needed. Value constraints can be equality or inequality constraints. An example of this kind of constraints would be: $\mathtt{at}(\mathtt{Aircraft1}, \mathtt{London}) \neq \mathtt{at}(\mathtt{Aircraft1}, \mathtt{Barcelona})$.

A plan can be found when a partial plan has no flaws. Flaws can be *open goals* or *threats*. An open goal is a goal that has no causal link. That is, a condition that no action in the plan can set to true. A threat is the negation of a needed precondition. For example, consider that action $a = \mathtt{fly}(\mathtt{Aircraft}, \mathtt{Barcelona}, \mathtt{London})$ needs $\mathtt{at}(\mathtt{Aircraft}, \mathtt{Barcelona}) = \top$, and action $b = \mathtt{fly}(\mathtt{Aircraft}, \mathtt{Berlin}, \mathtt{Barcelona})$ is responsible for setting variable $\mathtt{at}(\mathtt{Aircraft}, \mathtt{Barcelona}) = \top$. Consider that action $c = \mathtt{fly}(\mathtt{Aircraft}, \mathtt{Barcelona}, \mathtt{Berlin})$ is also part of the plan, with the effect $\mathtt{at}(\mathtt{Aircraft}, \mathtt{Barcelona}) = \bot$. To resolve the threat, a constraint can be imposed to prevent c from affecting the link between a and b: Make c go before b or after a or impose constraints over some state variables to prevent c from setting $\mathtt{at}(\mathtt{Aircraft}, \mathtt{Barcelona}) = \bot$.

Algorithm 2 depicts the idea of Plan-space Planning (PSP) [MR91]. It solves a planning problem by modifying a partial plan π, initialized with the facts entailed by the initial state and the goal, in which actions are partially ordered and partially grounded.

The objective is to produce a valid plan for $\prod$. It does this by repeatedly finding *flaws* and applying resolvers (a refinement that removes the flaw) for each one until it cannot find more flaws. Generally speaking, implementations of PSP search tend to run slower than the fastest state-space planner. This is due to very good heuristics in combination with greedy best-first search algorithms, and these heuristics are not directly applicable to PSP, because plan-space search have no explicit states. On the other hand, ideas from PSP have been transported to the realm of temporal planning, and have been useful for maintaining flexibility in uncertain environments. For more information, the reader can refer to [GNT16].

Algorithm 2 PSP Algorithm

```
Input: ($\prod, \pi$)
Output: A valid partially ordered plan $\pi$ or $UNSAT$
1: while true do
2:   if $Flaws(\pi) = \emptyset$ then
3:     return $\pi$
4:   end if
5:   Select a $f \in Flaws(\pi)$
6:   $R \leftarrow \{$all valid resolvers for$f\}$
7:   if $R = \emptyset$ then
8:     return $UNSAT$
9:   end if
10:  Select a $p \in R$
11:  $\pi \leftarrow p(\pi)$
12: end while
13: return $\pi$
```

2.5 Planning as SAT

Propositional satisfiability (SAT) was the first known NP-Complete problem, and the improvements in SAT technology during the last two decades have made it viable for solving many problems. Examples of its usefulness are its application to solve scheduling problems [BEG$^+$14] efficiently, solving classical planning problems [KSH06] or model checking applications [BCCZ99]. Some definitions follow to formally define the SAT problem.

Definition 2.5.1 (Atom)**.** *An atom can be seen as a statement, which can be true or false.*

An atom is the most simple formula of propositional logic, with no structure. For example, the atom p, can represent the statement "I like pizza".

Definition 2.5.2 (Literal)**.** *A literal is an atom (positive literal), or a negated atom (negative literal).*

For example, $\neg p$, could represent the negation of the statement "I like pizza".

Definition 2.5.3 (Clause)**.** *A clause is a disjunction of literals*

For example, $p \vee q \vee \neg r$ is a clause, where p, q and $\neg r$ are literals. Clauses are sometimes represented as a set of literals. The empty disjunction (*empty clause*) is represented by $\Box$.

A *unit clause* is a clause with only one literal, and a *binary clause* is a clause with two literals.

Definition 2.5.4 (CNF)**.** *A formula is a set of clauses. The formula is in conjunctive normal form (CNF) if it is a conjunction of clauses.*

For example, $(p \vee q) \wedge (\neg r \vee s)$ is a formula in CNF composed by two clauses: $p \vee q$ and $\neg r \vee s$.

Definition 2.5.5 (Truth Assignment)**.** *A truth assignment, interpretation, or model is a function that maps each atom to true ($\top$) or false ($\bot$).*

Now, given a truth assignment, a positive literal is said to be true only if the assignment maps its atom to $\top$. A negative literal is true only if the assignment maps its atom to $\bot$. Finally, a literal is false if its not true.

Now we can say that, given a truth assignment, a clause is true (satisfied) if at least one of its literals is true, and false if all of its literals are false. In particular, the empty clause is always false. A CNF is true if all of its clauses are true, and false otherwise. An empty CNF is always true.

Definition 2.5.6. *The Satisfiability Problem (SAT) for a CNF ϕ is the problem of deciding if there exists a truth assignment that satisfies all the clauses of ϕ.*

Example 6. *Let us consider a CNF formula ϕ having three clauses c_1, c_2 and c_3:*

$$\begin{aligned} c_1 &: p \vee \neg q \\ c_2 &: p \vee r \\ c_3 &: \neg p \vee q \vee r \end{aligned}$$

Under the (partial) truth assignment $\{\neg p, \neg q\}$, clauses c_1 and c_3 are satisfied and clause c_2 is undefined. Therefore, the CNF formula ϕ is undefined under this assignment.

Suppose now that this assignment is completed by adding the literal $\neg r$. Then, clause c_2 becomes unsatisfied. Finally, if we consider the assignment $\{\neg p, \neg q, r\}$, all the clauses are satisfied.

From now on, and for the sake of readability, we will use a more general syntax for expressing formulas. We will incorporate the use negations at any level and logical implications ($\rightarrow$). This kind of formulas can be transformed to equivalent CNFs by using distributivity, De Morgan's rules and the equivalence of $a \rightarrow b$ to $\neg a \vee b$.

2.5.1 Translation to SAT

Planning is a notoriously hard problem. In fact, the problem of finding out if there is a plan in a classical planning problem is PSPACE complete [Byl91, ENS92, Bäc92] The PSPACE hardness holds when the potential solutions can be of exponential length. If we are only interested in polynomial-length plans, then planning is indeed NP-complete.

In the planning as satisfiability approach, a planning problem is translated to a Boolean formula, with the property that any model of this formula corresponds to a valid plan. As the length of a valid plan is not known a priori, the basic idea is to bound the planning problem to a positive integer n, and then for $n = 1, 2, \ldots$ to take the problem of finding a plan of length n, rewrite it as a propositional formula $f(n)$, and try to solve it. If the planning problem is solvable, then $f(n)$ will be solvable for sufficiently large n.

We define $S^t = \{s^t | s \in S\}$, consisting of all state variables in S superscripted with an integer $t \geq 0$. From now on, given a formula ϕ, ϕ^t will represent the same formula but replacing all $s \in S$ by the corresponding $s^t \in S^t$. This superscripted integer t will represent the time step. So when $t = 0$, the formula will be representing the initial state, and with $t = 1$ the state after executing one action, and so on.

To represent the plan π, for all action $a \in A$ a Boolean variable a^t will represent if that action is executed at that time step t. The encoding, simplified from [Rin09], goes as follows. First, we express that the execution of an action implies its preconditions

$$a^t \rightarrow p^t \qquad \forall a = \langle p, e \rangle \in A \tag{2.1}$$

Then, if the action is executed, its effects will take place at the next time step.

$$a^t \rightarrow e^{t+1} \qquad \forall a = \langle p, e \rangle \in A \tag{2.2}$$

Note that p and e are defined as sets of valuations. These sets are translated to propositional form as conjunctions.

A change of value of a state variable must occur only if an action that can change that state variable has been executed.

$$(s^t \wedge \neg s^{t+1}) \rightarrow \bigvee \{a^t | \neg s \in e, a = \langle p, e \rangle \in A\} \qquad \forall s \in S \tag{2.3}$$

$$(\neg s^t \wedge s^{t+1}) \rightarrow \bigvee \{a^t | s \in e, a = \langle p, e \rangle \in A\} \qquad \forall s \in S \tag{2.4}$$

Finally, we have to restrict the execution of actions to one per time step:

$$\bigvee a^t \qquad a \in A \tag{2.5}$$

$$\neg(a_1^t \wedge a_2^t) \qquad a_1 \in A, a_2 \in A, a_1 \neq a_2 \tag{2.6}$$

If we retake Example 1, the actions `fly(Aircraft1,Barcelona,London)` and `fly(Aircraft1,Barcelona,Berlin)` can not be executed in parallel, as the effects would wrongly place the aircraft at two cities at the same time. In Chapter 4, the semantics to execute more than one action per time step will be explained.

To illustrate this encoding, let us consider Example 1 in Section 2.1. The set of state variables would be comprised by a total of 11 propositional variables, matching the variables reflected in Example 2. To express if an action with a given set of parameters is executed in a given time step, a propositional variable is used. We will be abbreviating Aircraft1 as *a1*, Barcelona as *bcn*, London as *lon* and Berlin as *ber* for brevity.

Then, we would need to express actions. To have a complete toy problem, we would need at least three actions to be able to express the transportation of persons between cities: fly, embark and debark. For the sake of brevity, if we only consider action fly in Example 3, the extra propositional variables to represent if an action is executed would be:

fly_bcn_bcn, fly_bcn_lon, fly_bcn_ber
fly_lon_bcn, fly_lon_lon, fly_lon_ber
fly_ber_bcn, fly_ber_lon, fly_ber_ber

Constraints 2.1 would be:

$$\begin{aligned}
&\text{fly\_a1\_bcn\_bcn}^t \rightarrow \text{at\_a1\_bcn}^t\\
&\text{fly\_a1\_bcn\_lon}^t \rightarrow \text{at\_a1\_bcn}^t\\
&\text{fly\_a1\_bcn\_ber}^t \rightarrow \text{at\_a1\_bcn}^t\\
&\text{fly\_a1\_lon\_bcn}^t \rightarrow \text{at\_a1\_lon}^t\\
&\text{fly\_a1\_lon\_lon}^t \rightarrow \text{at\_a1\_lon}^t\\
&\text{fly\_a1\_lon\_ber}^t \rightarrow \text{at\_a1\_lon}^t\\
&\text{fly\_a1\_ber\_bcn}^t \rightarrow \text{at\_a1\_ber}^t\\
&\text{fly\_a1\_ber\_lon}^t \rightarrow \text{at\_a1\_ber}^t\\
&\text{fly\_a1\_ber\_ber}^t \rightarrow \text{at\_a1\_ber}^t
\end{aligned}$$

Now, Constraints 2.2 would be:

$$\begin{aligned}
&\text{fly\_a1\_bcn\_bcn}^t \rightarrow (\text{at\_a1\_bcn}^{t+1} \wedge \neg\text{at\_a1\_bcn}^{t+1})\\
&\text{fly\_a1\_bcn\_lon}^t \rightarrow (\text{at\_a1\_lon}^{t+1} \wedge \neg\text{at\_a1\_bcn}^{t+1})\\
&\text{fly\_a1\_bcn\_ber}^t \rightarrow (\text{at\_a1\_ber}^{t+1} \wedge \neg\text{at\_a1\_bcn}^{t+1})\\
&\text{fly\_a1\_lon\_bcn}^t \rightarrow (\text{at\_a1\_bcn}^{t+1} \wedge \neg\text{at\_a1\_lon}^{t+1})\\
&\text{fly\_a1\_lon\_lon}^t \rightarrow (\text{at\_a1\_lon}^{t+1} \wedge \neg\text{at\_a1\_lon}^{t+1})\\
&\text{fly\_a1\_lon\_ber}^t \rightarrow (\text{at\_a1\_ber}^{t+1} \wedge \neg\text{at\_a1\_lon}^{t+1})\\
&\text{fly\_a1\_ber\_bcn}^t \rightarrow (\text{at\_a1\_bcn}^{t+1} \wedge \neg\text{at\_a1\_ber}^{t+1})\\
&\text{fly\_a1\_ber\_lon}^t \rightarrow (\text{at\_a1\_lon}^{t+1} \wedge \neg\text{at\_a1\_ber}^{t+1})\\
&\text{fly\_a1\_ber\_ber}^t \rightarrow (\text{at\_a1\_ber}^{t+1} \wedge \neg\text{at\_a1\_ber}^{t+1})
\end{aligned}$$

Frame axioms, represented in Constraints 2.3 would be:

$$\begin{aligned}
&(\text{at\_a1\_bcn}^t \wedge \neg\text{at\_a1\_bcn}^{t+1}) \rightarrow\\
&\qquad(\text{fly\_a1\_bcn\_lon}^t \vee \text{fly\_a1\_bcn\_ber}^t \vee \text{fly\_a1\_bcn\_bcn}^t)\\
&(\neg\text{at\_a1\_bcn}^t \wedge \text{at\_a1\_bcn}^{t+1}) \rightarrow\\
&\qquad(\text{fly\_a1\_lon\_bcn}^t \vee \text{fly\_a1\_ber\_bcn}^t \vee \text{fly\_a1\_bcn\_bcn}^t)\\
&(\text{at\_a1\_lon}^t \wedge \neg\text{at\_a1\_lon}^{t+1}) \rightarrow\\
&\qquad(\text{fly\_a1\_lon\_bcn}^t \vee \text{fly\_a1\_lon\_ber}^t \vee \text{fly\_a1\_lon\_lon}^t)\\
&(\neg\text{at\_a1\_lon}^t \wedge \text{at\_a1\_lon}^{t+1}) \rightarrow
\end{aligned}$$

$$
\begin{array}{l}
\qquad (\text{fly\_a1\_bcn\_lon}^t \vee \text{fly\_a1\_ber\_lon}^t \vee \text{fly\_a1\_lon\_lon}^t) \\
(\text{at\_a1\_ber}^t \wedge \neg \text{at\_a1\_ber}^{t+1}) \rightarrow \\
\qquad (\text{fly\_a1\_ber\_bcn}^t \vee \text{fly\_a1\_ber\_lon}^t \vee \text{fly\_a1\_ber\_ber}^t) \\
(\neg \text{at\_a1\_ber}^t \wedge \text{at\_a1\_ber}^{t+1}) \rightarrow \\
\qquad (\text{fly\_a1\_bcn\_ber}^t \vee \text{fly\_a1\_lon\_ber}^t \vee \text{fly\_a1\_ber\_ber}^t)
\end{array}
$$

Finally, the restriction of one action per time step expressed with Constraints 2.5:

$$
\begin{array}{l}
(\text{fly\_a1\_bcn\_bcn}^t \vee \text{fly\_a1\_bcn\_lon}^t \vee \text{fly\_a1\_bcn\_ber}^t \vee \\
\text{fly\_a1\_lon\_bcn}^t \vee \text{fly\_a1\_lon\_lon}^t \vee \text{fly\_a1\_lon\_ber}^t \vee \\
\text{fly\_a1\_ber\_bcn}^t \vee \text{fly\_a1\_ber\_lon}^t \vee \text{fly\_a1\_ber\_ber}^t)
\end{array}
$$

$$
\begin{array}{ll}
\neg(\text{fly\_a1\_bcn\_bcn}^t \wedge \text{fly\_a1\_bcn\_lon}^t), & \neg(\text{fly\_a1\_bcn\_bcn}^t \wedge \text{fly\_a1\_bcn\_ber}^t) \\
\neg(\text{fly\_a1\_bcn\_bcn}^t \wedge \text{fly\_a1\_lon\_bcn}^t), & \neg(\text{fly\_a1\_bcn\_bcn}^t \wedge \text{fly\_a1\_lon\_lon}^t) \\
\neg(\text{fly\_a1\_bcn\_bcn}^t \wedge \text{fly\_a1\_lon\_ber}^t), & \neg(\text{fly\_a1\_bcn\_bcn}^t \wedge \text{fly\_a1\_ber\_bcn}^t) \\
\neg(\text{fly\_a1\_bcn\_bcn}^t \wedge \text{fly\_a1\_ber\_lon}^t), & \neg(\text{fly\_a1\_bcn\_bcn}^t \wedge \text{fly\_a1\_ber\_ber}^t) \\
\neg(\text{fly\_a1\_bcn\_lon}^t \wedge \text{fly\_a1\_bcn\_ber}^t), & \neg(\text{fly\_a1\_bcn\_lon}^t \wedge \text{fly\_a1\_lon\_bcn}^t) \\
\neg(\text{fly\_a1\_bcn\_lon}^t \wedge \text{fly\_a1\_lon\_lon}^t), & \neg(\text{fly\_a1\_bcn\_lon}^t \wedge \text{fly\_a1\_lon\_ber}^t) \\
\neg(\text{fly\_a1\_bcn\_lon}^t \wedge \text{fly\_a1\_ber\_bcn}^t), & \neg(\text{fly\_a1\_bcn\_lon}^t \wedge \text{fly\_a1\_ber\_lon}^t) \\
\neg(\text{fly\_a1\_bcn\_lon}^t \wedge \text{fly\_a1\_ber\_ber}^t), & \neg(\text{fly\_a1\_bcn\_ber}^t \wedge \text{fly\_a1\_lon\_bcn}^t) \\
\neg(\text{fly\_a1\_bcn\_ber}^t \wedge \text{fly\_a1\_lon\_lon}^t), & \neg(\text{fly\_a1\_bcn\_ber}^t \wedge \text{fly\_a1\_lon\_ber}^t) \\
\neg(\text{fly\_a1\_bcn\_ber}^t \wedge \text{fly\_a1\_ber\_bcn}^t), & \neg(\text{fly\_a1\_bcn\_ber}^t \wedge \text{fly\_a1\_ber\_lon}^t) \\
\neg(\text{fly\_a1\_bcn\_ber}^t \wedge \text{fly\_a1\_ber\_ber}^t), & \neg(\text{fly\_a1\_lon\_bcn}^t \wedge \text{fly\_a1\_lon\_lon}^t) \\
\neg(\text{fly\_a1\_lon\_bcn}^t \wedge \text{fly\_a1\_lon\_ber}^t), & \neg(\text{fly\_a1\_lon\_bcn}^t \wedge \text{fly\_a1\_ber\_bcn}^t) \\
\neg(\text{fly\_a1\_lon\_bcn}^t \wedge \text{fly\_a1\_ber\_lon}^t), & \neg(\text{fly\_a1\_lon\_bcn}^t \wedge \text{fly\_a1\_ber\_ber}^t) \\
\neg(\text{fly\_a1\_lon\_lon}^t \wedge \text{fly\_a1\_lon\_ber}^t), & \neg(\text{fly\_a1\_lon\_lon}^t \wedge \text{fly\_a1\_ber\_bcn}^t) \\
\neg(\text{fly\_a1\_lon\_lon}^t \wedge \text{fly\_a1\_ber\_lon}^t), & \neg(\text{fly\_a1\_lon\_lon}^t \wedge \text{fly\_a1\_ber\_ber}^t) \\
\neg(\text{fly\_a1\_lon\_ber}^t \wedge \text{fly\_a1\_ber\_bcn}^t), & \neg(\text{fly\_a1\_lon\_ber}^t \wedge \text{fly\_a1\_ber\_lon}^t) \\
\neg(\text{fly\_a1\_lon\_ber}^t \wedge \text{fly\_a1\_ber\_ber}^t), & \neg(\text{fly\_a1\_ber\_bcn}^t \wedge \text{fly\_a1\_ber\_lon}^t) \\
\neg(\text{fly\_a1\_ber\_bcn}^t \wedge \text{fly\_a1\_ber\_ber}^t), & \neg(\text{fly\_a1\_ber\_lon}^t \wedge \text{fly\_a1\_ber\_ber}^t)
\end{array}
$$

It is obvious now that the size of the encoding is a factor to consider, and one cannot naively try to solve them if a solution is to be expected in reasonable time. Chapter 4 gives a more thorough view on this approach.

2.6 Other Planning Frameworks

The real world is very complex, and thus there are situations where the classical planning approach is not enough to be able to model some problems.

In some practical domains like robotics or control, the outcome of the actions cannot be accurately modeled and thus the execution of a given plan may not be fully predictable. In the planning community, this is modelled using a variant of classical planning that incorporates non-deterministic actions. Two close models can be used: Fully Observable Non-Deterministic planning (FOND) or Markov Decision Process (MDP). The former assumes non-determinism on the potential effects of the actions, while the latter breaks this assumption and assigns a probability distribution over the action outcomes.

In some scenarios, actions can be non-deterministic and the world partially observable, like in robot navigation problems. This kind of problems can be modelled as a Partially Observable Markov Decision Process (POMDP) or a Contingent Planning problem. Problems like modelling multi-player games, where each entity is independent, can choose to coordinate and has its own goals, can be expressed as multi-agent planning problems.

Until now, actions and effects are supposed to be instantaneous, but in the real world actions occur over a time span and conditions may have to hold during not only the beginning of the action but also during it. For example, in a transportation problem, the traveling time between places is not instantaneous, as it depends on distances. Also, trucks can only refuel in gas stations that are actually open. These problems are normally expressed as temporal planning problems.

These other interesting areas of automatic planning will not be further explored, as the focus of this dissertation diverts from them. If the reader is interested, in [GNT16] many of these interesting areas of automated planning are explored in detail.

Chapter 3

Satisfiability Modulo Theories

In this section we introduce Satisfiability Modulo Theories (SMT), as the encodings presented in this thesis will translate planning problems to SMT. We begin by explaining the Conflict-Driven Clause-Learning algorithm, which is a complete algorithm that tends to be the base algorithm used in state-of-the-art complete SAT and SMT solvers. First some concepts are introduced to help the reader understand the algorithm. The reader can refer to [BHvMW09] for a thorough explanation of various algorithms and its historical evolution.

The Resolution Method

Resolution is one of the complete methods used to solve SAT. It is based on the resolution rule, which provides a refutation complete inference system [Rob65]. The resolution rule produces a new clause implied by two clauses in CNF, containing complementary literals. Two literals are said to be complements if one is the negation of the other (in the following $\neg p$ is taken to be the complement of p).

Example 7. *Consider the two clauses $p \vee q_1 \vee \cdots \vee q_n$ and $\neg p \vee r_1 \vee \cdots \vee r_n$. As they have complementary literals, we can apply the resolution rule:*

$$\frac{p \vee q_1 \vee \cdots \vee q_n \quad \neg p \vee r_1 \vee \cdots \vee r_n}{q_1 \vee \cdots \vee q_n \vee r_1 \vee \cdots \vee r_n}$$

and produce the clause $q_1 \vee \cdots \vee q_n \vee r_1 \vee \cdots \vee r_n$.

The produced clause is called a *resolvent*, and the dividing line stands for logical entailment. Then, the resolvent can be used in further applications of the resolution rule.

When coupled with a complete search algorithm [DP60], the resolution rule yields a sound and complete algorithm for deciding the satisfiability of a propositional formula in CNF.

Unit Propagation

Unit propagation is a simple but incomplete method to prove the satisfiability of a formula. If a CNF formula ϕ contains a unit clause, then it can be simplified by *unit propagation*. To apply unit propagation to a CNF, Algorithm 3 can be used. This procedure receives a CNF and returns the simplified CNF and a set of literals U, corresponding to unit clauses.

Algorithm 3 Unit-Propagation

```
Input: ($\phi$ : CNF)
Output: An equisatisfiable formula $\phi'$ and a set of literals $U$
1: $\phi' \leftarrow \phi$
2: $U \leftarrow \emptyset$
3: while $\phi'$ contains no empty clause and has a unit clause $l$ do
4:     $\phi' \leftarrow \phi'|_l$
5:     $U \leftarrow U \cup \{l\}$
6: end while
7: return $\{\phi', U\}$
```

Given a clause ϕ and an atom a, $\phi|_a$ stands for the result of removing from ϕ all clauses containing literal a and removing literal $\neg a$ in all remaining clauses. The result of applying the unit propagation algorithm is that ϕ' will be equisatisfiable to ϕ and without unit clauses. U will contain the set of literals that must be satisfied.

Example 8. *Suppose we have the CNF formula $\phi = \phi_1 \land \phi_2 \land \phi_3$, where $\phi_1 = p$, $\phi_2 = \neg p \lor \neg q$ and $\phi_3 = \neg q \lor r$. We apply unit propagation to ϕ by invoking algorithm 3. After the first execution of the loop, we select ϕ_1 as* *the unit clause and*

$$\begin{aligned}&\phi_1 \textit{ is removed}\\&\phi_2 = \neg q\\&\phi_3 = \neg q \vee r\\&U = \{p\}\end{aligned}$$

and after the second iteration, $\neg q$ is selected,

$$\begin{aligned}&\phi_1 \textit{ was removed}\\&\phi_2 \textit{ is removed}\\&\phi_3 \textit{ is removed}\\&U = \{p, \neg q\}\end{aligned}$$

Finally, as all the clauses have been removed, ϕ has been proven satisfiable. A model can be found by satisfying the literals in U.

In Example 8, note that on the last step all clauses are removed, and as the original formula is in CNF, the empty conjunction left evaluates to $\top$.

Example 9. *Suppose now that we also have a CNF formula $\phi = \phi_1 \wedge \phi_2 \wedge \phi_3$, where $\phi_1 = q$, $\phi_2 = p$ and $\phi_3 = \neg p \vee \neg q$. After the first execution of the loop, the unit clause chosen is ϕ_2 and we have*

$$\begin{aligned}&\phi_1 = q\\&\phi_2 \textit{ is removed}\\&\phi_3 \textit{ is simplified to } \neg q\\&U = \{p\}\end{aligned}$$

On the second iteration, ϕ_1 is selected, giving the formula $\phi = q \wedge \neg q$, clearly a contradiction. If we apply another iteration

$$\begin{aligned}&\phi_1 \textit{ is removed}\\&\phi_2 \textit{ was removed}\\&\phi_3 \textit{ is simplified to } \Box\\&U = \{p, q\}\end{aligned}$$

Being $\Box$ the empty clause, we demonstrate the unsatisfiability of ϕ.

In Example 9, the empty clause ($\Box$) is found in ϕ_3, and therefore ϕ has been proven unsatisfiable. Remember that the empty disjunction evaluates to $\bot$.

The Davis-Putnam-Logemann Loveland Algorithm

Since the SAT problem is NP-complete, only algorithms with exponential worst-case complexity are known for it. Many SAT solvers are based on the Davis-Putnam-Logemann Loveland procedure, or DPLL [DLL62]. This procedure can decide if a CNF formula is satisfiable and find a satisfying interpretation if it is.

Algorithm 4 shows the Davis-Putnam-Logemann-Loveland (DPLL) procedure. It is an extension of the unit propagation method that can solve the satisfiability problem for any propositional formula.

Algorithm 4 DPLL

```
Input: $\phi$ : CNF formula
Output: unsatisfiable or a model of $\phi$
 1: $\phi, U' \leftarrow$ Unit-Propagation($\phi$)
 2: $U \leftarrow U \cup U'$
 3: if $\Box \in \phi$ then
 4:   return unsatisfiable
 5: end if
 6: if $\phi = \top$ then
 7:   return $U$
 8: end if
 9: $l \leftarrow$ a literal from $\phi$
10: L $\leftarrow$ DPLL($\phi|_l$, $U \cup \{l\}$)
11: if L $\neq$ unsatisfiable then
12:   return L
13: else
14:   L $\leftarrow$ DPLL($\phi|_{\neg l}$, $U \cup \{\neg l\}$)
15:   if L $\neq$ unsatisfiable then
16:     return L
17:   else
18:     return unsatisfiable
19:   end if
20: end if
```

The selection of the literal on line 9 can have a dramatic impact on the running time of the algorithm. Also note that the value of the selected literal is not assumed to be first true and then false, as literals can be positive or negated atoms. The state where the algorithm makes a decision on the polarity of an atom it is called a *decision level.*

The algorithm is recursive, making an implicit backtracking step in line 10. If the call returns *unsatisfiable*, then the parent caller continues by calling line 14 with the inverted polarity of the atom. If both values lead to a contradiction, then the algorithm backtracks to a previous decision level, and continues trying other values. The process of moving from the current decision level to a previous one only after trying both values is called *chronological backtracking.* The problem with chronological backtracking is that it does not take into account the information of the contradiction that triggered the backtrack.

Non-Chronological Backtracking

When a backtrack occurs, it can happen that the real cause of the conflict emerges from many levels before the current decision level. This leads the solver to explore many irrelevant branches before finding the real cause of the contradiction in a much higher level of decision. *Non chronological backtracking* addresses this problem by taking into account the set of variables that are actually involved in the contradiction. This technique was originally proposed as a technique for solving constraint satisfaction problems (CSPs) [SS77].

Non-chronological backtracking can be performed by first identifying the *conflict set*: Every assignment that contributes to the derivation of the empty clause. Then, when the empty clause is derived, instead of backtracking to the last level of decision, the algorithm backtracks to the last variable of the conflict set, while erasing all decisions between. The reader can refer to [BHvMW09] for further details.

Conflict-Driven Clause Learning

What can happen with non-chronological backtracking, is that the algorithm backtracks past every variable in a conflict set. This can occur due to a new decision triggering another conflict and then backtracking to a previous decision level with respect to the variables of the first conflict set. This new backtrack avoids the analysis of the first conflict and therefore the same mistakes that lead to the first conflict can be repeated in the future. To address this problem, a possible idea is to add clauses to the CNF to prevent this. But, how can these clauses can be identified? Each time unit resolution finds a conflict, there is a possibility to identify a clause implied by the CNF that can help unit resolution to detect this contradiction earlier. With this clause, unit resolution will be able to avoid the same mistake in the future much earlier.

For example, once a conflict set is identified, a conflict-driven clause can be created by simply negating all the assignments in the set. For example, if the conflict set is $\{x_1 = \top, x_2 = \bot, x_3 = \bot\}$, the conflict-driven clause should be $\neg x_1 \vee x_2 \vee x_3$. Once the conflict-driven clause is derived, it is added to the CNF formula. This process of adding a conflict-driven clause to the CNF is called *clause learning* [JS97, SS99, ZMMM01, BKS04].

Restarts

Another important technique employed by many modern SAT solvers are *restarts* [GSC97]. When a SAT solver restarts, it forgets all the current assignments and starts the search again at the root of the search tree, while it maintains other information, most notably the previously learned conflict-driven clauses. Restarting is a way of dealing with the heavy-tailed distribution of running time often found in combinatorial search [GSC97]. Intuitively, restarting prevents the solver from being stuck in an area of the search space that contains no solution. In practice, solvers normally restart after a given number of conflicts have been found, but also many other types of restart policies have been studied, such as arithmetic o geometric series over the number of conflicts.

The Conflict-Driven Clause-Learning Algorithm

One of the main reasons of the widespread usage of SAT solvers is the effectiveness of *Conflict-Driven Clause-Learning* (CDCL) SAT solvers. CDCL solvers are classically based on DPLL algorithms, incorporating many techniques, from which the most important ones are the non-chronological backtracking, the learning of conflict clauses and the policy-based restarts. Algorithm 5 shows the structure of the typical CDCL algorithm. It starts by simplifying the input formula by unit propagation. Then, while it has not assigned all variables a polarity, it keeps deciding on a variable, assigning it a polarity, and doing unit propagation again. If unit propagation finds the empty clause, it learns from the conflict: it adds new clauses to the CNF to prune the search space, and backtracks to the decision level it needs to undo the conflict. The newly introduced functions are the following:

- **PickBranchingVariable** selects a variable to assign, and its respective polarity. It returns an atom that reflects the decided polarity. The variable selection heuristic is considered decisive for finding as quick as possible a solution. A bad heuristic can lead to explore the

Algorithm 5 A typical CDCL algorithm

```
Input: $\phi$ : CNF clause
Output: unsatisfiable or a model of $\phi$
 1: $\phi, U' \leftarrow$ Unit-Propagation($\phi$)
 2: $U \leftarrow U \cup U'$
 3: if $\Box \in \phi$ then
 4:     return unsatisfiable
 5: end if
 6: $dl \leftarrow 0$
 7: while not AllVariablesAssigned($\phi$,$U$) do
 8:     $x \leftarrow$ PickBranchingVariable($\phi, U$)
 9:     $dl \leftarrow dl + 1$
10:     $\phi, U' \leftarrow$ UnitPropagation($\phi$)
11:     $U \leftarrow U \cup U' \cup \{x\}$
12:     if $\Box \in \phi$ then
13:         $\beta \leftarrow$ ConflictAnalysisAndLearning($\phi$,$U$)
14:         if $\beta < 0$ then
15:             return unstatisfiable
16:         else
17:             NonChronologicalBacktrack($\phi$,$U$,$\beta$)
18:             $dl \leftarrow \beta$
19:         end if
20:     end if
21: end while
22: return $U$
```

whole search space, whereas a good heuristic allows us to cut several regions. There are a lot of heuristic methods for selecting the variable to assign, but the most used are the ones based in the *Variable State Independent Decaying Sum* (VSIDS) heuristic [MMZ+01].

- **ConflictAnalysisAndLearning** consists in analyzing the most recent conflict, and learning a new clause from the conflict, as explained in Section 3. It returns the decision level from where the conflict originates.

- **NonChronologicalBacktrack** has three parameters: the formula ϕ, the trail of decisions U, and the decision level β. The function backtracks to the decision level computed by *ConflictAnalysisAndLearning*, as explained in section 3.

- **AllVariablesAssigned** tests if all variables have been assigned, in which case the algorithm terminates indicating a satisfiable result.

The Two-Watched Literals Scheme

In [GKSS08] many of the more recent techniques of modern SAT solvers are explained. Especially noticeable is the two-watched literals scheme, introduced initially in the Chaff SAT solver [MMZ$^+$01]. Now it is a standard method used by most SAT solvers for efficient constraint propagation. The key idea behind the watched literals scheme, as the name suggests, is to maintain and watch two special literals, for each not yet satisfied clause, which are not false under the current partial assignment (could still be either true or unassigned). Recall that empty clauses halt the DPLL process and unit clauses are immediately satisfied. Hence, one can always find such watched literals in all active clauses. It works as follows. Suppose a literal l is set to false. For each clause that had l as a watched literal, we examine it and find another candidate to watch (that is already true or unassigned), as we are not longer interested in the literal l, because it is already false. If the clause has no other candidate to watch, the remaining literal can still be true or unassigned. If its already true, nothing happens, as the clause is already satisfied. If it is unassigned, the clause is implied, as it has now become a unit clause. With this setup, the solver can test clause satisfiability by checking if at least one of its two watched literals is true. An interesting property is that when the solver unassigns l because of a backtracking, it does not have to do anything.

The two-watched literals scheme has played an important role in the efficiency of clause-learning SAT solvers, as it helps to cope with the length of the constantly learned clauses added to the database, allowing propagation to become very efficient.

3.1 Satisfiability Modulo Theories

Satisfiability Modulo Theories (SMT) is a decision problem for logical formulas where some predicates have predefined interpretations from background theories.

For example, if we consider the theory of linear integer arithmetic, a SMT problem could be the following formula: $p \vee 2x + y \leq 3z \wedge q$.

SMT is interesting, not only for his many applications in model checking [CFM12], static analysis [JHFK12], scheduling [ABP$^+$11] or general CSP solving [BSV10, MBL09], but because planning problems that integrate non-propositional reasoning can commonly be directly reformulated to SMT. We now introduce the main definitions and explain the solving approach.

Definition 3.1.1. *A* theory *is a set of first-order formulas closed under logical consequence. A theory T is said to be* decidable *if there is an effective method for determining whether arbitrary formulas are included in T.*

Definition 3.1.2. *A formula ϕ is* T-satisfiable or T-consistent *if $T \cup \{\phi\}$ is satisfiable in the first-order sense. Otherwise, it is called* T-unsatisfiable or T-inconsistent.

Definition 3.1.3. *A (partial) truth assignment M of a propositional formula can be seen either as a set or as a conjunction of literals, and hence as a formula. If M is a T-consistent partial truth assignment and ϕ is a formula such that $M \models \phi$, i.e., M is a (propositional) model of ϕ, then we say that M is a* T-model *of ϕ.*

Definition 3.1.4. *The* SMT problem *for a theory T is the problem of determining, given a formula ϕ, whether ϕ is T-satisfiable.*

There are two types of procedures for solving SMT, the so-called eager and the lazy approaches. In the eager approach, the input formula is fully translated into a propositional CNF formula, preserving satisfiability, which is then checked whether is satisfiable or not by a SAT solver. Sophisticated ad-hoc translations have been developed for several theories, but still on many practical problems either the translation process or the SAT solver run out of time or memory [dMR04]. In this thesis we focus on the lazy approach, since it has been shown to be the most efficient in many cases.

3.1.1 The Lazy SMT Approach

Currently most successful SMT solvers are based on a lazy approach. It consists of an efficient SAT solver integrated with a T-solver, which is a decision procedure for the given theory T. In this approach, while the SAT solver is in charge of the Boolean component of reasoning, the T-solver deals with sets of literals that belong to T. It is named *lazy* because the theory information is only used when checking the consistency of the truth assignment against the theory T.

The basic idea is to let the T-solver analyze the partial truth assignment that the SAT solver is building, and warn about conflicts with the theory T

(T-inconsistency). This idea combines the efficiency of the SAT solver and special-purpose algorithms inside the T-solver for non-Boolean reasoning.

Algorithm 6 shows an enumeration-based T-satisfiability procedure simplified (from [BCF$^+$06]), where the T-consistency is only checked for total Boolean assignments. The reader is referred to [Seb07] for a survey on the lazy SMT approach. The algorithm enumerates the Boolean models of the

Algorithm 6 Bool+T Algorithm

```
Input: $\phi$ : SMT formula
Output: Satisfiability of $\phi$
 1: $A^p \leftarrow T2B(Atoms(\phi))$;
 2: $\phi^p \leftarrow T2B(\phi)$;
 3: while $Bool\text{-}satisfiable(\phi^p)$ do
 4:     $\mu^p \leftarrow pick\_total\_assignment(A^p, \phi^p)$;
 5:     $\mu \leftarrow B2T(\mu^p)$;
 6:     $(\rho, \pi) \leftarrow T\text{-}satisfiable(\mu)$;
 7:     if $\rho =$ sat then
 8:         return sat;
 9:     else
10:         $\phi^p \leftarrow \phi^p \wedge \neg T2B(\pi)$;
11:     end if;
12: end while
13: return unsat;
```

propositional abstraction of the SMT formula ϕ and checks for their satisfiability in the theory T.

- The function *Atoms* takes a quantifier-free SMT formula ϕ and returns the set of atoms which occur in ϕ, where an atom is either a propositional variable or an expression of theory T.

- The function $T2B$ maps propositional variables to themselves, and ground atoms into fresh propositional variables, and is homomorphic with respect to Boolean operators and set inclusion.

- ϕ^p is initialized to be the propositional abstraction of ϕ using $T2B$.

- The function $B2T$ is the inverse of $T2B$.

- μ^p denotes a propositional assignment as a set (conjunction) of propositional literals.

- The function *pick_total_assignment* returns a total assignment to the propositional variables in ϕ^p. In particular, it assigns a truth value to all variables in A^p.
- The function T-*satisfiable* checks if a set of conjuncts μ is T-*satisfiable*, i.e., if there is a model for $T \cup \mu$, returning (`sat`,$\emptyset$) in the positive case and (`unsat`,π) otherwise, being $\pi \subseteq \mu$ a T-*unsatisfiable* set (the theory conflict set). Note that the negation of the propositional abstraction of π is added to ϕ^p in case of `unsat` (learning).

Algorithm 6 is illustrated with Example 10.

Example 10. *Consider the following SMT formula, expressed as a set of clauses, where T is assumed to be the theory of linear integer arithmetic:*

$$\begin{aligned}\phi = \{&\neg(x > 0) \vee a \vee b,\\ &\neg a \vee \neg b,\\ &\neg(x+1 < 0) \vee a,\\ &\neg b \vee \neg(y = 1)\}\end{aligned}$$

Then $\{x > 0, a, b, x+1 < 0, y = 1\}$ is its set of atoms and

$$A^p = \{p_{(x>0)}, a, b, p_{(x+1<0)}, p_{(y=1)}\}$$

is the Booleanization of this set, where $p_{(x>0)}, p_{(x+1<0)}$ and $p_{(y=1)}$ are three fresh propositional variables corresponding to the arithmetic atoms $x > 0, x+1 < 0$ and $y = 1$, respectively. The propositional abstraction of ϕ is then the following Boolean formula:

$$\begin{aligned}\phi^p = \{&\neg p_{(x>0)} \vee a \vee b,\\ &\neg a \vee \neg b,\\ &\neg p_{(x+1<0)} \vee a,\\ &\neg b \vee \neg p_{(y=1)}\}\end{aligned}$$

Note that ϕ^p is satisfiable. Suppose that pick_total_assignment(A^p, ϕ^p) returns us the following Boolean model for ϕ^p:

$$\mu^p = \{p_{(x>0)}, a, \neg b, p_{(x+1<0)}, \neg p_{(y=1)}\}$$

Now we need to check the T-satisfiability of $B2T(\mu^p)$. Since we are interested in checking the consistency of the current Boolean assignment with theory T, *here we only need to take into account the literals corresponding to the theory, i.e., we have to check the T-satisfiability of $\{x > 0, x + 1 < 0, \neg(y = 1)\}$. This is obviously T-unsatisfiable, so we get a subset of T-inconsistent literals from the T-solver, e.g., $\pi = \{x > 0, x + 1 < 0\}$, and we extend ϕ^p with the learned clause, namely $\neg p_{(x>0)} \vee \neg p_{(x+1<0)}$. Then the search starts again.*

In practice, the enumeration of Boolean models is carried out by means of efficient implementations of the CDCL algorithm [ZM02], where the partial assignments μ^p are incrementally built. These systems benefit of the spectacular progress in performance from SAT solvers in the last decade, achieved thanks to better implementation techniques and conceptual enhancements.

In the approach presented so far, the T-solver provides information only after a T-inconsistent partial assignment has been generated. In this sense, the T-solver is used only to validate the search a posteriori, not to guide it a priori. In order to overcome this limitation, the T-solver could also be used to detect literals l occurring in ϕ such that $M \models_T l$, where M is a partial assignment of ϕ. This is called *theory propagation*. The propagation capability is a very important aspect of theory solvers, since getting more general explanations (conflict sets) from the theory solver is essential in order to keep the learned lemmas as short as possible and will allow for more pruning in general.

Finally, as it happens in SAT solving, most SMT solvers do restart periodically in order to try to explore easier successful branches.

3.1.2 Theories and Logics

The Satisfiability Modulo Theories Library (SMT-LIB) [BST10] has the goal of establishing a common standard for the specification of benchmarks and of background theories, as well as to establish a library of benchmarks for SMT. The Satisfiability Modulo Theories Competition (SMT-COMP) is an associated yearly competition for SMT solvers. Among the logics considered in the SMT-LIB there are:

- The theory of *Equality and Uninterpreted Functions* (QF_EUF, or simply QF_UF) is the quantifier-free fragment of first order logic with equality and no restrictions on the signature (hence the name UF for Uninterpreted Functions). It is also known as the empty theory, as far as we are concerned with first order logic with equality (i.e., with equality built-in). It is a theory that is often integrated with other theories, like linear integer or real arithmetic. Uninterpreted functions have no other property than its name and arity, and are only subject to the following axiom: $x_1 = x'_1 \wedge \cdots \wedge x_n = x'_n \rightarrow f(x_1, \ldots, x_n) = f(x'_1, \ldots, x'_n)$. It is possible (and sometimes preferable) to eliminate all uninterpreted function symbols by means of Ackermann's reduction [Ack54]. In Ackermann's reduction, each application $f(a)$ is replaced by a variable f_a, and for each pair of applications $f(a)$, $f(b)$ the formula $a = b \rightarrow f_a = f_b$ is added, i.e., the single theory axiom $x = y \rightarrow f(x) = f(y)$ of the theory becomes instantiated as necessary. Some modern SMT solvers determine when doing this reduction is advantageous and do it dynamically by means of a technique called dynamic Ackermanization [dMB08a].

- *Linear Arithmetic* over the integers (QF_LIA) or the reals (QF_LRA). Closed quantifier-free formulas with Boolean combinations of inequations between linear polynomials over integer (real) variables, e.g., $(3x + 4y \geq 7) \rightarrow (z = 3)$ where x, y and z are integer variables. These inequalities can be placed in a normal form $c_0 + \sum_{i=1}^{n} c_i * x_i \leq 0$, where each c_i is a rational constant and the variables x_i are integer (real) variables. The most common approaches to solve linear arithmetic (real and integer variables) are based on the Simplex with Gomory cuts method. A description of how a QF_LIA and a QF_LRA solver is integrated into SMT can be found in [DdM06b].

- *Difference Logic* over the integers (QF_IDL) or the reals (QF_RDL). It is a very efficiently solvable fragment of linear arithmetic in which arithmetic atoms are restricted to have the form $x - y \bowtie k$, where x and y are numeric (integer or real) variables, k is a numeric (integer or real) constant and $\bowtie \in \{=, <, >, \leq, \geq\}$. In the usual solving method, first of all, the atoms are rewritten in terms of $\leq$. Then, the resulting atoms can be represented as a weighted directed graph with variables as vertices and edges from x to y labeled with k for every atom $x - y \leq k$. A formula is unsatisfiable iff there exists a path $x_1 \xrightarrow{k_1} x_2 \ldots x_n \xrightarrow{k_n} x_1$ such that $k_1 + k_2 + \cdots + k_n < 0$. A description of a QF_RDL solver can be found in [NO05].

- *Non-linear Arithmetic* over the integers (QF_NIA) or over the reals (QF_NRA). Quantifier free integer or real arithmetic with no linearity restrictions, i.e., with clauses like $(3xy > 2 + z^2) \vee (3xy = 9)$ where x, y and z are variables. The fragment this theory handles is not decidable. A possible technique to check the satisfiability of these formulas is to transform the problem into a linear approximation [BLO$^+$12].

- *Arrays* (QF_AX). Closed quantifier-free formulas over the theory of arrays with extensionality. The signature of this theory consists of two interpreted function symbols: *read*, used to retrieve the element stored at a certain index of the array, and *write*, used to modify an array by updating the element stored at a certain index. A possible approach to decide the satisfiability of ground literals in this theory is to transfer the atoms to the Equality and Uninterpreted Functions theory. Other approaches are based on a careful analysis of the problem that allows to infer, for each array, which are the relevant indices and which values are stored at these indices of the array [SBDL01, BNO$^+$08].

- *Bit vectors* (QF_BV). Closed quantifier-free formulas over the theory of fixed-size bit vectors. Bit vectors are normally used for representing memory contents. Common operations are: extraction of a sequence of bits, concatenation, arithmetic operations ($+$, $-$, $*$, ...), bit-wise operations (and, or, not, ...), etc. State-of-the-art methods for checking the satisfiability of a given bit vector formula are based on reduction to SAT (bit-blasting). Each bit vector is encoded into a set of Boolean variables and the operators are encoded into logical circuits [BKO$^+$07].

- *Other theories.* In the literature we can find some other theories of interest not considered in the SMT-LIB. For example, the Alldifferent theory [BM10], the theory of costs [CFG$^+$10] or the theory of sets [BRBT16].

The expressivity of each of these logics has its corresponding computational price. For example, checking consistency of a set of IDL constraints has polynomial time complexity while checking consistency of a set of LIA constraints is NP-complete.

Combination of Theories

Many SMT problems contain atoms from multiple theories. When dealing with two or more theories, a standard approach is to handle the integration of the different theories by performing some sort of search on the equalities between their shared (or *interface*) variables. First of all, formulas are purified by replacing terms with fresh variables, so that each literal only contains symbols belonging to one theory. For example,

$$a(1) = x + 2$$

is translated into

$$\begin{aligned} a(v_1) &= v_2 \\ v_1 &= 1 \\ v_2 &= x + 2 \end{aligned}$$

where the first literal belongs to UF, and the last two to LIA. Variables v_1 and v_2 are then called *interface variables*, as they appear in literals belonging to different theories. An *interface equality* is an equality between two interface variables. All theory combination schemata, e.g., Nelson-Oppen [NO79], Shostak [Sho84], or Delayed Theory Combination [BBC$^+$06], rely to some point on checking equality between interface variables, in order to ensure mutual consistency between theories. This may imply to assign a truth value to all the interface equalities. Since the number of interface equalities is given by $|\mathcal{V}| \cdot (|\mathcal{V}| - 1)/2$, where $|\mathcal{V}|$ is the number of interface variables, the search space may be enlarged in a quadratic factor in the number of interface variables.

In the case of combining UF with another theory T, an alternative approach is to eliminate the uninterpreted function symbols by means of Ackermann's reduction [Ack54], and then solving the resulting SMT problem only with theory T. However, this has the same disadvantage as theory combination since the number of additional literals is quadratic in the size of the input and, in fact, as shown in [BCF$^+$06], there is no clear winner between DTC and Ackermannization.

Chapter 4

Planning as Propositional Satisfiability

The problem of planning was born as a deduction problem, but around the year 1969 the deduction methods were not considered to be efficient enough. Therefore, the problem of planning started to be seen as a search problem. This was the only approach to the problem of planning until 1992, where a novel approach was presented: encoding it as a satisfiability problem [KS92]. The advances of SAT technology permitted to recover the interest in logic-based methods. Kautz and Selman developed a formal model of planning based on satisfiability, rather than deduction (at that time, the best-known logical formalization of planning was the situation calculus [McC69]). They devised how to create a set of axioms with the property that any model of the axioms corresponded to a valid plan. The ad-hoc encoding they presented was for the well-known blocksworld problem, where a set of wooden blocks in a table has to be used to build a vertical stack of blocks. Their encoding was a set of simple formulas, where they basically expressed:

- Rule out the possibility that an action executes despite the fact that its preconditions are false.
- Only one action occurs at a time.
- An action occurs every time.

With this, they could devise linear plans for any blocksworld problem. Then, we can describe the original SatPlan "system" [KS92] as a set of rules for encoding STRIPS-style linear planning problems (as explained in Section 2.2) to SAT. A similar encoding has been roughly explained in Section 2.5.

An efficient planning system emerged in 1995, by Blum and Furst [BF95, BF97], named GraphPlan. It translated STRIPS-style planning problems in a graph structure called a planning graph. It is an ordered graph, where alternating layers of nodes correspond to ground facts and fully instantiated actions, both indexed by time step. Arcs lead from each fact to the actions that contain it as a precondition in the next layer, and similarly from each action to its effects in the next layer. Then, a systematic search is made in this graph for a solution. A solution is a subgraph that contains the initial states and final states and no two actions in the same layer that conflict (i.e. one action deletes a precondition or an effect of the other).

In fact, the planning graph could be seen as a propositional representation of the problem. It was after the formalization of parallelism in plans [Kno94] that a key technical advance for planning as satisfiability was achieved. In 1996, Kautz et al. [KMS96] incorporated the notion of parallel plans in their planning as satisfiability approach. The main motivation for using parallel plans was that it could compactly represent all intermediate states of a sequential plan. The reduced number of explicitly represented states lead to smaller formulas and sometimes to a more easily solvable problem. Their approach also had to restrict what actions could appear at the same time step, using a notion of non-interference. This condition guaranteed that any total ordering on the actions executed at the same time step is a valid serial plan and it leads to the same state in all cases.

Shortly afterwards, the MEDIC [EMW97] planner appeared, which was the first complete implementation of SatPlan that took a STRIPS-style input and translated it into SAT. In the next year, the BlackBox [KS98, KS99] planner appeared, which performed a set of local computations called *mutex propagation* to infer mutexes (i.e. a negative binary clause). These mutexes were used to control interferences between actions. This process was based on an idea introduced by GraphPlan [BF95].

Solvers like MEDIC also implemented a *lifted* representation of actions, with the objective to lessen the size blowup of the formulas of big problems. But in that case the encoding was limited to linear plans, rendering the idea of parallel plans obsolete.

The biggest drawbacks to all the BlackBox successors is the enormously sized conjunctive normal form (CNF) formulas derived from encoding the plan graph. The encodings of some planning problems could become intractable due to their size blowup. This blowup is mitigated by some solvers by performing *reachability* and *neededness* analysis, but still those solvers are unable to tackle problems that planners with other approaches like forward-search guided by good heuristics routinely solved.

A new implementation of the idea, SatPlan04, entered in the 2004 International Planning Competition (IPC-4), on the optimal propositional track. Satplan04 worked similarly to Blackbox, by using a planning graph and translating the constraints implied by it to a set of clauses. It included four different encodings and a postprocessing step to remove redundant actions, but the main difference between them was the SAT solver used. In fact, it came up first in the IPC-4, thanks to two factors: the improvement in SAT solvers at the time [MMZ$^+$01, Rya04], and that the new problems were intrinsically hard. While general STRIPS-like planning is PSPACE-complete [Byl91, ENS92], some domains can be solved optimally with a polynomial number of backtracks [Hof02]. SatPlan06 [KSH06] improved over SatPlan04 and also won (tied with MaxPlan [CXZ07]) the IPC-5 competition. The 2004 version of SatPlan did not perform mutex propagation during the step where the plan graph was generated because the resulting formulas were so large due to mutex clauses that they were unsolvable due to memory constraints. For SatPlan06, mutex propagation was enabled, but only generated clauses for inferred mutexes for fluents, not for actions. This strategy allowed SatPlan06 to solve harder instances while avoiding the worst memory problems.

Nabeshima et al. proposed searching for plans in parallel [NII02], by having n SAT solvers simultaneously, by trying to solve the planning problem for horizons $1 \dots n$. If a formula is found satisfiable then a plan is found, and if its not satisfiable, start a new solver with horizon $n + 1$. Later, Rintanen et al. improved the idea in [RHN06]. The proposed algorithm tries to solve the planning problem with horizons lengths $1, 2, 3 \dots$ in parallel, assigning to the SAT solver with horizon length t CPU time g times that of horizon length $t - 1$, for some constant $g < 1$. That is, the rate at which SAT problems are solved form a decreasing geometric sequence. Some finite bound is used for the number of SAT solvers run at any given moment (as determined by available memory). Note that a prerequisite for the use of the efficient parallel search strategies is the compactness of the SAT encodings, as many instances will have to fit in memory.

Another approach that followed was the reduction on the number of queries to the SAT solver by relaxing constraints on action parallelism. This was proposed also by Rintanen et al. [RHN06], where they formalized a generalization of the notion of parallelism of [KMS96], calling it the $\forall$-step semantics. The $\forall$-step semantics considers that a set of actions can be executed simultaneously (i.e. in any order) if they are pairwise independent. Definition 4.0.1 expresses this idea more formally.

Rintanen et al. also proposed a linear-size encoding for the mutexes derived from the parallelism semantics.

Definition 4.0.1 ($\forall$-Step Plan)**.** *Given a set of actions A and an initial state I, for a state space S, a* $\forall$-step plan *for A and I is a sequence $P = \langle A_0, \dots, A_{l-1}\rangle$ of sets of actions for some $l \geq 0$, such that there is a sequence of states $s_0, \dots, s_l$ (the execution of P) such that*

1. *$s_0 = I$, and*
2. *for all $i \in \{0, \dots, l-1\}$ and every total ordering $a_1 < \cdots < a_n$ of A_i, $app_{a_1;\dots;a_n}(s_i)$ is defined and equals s_{i+1}.*

Where $app_{a_1;\dots;a_n}(s_i)$ is $app_{a_n}(\cdots app_{a_2}(app_{a_1}(s))\cdots)$, and $app_a(s)$ is the unique state resulting from applying action a to state s, assuming a is applicable in s.

Rintanen states in [Rin18] that it is sometimes believed that $\forall$-step plans represent "real" parallelism, and this is used as a justification when focusing on "optimal" (minimal horizon length) $\forall$-step plans. Minimal horizon length $\forall$-step plans do not in general have anything to do with any practically interesting optimality criterion (i.e. minimum cost or minimum real makespan). In particular, the definition of $\forall$-step plans does not guarantee that two actions that take place in the same step can actually be taken in parallel, or that two actions that interfere could not in reality be taken in parallel. Further, in those cases in which the problem modeling has guaranteed that the parallelism reflects reality, action durations also have to be taken into account. Durations of actions in the real world can vary so much between them that the minimal number of steps does not have much to do with minimal real makespan. All the actions in the real world would need to have exactly the same duration for this to really represent parallelism (a rare condition). So, we remark that the main purpose of the parallelism in planning as satisfiability is to reduce the size of the search space.

Rintanen et al. [RHN06] also introduced the $\exists$-step semantics, which exploits the concept of *post-serializability*, originally from Dimopoulos et al. [DNK97]. A linear encoding of the mutexes in the size of actions effects was also presented, based on a total ordering of the actions and requiring that no action affects a later action.

Definition 4.0.2 ($\exists$-Step Plan)**.** *Given a set of actions A and an initial state I, for a state space S, a* $\exists$-step plan *for A and I is a sequence $P = \langle A_0, \dots, A_{l-1}\rangle$ of sets of actions together with a sequence of states $s_0, \dots, s_l$ (the execution of P), for some $l \geq 0$, such that*

1. *$s_0 = I$, and*

2. *for all $i \in \{0, \dots, l-1\}$ there is a total ordering $a_1 < \cdots < a_n$ of A_i, such that $app_{a_1;\dots;a_n}(s_i)$ is defined and equals s_{i+1}.*

The idea of ∃-step plans is to allow possibly conflicting actions to be executed at the same time step if it can be guaranteed a priori that there exists a valid linearization (that is, there exists at least one valid order). Instead of requiring that each group A_i of actions can be ordered to any total order, as in ∀-step semantics, in ∃-step semantics it is sufficient that exists one order that maps state s_i to s_{i+1}. Note that under this semantics the successor s_{i+1} of s_i is not uniquely determined solely by A_i, as the successor depends on the implicit ordering of A_i and, hence, the definition has to make the execution $s_0, \dots, s_l$ explicit.

Rintanen et al. proposed shortly afterwards the relaxed ∃-step semantics [WR07]. This semantics is the same as the ∃-step semantics, but now an action can be executed in a time step even if it is not applicable at the start of the time step. Remember that the ∃-step semantics has to do the execution explicit. In this case, it is guaranteed that the action will be applicable after the execution of the previous action in the same time step.

Robinson et al. [RGPS09, RGPS08] proposed a factored encoding of ∀-step plans and demonstrated substantial speed-ups over some of the encodings from the SatPlan06 solvers. This factored representation reduces the size of the encoding by representing part of the grounding process as a propositional formula. For example, an action `move(`x`,`y`,`z`)` that moves object x from location y to a location z can be represented by the parameters $x \in X$, $y \in Y$ and $z \in Y$ where X is the set of objects and Y the set of locations [KMS96].

In the SAT solver community, VSIDS [MMZ+01] is the de-facto heuristic for deciding about what variable will be chosen to evaluate next. With the current encodings of planning to satisfiability, Rintanen [Rin12a, Rin12b] modified a SAT solver to implement new heuristics to replace VSIDS for planning problems. He also proposed a new way to represent clauses internally into the solver, exploiting the structure of planning problems. With these improvements, planning as SAT became state of the art again.

In the *relaxed relaxed ∃-step (R^2∃-step) semantics* presented by Balyo et al. [Bal13] the relaxed ∃-step semantics is further relaxed. The application of action effects is relaxed similarly to precondition requirements in the relaxed ∃-step semantics. Therefore the only requirement in the R^2∃-step semantics is that parallel actions can be ordered to form a valid sequential plan.

Definition 4.0.3 ($R^2\exists$-Step Plan)**.** *Given a set of actions A and an initial state I, for a state space S, a* relaxed relaxed $\exists$-step ($R^2\exists$-step) plan *for A and I is a sequence $P = \langle A_0, \ldots, A_{l-1}\rangle$ of sets of actions together with a sequence of states $s_0, \ldots, s_l$ (the execution of P), for some $l \geq 0$, such that $s_0 = I$, and for all $i \in \{0, \ldots, l-1\}$ there is a total ordering $a_1 < \cdots < a_n$ of A_i, such that $app_{a_1;\ldots;a_{j-1}}(s_i) \models Pre_{a_j}$ for all $a_j = \langle Pre_{a_j}, Eff_{a_j}\rangle \in A_i$, and $app_{a_1;\ldots;a_n}(s_i) = s_{i+1}$.*

Balyo et al. [BB15] tried to devise a rule to select the best encoding given a problem, by comparing the performance of various of the best performant recent encodings on the 2011 optimal track competition problems. The encodings compared were the family of Rintanen's encodings [RHN06], the reinforced encoding [BBT15] and the $R^2\exists$-step encoding [Bal13].

In fact, we remark that the definitions of $\exists$-step plan and $R^2\exists$-step plan are equivalent, because $app_{a_1;\ldots;a_n}(s_i)$ is defined iff $app_{a_1;\ldots;a_{j-1}}(s_i) \models Pre_{a_j}$ for all $a_j = \langle Pre_{a_j}, Eff_{a_j}\rangle \in A_i$. What really happens is that Rintanen uses a notion of happening that requires that the preconditions of actions that can be executed in parallel are able to be satisfied at the same time. But the definition that he gives of the $\exists$-step plans does not include this and therefore is sufficiently general to cover the $R^2\exists$-step plans.

4.1 Numeric Planning

In real-world logistic problems, reasoning about numeric variables is essential, as most problems include various magnitudes that cannot be avoided, like weights, sizes, costs, ... Taking into account these magnitudes brings the abstract problem representation closer to the real world problem.

More formally, a numeric planning problem can be defined as a tuple $\langle V, P, A, I, G\rangle$ where V is a set of numeric variables, P is a set of propositions (or Boolean variables), A is a set of actions, I is the initial state and G is a formula over $V \cup P$ that any goal state must satisfy. A state is a total assignment to the variables. Actions are formalized as pairs $\langle p, e\rangle$, where p are the preconditions and e the effects. p is a set of Boolean expressions over $V \cup P$, while e is a set of assignments. An assignment is a pair $\langle v, exp\rangle$, where v is a variable and exp is an expression of the corresponding type. For example, increasing a variable v by one is represented by the pair $\langle v, v+1\rangle$, indicating that $v+1$ is the value that v will hold in the next state.

An action $a = \langle p, e\rangle$ is executable in a given state s if $s \models p$ and the effects of a in state s are consistent, i.e., we do not have $exp \neq exp'$ for any variable $v \in V \cup P$ and assignments $\langle v, exp\rangle$ and $\langle v, exp'\rangle$ in the effects.

Example 11. *We retake Example 1, where an airline has to transport some passengers to their destinations. The objects of the problem where:*

$$\begin{aligned} \texttt{Cities} &= \{Barcelona, London, Berlin\} \\ \texttt{Aircrafts} &= \{Aircraft1\} \\ \texttt{Passengers} &= \{Person1, Person2\} \end{aligned}$$

The predicates were `at` *and* `in`*, that represented where a plane or person were, and if a passenger was inside a plane respectively. Now we add a state variable, named* `fuel`*, that represents how much fuel does an aircraft have. Each flight will decrease fuel by one, and we also have an action that refuels a plane. The initial state can now be represented as:*

$$\begin{aligned} s_0 = \{&at(Aircraft1) = Barcelona, at(Person1) = c_1, \\ &at(Person2) = Berlin, in(Person1) = nil, in(Person2) = nil, \\ &fuel(Aircraft1) = 10\} \end{aligned}$$

The `fly` *and* `refuel` *action would be:*

$$\begin{aligned} \alpha_1 = (&\texttt{fly}(\texttt{plane}, \texttt{from}, \texttt{to}), \\ &\{\texttt{at}(\texttt{plane}) = \texttt{from},\ \texttt{fuel}(\texttt{plane}) > \texttt{0}\}, \\ &\{\texttt{at}(\texttt{plane}) = \texttt{to},\ (\texttt{fuel}(\texttt{plane}), \texttt{fuel}(\texttt{plane}) - \texttt{1})\}) \end{aligned}$$

$$\alpha_2 = (\texttt{refuel}(\texttt{plane}),\ \{\},\ \{(\texttt{fuel}(\texttt{plane}), \texttt{10})\})$$

4.2 Numeric Planning as Satisfiability

The area of planning as satisfiability has seen few works that try to integrate numeric reasoning in the planning problem. The first to obtain notable results was LPSAT [WW99] by combining a SAT solver with a Simplex solver. The interesting part of LPSAT is the use of conflict sets for guiding the solvers, together with the learning and backjumping methods from the SAT solver. It can be said in some sense that this article hinted some of the ideas that gave birth to SMT. In parallel, Kautz and Walser presented the ILP-PLAN framework [KW99b] to solve planning problems presented as integer lineal problems, but few works followed [WW01, SD05] in the area. Some efforts that used traditional forward-search heuristic planners with a module to reason about magnitudes [Hof03, GSS08] appeared, but the area did not grow beyond.

Most of the works that dealt with numeric systems, shifted their interest on the planning and control of hybrid systems [LEKN12, DIMM10],... It wasn't until in 2007, after SMT started to gain some traction [NOT06], that the NumReach [HGSK07] planner appeared. Numreach translated classical planning with resources problems to SMT. It also used a planning as SAT approach, where it discretized the possible states of numerical variables. Thanks to the expressiveness of SMT, the numeric planning problem could be naturally translated. It also showed that at the time, the translation to SMT was not competitive. Planning as SMT found some traction with temporal planning [Rin15, RG15]. Although reductions of temporal or hybrid systems to SAT or SMT have been known since at least 2005 [SD05], SMT has not been normally viewed as a competitive approach.

Other approaches, related to SMT to some amount as well, have been developed. In [BM12], a set of encoding rules is defined for spatio-temporal planning, taking SMT as the target formalism. On the other hand, a modular framework named PMT [GLFB12], inspired in the architecture of lazy SMT, is developed for planning with resources. PMT can be seen as an abstraction of numeric planning. In fact, PMT can be seen as classical planning modulo any theory that could be needed for solving, like integer arithmetic or set theory.

Lately, some approaches to planning as SMT have been developed that try to exploit the expressiveness of SMT, like Springroll [SRHT16], SMT-Plan [CFLM16] or Rantanplan [BEV15, BEV16b]. Springroll uses the planning as SMT approach, with a $\forall$-step semantics. The planner focuses at producing more succinct encodings by "rolling up" an unbounded yet finite number of instances of an action into a single plan step. In problems where "foldable" actions occur, the planner is able to reduce the number of time steps importantly. SMTPlan proposes an approach to PDDL+ [FL02] planning through SMT, with an encoding that captures all the features of the PDDL+ language. Its encoding focuses on domains with nonlinear and continuous change. Rantanplan is the planner developed during this thesis, also focused on the planning as SMT approach. As far as we know, its the first planner to support the $\forall$-step, $\exists$-step and $R^2\exists$-step semantics for numerical planning problems, and it focuses its efforts on producing small encodings by assigning as much actions as possible in the least number of time steps. This reduction in formula sizes helps by reducing the search space and therefore helping in solving the problem. Its approach will be explained in detail in the following sections.

Finally, regarding the heuristic approach, the most influential planners that can handle numeric planning appeared in the third and fourth edition of the International Planning Competition. Those are SGPlan, Metric-FF and LPG-td. The SGPlan [CWH06] planner bootstraps heuristic search planners by applying Lagrange optimization to combine the solution of the planning subproblems. To split the problem, an ordering of the planning goals is derived. The incremental local search strategy that is applied for Lagrange optimization on top of the individual planners relies on the theory of extended saddle points for mixed integer linear programming. Metric-FF [Hof03] is based on the FF system by the same author. It uses forward state-space search using relaxed plans to give heuristic guidance in its choice between possible steps through the space. Metric-FF is an extension that includes delete-relaxation heuristics for numeric and Boolean theories. Other authors refined or improved the Metric-FF heuristics [AN17, CFLS08].

LPG-td [GSS05] is a planner based on a local-search algorithm, applied to plan graphs [BF95]. The approach has been generalised to support numeric and temporal problems. The use of local search allows the planner to be configured to trade-off time and plan quality.

Chapter 5

Encodings for Planning as SMT

In this chapter, a formal definition of the framework of planning modulo theories is given, together with two different encodings. The first encoding is a planning as SMT encoding, generalizing Rintanen's encodings of planning as SAT, and supporting $\forall$ and $\exists$-step semantics. Experimental results are presented on this encoding, together with a explanation of how interference between actions is determined. Finally, a lifted encoding is also explained. It takes advantage of the theory of uninterpreted functions that is normally embedded in numeric theories in many SMT solvers.

5.1 Planning Modulo Theories

Reasoning about resources, distances and other magnitudes is necessary to be able to solve many real world planning problems. Planning Modulo Theories (PMT) is an approximation inspired by Satisfiability Modulo Theories (SMT) that generalizes the integration of arbitrary theories with propositional planning. We explore the planning as SMT approach, following the concepts and notation defined in [GLFB12] for PMT.

A *state* is a valuation over a finite set of variables X, i.e., an assignment function, mapping each variable $x \in X$ to a value in its domain, D_x. The expression $s(x)$ denotes the value that state s assigns to variable x, and $s[x \mapsto v]$ is the state identical to s except that it assigns the value v to variable x. A *state space* for a set of variables X is the set of all valuations over X. By $var(S)$ we denote the state variables of a state space S.

A *first order sentence over a state space S modulo T* is a first order sentence over the variables of S, constant symbols, function symbols and predicate symbols, where T is a theory defining the domains of the state space variables and interpretations for the constants, functions and predicates.[1] A *state space modulo T* is a state space ranging over the domains defined in T. A *term* over S modulo T is, similarly, an expression constructed using the symbols defined by S and T. A formula ϕ is T-satisfiable if $\phi \wedge T$ is satisfiable in the first-order sense. By $eval_T^s(\phi)$ we denote the value of ϕ under the assignment s, according to the interpretation defined by theory T.

A *substitution* is a partial mapping from variables to terms. It can be represented explicitly as a function by a set of bindings of variables to terms. That is, if $\sigma = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ (assuming $x_i \neq x_j,\ \forall i \neq j \in 1 \ldots n$), then $\sigma(x_i) = t_i$ for all i in $1..n$, and $\sigma(x) = x$ for every other variable. We also define the domain of σ as $Dom(\sigma) = \{x_1, \ldots, x_n\}$.

Substitutions are extended homomorphically to a total mapping from terms to terms. We use the postfix notation $t\sigma$ for the image of a term t under a substitution σ. This is defined inductively on the structure of terms as follows:

$$t\sigma = \begin{cases} \sigma(t) & \text{if } t \text{ is a variable} \\ f(t_1\sigma, \ldots, t_m\sigma) & \text{if } t \text{ is of the form } f(t_1, \ldots, t_m) \end{cases}$$

In the second case of this definition, $m = 0$ is allowed: in this case, f is a constant symbol and $f\sigma$ is f. Thus $t\sigma$ is t with all variables replaced by terms as specified by σ. The image of a formula under a substitution is defined similarly.

The composition of two substitutions σ_1 and σ_2, denoted by juxtaposition, is defined as the composition of two functions, that is, $t\sigma_1\sigma_2 = (t\sigma_1)\sigma_2$.

Definition 5.1.1 (Action). *An* action *a, for a state space S modulo T, is a state transition function, comprising:*

- *A first-order sentence over S modulo T, Pre_a (the* precondition *of a).*
- *A set Eff_a (the* effects *of a), of assignments to a subset of the state variables in S, each assigning a distinct variable to a value defined by a term over S modulo T.*

[1]In some other contexts, such as mathematical logic, a theory is understood as being just a set of sentences.

An action a, for a state space S modulo T, is *applicable* (or *executable*) in a state $s \in S$ if $T, s \models Pre_a$ (that is, the theory together with the valuation s satisfies the precondition of a).

We represent actions a as pairs $\langle Pre_a, \mathit{Eff}_a \rangle$, with the effects Eff_a often written as a substitution $\sigma_a = \{x_1 \mapsto exp_1, \ldots, x_n \mapsto exp_n\}$, where exp_i is an expression that defines the value of variable x_i in the resulting state, for each i in $1..n$ (e.g. $x \mapsto x + k$, for increasing a numeric variable x by k). We use $\top$ and $\bot$ to denote the Boolean *true* and *false* values, respectively. Making abuse of notation, we will talk of a substitution as an assignment.

Following the application of a, the state is updated by the assignments in Eff_a to the variables that they affect, leaving all other variables unchanged. We denote the unique state resulting from applying action a, in a state s in which is applicable, by $app_a(s)$. Formally, the resulting state s' is the mapping where, for each variable $x \in var(S)$, $s'(x) = eval_T^s(x\sigma_a)$, where σ_a is the substitution representing the effects of a. For any given sequence of actions $a_1; a_2; \ldots; a_n$ we define $app_{a_1;a_2;\ldots;a_n}(s)$ as $app_{a_n}(\cdots app_{a_2}(app_{a_1}(s)) \cdots)$.

Definition 5.1.2 (Planning modulo Theory)**.** *A* Planning Modulo T *problem, for a theory T, is a tuple $\pi = \langle S, A, I, G \rangle$ where:*

- *S is a state space in which all variable domains are defined in T,*
- *A is a set of actions for S modulo T,*
- *I is a valuation in S (the* initial state*), and*
- *G is a first order sentence over S modulo T (the* goal*).*

A (sequential) plan *for π is a sequence of actions $a_1; \ldots; a_n$ such that, for all i in $1..n$, a_i is applicable in state s_{i-1} and s_i is the result of applying a_i to s_{i-1}, where $s_0 = I$ and $T, s_n \models G$.*

As it is usual in SMT, we assume that T is a first-order theory with equality, which means that the equality symbol $=$ is a predefined predicate, interpreted as the identity in the underlying domain. Sometimes we say that a sequence of actions is a plan starting from an initial state I, without specifying the goal. In this case we mean that the plan is executable starting from I.

This approach can be seen as a generalization of some previous works that use sub-solvers to work with theories. ILP-PLAN [KW99a] is a framework based on integer optimization of linear integer programs that can be seen as a particular case of this, taking linear inequalities as preconditions, and limiting effects to increasing, decreasing or setting the value of a variable. Numeric planning, as defined in [Hel02] or in [GSS08] is also a particular case, using a very limited fragment of first-order logic in the preconditions of actions, and taking T as the theory of rational functions, i.e., fractions between polynomials. The proposal in [RHN06] raises preconditions to general Boolean formulae, but does not consider numeric variables. In [GLFB12] the reader can find other examples of planners using subsolvers to work with theories.

5.2 Generalized SMT Encoding

In this section we propose an encoding for PMT as *planning as SMT*, that can adhere to the $\forall$-step and $\exists$-step semantics. The following encoding is a generalization of Rintanen's [Rin09] encoding of planning as SAT, to include reasoning with a theory T. The encoding is valid for any theory T under quantifier-free first-order logic with equality. In particular, for numeric planning we could take T as the theory of the integers (or the reals) and use quantifier free linear integer (or real) arithmetic formulae. This is the case for the upcoming examples in this section, but we emphasize that this encoding could be used for any theory T.

To fill a gap between the problem representation and the representation used to solve the problem, it should be noted that languages like the Planning Domain Definition Language [FL03] (PDDL from now on) use a lifted representation to express problem variables and actions.

Example 12. *For example, a typical way to express the position of an element is to define a Boolean predicate* `at(?o - object,?p - place)`. *This predicate states, given an object and place in the problem, if the object is in the place.*

These lifted representations are compact and practical to model the problem, but when solving, most approaches do not support these representations and therefore ground them. That is, each predicate or function is expanded so they lack any free variable.

Example 13. *For instance, a Boolean predicate stating the position of an aircraft such as* `at(?a - aircraft, ?c - city)`, *with three cities* `c1`, `c2` *and* `c3`, *and two planes* `plane1` *and* `plane2`, *when grounded will result into six ground instances* `at(plane1,c1)`, ..., `at(plane2,c3)`, *that will be mapped to six Boolean variables* $at^t_{plane1,c1}$, ..., $at^t_{plane2,c3}$ *for each time step* t.

Example 14. *Now let us consider the function (or object fluent*[2]*)* `at(?o - aircraft) - city` *that, given an aircraft, says where it is. In this case, the mapping would result into two numeric variables at_{plane1}, at_{plane2} with the domains being the possible cities* `c1`*,* `c2` *and* `c3` *(conveniently mapped to integers).*

Note that in the example, if we map objects to numbers, we can get a more compact encoding of states in the presence of object fluents than using a plain SAT approach.

Therefore, in our approach, when a lifted representation like PDDL is used to state the problem, every PDDL predicate is grounded and mapped to a Boolean variable, and every PDDL function is grounded and mapped to a numeric variable. Then, the Boolean and numeric variables resulting from grounding the predicates and functions, respectively, constitute the state variables.

Let $\pi = \langle S, A, I, G\rangle$ be a planning problem modulo T, for a theory T under a quantifier-free first-order logic with equality. For each variable x in $var(S)$ and each time step t, a new variable x^t of the corresponding type is introduced, denoting the value of x at step t. Moreover, for each action a and each time step t, a Boolean variable a^t is introduced, denoting whether a is executed at step t.

Given a term s, by s^t we denote term s, where all variables x in $var(S)$ have been replaced by x^t, and analogously for formulas. For example $(x + y)^t = x^t + y^t$, and $(p \wedge x > 0)^t = p^t \wedge x^t > 0$. For the case of effects, we define

$$\{x \mapsto \top\}^t \overset{def}{=} x^{t+1}$$
$$\{x \mapsto \bot\}^t \overset{def}{=} \neg x^{t+1}$$
$$\{x \mapsto s\}^t \overset{def}{=} (x^{t+1} = s^t)$$

where s is a non-Boolean term belonging to the theory T. For example, for an assignment $\{x \mapsto x + k\}$, where k is a constant, we have $\{x \mapsto x + k\}^t = (x^{t+1} = x^t + k)$. For sets of assignments, i.e., action effects, we define

$$(\{x \mapsto s\} \cup \mathit{Eff})^t \overset{def}{=} \{x \mapsto s\}^t \wedge \mathit{Eff}^t \quad \text{and} \quad \emptyset^t \overset{def}{=} \top$$

where s is a term (either Boolean or not) and *Eff* is a set of assignments.

[2]An object fluent is a mechanism adopted in PDDL 3.1 [Gef00] where a function can refer to problem objects.

The introduced constraints to represent the planning problem are the following:

First, if an action is executed during time step t, it implies that its preconditions are met.

$$a^t \rightarrow Pre_a^t \qquad \forall a = \langle Pre_a, Eff_a \rangle \in A \tag{5.1}$$

Also, each of its effects will hold at the next time step.

$$a^t \rightarrow Eff_a^t \qquad \forall a = \langle Pre_a, Eff_a \rangle \in A \tag{5.2}$$

Second, we need explanatory axioms to express the reason of a change in the value of Boolean state variables. For each variable x in $var(S)$,

$$x^t \neq x^{t+1} \rightarrow \bigvee_{\substack{\forall a = \langle Pre_a, Eff_a \rangle \in A \\ \text{such that } \exists \{x \mapsto s\} \in Eff_a}} a^t \tag{5.3}$$

That is, a change in the value of x implies the execution of at least one action that has an assignment to x among its effects.

Conditional Effects

In the PDDL language, there exists a modelling feature called *conditional effects*. These conditional effects, as their name suggest, can add conditions to any set of effects of an action. They can be used for expressing some domains more compactly.

Example 15. *For example, the following action in PDDL uses the* `when` *construct to express different fuel consumptions for a ship, depending on if it carries any load.*

```
(:action sail
  :parameters (?sh - ship ?from - location ?to - location)
  :precondition (at ?sh ?from)
  :effect (and
           (at ?sh ?to)
           (not (at ?sh ?from))
           (when (= (load ?sh) 0)
             (increase (fuel_used)
                       (/ (distance ?from ?to) 5)))
           (when (not (= (load ?sh) 0))
             (increase (fuel_used)
                       (/ (distance ?from ?to) 3)))))
```

Note that, as the PDDL grammar does not contemplate an else *branch, the alternative condition has been added explicitly.*

When solving, there are two ways of dealing with these conditional effects. The first one is the easiest, as they can be syntactically removed by creating new auxiliar actions to consider all the possible branches of the conditional effects. Those actions will need to incorporate on its general preconditions the effect preconditions of the particular branch. In the case of Example 15, the action is split in two, as shown in Example 16.

Example 16. *Following the previous example, the* `sail` *action has been split in two:* `sail_empty` *and* `sail_full`*, where the condition of each* when *has been added to the precondition.*

```
(:action sail_empty
  :parameters (?sh - ship ?from - location ?to - location)
  :precondition (and (at ?sh ?from)
                (= (load ?sh) 0))
  :effect (and (at ?sh ?to)
          (not (at ?sh ?from))
          (increase (fuel_used)
                    (/ (distance ?from ?to) 5))))

(:action sail_full
  :parameters (?sh - ship ?from - location ?to - location)
  :precondition (and (at ?sh ?from)
                (not (= (load ?sh) 0)))
  :effect (and (at ?sh ?to)
          (not (at ?sh ?from))
          (increase (fuel_used)
                    (/ (distance ?from ?to) 3))))
```

The second way of dealing with conditional effects, is encoding them directly into the SMT formula. For achieving this, we should consider that actions are now still defined as $a = \langle Pre_a, \mathit{Eff}_a \rangle$, but where Eff_a is now a set of $\langle f, d \rangle$ pairs, and f being a Boolean expression representing the condition of the effect, and d a set of assignments. Unconditional effects would have f set to $\top$.

The encoding previously presented would change, considerably. Equation 5.2 would have to consider f:

$$a^t \wedge f \rightarrow d^{t+1} \qquad \forall \langle f, d \rangle \in \mathit{Eff}_a, \forall a = \langle Pre_a, \mathit{Eff}_a \rangle \in A \tag{5.4}$$

Equation 5.3, that refers to the explanatory frame axioms, would have to also consider the preconditions:

$$x^t \neq x^{t+1} \rightarrow \bigvee_{a=\langle Pre_a, \mathit{Eff}_a\rangle \in A} \left(a^t \wedge (EPC_x(a))^t\right) \tag{5.5}$$

where, given an action $a = \langle Pre_a, \mathit{Eff}_a\rangle$ and a variable x,

$$EPC_x(a) = \bigvee_{f \Rightarrow d \in \mathit{Eff}_a} \{f \mid d \text{ contains an assignment for } x\}$$

that is, the *effect precondition* for the modification of x in action a, where the empty disjunction is defined as *false*. For Boolean variables, the expression $x^t \neq x^{t+1}$ can be written as $(x^t \wedge \neg x^{t+1}) \vee (\neg x^t \wedge x^{t+1})$.

To illustrate this encoding, let us expand Example 15 a bit. Lets say we have one ship named S, and two locations: A and B. We also decide to incorporate conditional effects directly into the encoding. If we only consider the action *sail*, at time step 1 Constraint 5.1 would give the following SMT-LIB [BST10] formulas:

```
(=> sail-S-A-B-1 at-S-A-1)
(=> sail-S-B-A-1 at-S-B-1)
```

where all the atoms appearing are Boolean. Atoms `sail-S-A-B-1` and `sail-S-B-A-1` are used to decide if the action with the parameters is executed or not at time step 1, and atoms `at-S-A-1` and `at-S-B-1` represent if the ship S is physically at location A or B at time step 1. Note that "=>" is the implication symbol in the SMT-LIB language. Now, Constraint 5.4 would be translated to SMT as:

```
(=> sail-S-A-B-1 (and at-S-B-2 (not at-S-A-2)))
(=> (and sail-S-A-B-1 (= load-S-1 0))
       (= fuel-used-2 (+ fuel-used-1 (/ distance-A-B 5))))
(=> (and sail-S-A-B-1 (not (= load-S-1 0)))
       (= fuel-used-2 (+ fuel-used-1 (/ distance-A-B 3))))

(=> sail-S-B-A-1 (and at-S-A-2 (not at-S-B-2)))
(=> (and sail-S-B-A-1 (= load-S-1 0))
       (= fuel-used-2 (+ fuel-used-1 (/ distance-B-A 5))))
(=> (and sail-S-B-A-1 (not (= load-S-1 0)))
       (= fuel-used-2 (+ fuel-used-1 (/ distance-B-A 3))))
```

Note that in this translation terms `distance-A-B` and `distance-B-A` do not have the time step in them. Our compiler is able to detect these static atoms, and as they will not change over time, its useless to create a different variable for them in each time step. Finally Constraint 5.5 would literally translate to:

```
(=> (distinct at-S-A-1 at-S-A-2)
     (or (and sail-S-A-B-1 true) (and sail-S-B-A-1 true)))

(=> (distinct at-S-B-1 at-S-B-2)
     (or (and sail-S-A-B-1 true) (and sail-S-B-A-1 true)))

(=> (distinct fuel-used-1 fuel-used-2)
     (or (and sail-S-A-B-1 (= load-S-1 0))
         (and sail-S-A-B-1 (not (= load-S-1 0)))
         (and sail-S-B-A-1 (= load-S-1 0))
         (and sail-S-B-A-1 (not (= load-S-1 0)))))
```

Note that the true atoms can be simplified at compile time, but has been left out in the example to help the reader match the result with the definition of Constraint 5.5. Note that most conditional effects will end being tautologies as most effects will be unconditional.

5.2.1 Sequential Plans

The encoding presented is still not complete, as we have not restricted what actions can be executed in a time step. We have to specify to what semantics we want to adhere.

The sequential encoding allows exactly one action per time step. This is achieved by imposing an `exactly-one` constraint on the action variables at each time step. We tested some well-known encodings such as the quadratic or the commander encoding [KK07], but we settled with the binary encoding (see [FG10]) as it gave us the best performance. The encoding introduces new variables $B_1, \ldots, B_{\lceil \log_2 n \rceil}$, where $n = |A|$, and associates each variable a_i^t with a unique bit string $s_i \in \{0,1\}^{\lceil \log_2 n \rceil}$. The encoding is:

$$\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{\lceil \log_2 n \rceil} \neg a_i^t \vee \odot(i,j) \tag{5.6}$$

$$\bigvee_{i=1}^{n} a_i^t \tag{5.7}$$

where $\odot(i,j)$ is B_j if the j^{th} bit of the bit string of s_i is 1, and $\neg B_j$ otherwise. The binary encoding of the `at-most-one` constraint (5.6), introduces $\lceil \log_2 n \rceil$ new variables and $n\lceil \log_2 n \rceil$ binary clauses. Together with the `at-least-one` constraint (5.7), we obtain the desired `exactly-one` constraint. Note that this is the same restriction as the one expressed in Section 2.5 but experimentally better, as it trades a good amount of formulas for a small number of new variables.

5.2.2 Parallel Plans

Encodings for two types of parallel plan semantics are considered with this encoding: ∀-step plans, and ∃-step plans. These semantics rely on a notion of interference between actions. For now, consider that an action a_1 can interfere with action a_2 if the execution of a_1 can prevent the execution of a_2 or change its effects. This notion will be discussed with detail in Section 5.4.

∀-step Plans

The notion of parallelism of a ∀-step plan is defined as the possibility of ordering the actions of each time step to any total order. Therefore, at each time step t we simply add a mutex between any pair of interfering actions a_i and a_j:

$$\neg(a_i^t \wedge a_j^t) \text{ if } a_i \text{ affects } a_j \text{ or } a_j \text{ affects } a_i \tag{5.8}$$

∃-step Plans

In ∃-step plans, instead of requiring that actions in a time step can be ordered to any total order (as in ∀-step semantics) it is sufficient that one possible order exists.

The quadratic encoding consists on a fixed (arbitrary) total ordering on the actions imposed beforehand, and the parallel execution of two actions a_i and a_j such that a_i affects a_j is forbidden only if $i < j$:

$$\neg(a_i^t \wedge a_j^t) \text{ if } a_i \text{ affects } a_j \text{ and } i < j \tag{5.9}$$

Since ∃-step plans are less restrictive than ∀-step plans, as they do not require that all orderings of parallel actions result in valid sequential plan, they normally allow more parallelism.

Note that this encoding is based on a fixed ordering of the actions. It is described in detail in [RHN06]. The selected order will determine which actions are allowed to appear in the same time step, and therefore more or less time steps will be needed to reach a valid plan.

5.3 Disabling graph

In [RHN04] Rintanen et al. defined a graph called the *disabling graph*. Thanks to this graph, sets of actions that might not be possible to execute in any total ordering can be identified.

Definition 5.3.1 (Disabling Graph)**.** *A graph $\langle O, E\rangle$ is a disabling graph for a planning problem $\pi = \langle S, A, I, G\rangle$ when $E \subseteq A \times A$ is the set of directed edges so that $\langle a_1, a_2\rangle \in E$ if a_1 can interfere with a_2*

That is, a directed graph where nodes are the grounded actions from the planning problem and an edge exists from action a_1 to action a_2 if the execution of a_1 can interfere with a_2. Figure 5.1 depicts a disabling graph. For a given set of actions there are normally many disabling graphs, as adding an edge to a disabling graph makes it also a disabling graph. For every set of actions and initial state, there is a minimal disabling graph, but computing it is theoretically expensive, as the reachability tests are already PSPACE-hard [RHN06].

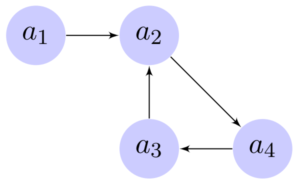

Figure 5.1: An example of a disabling graph

Example 17. *Imagine a logistics problem in PDDL where there are two kind of objects: trucks and packages. The variables that represent properties of the packages are the reason of all edges in the disabling graph. The minimal disabling graph for a problem instance where there are no packages, only trucks, would be an empty graph.*

The connectivity of the disabling graph is important, as it will determine what actions will be allowed to execute in a same time step. In fact, the application of all the actions pertaining to the same strongly connected component of the disabling graph is not possible [RHN06]. Roughly speaking this is because given all the possible total orders between actions, all of them contain a cycle.

This entails that the simultaneous application of a subset of the actions belonging to a strongly connected component can generate a valid plan if the disabling graph made only with the actions of this subset does not contain a cycle. Acyclicity is a sufficient but not a necessary condition for a set of actions to be executable in some order, even for minimal disabling graphs. This is because the edges are independent of the state.

Example 18. *Suppose that action a_2 in Figure 5.2 can never realistically appear in a feasible plan, or that its preconditions makes it impossible to be executed. The appearance of a_2 in the disabling graph generates a strongly connected component and, depending on the rest of the graph, possibly rendering a_1 and a_3 impossible to be executed at the same time step.*

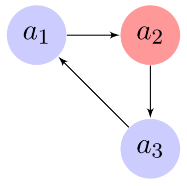

Figure 5.2: An unfeasible action in a disabling graph

The notion of interference is key to determine the density of the disabling graph. If a graph has few edges, its less probable to have strongly connected components and thus more actions will be allowed to appear in parallel. This will imply that probably less steps will be needed to reach a valid plan.

5.3.1 Sort and Cut Order

As previously said, for the ∃-step encoding, the selected order will define what actions can be set in the same time step, and therefore more or less time steps will be needed to find a valid plan.

One preprocessing step that has been tested before establishing any order is to "cut" all cycles made of only two vertices. This means that, if vertex n affected vertex m, and vertex m affected vertex n, m and n are directly prohibited to be scheduled on the same time step, and the two edges removed from the graph. The reason behind this preprocessing is that the quadratic ∃-step encoding will always need to add a mutex between n and m, and cutting these pairs of edges beforehand can potentially minimize the number of strongly connected components afterwards. In practise, this preprocessing only offered small marginal improvements in solving times on the benchmarks used, and therefore it was discarded in the implementation.

A newly implemented order is what we defined as the *Sort and Cut* order, a simple modification of the topological sort algorithm. The topological sort [Kah62] of a directed graph like the disabling graph gives an ordering of its vertices such that for every directed edge $\langle u, v\rangle$ from vertex u to vertex v, $u < v$ (that is, u comes before v in the ordering). Algorithm 7 depicts the topological sort.

Algorithm 7 Topological Sort

Input: L : Empty list, E : The set of all edges, V: The set of all vertices
Output: L contains a topological order of V according to E

```
 1: $S \leftarrow \{v \mid v \in V, \nexists \langle x, v\rangle \in E\}$
 2: while $S \neq \emptyset$ do
 3:     $n \leftarrow$pop($S$)
 4:     push_back($n$,$L$)
 5:     for all vertex $m$ such that $\langle n, m\rangle \in E$ do
 6:         $E \leftarrow E \setminus \langle n, m\rangle$
 7:         if $\nexists\langle x, m\rangle \in E$ then
 8:             $S \leftarrow S \cup \{m\}$
 9:         end if
10:     end for
11: end while
12: if $V \neq \emptyset$ then
13:     return error, there is a cycle
14: end if
```

In line 1 we define set S that contains all vertices with no incoming edges. Function *pop* on line 3 removes and returns a vertex from the set S, and in line 4 vertex n is added to the tail of list L. Line 6 removes all outgoing edges from n and line 8 finally adds to S all vertices that have no incoming edges.

Algorithm 7 depicts the classical topological sort. In our implementation, we modified it to add a set of vertices that records what vertices have been visited. Then, when visiting an edge, if it points to an already visited vertex, we remove it and prohibit concurrent execution of the actions represented by the two vertices. This cuts all the cycles and therefore the order in L adheres to the $\exists$-step semantics.

5.4 Interferences between actions

What we want is to be sure that the generated plans are always valid when serialized. To accomplish this, we add constraints to the problem that guarantee that sets of actions applied simultaneously can be ordered to form an executable plan.

In [Rin09], where conditional effects are considered, an action a_1 is defined to *affect* another action a_2 if a_1 may prevent the execution of a_2 or change its effects. Two actions a_1 and a_2 are considered to *interfere* if a_1 affects a_2 or a_2 affects a_1. In $\forall$-step plans, where all possible serializations must be valid, no two interfering actions can occur in parallel. In the more relaxed notion of parallelism of $\exists$-step plans, where it is only required that no action affects a later one in some total ordering, often much more parallelism is allowed in practice.

For efficiency reasons, typically syntactic (rather than semantic) restrictions are imposed on parallel actions. More precisely, in [Rin09], where only Boolean variables and conditional effects are considered, $a_1 = \langle \mathit{Pre}_1, \mathit{Eff}_1 \rangle$ is determined to affect $a_2 = \langle \mathit{Pre}_2, \mathit{Eff}_2 \rangle$ if, for some variable x,

1. x is set to *true* in d_1 for some $f_1 \Rightarrow d_1 \in \mathit{Eff}_1$, and x occurs in a negative literal in p_2 or occurs in f_2 for some $f_2 \Rightarrow d_2 \in \mathit{Eff}_2$, or

2. x is set to *false* in d_1 for some $f_1 \Rightarrow d_1 \in \mathit{Eff}_1$, and x occurs in a positive literal in p_2 or occurs in f_2 for some $f_2 \Rightarrow d_2 \in \mathit{Eff}_2$.

That is, a_1 affects a_2 if a_1 can impede the execution of a_2, or change its effects. Notice that this relation is not symmetric.

This is a fully syntactic check which can be used to establish sufficient although not necessary conditions for finding serializable parallel plans. We can observe that interference between effects is not considered. This is because, in the case two actions have contradictory effects, any formula encoding a plan with those two actions running in parallel will raise a contradiction. We generalized the previous approach to the case of numeric variables as follows:

Given an action $a_1 = \langle \mathit{Pre}_1, \mathit{Eff}_1 \rangle$, it affects $a_2 = \langle \mathit{Pre}_2, \mathit{Eff}_2 \rangle$ if, for some numeric variable x:

$$\begin{aligned} & \mathit{mod}(x, \mathit{exp}) \in d_1 \text{ for some } \langle f_1, d_1 \rangle \in \mathit{Eff}_1 \\ \wedge & \big(x \text{ occurs in } f_2 \text{ for some } \langle f_2, d_2 \rangle \in \mathit{Eff}_2 \vee \quad x \text{ occurs in } p_2\big) \end{aligned} \tag{5.10}$$

where $\mathit{mod}(x, \mathit{exp})$ is an assignment to variable x.

5.4.1 Plan Serialization

Finally, to obtain a serial plan from the solution, for each time step where there is more than one action, a subgraph of the disabling graph is extracted, containing only the actions at that time step. A valid order between actions can then be computed.

In all the implemented parallel encodings acyclicity is guaranteed between the executed actions. Therefore, the reverse of the order of the topological sort (depicted in Algorithm 7) of the subgraph can be used as a valid serialization.

Example 19. *Consider the disabling graph in Figure 5.1. In a plan with one time step consisting of* $\{a_1, a_2, a_4\}$ *the disabling graph in Figure 5.1 would be extracted.*

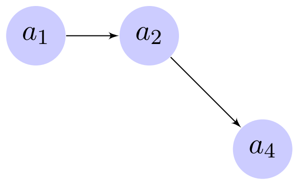

Figure 5.3: The extracted subgraph of the disabling graph

A trivial topological order would be $a_1 > a_2 > a_4$*, and reversing it would give us the sequential valid plan* a_4, a_2, a_1*.*

5.5 Lifted Encoding

As the previously introduced encoding grows considerably with the time horizon, to the point of getting unmanageable instances in large problems, a more compact encoding is proposed, using the theory of uninterpreted functions to express predicates, functions and actions in a compact manner. This encoding is reminiscent of the lifted causal encodings in [KMS96]. In the SMT-LIB standard [BST10], the uninterpreted functions theory is often integrated with other theories, like linear integer or real arithmetic. Uninterpreted functions have no other property than its name and arity, and are only subject to the following axiom: $x_1 = x'_1 \wedge \cdots \wedge x_n = x'_n \rightarrow f(x_1, \dots, x_n) = f(x'_1, \dots, x'_n)$.

To represent a planning problem, now we will use functions of the form $\varphi_a(c^a_1, \dots, c^a_{na})$ to represent actions in A. Now, for actions to be able to talk about state variables, preconditions and effects will also use functions instead of simple variables. Action effects will take the form $\varphi_f(\dots) \mapsto s$, where s is an expression and $\varphi_f(\dots)$ is the state variable (with its possible parameters) that is being assigned a new value. Boolean effects will omit the s and be treated as literals.

Example 20. *Following Example 16, the* `sail_empty` *action could be expressed as*

$$\begin{aligned}\varphi_{sail\_empty}(x, y, z) = \langle &\varphi_{at}(x, y) \wedge \varphi_{load}(x) = 0,\\ &\varphi_{at}(x, z) \wedge \neg\varphi_{at}(x, y) \ \wedge \varphi_{fuel\_used}() \mapsto \varphi_{fuel\_used}() + \varphi_{distance}(y, z)\ /\ 5\rangle\end{aligned}$$

The encoding goes as follows. Every defined object in the problem is mapped to a number. For each ungrounded fluent and action, an uninterpreted function is declared. As each object has been mapped to a number, each parameter of the uninterpreted functions is also being declared as a numeric variable. Also, a new integer parameter is added to each of them, representing a time step. Uninterpreted functions corresponding to Boolean fluents and actions return a Boolean value, whilst the ones for numeric fluents return a numeric value. Moreover, for each action, parameter and time step, a new integer variable is defined, representing the value of that parameter in the action if executed at the corresponding time step.

For example, the Boolean function $\varphi_a(x^t_{a,1}, \dots, x^t_{a,n}, t)$ is used to determine whether action a with parameters $x^t_{a,1}, \dots, x^t_{a,n}$ is executed at time step t. The parameter t is a constant, which is shared between all uninterpreted functions for the actions, predicates and functions in the same time step. Contrarily, $x^t_{a,1}, \dots, x^t_{a,n}$ are variables with finite domains, and constraints are added to restrict their possible values, depending on the parameter types. Regarding Boolean and numeric fluents, no new variables are defined, since their arguments will be either constants or variables occurring in some action.

In this new setting, a state is defined by the value of the uninterpreted functions corresponding to predicates and functions, for a given value of their arguments. Equations (5.1) and (5.2) of the previous encoding are generalized here as:

$$\varphi_a(x^t_{a,1}, \dots, x^t_{a,n}, t) \rightarrow \mathit{Pre}^t_a \qquad \forall a = \langle \mathit{Pre}_a, \mathit{Eff}_a \rangle \in A \tag{5.11}$$

$$\varphi_a(x^t_{a,1}, \dots, x^t_{a,n}, t) \rightarrow \mathit{Eff}^{t+1}_a \qquad \forall a = \langle \mathit{Pre}_a, \mathit{Eff}_a \rangle \in A \tag{5.12}$$

Note that this results in a much more compact encoding, since here we are using variables as arguments of functions, and it is the SMT solver who is in charge of guessing the concrete values of the parameters of the executed actions. The considered set of actions A is now parametrized, and hence similar to that of PDDL, with actions like $\mathit{fly}(x, y, z)$, instead of grounded actions like $\mathit{fly}_{p1,c1,c1}$, $\mathit{fly}_{p1,c1,c2}$, etc. Equation (5.3) is generalized as:

$$\varphi_h(c_1, \dots, c_n, t) \neq \varphi_h(c_1, \dots, c_n, t+1) \rightarrow \bigvee_{\substack{\varphi_a(d_1,\dots,d_m) \in A \text{ s.t.} \\ \varphi_h(e_1,\dots,e_n) \mapsto s \in \mathit{Eff}_{\varphi_a(d_1,\dots,d_m)}}} \Big(\varphi_a(d^t_1, \dots, d^t_m, t) \wedge c_1 = e_1 \wedge \dots \wedge c_n = e_n \Big) \tag{5.13}$$

Note that $e_1, \dots, e_n$ is a permutation of a subset of $d_1, \dots, d_m$ and by imposing equalities $c_i = e_i$ we state that the action is executed with the values that explain the reason of change in φ_h. To help the reader understand the formula, we provide an example. Suppose we have the following simple PDDL problem:

- Objects: `A,B - truck, L1,L2,L3 - loc`
- Boolean fluent: `at(?t - truck, ?l - loc)`
- Numeric fluent: `fuel(?t - truck) - number`
- And the following two actions:
  - `travel(?t - truck, ?from ?to - loc)`
  - `refuel(?x - truck, ?where - loc)`

where `travel` has `(decrease (fuel ?t) 10)` among its effects, and `refuel` has `(increase (fuel ?x) 20)` as its only effect. Constraint 5.13 for the `fuel` function would be encoded into SMT at time step 0 as follows:

```
(=> (distinct (fuel A 0) (fuel A 1))
 (or (and (travel x1_0 x2_0 x3_0 0) (= x1_0 A))
     (and (refuel x4_0 x5_0 0) (= x4_0 A))))

(=> (distinct (fuel B 0) (fuel B 1))
 (or (and (travel x1_0 x2_0 x3_0 0) (= x1_0 B))
     (and (refuel x4_0 x5_0 0) (= x4_0 B))))
```

That is, we are saying that if the fuel of truck `A` (or `B`) has changed, this should be because it has been the protagonist of some action implying a modification in its fuel, namely traveling or refueling. Again, this is much more compact than its grounded counterpart. With respect to the parallelism, for now this encoding only supports the sequential plan semantics.

We do not report results for this encoding, as they are comparable to that of the previous encoding without parallelism and, moreover, the extension of the this encoding to parallel plans is a non trivial task, as the encoding of the explanatory axioms relies on the premise that only one action is executed. Consequently we leave it as future work.

As we said, this encoding is more compact, but what is most important is that it retains most of the problem original structure. It remains to be seen if a parallelized version of this encoding could lead to better results than the encoding without functions. To the best of our knowledge, there were no works using parallelized encodings with uninterpreted functions.

Conditional Effects

Similarly to the previously presented encoding, conditional effects can be removed either by splitting actions, or by encoding the effects. If we choose to encode them, we should consider (as previously proposed) that actions are now still defined as $a = \langle Pre_a, Eff_a \rangle$, but where Eff_a is now a set of $\langle f, d \rangle$ pairs, and f being a Boolean expression representing the condition of the effect, and d a set of assignments. Unconditional effects would have f set to $\top$.

Then, Equation (5.12) would be expressed as:

$$\varphi_a(x^t_{a,1}, \ldots, x^t_{a,n}, t) \wedge f^t \rightarrow d^{t+1} \qquad \forall a = \langle Pre_a, Eff_a \rangle \in A, \forall \langle f, d \rangle \in Eff_a \quad (5.14)$$

where a condition for the effects is added in the premise, and Equation 5.13

would be:

$$\varphi_h(c_1,\ldots,c_n,t) \neq \varphi_h(c_1,\ldots,c_n,t+1) \rightarrow \bigvee_{\substack{\varphi_a(d_1,\ldots,d_m)\in A \text{ s.t.} \\ \varphi_h(e_1,\ldots,e_n)\mapsto s \in Eff_{\varphi_a(d_1,\ldots,d_m)}}} \left(\begin{array}{c} \varphi_a(d_1^t,\ldots,d_m^t,t) \\ \wedge EPC_{\varphi_h(c_1,\ldots,c_n)}(\varphi_a) \\ \wedge\ c_1 = e_1\ \wedge\ \ldots\ \wedge\ c_n = e_n \end{array} \right) \tag{5.15}$$

where:

$$EPC_{\varphi_h(c_1,\ldots,c_n)}(\varphi_a) = \bigvee_{f \Rightarrow d \in Eff_a} \{f \mid d \text{ contains an assignment for } \varphi_h(c_1,\ldots,c_n)\}$$

5.6 Experimental Results

The Petrobras domain was posed as a challenge problem at the International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS 2012). This domain[3] is an interesting real-life problem, that lies in the border between scheduling and planning.

Generically speaking, the problem is described as the need to transport various cargos of goods and tools from two ports to various platforms located in the ocean at various distances. The strips are divided in two parts: Rio de Janeiro and Santos. The basic elements and agents of the problem are: ports, platforms, waiting areas, cargo items and vessels. The actions that can be performed are:

Sail Navigates a ship from one location to another.

Dock Docks a vessel in a port or platform.

Undock Undocks a vessel in a port or platform.

Load Loads a cargo item into the ship.

Unload Unloads a cargo item from the ship to a platform or port.

Refuel Refuels a ship at a refueling location (a port or any specified platform).

[3] *http://icaps12.icaps-conference.org/ickeps/petrobrasdomain.html*

Although the proposal gives various optimization criteria, we only consider the satisfiability of the problem, minimizing the number of time steps.

We compared the performance of the presented approach to that of NumReach [HGSK07], which approximates the reachable domains of the numeric state variables. That is, it generates a set of values $D_t(v)$ for every numeric variable v, so that every value that v can have after t time steps is contained in $D_t(v)$. Then a SAT encoding is generated, by introducing a Boolean variable $a_{v,c,t}$ for every t, v and $c \in D_t(v)$. As we will see, this method is very sensitive to the size of $D_k(v)$.

As NumReach [HGSK07] does not support conditional effects, we created a second model from the original, removing all the conditional effects by splitting actions, as explained in Section 5.5. From now on, those will be referenced as the natural (with conditional effects) and unconditional models (without conditional effects). This will allow to make a comparison between our encoding and NumReach. In order to make the minimum changes from the natural model, we do not change any predicate or function. We only make the following changes: We split the `sail` action in two, namely `sail_empty` and `sail_full`, with the (unconditional) effects corresponding to the case where `current_load` is zero or not, respectively. The `refueling_at_port` and `refueling_at_platform` actions also make use of conditional effects, so they also need to be split in two. Note that although this apparently seems a minor change, it may cause many new variables to appear in the final encoding, as more actions are present.

The input language of the NumReach solver is PDDL, and it has two strategies for solving: NumReach/SAT and NumReach/SMT. The NumReach/SMT approach is similar to NumReach/SAT, except for the encoding of the numeric variables. NumReach uses a different backend solver for each one. For the SAT approach, it uses MiniSAT or ZChaff, but we only used the latest version of MiniSAT (2.2.0) [ES03] in the experiments, as we couldn't find any modern version of ZChaff. For the SMT backend, it was not possible to use any modern version of a SMT solver, and we had to restrict MathSAT 3. This is because NumReach generates the SMT instances in a file format which is different from SMT-LIB and not known by modern SMT solvers. We also could not find any documentation on the format used. For these reasons and due to the poor observed performance, we do not include the results for NumReach/SMT.

During the experiments with NumReach, we found out that MiniSAT dedicated most of its solving time into simplifying the formula. So we decided to execute the same experiments in two ways: instructing MiniSAT not to simplify the input formula, and with the default options. In the tables of results we refer to both solving options as SAT and SAT w/o pre, respectively.

The nature of this problem involves linear integer arithmetic expressions. Translated to the world of SMT, these expressions naturally fall into the QF_LIA logic. In the SMT-LIB standard [BST10], QF_LIA stands for the logic of Quantifier-Free Boolean formulas, with Linear Integer Arithmetic constraints. This logic has a good compromise between expressivity and performance, and is the natural choice for many planning problems, like transportation style problems with limited resources or resource management problems using numeric variables.

To test our encoding (QF_LIA encoding from now on), we created a very similar set of benchmarks to the ones used in [TDT$^+$12]. It consists of 4 groups of generated instances, with an increasing number of cargo items, ranging from 1 to 15. Every cargo is assigned randomly to one of the two ports, and each ship is randomly docked in one of them. The groups differ in the number of ships and in the total fuel capacity of each ship:

- Group A: 3 ships with 600 liters of fuel capacity.
- Group B: 10 ships with 600 liters of fuel capacity.
- Group C: 10 ships with 800 liters of fuel capacity.
- Group D: 10 ships with 1000 liters of fuel capacity.

The experiments were run on a cluster of machines, running the CentOS operating System, equipped with Intel® Xeon® E3-1220v2 Processors at 3.10 GHz with Turbo Boost disabled, and 8GB of main memory.

For each instance, we made executions for the encoding with the three semantics: sequential, $\forall$-step and $\exists$-step, with two SMT solvers via API: Yices-1.0.38 [DDM06a] and Z3-4.3.1 [dMB08c]. The results depicted are from the Yices executions, as although it wasn't always faster, it solved more instances than Z3. The executions were made through the APIs, because when we tried to use plain files we found that some of the generated files for the biggest instances were too big for the solvers, spanning to some gigabytes.

Tables 5.1 to 5.4 show the execution time in seconds and number of time steps checked for each group of instances. TO denotes that the solver could not find a plan in the given time of one hour, and the fastest solver for each instance is highlighted. Between parenthesis there is the last time step checked by the solver (which corresponds to the length of the shortest

Inst	QF_LIA encoding						NumReach			
	Sequential		∀-step		∃-step		SAT		SAT w/o pre	
A1	2.73	(5)	45.27	(5)	22.95	(5)	105.77	(6)	**1.20**	(6)
A2	45.67	(13)	76.37	(8)	40.51	(8)	780.93	(9)	**13.65**	(9)
A3	TO	(17)	105.44	(10)	55.29	(9)	1231.98	(10)	**26.28**	(10)
A4	TO	(17)	1727.77	(13)	2674.75	(12)	2067.24	(11)	**137.47**	(11)
A5	TO	(19)	TO	(13)	TO	(13)	2992.79	(12)	**243.45**	(12)
A6	TO	(19)	TO	(14)	TO	(13)	TO	(13)	**933.22**	(14)
A7	TO	(19)	TO	(14)	TO	(13)	TO	(13)	**431.90**	(13)
A8	TO	(20)	TO	(14)	TO	(14)	TO	(13)	TO	(14)
A9	TO	(18)	TO	(14)	TO	(13)	TO	(12)	TO	(14)
A10	TO	(19)	TO	(15)	TO	(14)	TO	(12)	TO	(12)
A11	TO	(20)	TO	(14)	TO	(14)	TO	(12)	TO	(12)
A12	TO	(20)	TO	(14)	TO	(15)	TO	(12)	TO	(12)
A13	TO	(20)	TO	(15)	TO	(15)	TO	(12)	TO	(12)
A14	TO	(19)	TO	(14)	TO	(14)	TO	(12)	TO	(12)
A15	TO	(21)	TO	(15)	TO	(15)	TO	(12)	TO	(12)

Table 5.1: Execution times for group A in seconds, and number of time steps checked.

step-wise plan found for the solved instances). Note that it is not clear for us what notion of interference or parallelism is NumReach using, so the plan lengths between correct solutions given by NumReach and our encoding for the same instance may differ. Table 5.5 summarizes how many instances each solving approach could finish in the given time, and among those in how many it was the fastest. The natural model could not be directly compared to the unconditional model using NumReach (recall that it does not support conditional effects), so the NumReach results shown in Tables 5.1 to 5.5 are from the unconditional model.

If we look at the NumReach/SAT executions, all the instances have a better solving time without simplifying the input formula. But, although we observe an speedup of more than one order of magnitude on most of the solved instances, only a few more instances can be solved without preprocessing, due to the combinatorial explosion. This indicates that the problem is inherently hard.

After analyzing the computed interferences between actions, we could see that the problem is highly parallel in the number of ships. Ships can operate independently, with the only limitation of the docking space. This can be seen for example in instance D7: in 4 time steps, 7 cargo items

Inst	QF_LIA encoding						NumReach			
	Sequential		∀-step		∃-step		SAT		SAT w/o pre	
B1	11.47	(5)	154.01	(5)	79.20	(5)	366.29	(6)	**9.02**	(6)
B2	154.53	(10)	158.88	(5)	82.80	(5)	544.15	(6)	**12.76**	(6)
B3	TO	(12)	162.81	(5)	86.02	(5)	1344.00	(7)	**35.97**	(7)
B4	TO	(13)	168.30	(5)	**89.74**	(5)	2761.95	(8)	151.22	(8)
B5	TO	(13)	173.05	(5)	**93.77**	(5)	2864.55	(8)	167.65	(8)
B6	TO	(14)	178.21	(5)	**96.92**	(5)	2952.88	(8)	173.83	(8)
B7	TO	(14)	183.16	(5)	**101.81**	(5)	TO	(10)	TO	(10)
B8	TO	(14)	300.96	(7)	**189.73**	(7)	TO	(9)	TO	(9)
B9	TO	(14)	TO	(8)	TO	(7)	TO	(9)	TO	(9)
B10	TO	(15)	748.626	(8)	**358.57**	(7)	TO	(9)	TO	(9)
B11	TO	(14)	TO	(8)	TO	(8)	TO	(9)	TO	(9)
B12	TO	(14)	TO	(8)	TO	(8)	TO	(9)	TO	(9)
B13	TO	(14)	TO	(8)	TO	(8)	TO	(9)	TO	(9)
B14	TO	(16)	TO	(9)	TO	(8)	TO	(9)	TO	(9)
B15	TO	(14)	TO	(9)	TO	(8)	TO	(9)	TO	(9)

Table 5.2: Execution times for group B in seconds, and number of time steps checked.

are transported from the port of origin to its destination. The difference of 2 time steps between D7 and D8 is caused only by the docking space capacities.

Note also that ∃-step plans are easier to find than ∀-step plans in this domain. However, contrarily to what could be expected, in most of the cases they are not shorter. This is due to the nature of the domain: as said, ships can operate independently and hence, in many cases, requiring parallel actions to result in a valid plan if putting them to any total order, is not stronger than requiring this for some fixed order. On the other hand, under our approach, with the natural model the solver found a solution for 7 more instances than with the unconditional model.

Intuitively, a higher ship fuel capacity should make the problem easier, as less actions will be necessary as ships will need to refuel less often. Instead, it is interesting to note that for NumReach/SAT the groups C and D become the hardest instances. This is because with the higher numbers, state-space exploration seems to grow too large to be manageable, as we suspected.

Other works have provided efficient solutions to the Petrobras challenge proposal. In [TDT$^+$12, BZ13] various heuristic (incomplete) solvers are used to solve the Petrobras challenge under different optimization criteria.

Inst	QF_LIA encoding						NumReach			
	Sequential		∀-step		∃-step		SAT		SAT w/o pre	
C1	11.58	(5)	154.46	(5)	79.05	(5)	510.45	(6)	**14.78**	(6)
C2	149.59	(10)	159.05	(5)	81.59	(5)	744.29	(6)	**21.80**	(6)
C3	TO	(13)	163.17	(5)	86.21	(5)	1826.95	(7)	**71.88**	(7)
C4	TO	(13)	168.05	(5)	**89.76**	(5)	TO	(9)	TO	(9)
C5	TO	(13)	173.13	(5)	**93.44**	(5)	TO	(8)	TO	(9)
C6	TO	(13)	178.25	(5)	**97.28**	(5)	TO	(8)	TO	(8)
C7	TO	(14)	183.30	(5)	**101.06**	(5)	TO	(8)	TO	(8)
C8	TO	(14)	298.40	(7)	**168.80**	(7)	TO	(8)	TO	(8)
C9	TO	(14)	TO	(8)	TO	(7)	TO	(8)	TO	(8)
C10	TO	(14)	758.50	(8)	**351.54**	(7)	TO	(8)	TO	(8)
C11	TO	(14)	TO	(8)	TO	(8)	TO	(8)	TO	(8)
C12	TO	(14)	TO	(8)	TO	(8)	TO	(8)	TO	(8)
C13	TO	(14)	TO	(8)	TO	(8)	TO	(8)	TO	(8)
C14	TO	(17)	TO	(9)	TO	(8)	TO	(8)	TO	(8)
C15	TO	(14)	TO	(9)	TO	(8)	TO	(8)	TO	(8)

Table 5.3: Execution times for group C in seconds, and number of time steps checked.

In Table 5.6 we can see that, with the unconditional model, only 3 instances less are solved than with SGPlan [CWH06]. But if we consider the natural model, 7 more instances are solved, outperforming SGPlan.

We can conclude that the proposed encoding makes use of SMT to tightly integrate arithmetic into the problem, where other approximations rely into making state-space exploration on the numerical variables, or loosely integrate external solvers for evaluating arithmetic constraints (and therefore not being able to infer anything from the numerical side). Our approximation seems to be competitive with other exact and complete methods for planning with resources on this problem, and also with some incomplete (heuristic) ones. In particular, we have obtained better results than NumReach [HGSK07] and similar results to SGPlan [CWH06]. We have seen that the method of [HGSK07], which is based on approximating the reachable domains of numeric variables, is very sensitive to the number of distinct possible values, and it is not well-suited for this real real-life problem.

Although SAT and SMT solvers have generic preprocessing steps to simplify the input formulas, we observed that for MiniSAT those were harmful for this problem. Nevertheless, it would be interesting to consider some more ad hoc preprocessing steps to help reduce the search space.

Inst	QF_LIA encoding						NumReach			
	Sequential		∀-step		∃-step		SAT		SAT w/o pre	
D1	**11.58**	(5)	154.47	(5)	78.80	(5)	1097.26	(6)	24.71	(6)
D2	139.56	(10)	158.66	(5)	81.80	(5)	1777.23	(6)	**37.44**	(6)
D3	TO	(12)	163.36	(5)	**85.86**	(5)	TO	(7)	144.29	(7)
D4	TO	(13)	168.01	(5)	**89.39**	(5)	TO	(7)	TO	(7)
D5	TO	(13)	173.36	(5)	**93.17**	(5)	TO	(7)	TO	(7)
D6	TO	(13)	177.89	(5)	**97.26**	(5)	TO	(7)	TO	(7)
D7	TO	(14)	182.66	(5)	**100.92**	(5)	TO	(7)	TO	(7)
D8	TO	(14)	302.91	(7)	**240.33**	(7)	TO	(7)	TO	(7)
D9	TO	(14)	TO	(8)	TO	(8)	TO	(7)	TO	(7)
D10	TO	(14)	762.78	(8)	**333.21**	(7)	TO	(7)	TO	(7)
D11	TO	(15)	TO	(8)	TO	(8)	TO	(7)	TO	(7)
D12	TO	(14)	TO	(8)	TO	(8)	TO	(7)	TO	(7)
D13	TO	(14)	TO	(8)	TO	(8)	TO	(7)	TO	(7)
D14	TO	(17)	TO	(9)	TO	(8)	TO	(7)	TO	(7)
D15	TO	(15)	TO	(9)	TO	(8)	TO	(7)	TO	(7)

Table 5.4: Execution times for group D in seconds, and number of time steps checked.

60 instances	QF_LIA encoding			NumReach	
	Sequential	∀-step	∃-step	SAT	SAT w/o pre
Total solved	8	31	31	16	19
Faster instances	1	0	19	0	14

Table 5.5: Summary of the results.

Instance set	Unconditional model			Natural model			SGPlan
	Sequential	∀-step	∃-step	Sequential	∀-step	∃-step	
Group A	2	4	4	3	5	5	6
Group B	2	9	9	2	11	11	6
Group C	2	9	9	2	11	11	10
Group D	2	9	9	2	11	11	12
Total	8	31	31	9	38	38	34

Table 5.6: Number of instances solved by each approximation.

Chapter 6

A Semantic Notion of Interference for Planning Modulo Theories

Performing only syntactic checks for detecting interferences between actions like the previous approaches seems too restrictive for numeric variables. Our assumption is that with less edges in the disabling graph, less strongly connected components will form and thus more actions will be able to be scheduled to execute at the same time step. At the same time, formula sizes will lower, less time steps will be needed and thus more problems will be able to be solved.

One could try to build a set of rules that brings the number of edges of the disabling graph close to its minimal. To be effective, this set of rules would need to reason about the theory, and therefore they would depend on the theory considered. In addition, every combination of theories would also need its own set of rules. For this reason, in this chapter we propose a new idea, which is to use SMT technology to perform semantic checks of interference at compile time, in order to increase the amount of parallelization of plans.

The method presented is independent of any test suite or theory and does not require any special purpose algorithm, as it relies on encoding the possible interference situations between pairs of actions as SMT formulas and checking their satisfiability, by calling an SMT solver, at compile time.

A new and relaxed notion of interference for the parallel execution of actions is introduced, suitable for both $\forall$-step and $\exists$-step semantics. We prove its correctness and motivate its usefulness with some examples.

Finally, an encoding that takes advantage of this semantic notion of interference is presented, together with some experimental results that show its usefulness.

6.1 A Semantic Notion of Interference

In the following, we consider plans as sequences of *sets of actions*. A set of actions planned at the same time is commonly called a *happening* [FL03]. Two actions can be concurrently planned if, roughly, they do not interfere. It is commonly accepted that two actions are non-interfering only if the composition of their effects is commutative, and there is no interaction between effects and preconditions. In [FL03, GSS08] the state resulting from executing a happening is defined as the one obtained after applying the composition of effects of the actions in the happening.

Example 21. *Let* $a = \langle \top, \{x \mapsto x + y + z\} \rangle$ *and* $b = \langle \top, \{x \mapsto x + 1, y \mapsto y + 1, z \mapsto z - 1\} \rangle$*. These actions do not interfere, as their preconditions are true (and hence cannot interact with effects) and their effects commute: executing first* a *and then* b*, as well as executing first* b *and then* a*, produces the same effect, which is that of an action of the form* $\langle \top, \{x \mapsto x + y + z + 1, y \mapsto y + 1, z \mapsto z - 1\} \rangle$*,*

Example 22. *Let* $c = \langle \top, \{x \mapsto x+y+z\} \rangle$ *and* $d = \langle \top, \{x \mapsto x+1, y \mapsto y+2, z \mapsto z - 1\} \rangle$*. These actions interfere, since their effects do not commute. Executing first* c *and then* d *is equivalent to executing* $\langle \top, \{x \mapsto x + y + z + 1, y \mapsto y + 2, z \mapsto z - 1\}$*, whereas executing first* d *and then* c *is equivalent to executing* $\langle \top, \{x \mapsto x + y + z + 2, y \mapsto y + 2, z \mapsto z - 1\}$*. Then they would not be allowed to be planned in parallel.*

Thanks to the commutativity requirement, effects of non-interfering actions can be composed in any order, allowing parallel plans to be serialized in any order, while preserving their semantics. This adheres to the $\forall$-step semantics of [RHN06], but it does not lift to the $\exists$-step semantics (introduced in the same work for the Boolean case), where it is only necessary that actions can be executed it at least one order, making it possible to increase the number of parallel actions.

The main contributions on this chapter are a new relaxed semantics of *happening execution*, and a new notion of interference that are suitable for both $\forall$-step and $\exists$-step semantics [RHN06], in the context of PMT.

As we show in Section 6.2, the proposed notion of interference can moreover be fully checked at compile time by means of satisfiability checks. As far as we know, previous approaches used syntactic or limited semantic approaches [KW99a, FL03, GSS08]. Note that non-interference of actions such those in Example 21 cannot be easily determined syntactically.

In the rest of this section we introduce the new semantics of happening execution, define the new notion of interference, and prove that their combination is valid for both $\forall$-step and $\exists$-step semantics.

Definition 6.1.1 (Commuting Assignments)**.** *Assignments* $\{x \mapsto exp_1\}$ *and* $\{x \mapsto exp_2\}$ commute*, for a variable x and two expressions (terms) exp_1 and exp_2 over a state space S modulo T, if* $T \models (exp_2\{x \mapsto exp_1\} = exp_1\{x \mapsto exp_2\})$*.*

Example 23. *If T is the theory of real numbers, then* $\{x \mapsto x+1\}$ *and* $\{x \mapsto x-2\}$ *commute, since* $T \models ((x-2)+1 = (x+1)-2)$*, whereas* $\{x \mapsto x+1\}$ *and* $\{x \mapsto x*2\}$ *do not commute, since* $T \not\models ((x+1)*2 = (x*2)+1)$*.*

Definition 6.1.2 (Simply Commuting Actions)**.** *We will refer to a set* $A = \{a_1, \ldots, a_n\}$ *of actions as* simply commuting*, for a state space S modulo T, if for every variable $x \in var(S)$ and every pair of assignments* $\{x \mapsto exp_1\}$ *and* $\{x \mapsto exp_2\}$ *in the effects of actions in A,* $\{x \mapsto exp_1\}$ *and* $\{x \mapsto exp_2\}$ *commute.*

Definition 6.1.3 (Happening Action)**.** *Let* $A = \{a_1, \ldots, a_n\}$ *be a set of simply commuting actions. We define the* happening action *for A as an action* $h(A) = \langle Pre_{h(A)}, \sigma_{h(A)} \rangle$ *with*

$$Pre_{h(A)} = \bigwedge_{a \in A} Pre_a$$

and

$$\sigma_{h(A)} = \bigcup_{x \in var(S)} \{\sigma_{x,1} \circ \cdots \circ \sigma_{x,n}\}$$

where $\sigma_{x,i}$, for i in $1..n$, is the mapping of variable x in the effects of action a_i, and $\circ$ denotes the composition of functions.

Note that the effects on each variable can be composed in any order, because of the commutation requirement. Therefore, $h(A)$ is well-defined.

Definition 6.1.4 (Happening Execution)**.** *Let* $A = \{a_1, \ldots, a_n\}$ *be a set of simply commuting actions. Then, the state resulting from the execution of the happening A in state s, denoted $app_A(s)$, is defined as $app_{h(A)}(s)$, where $h(A)$ is the happening action corresponding to A.*

Note that if some action in A is not applicable in state s then $app_A(s)$ is undefined.

Example 24. *Let $a = \langle \top, \{x \mapsto x+1, y \mapsto y+1\}\rangle$ and $b = \langle \top, \{y \mapsto y+x\}\rangle$. Then $app_{\{a,b\}}(s)$, for a state s, is $app_{h(\{a,b\})}(s)$, with $h(\{a,b\}) = \langle \top, \{x \mapsto x+1, y \mapsto (y+x)+1\}\rangle$.*

A key difference with the transition functions for happenings defined in [FL03] and [GSS08] is that , instead of considering the *composition of functions* (i.e., the composition of effects of actions, seen as functions on all variables), we are considering the *function of compositions* (i.e., the function defined by the composition of assignments to each single variable across all actions). We consider the possibility of composing the effects on each variable in any order, as a minimal requirement to be able to serialize plans in some order (see definitions and proofs below). Our aim is to show that the proposed semantics for happenings allows us to increase the number of parallel actions in the context of $\exists$-step plans, where parallel semantics and interference notions of existing approaches to numeric planning are too restrictive.

Definition 6.1.5 (Affecting Action)**.** *Given two actions $a = \langle Pre_a, \sigma_a\rangle$ and $b = \langle Pre_b, \sigma_b\rangle$, for a state space S modulo T, we consider a to* affect *b if*

1. *$Pre_a \wedge Pre_b \wedge \neg(Pre_b\sigma_a)$ is T-satisfiable, or*
2. *either a and b are not simply commuting, or $Pre_a \wedge Pre_b \wedge \neg(x\sigma_{h(\{a,b\})} = x\sigma_b\sigma_a)$ is T-satisfiable for some variable $x \in var(S)$, where $h(\{a,b\})$ denotes the happening action for a and b,*

that is, a can impede the execution of b, or they are not simply commuting, or they are simply commuting but executing first a and then b has a different effect than that of the happening $\{a,b\}$.

Recall that $h(\{a,b\})$ is defined only for simply commuting actions.

Example 25. *Following Example 24, where actions a and b are simply commuting, we have that a affects b since $y\sigma_{h(\{a,b\})} = (y+x)+1$, while $y\sigma_b\sigma_a = (y+x)\sigma_a = (y+1)+(x+1)$, and thus $Pre_a \wedge Pre_b \wedge \neg(y\sigma_{h(\{a,b\})} = y\sigma_b\sigma_a)$ is T-satisfiable. On the contrary, b does not affect a, since the preconditions of both actions are true, $x\sigma_{h(\{a,b\})} = x+1 = x\sigma_b\sigma_a$, and $y\sigma_a\sigma_b = (y+1)\sigma_b = (y+x)+1$. This is to say that the effect of the happening $\{a,b\}$ is the same as executing first b and then a, but not first a and then b. In fact, in this example we have $app_{\{a,b\}}(s) = app_{b;a}(s) \neq app_{a;b}(s)$ for all s.*

Definition 6.1.6 (Interference)**.** *Given two actions a and b, we consider a and b to* interfere *if a affects b or b affects a.*

6.1.1 ∀-Step Plans

Lack of interference guarantees that actions in a happening can be executed sequentially in any total order and that the final state is independent of the ordering (see Theorem 6.1.15). The notion of ∀-step plan, defined in [RHN06], can be generalized to the setting of PMT as follows.

Definition 6.1.7 (∀-Step Plan)**.** *Given a set of actions A and an initial state I, for a state space S modulo T, a* ∀-step plan *for A and I is a sequence $P = \langle A_0, \dots, A_{l-1}\rangle$ of sets of actions for some $l \geq 0$, such that there is a sequence of states $s_0, \dots, s_l$ (the execution of P) such that*

1. *$s_0 = I$, and*
2. *for all $i \in \{0, \dots, l-1\}$ and every total ordering $a_1 < \cdots < a_n$ of A_i, $app_{a_1;\dots;a_n}(s_i)$ is defined and equals s_{i+1}.*

Lemma 6.1.8. *Let A be a set of actions, for a state space S modulo T, and let $s \in S$ be a state such that all actions in A are applicable in s. Then $app_{a_1;\dots;a_n}(s)$ is defined for every ordering $a_1 < \cdots < a_n$ of A such that if $a_i < a_j$ then a_i does not affect a_j.*

Proof. By induction on the number of actions n in A. If $n = 1$ we are trivially done. If $n \geq 2$, consider any ordering $a_1 < \cdots < a_n$ of A such that if $a_i < a_j$ then a_i does not affect a_j. Let $a_1 = \langle Pre_{a_1}, \sigma_{a_1}\rangle$. First of all we show, by contradiction, that $app_{a_i}(app_{a_1}(s))$ is defined for every $a_i = \langle Pre_{a_i}, \sigma_{a_i}\rangle$ such that $a_1 < a_i$. Suppose that $T, app_{a_1}(s) \not\models Pre_{a_i}$, i.e., that a_i is not applicable after applying a_1 in state s. This is equivalent to say that $T, s \not\models Pre_{a_i}\sigma_{a_1}$ and, since s is an assignment, to $eval^s_T(Pre_{a_i}\sigma_{a_1}) = \bot$. Now, by assumption, we have $T, s \models Pre_{a_1}$ and $T, s \models Pre_{a_i}$, since all actions are applicable in state s. Therefore, $T, s \models Pre_{a_1} \wedge Pre_{a_i} \wedge \neg(Pre_{a_i}\sigma_{a_1})$, i.e., $Pre_{a_1} \wedge Pre_{a_i} \wedge \neg(Pre_{a_i}\sigma_{a_1})$ is T-satisfiable, contradicting that a_1 does not affect a_i. Finally, since all actions a_i such that $a_1 < a_i$ are applicable in state $app_{a_1}(s)$, by the induction hypothesis we have that $app_{a_2;\dots;a_n}(app_{a_1}(s))$ is defined for any ordering $a_2 < \cdots < a_n$ of $A \setminus \{a_1\}$ such that if $a_i < a_j$ then a_i does not affect a_j, and hence so is $app_{a_1;a_2;\dots;a_n}(s)$ for the ordering we have considered. □

Lemma 6.1.9. *Let a and b be two simply commuting actions, for a state space S modulo T, such that a does not affect b, and let $s \in S$ be a state such that a and b are applicable in s. Then $app_{\{a,b\}}(s) = app_{a;b}(s)$.*

Proof. Let $a = \langle Pre_a, \sigma_a \rangle$ and $b = \langle Pre_b, \sigma_b \rangle$. Since a and b are applicable in s, we have that $app_{\{a,b\}}(s)$ is defined. Moreover, since a does not affect b, by Lemma 6.1.8 we have that $app_{a;b}(s)$ is defined.

We conclude by showing that $app_{\{a,b\}}(s)[x] = app_{a;b}(s)[x]$ for every variable x. Recall that $app_{\{a,b\}}(s) = app_{h(\{a,b\})}(s)$, where $h(\{a,b\})$ denotes the happening action for a and b. Now, by definition of application, we have $app_{h(\{a,b\})}(s)[x] = eval^s_T(x\sigma_{h(\{a,b\})})$ and $app_{a;b}(s)[x] = eval^s_T(x\sigma_b\sigma_a)$, for every variable x. On the other hand, since a does not affect b, $Pre_a \wedge Pre_b \wedge \neg(x\sigma_{h(\{a,b\})} = x\sigma_b\sigma_a)$ is T-unsatisfiable. And, since a and b are both applicable in state s, we have $T, s \models Pre_a$ and $T, s \models Pre_b$. Therefore $T, s \models Pre_a \wedge Pre_b \wedge (x\sigma_{h(\{a,b\})} = x\sigma_b\sigma_a)$, and thus $eval^s_T(x\sigma_{h(\{a,b\})}) = eval^s_T(x\sigma_b\sigma_a)$, which lets us conclude. □

Lemma 6.1.10. *Let a and b be two non-interfering actions, for a state space S modulo T, and let $s \in S$ be a state such that a and b are applicable in s. Then $app_{a;b}(s) = app_{b;a}(s)$.*

Proof. Since a and b are non-interfering, then they are simply commuting, and neither a affects b nor b affects a. Then, by Lemma 6.1.9, we have $app_{\{a,b\}}(s) = app_{a;b}(s)$, and $app_{\{a,b\}}(s) = app_{b;a}(s)$. □

Lemma 6.1.11. *Let A be a set of non-interfering actions, for a state space S modulo T, and let $s \in S$ be a state such that all actions in A are applicable in s. Then $app_{a_1;\ldots;a_n}(s)$ is the same state for every total ordering $a_1 < \cdots < a_n$ of A.*

Proof. Since actions in A are non-interfering, and applicable in state s, by Lemma 6.1.8 we have that $app_{a_1;\ldots;a_n}(s)$ is defined for any total ordering $a_1 < \cdots < a_n$ of A. We conclude by showing that any two consecutive actions in the sequence $a_1;\ldots;a_n$ can be permuted, preserving the final state. Consider any two consecutive actions a_i and a_{i+1} in the sequence $a_1;\ldots;a_n$. Since $app_{a_1;\ldots;a_n}(s)$ is defined, so is $app_{a_1;\ldots;a_i}(s)$, and a_i is applicable in state $app_{a_1;\ldots;a_{i-1}}(s)$ (in case that $i = 1$, let $app_{a_1;\ldots;a_{i-1}}(s)$ denote the state s). Now, since actions in A are non-interfering, by Lemma 6.1.8 we have that $app_{a_1;\ldots;a_{i-1};a_{i+1}}(s)$ is also defined, so a_{i+1} is also applicable in state $app_{a_1;\ldots;a_{i-1}}(s)$. Finally, by the Lemma 6.1.10, it follows that

$$app_{a_i;a_{i+1}}(app_{a_1;\ldots;a_{i-1}}(s)) = app_{a_{i+1};a_i}(app_{a_1;\ldots;a_{i-1}}(s))$$

which, by definition of application, is equivalent to

$$app_{a_1;\ldots;a_{i-1};a_i;a_{i+1};\ldots;a_n}(s) = app_{a_1;\ldots;a_{i-1};a_{i+1};a_i;\ldots;a_n}(s)$$

. □

Lemma 6.1.12. *Let A be a set of simply commuting actions, for a state space S modulo T, such that $|A| \geq 2$. Then, for every action $a \in A$, we have that a and $h(A \setminus \{a\})$ are simply commuting, and $h(\{a, h(A \setminus \{a\})\}) = h(A)$.*

Proof. Let $A = \{a_1, a_2, \ldots, a_n\}$, $a = a_1$ and $A' = A \setminus \{a\} = \{a_2, \ldots, a_n\}$. According to the definition of happening action, we have that $\sigma_{h(A')} = \cup_{x \in var(S)} \{\sigma_{x,2} \circ \cdots \circ \sigma_{x,n}\}$, where $\sigma_{x,i}$, for i in $2..n$, is the mapping of variable x in the effects of action a_i. Now, since composition of functions is associative, we have that $\sigma_{x,1} \circ (\sigma_{x,2} \circ \cdots \circ \sigma_{x,n}) = \sigma_{x,1} \circ \sigma_{x,2} \circ \cdots \circ \sigma_{x,n}$ for every variable x, being $\sigma_{x,1}$ the mapping of variable x in the effects of action a_1. And, since actions in A are simply commuting, we have that $\sigma_{x,1} \circ \sigma_{x,2} \circ \cdots \circ \sigma_{x,n} = (\sigma_{x,2} \circ \cdots \circ \sigma_{x,n}) \circ \sigma_{x_1}$, which lets us conclude that a and $h(A')$ are simply commuting.

Now, provided that a and $h(A')$ are simply commuting, in order to prove that the happening actions $h(\{a, h(A')\})$ and $h(A)$ are equivalent, we need to show that they have equivalent preconditions and effects. For preconditions, we have $Pre_{h(\{a,h(A')\})} = Pre_a \wedge Pre_{h(A')} = \wedge_{a \in A} Pre_a = Pre_{h(A)}$. For effects, we have $\sigma_{h(\{a,h(A')\})} = \cup_{x \in var(S)} \{\sigma_{x,1} \circ (\sigma_{x,2} \circ \cdots \circ \sigma_{x,n})\}$ which, as seen before, is equivalent to $\cup_{x \in var(S)} \{\sigma_{x,1} \circ \sigma_{x,2} \circ \cdots \circ \sigma_{x,n}\}$. □

Lemma 6.1.13. *Let a, b and c be three simply commuting actions, for a state space S modulo T. If a affects neither b nor c, then a does not affect the happening action $h(\{b, c\})$.*

Proof. Let $a = \langle Pre_a, \sigma_a \rangle$, $b = \langle Pre_b, \sigma_b \rangle$ and $c = \langle Pre_c, \sigma_c \rangle$. We need to prove that

1. $Pre_a \wedge Pre_{h(\{b,c\})} \wedge \neg(Pre_{h(\{b,c\})}\sigma_a)$ is T-unsatisfiable,
2. a and $h(\{b, c\})$ are simply commuting, and
3. $Pre_a \wedge Pre_{h(\{b,c\})} \wedge \neg(x\sigma_{h(\{a,h(\{b,c\})\})} = x\sigma_{h(\{b,c\})}\sigma_a)$ is T-unsatisfiable for every variable $x \in var(S)$.

For condition 1, since $Pre_{h(\{b,c\})} = Pre_b \wedge Pre_c$, we have that $Pre_a \wedge Pre_{h(\{b,c\})} \wedge \neg(Pre_{h(\{b,c\})}\sigma_a) = Pre_a \wedge Pre_b \wedge Pre_c \wedge \neg((Pre_b \wedge Pre_c)\sigma_a) = (Pre_a \wedge Pre_b \wedge Pre_c \wedge \neg(Pre_b\sigma_a)) \vee (Pre_a \wedge Pre_b \wedge Pre_c \wedge \neg(Pre_c\sigma_a))$. Now assume that $Pre_a \wedge Pre_b \wedge Pre_c \wedge \neg(Pre_b\sigma_a)$ is T-satisfiable (the other case is analogous). Then $Pre_a \wedge Pre_b \wedge \neg(Pre_b\sigma_a)$ would also be T-satisfiable, contradicting that a does not affect b.

Condition 2 follows directly from Lemma 6.1.12.

For condition 3, we proceed by contradiction. Let us assume that $Pre_a \wedge Pre_{h(\{b,c\})} \wedge \neg(x\sigma_{h(\{a,h(\{b,c\})\})} = x\sigma_{h(\{b,c\})}\sigma_a)$ is T-satisfiable for some variable $x \in var(S)$. Then, by definition of happening action, we have $Pre_a \wedge Pre_b \wedge Pre_c \wedge \neg(x(\sigma_{x,b} \circ \sigma_{x,c} \circ \sigma_{x,a}) = x(\sigma_{x,b} \circ \sigma_{x,c})\sigma_a)$ is T-satisfiable, where $\sigma_{x,a}$, $\sigma_{x,b}$ and $\sigma_{x,c}$ are the mappings of variable x in the effects of actions a, b and c, respectively. So there exists some assignment s such that $T, s \models Pre_a$, $T, s \models Pre_b$, $T, s \models Pre_c$, and $eval^s_T(x\sigma_{x,b}\sigma_{x,c}\sigma_{x,a}) \neq eval^s_T(x\sigma_{x,b}\sigma_{x,c}\sigma_a)$. This implies the existence of some variable y different from x such that $\sigma_a[y] \neq y$. Moreover, since $\sigma_{x,b}$ and $\sigma_{x,c}$ are substitutions replacing only variable x, y must be a variable in $x\sigma_{x,b}$ or in $x\sigma_{x,c}$ and, necessarily, $eval^s_T(x\sigma_{x,b}\sigma_{x,a}) \neq eval^s_T(x\sigma_{x,b}\sigma_a)$ or $eval^s_T(x\sigma_{x,c}\sigma_{x,a}) \neq eval^s_T(x\sigma_{x,c}\sigma_a)$. But this, together with $T, s \models Pre_a$, $T, s \models Pre_b$ and $T, s \models Pre_c$, contradicts a affecting neither b nor c. $\square$

Lemma 6.1.14. *Let A be a set of actions, and a an action, for a state space S modulo T, such that the actions in $A \cup \{a\}$ are simply commuting. If a affects none of the actions in A, then a does not affect the happening action $h(A)$.*

Proof. Let $A = \{a_1, \ldots, a_n\}$. We proceed by induction on the number of actions n in A. If $n = 1$ then we are trivially done, since $h(A) = a_1$ and, by assumption, a does not affect a_1. If $n \geq 2$, let $A' = A \setminus \{a_1\}$. Then a neither affects a_1 nor the happening action $h(A')$ (by the induction hypothesis). Moreover, since actions in $A \cup \{a\}$ are simply commuting, so are a, a' and $h(A')$. Then, by Lemma 6.1.13, we have that a does not affect $h(\{a', h(A')\})$ and, by Lemma 6.1.12, $h(\{a', h(A')\}) = h(A)$. $\square$

Theorem 6.1.15. *Let A be a set of non-interfering actions, for a state space S modulo T, and $s \in S$ a state such that $app_A(s)$ is defined. Then $app_A(s) = app_{a_1;\ldots;a_n}(s)$ for any total ordering $a_1 < \cdots < a_n$ of A.*

Proof. By induction on the number of actions n in A. If $n = 1$ then we are trivially done. If $n \geq 2$, then let $A = \{a\} \cup A'$. Since actions in A are non-interfering, then they are simply commuting and a affects none of the actions in A'. Then, by Lemma 6.1.14, we have that a does not affect the happening action $h(A')$. Now observe that, since $app_A(s)$ is defined and $Pre_{h(A)} = \bigwedge_{a \in A} Pre_a$, both a and $h(A')$ are applicable in state s. Then, by Lemma 6.1.9, we have that $app_{\{a,h(A')\}}(s) = app_{a;h(A')}(s)$. We conclude by showing that $app_A(s) = app_{\{a,h(A')\}}(s)$ and $app_{a;h(A')}(s) = app_{a_1;\ldots;a_n}(s)$ for any total ordering $a_1 < \cdots < a_n$ of A.

Equality $app_A(s) = app_{\{a,h(A')\}}(s)$ holds by Lemma 6.1.12. For equality $app_{a;h(A')}(s) = app_{a_1;\ldots;a_n}(s)$, observe that $app_{a;h(A')}(s) = app_{A'}(app_a(s))$.

Since actions in A' are non-interfering and $app_{A'}(app_a(s))$ is defined, by the induction hypothesis we have $app_{A'}(app_a(s)) = app_{a_1;\ldots;a_{n-1}}(app_a(s))$ for any total ordering $a_1 < \cdots < a_{n-1}$ of A', i.e., $app_{a;h(A')}(s) = app_{a;a_1;\ldots;a_{n-1}}(s)$ for any total ordering $a_1 < \cdots < a_{n-1}$ of A'. Finally, since actions in A are non-interfering and all of them are applicable in state s, by Lemma 6.1.11 we have that $app_{a;h(A')}(s) = app_{a_1;\ldots;a_n}(s)$ for any total ordering $a_1 < \cdots < a_n$ of A

$\square$

6.1.2 ∃-Step Plans

Here we generalize the notion of ∃-step plan, proposed in [DNK97] and further developed in [RHN06], to the setting of Planning modulo Theories. Under the ∃-step semantics, it is not necessary that all actions are non-interfering as long as they can be executed it at least one order, which makes it possible increase the number of parallel actions still further.

Definition 6.1.16 (∃-Step Plan)**.** *Given a set of actions A and an initial state I, for a state space S modulo T, a* ∃-step plan *for A and I is a sequence $P = \langle A_0, \ldots, A_{l-1}\rangle$ of sets of actions together with a sequence of states $s_0, \ldots, s_l$ (the execution of P), for some $l \geq 0$, such that*

1. *$s_0 = I$, and*
2. *for all $i \in \{0, \ldots, l-1\}$ there is a total ordering $a_1 < \cdots < a_n$ of A_i, such that $app_{a_1;\ldots;a_n}(s_i)$ is defined and equals s_{i+1}.*

Instead of requiring that each group A_i of actions can be ordered to any total order, as in ∀-step semantics, in ∃-step semantics it is sufficient that there is one order that maps state s_i to s_{i+1}. Note that under this semantics the successor s_{i+1} of s_i is not uniquely determined solely by A_i, as the successor depends on the implicit ordering of A_i and, hence, the definition has to make the execution $s_0, \ldots, s_l$ explicit.

Theorem 6.1.17. *Let A be a set of simply commuting actions, for a state space S modulo T, such that, for some total ordering $a_1 < \cdots < a_n$ of A, if $a_i < a_j$ then a_i does not affect a_j, and let $s \in S$ be a state such that $app_A(s)$ is defined. Then $app_A(s) = app_{a_1;\ldots;a_n}(s)$.*

Proof. The proof is analogous to the proof of Theorem 6.1.15, but without using Lemma 6.1.11. We proceed by induction on the number of actions n in A. If $n = 1$ then we are trivially done. If $n \geq 2$, then let $A' = \{a_2, \ldots, a_n\}$.

We have that actions in A are simply commuting and a_1 affects none of the actions in A'. Then, by Lemma 6.1.14, we have that a_1 does not affect the happening action $h(A')$. Now observe that, since $app_A(s)$ is defined and $Pre_{h(A)} = \bigwedge_{a \in A} Pre_a$, both a_1 and $h(A')$ are applicable in state s. Then, by Lemma 6.1.9, we have that $app_{\{a_1, h(A')\}}(s) = app_{a_1;h(A')}(s)$. We conclude by showing that $app_A(s) = app_{\{a_1,h(A')\}}(s)$ and $app_{a_1;h(A')}(s) = app_{a_1;\ldots;a_n}(s)$.

Equality $app_A(s) = app_{\{a_1,h(A')\}}(s)$ holds by Lemma 6.1.12. For equality $app_{a_1;h(A')}(s) = app_{a_1;\ldots;a_n}(s)$, observe that $app_{a_1;h(A')}(s) = app_{A'}(app_{a_1}(s))$. Since actions in A' are simply commuting and $app_{A'}(app_{a_1}(s))$ is defined, by the induction we have that $app_{A'}(app_{a_1}(s)) = app_{a_2;\ldots;a_n}(app_{a_1}(s))$ according to the given ordering, and hence $app_{a_1;h(A')}(s) = app_{a_1;\ldots;a_n}(s)$. □

6.2 Checking Interference with SMT

We can check the proposed notion of interference, according to Definitions 6.1.1, 6.1.2 and 6.1.5, by means of checking the satisfiability of some SMT formulas at compile time. The following simplified example, extracted from the *Planes* domain, demonstrates how this can be achieved. The *Planes* problem consists in transporting people between several cities using planes, with a limited number of seats. The considered actions are `board` and `fly`. Boarding is limited by seat availability, and a plane can only fly if it is transporting somebody. If we consider action a as:

$$\begin{aligned}
&\text{board\_person1\_plane1\_city1} = \langle \\
&\quad \text{seats\_plane1} > \text{onboard\_plane1} \land \text{at\_person1\_city1} \land \text{at\_plane1\_city1}, \\
&\quad \{\text{at\_person1\_city1} = \bot,\ \text{in\_person1\_plane1} = \top, \\
&\qquad\qquad \text{onboard\_plane1} = \text{onboard\_plane1} + 1\}\rangle
\end{aligned}$$

and action b as:

$$\begin{aligned}
&\text{fly\_plane1\_city1\_city2} = \langle \\
&\quad \text{onboard\_plane1} > 0 \land \text{at\_plane1\_city1}, \\
&\quad \{\text{at\_plane1\_city1} = \bot,\ \text{at\_plane1\_city2} = \top\}\rangle
\end{aligned}$$

then most planners, checking interference syntactically, would determine interference, since action a modifies the `onboard_plane1` variable and action

b uses this variable in its precondition. If we consider

$$\begin{aligned}
Pre_a &= \{\text{seats\_plane1} > \text{onboard\_plane1} \wedge \\
&\qquad \text{at\_person1\_city1} \wedge \text{at\_plane1\_city1}\} \\
Pre_b &= \{\text{onboard\_plane1} > 0 \wedge \text{at\_plane1\_city1}\} \\
Eff_a &= \{\text{at\_person1\_city1} = \bot,\ \text{in\_person1\_plane1} = \top, \\
&\quad \text{onboard\_plane1} = \text{onboard\_plane1} + 1\} \\
Eff_b &= \{\text{at\_plane1\_city1} = \bot,\ \text{at\_plane1\_city2} = \top\}
\end{aligned}$$

according to Definition 6.1.5 it can be seen that a does not affect b, since:

1. $Pre_a \wedge Pre_b \wedge \neg(Pre_b\sigma_a)$ is T-unsatisfiable. In this case Pre_b cannot be falsified by Eff_a because, if we consider Pre_b, there is no effect that modify at_plane_city1 and the effect onboard_plane1 $=$ onboard_plane1 $+ 1$ will never be able to falsify onboard_plane1 > 0 on Pre_b

2. a and b are simply commuting because there is no common variable in Eff_a and Eff_b, and

3. $Pre_a \wedge Pre_b \wedge \neg(x\sigma_{h(\{a,b\})} = x\sigma_b\sigma_a)$ is T-unsatisfiable for all variables x, because Eff_a and Eff_b do not share any variable and therefore there cannot be any problem derived from the ordering of effects

The first check of Definition 6.1.5 can be modelled in the SMT-LIB language [BST10] as follows:

```
;; declaration of problem variables.
(declare-fun at_person1_city1 () Bool)
(declare-fun at_plane1_city1 () Bool)
(declare-fun seats_plane1 () Int)
(declare-fun onboard_plane1 () Int)

;; preconditions of actions "board" and "fly"
(assert (and (> seats_plane1 onboard_plane1)
             at_person1_city1
             at_plane1_city1))

(assert (and (> onboard_plane1 0)
             at_plane1_city1))
```

```
;; negated precondition of fly after board
(assert (not (and (> (+ onboard_plane1 1) 0)
                  at_plane1_city1)))
(check-sat)
```

Note that in the negated precondition of fly, we are replacing each variable by the term which represents its value after the execution of `board`, i.e., we replace onboard_plane1 by onboard_plane1 + 1. A negative answer should be obtained from the SMT solver.

The check of simply commutativity would consist in checking for all variables commonly modified by the two actions, if the effects can be commuted. In the example, there are no common variables modified by both actions. Hence, suppose we are checking whether two arbitrary assignments $\{x \mapsto exp_1\}$ and $\{x \mapsto exp_2\}$ commute. According to Definition 6.1.1, this would consist in checking whether $\neg(exp_2\{x \mapsto exp_1\} = exp_1\{x \mapsto exp_2\})$ is T-satisfiable. A negative answer would imply T-unsatisfiability of this negation and, hence, commutativity of the assignments.

The third check can be implemented analogously by means of satisfiability checks.

An important difference with the purely syntactic definition of interference of [Rin09] is that we include preconditions of the checked actions in our checks. More precisely, the reason for adding the preconditions in all satisfiability checks is that we require that the two actions for which we check potential interference can occur in parallel. This way, we are able to avoid many "false positive" interference relationships, which would make the final formula grow unnecessarily. It can also be seen as a combination of a interference and reachability check, all in one. All in all, we obtain a much more fine-grained notion of interference, that will help to increase the parallelization of actions. Note that the interference relationships determined semantically will always be a subset of the interference relationships determined syntactically. Interestingly, we will be using an SMT solver both at compile time, as an oracle to predict interference relationships, and at solving time.

Ungrounded checking

Although these checks using a modern SMT solver are negligible in terms of time, with big planning problems the number of checks can grow considerably. Here we propose an optimization to be able to check interferences between actions without the need of grounding them first.

The main idea is to model the interference queries as before, but substituting the action parameters appearing in the preconditions and effects not with concrete values, but with variables. Then, incorporate to the formula the disequality relationships between variables of different types and ask the solver for a model. If the first action can affect the second, a query to the SMT solver would give a concrete set of values that explain why the first action can interfere with the second. But note that what we would really need is not one but all models of the formula, because to finally add the mutexes to the encoding we need the concrete grounded actions.

The #SMT problem is the problem of counting the number of satisfying assignments of a given SMT formula. In our case, we do not need to count them, but to enumerate them. Unfortunately no efficient implementation of an SMT solver that enumerate models has been found. The alternative is using a SMT solver, encode the problem as before, get a model, and then iteratively add a clause prohibiting the model given and ask the solver again. This approximation would need at least as many queries to the SMT solver as concrete interferences exist, so at first hand it seemed very inefficient. This is the reason why we propose an alternative.

Lets consider the original lifted actions in the PDDL model of the previous example:

```
(:action board
 :parameters (?p - person ?a1 - aircraft ?c1 - city)

 :precondition (and (at ?p ?c1)
                    (at ?a1 ?c1)
                    (> (seats ?a1) (onboard ?a1)))

 :effect (and (not (at ?p ?c1))
              (in ?p ?a1)
              (increase (onboard ?a1) 1)))

(:action fly
 :parameters (?a2 - aircraft ?c2 ?c3 - city)

 :precondition (and (at ?a2 ?c2)
                    (> (onboard ?a2) 0))

 :effect (and (not (at ?a2 ?c2))
              (at ?a ?c3)))
```

We have three planes in the problem: A320-1, A320-2 and A320-3. If we assign A320-1 to parameters ?a1 and ?a2, one should find the same interferences than if we assign A320-2 to parameters ?a1 and ?a2. The same should happen if we assign A320-1 and A320-2 or A320-2 and A320-3 to ?a1 and ?a2 respectively. So, to reason about interference between actions `board` and `fly`, we will need to determine, for example, if the actions interfere in the case that `?a1` and `?a2` are the same aircraft. Or in the case that cities `?c1` and `?c2` are the same, etc. That is, interference is determined depending on the equality relationship between parameters.

Since equality or disequality between parameters of different types has no sense, the first thing we need is to group the parameters of the same type in sets, by its most general declared type.

Then, one should need to consider all different possible equality and disequality relationships between the parameters of the same type, to find out in which cases one action can interfere with another action.

Following the previous example, in total we have three parameters `c1`, `c2` and `c3` of the type `city`, two parameters `a1` and `a2` of the type `aircraft` and one parameter `p` of the type `person`. Therefore, to enumerate all interferences we have to check the following situations:

$$
\begin{array}{l}
c1 = c2 = c3, a1 = a2 \\
c1 = c2, c2 \neq c3, a1 = a2 \\
c1 = c3, c2 \neq c3, a1 = a2 \\
c1 \neq c2, c2 \neq c3, a1 = a2 \\
\ldots
\end{array}
$$

If we consider all the possible partitions of the set, they map directly to all the possible equalities and disequalities between elements of the set.

Example 26. *Consider set $\{A, B, C\}$. All the partitions of this set are:*

- $\{\{A\}, \{B\}, \{C\}\}$
- $\{\{A, B\}, \{C\}\}$
- $\{\{A, C\}, \{B\}\}$
- $\{\{A\}, \{B, C\}\}$
- $\{\{A, B, C\}\}$

When two elements appear in the same set, we consider them to be equal, and when they appear on different sets, we consider them to be different. So, on partition $\{\{A, C\}, \{B\}\}$ we should consider that $A = C, A \neq B$ and $C \neq B$.

Once the set partitions have been generated for each set of parameters, the Cartesian product between all the sets has to be done to obtain the combination of equality and disequality relations between the parameters of the two actions.

We propose to model interference as shown in Section 6.2, and do one query for each possible combination of equality and disequality between parameters of the two actions. Instead of using variables or constant values, for convenience when we intent for two parameters to be equal we substitute them for the same integer, and by different integers when we want them to be different.

This approach results in much fewer queries to the SMT solver. Having the same example, with a total of 2 planes, 4 persons and 6 cities, a grounded checking would result in $4 \times 2 \times 6 = 48$ grounded fly actions and $2 \times 6 \times 6 = 72$ grounded board actions. This would result in $48 \times 72 = 3456$ grounded checks.

The total number of partitions of an n-element set is the Bell number B_n. If we now consider the proposed ungrounded checking method, the sets of parameters will be the following: for planes $S_{planes} = \{a1, a2\}$ and for cities $S_{cities} = \{c1, c2, c3\}$. Bell numbers for these sets are $B_2 = 2$ and $B_3 = 5$, so we will have a total of $2 \times 5 = 10$ ungrounded checks. As it can be seen, the number of checks needed using this technique is much lower than using the grounded checking, and thus scales much better with large problems.

Algorithm 8 Mutex Generation

```
Input: $A$ : A set of PDDL actions
Output: $M$ : A set of actions pairs $\langle a, b\rangle$ where $a$ interferes with $b$
1: $A_p \leftarrow$GeneratePairs($A$)
2: $M \leftarrow \emptyset$
3: for all $\{a_1, a_2\} \in A_p$ do
4:     $M \leftarrow M \cup$ InterferenceChecking($a_1$,$a_2$)
5: end for
6: return $M$
```

Algorithm 8 discovers, for a given set of ungrounded actions, the minimum set of interferences between them. It starts by generating all the possible pairs of actions. Then, for each generated pair, it calls Algorithm 9 to discover the minimum set of grounded interferences between the two actions.

Algorithm 9 receives two ungrounded actions and returns the minimum set of grounded interferences between them. It uses the following functions:

Algorithm 9 Interference Checking

```
Input: $a_1$ : first action, $a_2$ : second action
Output: $M$ : The set of ground interferences between $a_1$ and $a_2$
1: $[p_1, p_2, \ldots, p_n] \leftarrow$ groupByType(parameters($a_1$) $\cup$ parameters($a_2$))
2: $L \leftarrow$ setPartitions($p_1$) $\times$ setPartitions($p_2$) $\times \cdots \times$ setPartitions($p_n$)
3: for $l \in L$ do
4:     com $\leftarrow$pairWithIntegers($l$)
5:     if check1(com,$a_1$,$a_2$) $\vee$ check2(com,$a_1$,$a_2$) then
6:         $M \leftarrow$ generateInterferences($a_1$,$a_2$,com)
7:     end if
8: end for
9: return $M$
```

parameters given an action, it returns a set with all the parameters of that action.

groupByType receives a set of parameters, and returns a list of sets. Each set groups the original parameters by its most general type.

setPartitions receives a set, and efficiently [KN05] generates all the possible partitions of the original set.

pairWithIntegers receives a n-tuple of sets of sets of parameters (i.e., one of the possible partitions above), where each component corresponds to a type, whose elements (sets) denote parameters with the same value (and different to the parameters in the other sets). Then, it returns the same n-tuple but with each parameter paired to an integer. This integer will be equal to integers on the same set and different to others.

For example, given the n-tuple $(\{\{A, B\}, \{C\}\}, \{\{D\}, \{E\}\})$, as A and B belong to the same set, they will have the same integer. The other elements belong to different sets, so they will have different integers. Given this example, a possible return value would be:

$$\{\{\{\langle A, 1\rangle, \langle B, 1\rangle\}, \{\langle C, 2\rangle\}\}, \{\{\langle D, 3\rangle\}, \{\langle E, 4\rangle\}\}\}$$

check1 This function encodes the first condition of interference explained in Definition 6.1.5 to SMT: substitutes each parameter of the action by the integer paired with it and finally checks and returns the satisfiability of the resulting formula.

check2 Does the same as the former, but with the second condition in Definition 6.1.5.

generateInterferences generates all the ground instances of the pair of actions (a_1, a_2) that correspond to the equalities and disequalities induced by *com*.

For example, consider a problem with persons `p1` and `p2` and cities `Barcelona` and `London`. The two considered actions are: $a_1 =$ `board` with parameters `(?p - person ?c - city)` and action $a_2 =$ `fly` with parameters `(?c1 -city ?c2 - city)`. If we consider the n-tuple *com* to be $\{\{\langle \text{c}, 1\rangle, \langle \text{c1}, 1\rangle, \langle \text{c2}, 1\rangle\}, \{\langle \text{p}, 2\rangle\}\}$ (all city parameters need to be equal), the generated interferences would be:

$$\langle \text{board\_p1\_barcelona}, \text{fly\_barcelona\_barcelona}\rangle$$
$$\langle \text{board\_p1\_london}, \text{fly\_london\_london}\rangle$$
$$\langle \text{board\_p2\_barcelona}, \text{fly\_barcelona\_barcelona}\rangle$$
$$\langle \text{board\_p2\_london}, \text{fly\_london\_london}\rangle$$

6.3 Chained SMT Encoding

It is not difficult to see that the encoding described in Section 5.2 would be correct for sequential plans, but it does not adhere to the parallel plan semantics of Definition 6.1.4. If two actions planned at the same time modify a same variable, two different situations can arise. On the one hand, if the assignments are not equivalent, then the SMT formula encoding the planning problem will become unsatisfiable. Although this is right for the Boolean case, it is more subtle for other theories, where effects can be cumulative. For example, two assignments $\{x \mapsto x + 1\}$ and $\{x \mapsto x + 2\}$ would result into subformulas $x^{t+1} = x^t + 1$ and $x^{t+1} = x^t + 2$ which, together, are unsatisfiable. This, in practice, would rule out many parallel plans.

On the other hand, if assignments were equivalent, then all but one would become redundant in the SMT formula. Then, the formula would possibly be satisfiable but, in this case, solutions would not adhere to the semantics given in Definition 6.1.4, where effects of actions planned at the same time are composed. A simple way of overcoming this problem could be to forbid the parallel execution of actions modifying a same non-Boolean variable, but this would rule out the parallelization of actions with cumulative effects.

For this reason, in this section we propose a finer encoding for *planning as SMT* as a particular case of PMT (Chained SMT Encoding from now on). It is valid for any theory T under a quantifier-free first-order logic with equality. This encoding is an extension to the planning as SMT encoding presented in Section 5.2. It builds onto the former in order to add support for cumulative effects in parallel plans.

Let $\pi = \langle S, A, I, G\rangle$ be a planning problem modulo T, for a theory T under a quantifier-free first-order logic with equality. For each variable x in $var(S)$ and each time step t, a new variable x^t of the corresponding type is introduced, denoting the value of x at step t. Moreover, for each action a and each time step t, a Boolean variable a^t is introduced, denoting whether a is executed at step t.

Given a term s, by s^t we denote same term s, where all variables x in $var(S)$ have been replaced by x^t, and analogously for formulas. For example $(x + y)^t = x^t + y^t$, and $(p \wedge x > 0)^t = p^t \wedge x^t > 0$.

For the case of effects, we define

$$\begin{aligned}
\{x \mapsto \top\}^t &\overset{def}{=} x^{t+1} \\
\{x \mapsto \bot\}^t &\overset{def}{=} \neg x^{t+1} \\
\{x \mapsto s\}^t &\overset{def}{=} (x^{t+1} = s^t)
\end{aligned}$$

where s is a non-Boolean term belonging to theory T. For example, for an assignment $\{x \mapsto x + k\}$, where k is a constant, we have $\{x \mapsto x + k\}^t = (x^{t+1} = x^t + k)$.

For sets of assignments, i.e., action effects, we define

$$\begin{gathered}
(\{x \mapsto s\} \cup \mathit{Eff})^t \overset{def}{=} \{x \mapsto s\}^t \wedge \mathit{Eff}^t \\
\emptyset^t \overset{def}{=} \top
\end{gathered}$$

where s is a term (either Boolean or not) and Eff is a set of assignments.

Let N be the set of non-Boolean variables from $var(S)$. For each action $a = \langle Pre_a, \mathit{Eff}_a\rangle$ and each variable $n \in N$, let $\mathit{Eff}_{a,n}$ be the assignment $\{n \mapsto exp\} \in \mathit{Eff}_a$, or the empty set if there is no such assignment. For each $n \in N$, let $A_n = \{a \mid a \in A \wedge \mathit{Eff}_{a,n} \neq \emptyset\}$, i.e., the set of actions that modify variable n.

The constraints of the proposed encoding are as follows. As on the previous encoding, for each time step t, execution of an action implies that its precondition is met:

$$a^t \to Pre_a^t \qquad \forall a = \langle Pre_a, \mathit{Eff}_a\rangle \in A \tag{6.1}$$

On the previous encoding, constraint 5.2 stated that, if the action is executed, each of its effects will hold at the next time step. It was encoded as follows:

$$a^t \rightarrow \mathit{Eff}^t_a \qquad\qquad \forall a = \langle \mathit{Pre}_a, \mathit{Eff}_a \rangle \in A$$

Now, this constraint has to be split and rewritten as follows, in order to take into account a possible arbitrary number of consecutive assignments (or "chain of assignments") on each variable $n \in N$. First of all, we state the constraints for variables $n \in N$ such that $|A_n| = 1$, i.e., those that are modified only by one action:

$$a^t \rightarrow (\mathit{Eff}_a \setminus \cup_{n \in N, |A_n|>1} \{\mathit{Eff}_{a,n}\})^t \qquad \forall a = \langle \mathit{Pre}_a, \mathit{Eff}_a \rangle \in A \quad (6.2)$$

Then, for each variable $n \in N$ such that $|A_n| > 1$, and for each time step t, the following constraints are introduced, using additional variables $n^t_0, \ldots, n^t_{|A_n|}$ of the type of n, and considering an enumeration $a_1, \ldots, a_{|A_n|}$ of the actions in A_n:

$$\begin{aligned} & n^t = n^t_0 \\ & a^t_i \rightarrow \mathit{Eff}^t_{a_i,n}\{n^{t+1} \mapsto n^t_i, n^t \mapsto n^t_{i-1}\} && \forall a_i \in a_1, \ldots, a_{|A_n|} \\ & \neg a^t_i \rightarrow n^t_i = n^t_{i-1} && \forall a_i \in a_1, \ldots, a_{|A_n|} \\ & n^{t+1} = n^t_{|A_n|} \end{aligned} \quad (6.3)$$

Finally, as before, we need explanatory axioms to express the reason of a change in state variables. For each variable x in $var(S)$:

$$x^t \neq x^{t+1} \rightarrow \bigvee_{\substack{\forall a = \langle \mathit{Pre}_a, \mathit{Eff}_a \rangle \in A \\ \text{such that } \exists \{x \mapsto s\} \in \mathit{Eff}_a}} a^t \quad (6.4)$$

That is, a change in the value of x implies the execution of at least one action that has an assignment to x among its effects.

Example 27. *Lets suppose we have actions $A = \{a_1, a_2\}$, being $a_1 = \langle \top, \{x \mapsto x+1, y \mapsto 0\} \rangle$ and $a_2 = \langle \top, \{x \mapsto x+2, z \mapsto \top\} \rangle$. The set of actions that modify variable x is $A_x = \{a_1, a_2\}$, and the one for variable y is $A_y = \{a_1\}$. Given the encoding and a time step t, Constraint 6.1 would give:*

$$a^t_1 \rightarrow \top \qquad\qquad a^t_2 \rightarrow \top$$

as both actions have $\top$ *as its preconditions. Constraint 6.2 gives:*

$$a_1^t \rightarrow y^{t+1} = 0 \qquad\qquad a_2^t \rightarrow z^{t+1} = \top$$

Note that all effects that modify variable x *are not expressed, as* $|A_x| > 1$*. For expressing the possible chain of assignments on variable* x*, Constraint 6.3 is used instead. We would need the extra variables* x_0^t*,* x_1^t*,* x_2^t *together with constraints:*

$$\begin{array}{ll} x^t = x_0^t & \\ a_1^t \rightarrow x_1^t = x_0^t + 1 & \neg a_1^t \rightarrow x_1^t = x_0^t \\ a_2^t \rightarrow x_2^t = x_1^t + 2 & \neg a_2^t \rightarrow x_2^t = x_1^t \\ x^{t+1} = x_2^t & \end{array}$$

Finally, Constraint 6.4 encodes the frame axioms:

$$\begin{array}{l} x^t \neq x^{t+1} \rightarrow a_1^t \vee a_2^t \\ y^t \neq y^{t+1} \rightarrow a_1 \\ z^t \neq z^{t+1} \rightarrow a_2 \end{array}$$

6.3.1 Sequential and Parallel Plans

The previous constraints have to be complemented, depending on the type of parallelism we wish. Sequential plans with this encoding would not make sense, as it is basically an extension of the previous encoding in Section 5.2, expanded to support more parallelism. $\forall$-step and $\exists$-step plans can be implemented the same way as described in Section 5.2.2.

6.3.2 Path-based strong Components Order

If we choose to consider $\exists$-step plans, an order between actions is needed to decide what mutexes should be added. In this section we propose a new algorithm to obtain an order, based on an algorithm [Gab00] that uses depth-first search to compute the strongly connected components of a given graph.

Note that, with the considered notion of interference, and only with respect to effects, the order between actions is not important. This is because the second point of the interference notion presented in Definition 6.1.5

requires that effects are commutative. Therefore, an interference detected between a_1 and a_2 due to the non-commutativity of its effects will generate two edges in the disabling graph: one from a_1 to a_2 and the second from a_2 to a_1. Therefore, the order that is later used to traverse the disabling graph to add the mutexes is irrelevant.

When order is important is when preconditions are considered. Consider the first part of Definition 6.1.5. For example, action a_1 potentially disabling the precondition of action a_2 does not mean that a_2 has to disable the precondition of a_1. So, interferences found using this part generate only one vertex in the disabling graph.

As previously said, for the $\exists$-step encoding, the selected order will affect which actions can be set in the same time step, and therefore more or less time steps will be needed to find a valid plan.

Algorithm 10 detects the strongly connected components in the disabling graph and labels each vertex with the order in which is visited. The algorithm maintains two stacks, S and P. Stack S contains all the vertices that have not yet been assigned to a strongly connected component, in the order in which the depth-first search reaches the vertices. Stack P contains vertices that have not yet been determined to belong to different strongly connected components from each other. C is an integer that counts the number of vertices reached, which is used to assign a number (the order) to each vertex. L is a list which will contain the order the vertexes are visited, and N is also a list that will state to which SCC a node belongs. For example, $N[1] = 3$ will state that vertex 1 belongs to the third SCC. Finally, CC is an integer that will count the strongly connected components.

The overall algorithm consists of a loop through the vertices of the graph, calling this recursive search on each vertex that does not yet have a number assigned to it. L and N are initialized to -1 on all positions.

The advantage of using this order is that, for the quadratic $\exists$-step encoding, the mutexes added to the problem are not global, but local for each SCC. That is, a mutex is generated only if the two actions belong to the same SCC.

The condition for the quadratic encoding for the $\exists$-step semantics is that a mutex is added if $i < j$ and a_i affects a_j. But this is over restrictive, as if a_i and a_j does not belong to the same SCC, it is guaranteed [RHN06] that it exists a valid order between them.

Example 28. *Suppose that given two actions a and b, a affects b because it modifies some variable in the precondition of b, but b does not affect a.*

If we consider the quadratic encoding of the $\exists$-step semantics, depending

Algorithm 10 PBSCC

```
 1: $C \leftarrow 1$
 2: $CC \leftarrow 1$
Input: $v$ : vertex, $E$ : set of edges of the graph
Output: $L$ contains an order, and $N$ will have each vertex assigned to a
    SCC.
 3: $L[v] \leftarrow C$
 4: $C \leftarrow C + 1$
 5: $push(v, P)$
 6: $push(v, S)$
 7: for all $\langle v, w \rangle \in E$ do
 8:   if $L[w] = -1$ then
 9:     $PBSCC(w)$
10:   else
11:     if $N[w] = -1$ then
12:       while $L[top(P)] > L[w]$ do
13:         $pop(P)$
14:       end while
15:     end if
16:   end if
17: end for
18: if $v == top(P)$ then
19:   repeat
20:     $w \leftarrow pop(S)$
21:     $N[w] \leftarrow CC$
22:   until $w \neq v$
23:   $CC \leftarrow CC + 1$
24:   $pop(P)$
25: end if
```

on the order we pick ($a < b$ or $b < a$), a mutex should be added or not. But it does not matter what order we pick, because either way the mutex is unnecessary. This is because those two actions do not form a cycle, and thus we can guarantee that there will be a valid order to serialize them when a plan is given with them in the same time step.

Therefore, less mutexes can be added if we consider to what SCC each action belongs. To do this, the condition for the quadratic encoding for the $\exists$-step semantics should be: A mutex is added if $i < j$, a_i affects a_j and a_i belongs to the same SCC as a_j. This order gave the best results, and will be the one used on the following section.

6.4 Experimental Results

In this section we evaluate the impact of the proposed notion of interference on the length of parallel plans, using ∃-step semantics. Experiments have been performed using both syntactic an semantic checks of interference at compile time. For the case of semantic checks, we have additionally considered the chained SMT encoding. These executions are noted as SYN, SEM, and SEM+C, respectively, in Tables 6.1 and 6.2. In syntactic checking we forbid concurrent assignment or assignment and inspection to the same numeric variable. Semantic checks are the ones we have introduced, by means of calls to a SMT solver in Section 6.2.

Experiments have been run on 8GB Intel® Xeon® E3-1220v2 machines at 3.10 GHz, using Yices [DDM06a] v2.3.0 as back-end SMT solver, with the QF_LIA logic [BST10] and a two hours timeout. For the sake of completeness, we compare the performance of our implementation with the numeric planner NumReach/SAT [HGSK07] using MiniSAT 2.2.0 (column NR1), and NumReach/SMT using Yices v2.3.0 (column NR2).

Five domains are considered: the numeric versions of *ZenoTravel*, *DriverLog* and *Depots*, the real-life challenging *Petrobras* domain, and a crafted domain called *Planes*.

ZenoTravel and *DriverLog* are some of the domains in the literature with a higher numeric interaction between actions. Domains like *Rovers* or *Settlers* have been excluded because they are too big to show meaningful results with the encoding at hand and the chosen timeout. The *Petrobras* domain is the same explained on the previous chapter. In this chapter we used only instances of the group A, as they were the most challenging.

Due to the limited numeric interactions between actions in the domains found in the literature, we additionally propose a new domain called *Planes* which is created from *ZenoTravel*, by adding some plausible numeric constraints, in order to help us demonstrate the benefits from checking interference between actions semantically. Figure 2 depicts the full PDDL model of the Planes domain.

Table 6.1 shows the number of instances solved by each approach. Checking interference semantically and using the chained SMT encoding is best in *Petrobras*, *ZenoTravel* and *Planes*, while NumReach/SAT is best in *Depots* and *DriverLog*. The big gap in the number of solved instances in *Depots* is twofold: lack of intrinsic parallelism in the domain, and being the per-

Domain	NR1	NR2	SYN	SEM	SEM+C
Depots	**13**	13	5	5	5
Petrobras A	3	3	5	6	**7**
Planes	5	8	8	8	**8**
ZenoTravel	13	14	13	13	**15**
DriverLog	**18**	12	14	14	15
Total	**52**	50	45	46	50

Table 6.1: Total number of instances of each domain solved by each approach. For each domain, the approach solving more instances is marked in bold. In case of draw, the faster is marked.

Domain	NR	SYN	SEM	SEM+C
Depots (5)	54	52	52	**51**
Petrobras A (3)	19	12	12	**11**
Planes (4)	90	82	62	**45**
Zenotravel (13)	97	69	69	**42**
DriverLog (12)	104	85	84	**63**
Total (37)	364	300	279	**212**

Table 6.2: Sum of the number of time steps of the plans found, restricted to commonly solved instances. First column shows, in parenthesis, the number of instances solved by all approaches. NumReach uses the same parallelism approach when using different background solvers, so only one column is included. The winning approach is shown in bold.

fect scenario for the reachability approach of NumReach (small numeric domains).

Table 6.2 shows the sum of the number of time steps of the plans found, for commonly solved instances. Note that the domains where our implementation solves more instances are also the ones that exhibit more gains in parallelism. Note also the significant reduction in time steps from the syntactic approach to the semantic approach with the chained SMT encoding, especially in the *Planes* domain.

The importance of the semantic notion of interference and its checking using the SMT solver, is that it generates the minimum set of a-priori interferences between actions. At the same time, this reduction of interferences reduces the number of strongly connected components of the graph and therefore also reduces the number of mutexes added.

Table 6.3 compares the SYN and SEM+C approaches on the commonly

Domain	SYN	SEM+C	Difference	% Removed
Depots (5)	5.35e6	1.09e6	4.26e6	79%
Petrobras A (5)	1.90e7	3.10e5	1.87e7	96%
Planes (8)	8.60e4	7.92e3	7.80e4	98%
Zenotravel (13)	1.17e6	5.26e4	1.12e6	95%
DriverLog (14)	2.95e6	6.34e5	2.26e6	78%

Table 6.3: Reduction of interferences thanks to the semantic notion of interference.

solved instances. For each family, it shows the sum of interferences, the difference between the two approaches, and the percentage of interferences that could be avoided thanks to the semantic notion of interference.

n	NR/SAT		NR/SMT		Syntactic			Semantic			Sem+chain		
	sec.	ts	sec.	ts	sec.	ts	aff.	sec.	ts	aff.	sec.	ts	aff.
1	0.00	6	1.45	6	3.96	6	5e4	1.74	6	2e4	1.70	6	1e4
2	0.49	9	8.39	9	32.44	9	3e5	13.27	9	1e5	14.99	8	7e4
3	5.52	13	42.99	13	165.36	13	1e6	82.15	13	3e5	307.01	13	2e5
4	9.75	15	134.36	15	484.81	14	2e6	288.75	14	8e5	4292.05	14	5e5
5	TO	-	5187.96	21	TO	-	-	TO	-	-	TO	-	-
7	2.48	11	37.45	11	241.37	10	2e6	117.99	10	6e5	1142.86	10	3e5
8	15.25	15	403.05	15	MO	-	-	TO	-	-	TO	-	-
10	4.42	11	101.53	11	TO	-	-	MO	-	-	TO	-	-
11	43.30	18	TO	-	TO	-	-	TO	-	-	MO	-	-
13	2.68	10	84.33	10	TO	-	-	TO	-	-	TO	-	-
14	12.40	13	1314.04	13	TO	-	-	TO	-	-	TO	-	-
16	2.03	9	142.49	9	TO	-	-	TO	-	-	TO	-	-
17	6.82	8	395.62	8	TO	-	-	TO	-	-	TO	-	-
19	17.58	11	853.63	11	TO	-	-	TO	-	-	TO	-	-

Table 6.4: Detailed results on the execution of the *Depots* domain.

Tables 6.4, 6.5, 6.6, 6.7 and 6.8 show the full results on the solved instances of each domain. NumReach columns show the results of the NumReach reachability approach, with the SAT and the SMT solvers. *Syntactic* shows the SMT Encoding with a syntactic notion of interference. *Semantic* shows the presented parallelism approach with the SMT Encoding, while *Sem + chain* replaces the SMT encoding with the Chained SMT Encoding. Column *sec.* show time in seconds, with TO denoting a time out, and MO a memory out. *ts* denote the number of time steps of the plan, and *aff.*

n	NR/SAT		NR/SMT		Syntactic			Semantic			Sem+chain		
	sec.	ts	sec.	ts	sec.	ts	aff.	sec.	ts	aff.	sec.	ts	aff.
1	9.00	6	329.76	6	153.75	3	3e6	151.23	3	3e6	1209.26	3	4e4
2	17.53	6	357.70	6	197.94	4	4e6	193.47	4	3e6	1590.52	4	5e4
3	98.62	7	958.90	7	282.54	5	4e6	260.59	5	3e6	1335.50	4	6e4
4	TO	-	TO	-	467.61	6	4e6	391.40	6	3e6	1911.95	4	7e4
5	TO	-	TO	-	1435.65	7	4e6	1398.75	7	3e6	2667.87	4	8e4
6	TO	-	TO	-	TO	-	-	3373.09	8	3e6	1939.70	4	9e4
7	TO	-	TO	-	TO	-	-	TO	-	-	2666.77	4	1e5

Table 6.5: Detailed results on the execution of the *Petrobras* domain.

n	NR/SAT		NR/SMT		Syntactic			Semantic			Sem+chain		
	sec.	ts	sec.	ts	sec.	ts	aff.	sec.	ts	aff.	sec.	ts	aff.
1	TO	-	36.44	15	0.99	13	4e3	0.29	10	6e2	1.90	9	4e2
2	3.30	18	37.79	18	6.54	16	4e3	1.17	12	6e2	3.93	10	4e2
3	TO	-	228.05	20	42.21	18	1e4	6.37	13	1e3	78.74	10	1e3
4	4.45	23	633.29	23	401.32	21	1e4	78.89	15	1e3	307.94	11	1e3
5	TO	-	763.96	22	179.97	20	1e4	47.52	15	2e3	46.51	11	1e3
6	5.39	25	1153.02	25	1971.53	23	1e4	585.63	18	2e3	331.89	13	1e3
7	TO	-	1238.43	23	374.51	21	1e4	54.48	16	2e3	41.85	11	1e3
8	5.00	24	1247.73	24	1508.65	22	1e4	119.42	17	2e3	66.51	11	1e3
12	15.54	21	TO	-	TO	-	-	TO	-	-	TO	-	-

Table 6.6: Detailed results on the execution of the *Planes* domain.

the number of resulting computed affecting relations between the problem actions.

Is specially noticeable in these tables the reduction of one or two orders of magnitude on the number of computed affecting relations between actions when using the Semantic + chain encoding with respect to other encodings.

n	NR/SAT		NR/SMT		Syntactic			Semantic			Sem+chain		
	sec.	ts	sec.	ts	sec.	ts	aff.	sec.	ts	aff.	sec.	ts	aff.
1	0.00	2	0.15	2	0.05	1	5e2	0.05	1	1e2	0.01	1	1e2
2	0.00	7	1.59	7	0.09	3	8e2	0.09	3	2e2	0.04	3	1e2
3	0.00	6	3.69	6	0.15	3	3e3	0.14	3	1e3	0.11	3	6e2
4	0.00	6	2.38	6	0.27	4	4e3	0.17	4	1e3	0.12	3	7e2
5	0.08	7	6.86	7	0.40	4	7e3	0.26	4	3e3	0.18	3	1e3
6	0.03	7	4.12	7	0.81	6	9e3	0.45	6	3e3	0.20	3	1e3
7	0.07	8	9.01	8	0.79	5	1e4	0.43	5	4e3	0.26	3	2e3
8	0.38	7	7.78	7	2.58	5	4e4	1.65	5	2e4	0.55	3	5e3
9	0.34	9	18.13	9	24.82	8	5e4	20.98	8	2e4	1.36	4	6e3
10	0.65	9	24.42	9	70.02	8	5e4	42.16	8	2e4	1.83	4	6e3
11	3.38	8	18.40	8	8.13	6	8e4	5.72	6	3e4	3.24	4	8e3
12	3.67	10	99.04	10	73.60	7	9e4	76.81	7	3e4	4.47	4	1e4
13	22.07	11	565.39	11	1324.96	9	1e5	1270.29	9	4e4	3.33	4	1e4
14	TO	-	540.10	9	TO	-	-	TO	-	-	779.78	4	8e4
15	TO	-	TO	-	TO	-	-	TO	-	-	2850.68	4	2e5

Table 6.7: Detailed results on the execution of the *Zenotravel* domain.

n	NR/SAT		NR/SMT		Syntactic			Semantic			Sem+chain		
	sec.	ts	sec.	ts	sec.	ts	aff.	sec.	ts	aff.	sec.	ts	aff.
1	0.00	7	0.49	7	0.42	5	5e3	0.37	5	2e3	0.24	5	1e3
2	0.00	10	5.53	10	1.19	8	9e3	0.66	8	3e3	0.60	7	2e3
3	0.00	8	3.10	8	0.81	6	8e3	0.47	6	3e3	0.41	4	2e3
4	0.00	8	4.07	8	1.34	6	1e4	0.72	6	4e3	0.57	4	3e3
5	0.01	9	5.59	9	1.52	7	1e4	0.70	6	5e3	0.43	4	4e3
6	0.00	6	2.42	6	1.23	4	2e4	0.66	4	7e3	0.61	4	6e3
7	0.00	7	3.66	7	2.08	5	3e4	1.09	5	1e4	0.82	4	8e3
8	0.01	8	5.10	8	3.40	7	3e4	2.09	7	1e4	1.14	5	9e3
9	0.00	11	10.36	11	12.10	10	8e4	5.48	10	2e4	5.16	8	2e4
10	0.01	8	6.48	8	18.06	7	2e5	7.62	7	5e4	4.41	4	4e4
11	0.04	10	12.01	10	36.65	9	2e5	18.95	9	7e4	6.62	6	5e4
12	TO	-	TO	-	620.88	16	6e5	495.77	16	2e5	296.06	12	1e5
13	TO	-	TO	-	159.63	11	1e6	87.63	11	3e5	47.94	7	2e5
14	0.07	12	208.94	12	271.94	11	8e5	453.58	11	2e5	167.80	8	1e5
15	0.26	12	TO	-	MO	-	-	TO	-	-	5459.28	8	6e5
16	0.83	12	TO	-	TO	-	-	MO	-	-	TO	-	-
17	1.19	12	TO	-	MO	-	-	TO	-	-	TO	-	-
18	2.46	13	TO	-	TO	-	-	TO	-	-	TO	-	-
19	2.51	12	TO	-	TO	-	-	TO	-	-	TO	-	-
20	4.62	10	TO	-	TO	-	-	TO	-	-	TO	-	-

Table 6.8: Detailed results on the execution of the *Driverlog* domain.

Chapter 7

More Relaxed Semantics for Planning as SMT

Previous chapters showed that in the setting of parallel plans, before encoding a planning problem to SAT or SMT, interferences between pairs of actions have to be determined. Then, mutex clauses are commonly added to the problem, to forbid those actions to be executed in parallel at any time step. Especially in hard planning problems, mutex clauses are a big part of the resulting formula. This reaches the point that many times they become unsolvable due to memory constraints. For this reason, some SAT based planners implement dedicated algorithms to speed up the evaluation of this kind of binary clauses [Rin12a, Kau06].

On the previous chapter, a semantic notion of interference was introduced. This notion allows the presented encoding to be more parallel by reducing the number of unnecessary mutexes. In this chapter, we go a step further in the pursuit of parallelism in planning as SMT.

Inspired by the highly relaxed semantics of [Bal13] for planning as SAT, in this Section we avoid the need to add any mutex clause to avoid the parallel execution of (potentially) interfering actions.

The avoidance of mutexes is accomplished by a non-trivial encoding to SMT, which is presented later in this chapter. Broadly speaking, this encoding makes a trade-off between the mutexes and extra variables and clauses, with the idea to allow more actions to be executed at the same time step. Experiments show that using the presented encoding less time steps are needed to reach a valid plan compared to other similar planners, resulting in more instances solved.

Sometimes a more parallel encoding can result in a worse plan in terms of makespan. This happens because normally encodings do not guarantee that every action that is true in the solution is actually needed in order to achieve the goals of the original plan.

The makespan of a plan is a common measure for its quality. In the setting of classical planning as SAT, works like [RGPS10] use a MaxSAT solver to be able to extract makespan-optimal solutions. In this chapter, an approach that uses a MaxSMT solver to prune valid plans from redundant actions is also presented, similarly as in [BCK14]. With the proposed pruning approach, the quality of plans obtained with the new encoding become similar or even better than with other ∃-step encodings.

7.1 A further relaxation of ∃-step semantics

As we explained in Definition 4.0.2, under the ∃-step semantics, it is not necessary that all actions are non-interfering as long as they can be executed in at least one order. It is sufficient that there is one order that maps state s_i to s_{i+1}.

The notion of $R^2\exists$-step plan in the context of PMT is the following.

Definition 7.1.1 ($R^2\exists$-Step Plan)**.** *Given a set of actions A and an initial state I, for a state space S modulo T, a* relaxed-relaxed ∃-step ($R^2\exists$-step) plan *for A and I is a sequence $P = [A_0, \ldots, A_{l-1}]$ of sets of actions together with a sequence of states $s_0, \ldots, s_l$ (the execution of P), for some $l \geq 0$, such that $s_0 = I$, and for all $i \in \{0, \ldots, l-1\}$ there is a total ordering $a_1 < \cdots < a_n$ of A_i, such that $T, app_{a_1;\ldots;a_{j-1}}(s_i) \models Pre_{a_j}$ for all $a_j = \langle Pre_{a_j}, Eff_{a_j} \rangle \in A_i$, and $app_{a_1;\ldots;a_n}(s_i) = s_{i+1}$.*

This is a weakening of the definition of relaxed ∃-plan in [WR07], where the consistency requirement between effects of actions occurring at the same time step has been removed and, hence, the only requirement left is that those actions can be ordered to form a valid sequential plan. The definition also generalizes to the setting of PMT. Notice that no formal definition of $R^2\exists$-step plan is given in [Bal13]. Definition 7.1.1 is in fact equivalent to the definition of ∃-step plan in [BEV16a], as well as to the definition of ∃-step plan in [RHN06] for the propositional case, which already capture $R^2\exists$-step plans. In those works, however, the encodings given for ∃-step plans are restricted to *happenings* which require, among other things, that preconditions of actions in each parallel step hold at the same time. Here no notion of happening is used and, hence, we are properly considering $R^2\exists$-step plans in the sense of [Bal13]. Let us introduce a motivating example.

Example 29. *A merchant is looking to maximize his benefits in the fastest way. We will use variable x to represent his gains. Suppose the merchant starts with no money ($x = 0$) and can do two actions. Work carrying boxes of tulips and gaining 10 coins a month, or investing some of his money in the tulip industry, doubling his earnings. His objective is reaching the sum of 20 coins. Actions can be modeled as follows:*

$$\begin{aligned} \textit{work:} \quad & a_1 = \langle \top, x \mapsto x + 10 \rangle \\ \textit{invest:} \quad & a_2 = \langle x > 5, x \mapsto x * 2 \rangle \end{aligned}$$

If we consider the ∃-step semantics in [RHN06], one of the requirements is that the preconditions of actions must be satisfiable at the start of the same time step to be applicable. Therefore there is no ordering between actions that allows a_1 and a_2 to occur in one time step, since precondition of a_2 is not satisfied at the start. This would make the shortest ∃-step plan the following: $\prod = [\{a_1\}, \{a_2\}]$.

With the Relaxed ∃-step semantics of [WR07], there is no requirement that forces preconditions of actions in the same time step to be satisfied at the start of the time step. Still, there is a requirement of consistency between the set of effects applied at the same time step. The concept of consistency with numerical variables could be generalized as that all effects should be commutative. Since the effects $x \mapsto x + 10$ and $x \mapsto x * 2$ are not commutative, the shortest plan would also be $\prod = [\{a_1\}, \{a_2\}]$.

With the R^2∃-step semantics, these requirements are lifted, so, considering the ordering between actions $a_1 < a_2$, our merchant can reach its goal with the one step plan $\prod = [\{a_1, a_2\}]$. The same ordering can be used to model the application of non-commutative effects. This small problem could be encoded as follows, where x^t (resp. a^t) denotes the value of variable x

(resp. execution of action a) at time step t:

$$
\begin{array}{lr}
x^0 = 0 & \textit{initial state} \\
a_1^0 \rightarrow \top & \textit{precondition of } a_1 \\
\left.\begin{array}{l} a_1^0 \rightarrow x_1^0 = x_0^0 + 10 \\ \neg a_1^0 \rightarrow x_1^0 = x_0^0 \end{array}\right\} & \textit{effect of } a_1 \\
a_2^0 \rightarrow x_1^0 > 5 & \textit{precondition of } a_2 \\
\left.\begin{array}{l} a_2^0 \rightarrow x_2^0 = x_1^0 * 2 \\ \neg a_2^0 \rightarrow x_2^0 = x_1^0 \end{array}\right\} & \textit{effect of } a_2 \\
x^0 = x_0^0 \wedge x_2^0 = x^1 & \textit{tying constraints} \\
x^1 = 20 & \textit{goal}
\end{array}
$$

Additional variables x_0, x_1 and x_2 (conveniently superindexed with the time step) permit us to accumulate effects over variable x. Variable x_0 denotes the initial value of x. Variable x_1 embodies the value of variable x after the possible execution of action a_1. Note that x_1 gets an updated value if action a_1 is executed, or keeps the previous value x_0 otherwise. Therefore, the precondition of action a_2 has to check x_1 instead of x. The effects of action a_2 are applied on x_1 and captured on x_2. Finally, the tying constraints link these additional variables to the initial and final values of x.

This encoding of chained effects will be the key to increase parallelism.

7.2 Relaxed Chained Encoding

We propose an encoding for *planning as SMT*, as a particular case of PMT, that adhere to the $R^2\exists$-step semantics. The given encoding is valid for any theory T under quantifier-free first-order logic with equality. In particular, for numeric planning we could take T as the theory of the integers (or the reals) and use quantifier free linear integer (or real) arithmetic formulae. This is the case for the upcoming examples in this section, but we emphasize that this encoding could be used for any theory T.

Let $\pi = \langle S, A, I, G \rangle$ be a planning problem modulo T, for a theory T under a quantifier-free first-order logic with equality. For each variable x in $var(S)$ and each time step t, a new variable x^t of the corresponding type is introduced, denoting the value of x at step t. Moreover, for each action a and each time step t, a Boolean variable a^t is introduced, denoting whether a is executed at step t.

Given a term s, by s^t we denote term s, where all variables x in $var(S)$ have been replaced by x^t, and analogously for formulas. For example $(x+y)^t$ means $x^t + y^t$, and $(p \wedge x > 0)^t$ means $p^t \wedge x^t > 0$. For the case of effects, we define

$$\{x \mapsto \top\}^t \overset{def}{=} x^{t+1}$$
$$\{x \mapsto \bot\}^t \overset{def}{=} \neg x^{t+1}$$
$$\{x \mapsto s\}^t \overset{def}{=} (x^{t+1} = s^t)$$

where s is a non-Boolean term belonging to theory T. For example, for an assignment $\{x \mapsto x + k\}$, where k is a constant, we have that $\{x \mapsto x + k\}^t$ is $(x^{t+1} = x^t + k)$. For sets of assignments, i.e., action effects, we define

$$(\{x \mapsto s\} \cup \mathit{Eff})^t \overset{def}{=} \{x \mapsto s\}^t \wedge \mathit{Eff}^t \quad \text{and} \quad \emptyset^t \overset{def}{=} \top$$

where s is a term (either Boolean or not) and Eff is a set of assignments.

For each action $a = \langle Pre_a, \mathit{Eff}_a \rangle$ and each variable $x \in var(S)$, let $\mathit{Eff}_{a,x}$ denote the assignment $\{x \mapsto exp\}$ in Eff_a if any, or the empty set if there is no such assignment. For each $x \in var(S)$, let $A_x = \{a \mid a \in A \wedge \mathit{Eff}_{a,x} \neq \emptyset\}$, i.e., the set of actions that modify x.

As it has already been said, the only requirement in the $R^2\exists$-step semantics is that actions in each parallel step can be ordered to form a valid sequential plan. Then, let $L = a_1, a_2, \ldots, a_{|A|}$ be a list enumerating all actions. The relative position of each action in L will give us the total ordering $<_L$ to serialize the actions in each parallel step.

It is important to remark that the definition of $R^2\exists$-step allows a different ordering to be used in each parallel step. Here we will use the same ordering on all time steps. It is worth noting that the chosen ordering will be governing the amount of possible parallelism. Therefore the optimal order in terms of parallelism for any given time step t it is not necessarily the same for time step $t + 1$. In any case, completeness of the method is guaranteed by the fact that the possibility of choosing exactly one action per time step is retained.

The encoding will need to refer to the i-th action (according to $<_L$) in each set A_x. To this purpose, a mapping ρ_x is defined, such that $\rho_x(i) = j$ if the i-th action in A_x, according to $<_L$, is a_j. Formally: for each $x \in var(S)$, let $\rho_x^{-0} : \{1, \ldots, |A_x|\} \to \{1, \ldots, |A|\}$ be a mapping such that $a_{\rho_x^{-0}(i)} \in A_x$ for all i in $1..|A_x|$ and $a_{\rho_x^{-0}(i)} <_L a_{\rho_x^{-0}(i+1)}$ for all i in $1..|A_x| - 1$. Let $\rho_x : \{0, \ldots, |A_x|\} \to \{0, \ldots, |A|\}$ be $\rho_x^{-0} \cup \{0 \mapsto 0\}$. The mapping $\{0 \mapsto 0\}$ is added for notational convenience (see below).

Example 30. *Consider a set of actions $A = \{a_1, a_2, a_3, a_4\}$ and $L = [a_3, a_2, a_1, a_4]$, i.e., $a_3 <_L a_2 <_L a_1 <_L a_4$. Suppose that variable x is modified by actions a_1 and a_3, so $A_x = \{a_1, a_3\}$ and $|A_x| = 2$. Then we have $\rho_x(1) = 3$ and $\rho_x(2) = 1$, because $a_3 <_L a_1$. Semantically, $\rho_x(1)$ could be read as "What is the first action that modifies x, given the ordering $<_L$?".*

For each time step t and variable x, we introduce $x^t_{\rho_x(0)}, \ldots, x^t_{\rho_x(|A_x|)}$ new "chaining" variables of the same type of x. Variable $x^t_{\rho_x(0)}$ (i.e. x^t_0) will denote the value of x at time step t and, for all i in $1..|A_x|$, $x^t_{\rho_x(i)}$ will denote the value of x after the sequential application (or not) of actions $a_{\rho_x(1)}, \ldots, a_{\rho_x(i)}$. These variables allow us to encode a possible "chain of assignments" in effects of a parallel step. The formulation here is pretty much involved than in [Bal13], where only Boolean variables are considered and so serialization of actions is very simple.

To represent "chains of assignments", in the encoding we need to refer, for a given action a_i and variable x, to the last action before a_i (according to $<_L$) that may have modified x. Therefore, we define $prev_x : \{1, \ldots, |A|\} \to \{0, \ldots, |A|\}$ to be the mapping satisfying $prev_x(i) = \rho_x(max(\{0\} \cup \{k \in 1..|A_x| \mid a_{\rho_x(k)} <_L a_i\}))$. Notice that, if there is no previous action that may modify x, it returns 0.

Example 31. *Continuing with Example 30, we would have $prev_x(1) = 3$, $prev_x(2) = 3$, $prev_x(3) = 0$ and $prev_x(4) = 1$. Semantically, $prev_x(1) = 3$ could be read as "Given the ordering $<_L$, what action that modifies x comes before a_1?". Note that $prev_x(3) = 0$ because no action before a_3 modifies x, given the ordering $<_L$.*

The constraints of the encoding are the following (Example 33 illustrates a particular case):

The execution of an action implies its preconditions, with the variables conveniently renamed in order to consider the effects of the execution of *previous* (according to $<_L$) actions in the same time step t:

$$a^t_i \to Pre^t_{a_i} \sigma^t_{prev}(i) \qquad \forall a_i \in A \tag{7.1}$$

where

$$\sigma^t_{prev}(i) = \cup_{x \in var(S)} \{x^t \mapsto x^t_{prev_x(i)}\}$$

This substitution is in charge of renaming all variables in $Pre^t_{a_i}$ that may have been modified previously in the same time step.

The execution of an action implies its effects (again, with the variables conveniently renamed):

$$a_i^t \rightarrow \mathit{Eff}_{a_i}^t \sigma_{mod_i}^t \sigma_{prev}^t(i) \qquad\qquad \forall a_i \in A \qquad (7.2)$$

where

$$\sigma_{mod_i}^t = \cup_{x \in Dom(\mathit{Eff}_{a_i})} \{x^{t+1} \mapsto x_i^t\}$$

Recall that each effect in Eff_{a_i} is translated as an equality. Substitution $\sigma_{mod_i}^t$ only renames the left hand side of the equality x^{t+1} by x_i^t, while $\sigma_{prev}^t(i)$ renames all variables occurring in the right hand side that could have been possibly modified by previous actions according to $<_L$. See Example 32 for a particular case of this renaming.

If an action is not executed, the previous value of each variable it would have modified is carried forward:

$$\neg a_i^t \rightarrow \bigwedge_{x \in Dom(\mathit{Eff}_{a_i})} x_i^t = x_{prev_x(i)}^t \qquad\qquad \forall a_i \in A \qquad (7.3)$$

Moreover, initial and final auxiliary variables are linked with the original variables x^t and x^{t+1}:

$$x^t = x_0^t \qquad \text{and} \qquad x^{t+1} = x_{\rho_x(|A_x|)}^t \qquad \forall x \in var(S) \qquad (7.4)$$

Finally, explanatory axioms express the reason of a change in state variables:

$$x^t \neq x^{t+1} \rightarrow \bigvee_{a \in A_x} a^t \qquad\qquad \forall x \in var(S) \qquad (7.5)$$

That is, a change in the value of x implies the execution of at least one action that has an assignment to x among its effects.

Remark 7.2.1. *Explanatory axioms* (7.5) *are redundant in our setting, since they follow from Equation* (7.4) *and the inductive application of Equation* (7.3)*.*

The following example shows the behavior of the two substitutions in Equation 7.2.

Example 32. *Let* $\{a_1, a_2\}$ *be a set of actions such that* $a_1 <_L a_2$ *. If actions are defined as*

$$a_1 = \langle \top, y \mapsto y + 1 \rangle$$
$$a_2 = \langle \top, x \mapsto x + y \rangle$$

then the effect of a_1 at time step t will be encoded as

$$a_1^t \to y_1^t = y_0^t + 1$$

while the effect of a_2 will be encoded as

$$a_2^t \to x_1^t = x_0^t + y_1^t.$$

The following example provides a full picture of the presented encoding.

Example 33. *Let $A = \{a_1, a_2, a_3\}$, with*

$$\begin{aligned} a_1 &= \langle \top, \{y \mapsto y + 1\} \rangle \\ a_2 &= \langle \{x > 0\}, \{y \mapsto y + 2\} \rangle \\ a_3 &= \langle \{y > x\}, \{x \mapsto x + y\} \rangle \end{aligned}$$

Then $A_x = \{a_3\}$ and $A_y = \{a_1, a_2\}$. Let $L = [a_1, a_2, a_3]$. For time step t, we would add variables x_0^t, x_3^t, y_0^t, y_1^t, y_2^t and the following constraints (we make the substitutions explicit):

$$\begin{aligned} & a_1^t \to \top\{x^t \mapsto x_0^t, y^t \mapsto y_0^t\} && \textit{cf. (1)} \\ & a_2^t \to (x^t > 0)\{x^t \mapsto x_0^t, y^t \mapsto y_1^t\} \\ & a_3^t \to (y^t > x^t)\{x^t \mapsto x_0^t, y^t \mapsto y_2^t\} \end{aligned}$$

$$\begin{aligned} & a_1^t \to (y^{t+1} = y^t + 1)\{y^{t+1} \mapsto y_1^t\}\{x^t \mapsto x_0^t, y^t \mapsto y_0^t\} && \textit{cf. (2)} \\ & a_2^t \to (y^{t+1} = y^t + 2)\{y^{t+1} \mapsto y_2^t\}\{x^t \mapsto x_0^t, y^t \mapsto y_1^t\} \\ & a_3^t \to (x^{t+1} = x^t + y^t)\{x^{t+1} \mapsto x_3^t\}\{x^t \mapsto x_0^t, y^t \mapsto y_2^t\} \end{aligned}$$

$$\begin{aligned} & \neg a_1^t \to y_1^t = y_0^t && \textit{cf. (3)} \\ & \neg a_2^t \to y_2^t = y_1^t \\ & \neg a_3^t \to x_3^t = x_0^t \end{aligned}$$

$$\begin{aligned} & x^t = x_0^t \qquad x^{t+1} = x_3^t && \textit{cf. (4)} \\ & y^t = y_0^t \qquad y^{t+1} = y_2^t \end{aligned}$$

With all substitutions applied we get:

$a_1^t \to \top$	$a_1^t \to y_1^t = y_0^t + 1$
$a_2^t \to x_0^t > 0$	$a_2^t \to y_2^t = y_1^t + 2$
$a_3^t \to y_2^t > x_0^t$	$a_3^t \to x_3^t = x_0^t + y_2^t$
cf. (1)	*cf. (2)*

$\neg a_1^t \to y_1^t = y_0^t$	$x^t = x_0^t$
$\neg a_2^t \to y_2^t = y_1^t$	$x^{t+1} = x_3^t$
$\neg a_3^t \to x_3^t = x_0^t$	$y^t = y_0^t$
	$y^{t+1} = y_2^t$
cf. (3)	*cf. (4)*

As an example of the parallelism achieved, note that the precondition $y_2^t > x_0^t$ of a_3 could be possibly satisfied thanks to the execution of a_1 or a_2 in the same step, by making the variable y bigger. This is possible because a_1 and a_2 come before a_3 in the given ordering $<_L$.

If we think in terms of what differentiates the presented encoding from the previous ones, the difference can be explained as a trade-off between the original mutexes between actions for an additional set of variables and constraints. If we consider the previous encoding presented in Section 6.3 and compare it with the encoding in Section 5.2, it reduces the number of mutexes using semantic checks and then adds variables and clauses for creating chains of assignments for some effects. The encoding introduced in this section tries to add even more actions per time step by trading not some, but all mutexes between actions and adding additional constraints and variables. A graphical representation of these differences is represented in Figure 7.1. If we consider the generalized encoding (Section 5.2), most of the resulting formula is composed of the needed mutexes. As we have seen, mutexes are small binary clauses, but still there are lots of them. In the Chained encoding (Section 6.3), some of these mutexes are replaced by the chains of assignments to variables belonging to the theory T. The mutexes caused by effects on Boolean variables are still left in the formula. Finally, in the Relaxed Chained encoding (Section 7.2), all effects are chained, and therefore mutexes disappear completely from the formula. Section 7.3 goes into a bit more detail on the sizes of each part.

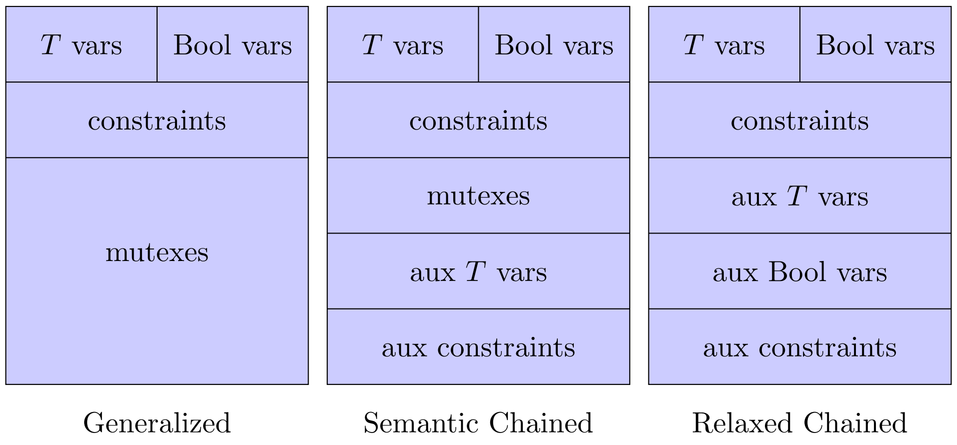

Figure 7.1: A graphical representation of the differences between the Generalized, Semantic Chained and Relaxed Chained encodings (Sections 5.2, 6.3 and 7.2 respectively)

7.2.1 Proofs

Here we present proofs of completeness and correctness of the encoding presented in this Chapter.

Definition 7.2.2. *Let $\pi = \langle S, A, I, G \rangle$ be a PMT problem, $<_L$ a total order on the actions in A and n a number of steps greater than 0. We denote by $E(\pi, n, <_L)$, the SMT formula resulting from the encoding of π described in Section 3.1 using order $<_L$, for n consecutive time steps.*

For each $t \in \{0..n\}$, we define $X^t = \{x^t | x \in var(S)\}$.

We define I^0 as the formula describing the initial state I with variables superscripted by time point 0.

We define G^n as the formula describing the goal G with variables superscripted by time point n.

Theorem 7.2.3 (Soundness)**.** *Given a PMT problem $\pi = \langle S, A, I, G \rangle$, a number of steps n and a total order $<_L$ between actions in A, if M is a model of the SMT formula $\varphi = E(\pi, n, <_L) \wedge I^0 \wedge G^n$, then we can infer a valid sequential plan for π from M.*

Proof. We first prove that theorem is true for $n = 1$ and then argue why this can be generalized to $n > 1$.

Let $n = 1$. Let $a_0^1, \ldots, a_0^k$ be the, according to $<_L$, ordered action variables set to true in the model of φ. Then, the corresponding sequence of actions $a_0^1; \dots ; a_0^k$ is a sequential plan of π: if $k = 0$ we are done since this means that G is already satisfied from X^0 without executing any action. Notice that if no action is executed, Constraints 7.3 and 7.4 enforce equality between chained variables in X^0 and X^1. If $k > 0$ we need to prove that each action a_0^i is applicable after applying $a_0^1; \dots ; a_0^{i-1}$. This is guaranteed by Constraint 7.1 and 7.2. Constraint 7.2 force the effects of previous actions to be applied resulting in a "temporal state" and constraints 7.1 ensure that precondition of a_0^i is satisfied by this temporal state. Roughly, this temporal state consists of the valuation assigning, to each $x \in S$, its updated value due to effects of actions $a_0^1; \dots ; a_0^{i-1}$. This value is captured by the "closest" previous chaining variable $x^0_{prev(i)}$, thanks to Constraint 7.2 and 7.3.

For $n > 1$, Constraint 7.4 properly links, for each $m \in \{1..n\}$, variables X^{m-1} to the first temporal state of step m and the last "temporal state" of step m to X^m. Hence, if $a_0^1, \dots , a_0^{k_0}, \dots , a_n^1, \dots , a_n^{k_n}$ are the action variables set to true in the model of $E(\pi, n, <_L)$, where each subset $a_i^1, \dots , a_i^{k_0}$ is ordered according to $<_L$, the corresponding sequence of actions $a_0^1; \dots ; a_0^{k_0}; \dots ; a_n^1; \dots ; a_n^{k_n}$ is a valid sequential plan of π. □

Completeness is guaranteed since the possibility of executing exactly one action per time step is retained.

Theorem 7.2.4 (Completeness)**.** *Given a PMT problem $\pi = \langle S, A, I, G\rangle$, if there exists a valid sequential plan $a_1; \dots ; a_n$ for π then, for any total order $<_L$ on the actions of A, the SMT formula $\varphi = E(\pi, n, <_L) \wedge I^0 \wedge G^n$ is satisfiable.*

Proof. Let $<_L$ be an arbitrary order on the actions in A. Mimicking the valid sequential plan $a_1; \dots ; a_n$, we build an assignment M to the variables of φ such that I^0 holds, and for all i in $1..n$,

1. a_i^i is true in M, and a_j^i is false in M for all $j \neq i$,
2. $\mathit{Eff}^i_{a_i} \sigma^i_{mod_i} \sigma^i_{prev}(i)$ holds under M, and
3. $\bigwedge_{x \in Dom(\mathit{Eff}_{a_j})} x_j^i = x^i_{prev_x(j)}$ holds under M for all $j \neq i$.
4. $x^i = x_0^i$ and $x^{i+1} = x^i_{\rho_x(|A_x|)}$ hold under M for all $x \in var(S)$.

Now let us consider any particular step number i. According to 3, and by transitivity of equality, we have that $x^i_{prev_x(i)}$, which corresponds to x_j^i for some $a_j <_L a_i$, has the same value as x_0^i under M, for every variable x. Moreover, we have that $x^i = x_0^i$ holds by 4. Therefore, (the succedent of)

Constraint 7.1 will hold if a_i can be executed in step i of the sequential plan, which is the case, since by assumption the sequential plan is valid, and I^0 holds by construction.

Analogously to before, by 3 and 4, we have that x^{i+1} has the same value as x_i^i under M, for every variable x modified by a_i, i.e., the new value of x is carried forward to the next step. By induction, and validity of the sequential plan, this implies that G^n holds under M.

Note that Constraint 7.2 holds by 1 and 2, Constraint 7.3 holds by 1 and 3, and Constraint 7.4 by 4. Consistency of M follows from the validity of the sequential plan at hand. □

7.2.2 Removal of Redundant Actions

Roughly speaking, redundant actions are those that can be removed from a plan, resulting in a still valid plan. For example, in a logistics domain the objective normally specifies where the packets should end, but not the transports. Therefore, a transport could bring a packet to its destination, fulfilling part of the goal, but moving afterwards without any purpose. Hence, this last movement could be a redundant action.

The SAT and SMT translations of planning problems does not guarantee that every action that is true in the solution is actually needed in order to achieve the goals of the original plan. That is because during the search, there is the possibility that some variables denoting execution of actions that may be not relevant for achieving the goal are set to true by the SMT solver. Moreover, when a SAT or SMT solver reaches the point that it has a partial assignment that satisfies the goal condition of the problem and is asked for a model, it normally keeps giving values to the rest of the variables until it has a total assignment. These two factors, when paired with very parallel encodings, make that plans given by the SMT solver tend to contain some redundant actions. This effect is strictly needed to achieve the goal condition. The highly parallel encoding proposed makes the aforementioned issues more noticeable than with other encodings, so here an idea for optimizing plans given by the SMT solver is presented. For this approach, a MaxSMT solver will be used, as the idea is to add soft clauses that penalize the optimum when Boolean variables that represent the execution of actions are assigned to true by the solver.

Given a plan $\prod$ for the PMT problem, two new sets of clauses are added. The first is a set of *soft* clauses

$$\neg a^t \qquad\qquad a \in \prod \tag{7.6}$$

that will (softly) ask the solver to set to false all actions included in the plan. This will force the MaxSMT solver to set to true the minimum number of actions already included in the plan.

The second is a set of *hard* clauses to force the solver to not consider any action that was outside of the original plan

$$\neg a^t \qquad\qquad a \notin \prod \tag{7.7}$$

These last clauses will cause cheap unit propagation.

After solving the new problem, resulting plans will not be necessarily makespan-optimal, but the optimization cost with respect to solving time will almost always be negligible, while the quantity of actions pruned will be usually notable, as shown in the next section.

7.3 Empirical Evaluation

In this section we evaluate the impact of the presented encoding under the $R^2\exists$-step plan semantics and the proposed strategy for eliminating redundant actions. The proposed encoding, implemented in the RANTANPLAN system [BEV17] (R^2Chained onwards), is compared with the Semantic Chained encoding from Section 6.3, which uses the $\exists$-step semantics (from now on noted as *SEM+C*), and the two planners Springroll [SRHT16] and SMTPlan [CFLM16]. These two systems are the most recent non-heuristic numeric planners available. Springroll uses the planning as SMT approach, but with a $\forall$-step semantics. The planner focuses on producing more succinct encodings by "rolling up" an unbounded yet finite number of instances of an action into a single plan step. In problems where "foldable" actions occur, the planner is able to greatly reduce the number of time steps. SMTPlan proposes an approach to PDDL+ planning through SMT, with an encoding that captures all the features of the PDDL+ language. Its encoding focuses on domains with nonlinear and continuous change.

In [BB15] it was found experimentally that none of the considered orderings between actions could be clearly defined as the best, for the considered encoding under the $R^2\exists$-Step semantics for planning as SAT. In the first experiments, the ordering considered for the R^2Chained encoding is the order from which actions are read from the input files, which we refer as *dec*.

Some insights and results on more clever orderings, other than *dec*, are given in Section 7.4. It is also worth noticing that, although we choose the same order for each time step, the encoding is general enough to allow for a different order in each time step.

Solved	Springroll	SMTPlan	SEM+C	R^2C
Depots (22)	**7**	1	4	7
Driverlog (20)	12	7	11	**12**
Petrobras (60)	0	3	15	**51**
Planes (12)	3	5	7	**8**
Rovers (20)	12	4	6	**16**
Zenotravel (20)	*	6	**16**	15
Total	34	26	59	**109**

Table 7.1: Number of instances solved by each planner in each domain (total number of instances between parentheses), with a timeout of 1 hour. Boldface indicates the best results, with ties broken by total solving time. "*" denotes an execution problem.

We consider the domains of the third IPC [LF03] with integer numeric fluents and without quantified preconditions, as the rest of the domains contain features that are not commonly supported by the considered planners. These domains are: *Zenotravel*, *Driverlog*, *Depots* and *Rovers*. The previous *Petrobras* and *Planes* domains are also considered since they have a higher numerical component.

Experiments have been run on 8GB Intel® Xeon® E3-1220v2 machines at 3.10 GHz, using Yices [DDM06a] v2.5.1 as the back-end SMT solver, under the quantifier-free linear integer arithmetic logic [BST10]. Z3 [dMB08b] v4.5.1 is used as the MaxSMT solver to remove redundant actions. The total timeout is set to 1 hour.

Table 7.1 shows that with the R^2Chained (R^2C) encoding we are able to solve notably more instances than the rest of the approaches. Springroll is unable to process the Zenotravel domain.[1] The R^2Chained encoding dominates in most of the families. Thanks to the increased number of actions selected at each step, it generally needs fewer steps to find a valid plan than the rest of the planners. The number of Petrobras instances solved by this approach is noticeable. This domain is notably bigger in terms of formula size and more constrained in terms of resources than the rest. If this domain is set aside, the R^2Chained encoding still solves a few more instances than the other approaches.

[1]After reading the instance, it reports "Error in the encoding".

	Springroll	SMTPlan	SEM+C	R^2C	R^2O
depots1	13/6	13/6	13/6	13/2	13/2
depots7	29/10	-	25/10	30/4	25/4
depots16	34/8	-	-	59/3	33/3
driverlog5	30/8	25/8	18/4	30/4	20/4
driverlog6	26/5	17/5	21/4	22/3	22/3
petro-A2	-	9/3	11/4	11/2	10/2
petro-A6	-	-	33/9	38/5	34/5
petro-B15	-	-	-	82/2	57/2
petro-C1	-	-	14/3	34/2	5/2
planes2	17/16	18/11	17/11	19/7	19/7
planes3	19/17	27/13	19/10	23/7	23/7
planes8	-	-	24/12	27/7	25/7
rovers1	11/9	11/8	11/8	13/3	9/3
rovers4	31/6	10/6	8/5	8/1	8/1
rovers14	50/11	-	-	41/3	32/3
zeno7	*	16/6	19/3	18/2	16/2
zeno8	*	-	22/3	37/2	22/2

Table 7.2: Number of actions / number of steps, per instance and planner. "-" denotes a timeout. R^2O denotes the R^2C approach plus the redundant action removal presented in the previous section. "*" denotes an execution problem.

7.3.1 Plan Quality

Next we evaluate the plan quality in terms of makespan. As it has already been stated, the R^2Chained approach allows many more actions per time step. This increases the number of instances solved, since the number of time steps is in general smaller and, hence, so is the resulting formula. But unfortunately, this is bad in terms of plan quality, since it may add some redundant actions in the plan.

Table 7.2 shows the number of actions and time steps of the plans found in a selected number of instances. Comparing the R^2C and the R^2O columns we can see that the reduction is notable in all domains, except for Planes. We remark that the time spent on the process of removing redundant actions is negligible (typically less than two seconds).

In general, in instances where the reduction is small, the plan was already reasonably good, in terms of number of actions, compared to the rest of the planners. See for example instances driverlog6, petro-A2, planes2 or rovers4. In instances where the reduction is significant, the original plan was too long and the optimized one turns to be reasonably good compared to the others. See for instance depots7, depots16, driverlog5 or zeno8.

In particular, (see petro-C1 for example) there can be many agents (namely, the ships) that are not relevant for the plan objective, so the procedure can remove many actions. In contrast, the Planes domain is very tight, as there are very few agents that need to act, and thus all planners produce similar plans in terms of makespan.

7.3.2 Comparison of the R^2C against the SEM+C encoding

Since the R^2Chained encoding relaxes the SEM+C encoding, we provide some insights on why, in general, it behaves better. The relaxation is done by applying the idea of creating chains of assignments to all variables, while in the SEM+C encoding only a subset of them are eligible to be chained, due to the interference notion considered. In contrast to R^2Chained, there is no possible chain for Boolean variables in SEM+C. In fact, the difference between the two encodings can be seen as a trade-off between the need for mutex clauses and the extra number of chaining variables and linking constraints.

Moreover, the relaxation by the $R^2\exists$-step semantics of actions' applicability at the beginning of a time step, as well as that of the consistency of effects, not only allows us to solve more instances, but also to use many less time steps on the commonly solved ones.

R^2C w.r.t. SEM+C	clauses/ time step	variables/ time step	final num. variables	final num. clauses
Depots(4)	+24.1%	+39.3%	-48.1%	-54.1%
Driver.(11)	-1.7%	+18.1%	-12.5%	-28.2%
Petrobras(15)	-54.9%	+27.7%	-27.1%	-74.2%
Planes(7)	+2.1%	+14.3%	-26.2%	-29.7%
Rovers(6)	+65.5%	+52.4%	-48.2%	-52.0%
Zeno.(15)	-0.6%	+17.2%	-10.0%	-23.2%

Table 7.3: Average problem size difference between the R^2*Chained* encoding and the *SEM+C* encoding for each domain. The number of commonly solved instances is between parentheses. The first two columns show the average size difference in a single time step, and the second pair similarly but at the step where the solution is found.

Table 7.3 essentially shows that although R^2C uses more clauses and variables in each formula tested for satisfiability, it is able to solve the problem at an earlier step than SEM+C. This reduction in steps is especially noticeable in Depots and Rovers, where although the formula size per time step is bigger, the size of the last checked formula (i.e., the first satisfiable formula) is nearly cut in half. In the Petrobras domain, even using a semantic notion of interference, the SEM+C approach generates many mutexes, as there are many incompatibilities between actions. The removal of mutexes lets the R^2Chained encoding state the problem more compactly at each time step. This decrease in size per step, combined with the decrease in the number of needed time steps, lets the planner find a feasible solution with a reduction of nearly 75% of the problem clauses. The gains of these reductions are also reflected in Table 7.1, where the difference in the number of problems solved on the Depots, Rovers and Petrobras domains is noticeable.

To better illustrate the size of the final formulas shown on the second pair of columns of Table 7.3, in Table 7.4 the results on the number of steps are shown. Note that Depots and Rovers instances with the R^2Chained encoding need nearly a third of the steps needed by the SEM+C approach, and in the Petrobras domain this is nearly the half.

7.4 Orderings

In this section we try to reason about the effect of the selected ordering between actions on the efficiency of the encoding. Imagine a planning task

Time steps	SEM+C		R^2C	
	t. steps	avg. steps	t. steps	avg. steps
Depots (4)	37	9.25	14	3.50
Driverlog (11)	51	4.64	38	3.45
Petrobras (15)	72	4.80	34	2.26
Planes (7)	76	10.86	49	7.00
Rovers (6)	41	6.83	13	2.17
Zenotravel (15)	49	3.27	37	2.47

Table 7.4: Results on the number of time steps needed for the commonly solved instances between the R^2*Chained* and the *SEM+C* approaches. The number of commonly solved instances is shown between parentheses. Columns *t. steps* show the sum of all the steps of the commonly solved instances of each family, and columns *avg. steps* show the average steps per instance commonly solved.

with three actions a_1, a_2, a_3, applicable in the initial state and with a goal state that can be reached by the execution of the three tasks. Consider that a_1 disables a_2, a_2 disables a_3, and there are no further disabling relations. The disabling graph based encodings for $\exists$-step semantics of [RHN06] will not produce any constraints with respect to parallelism, because there are no cyclic disabling relations. Supposing that all actions are initially applicable and have consistent effects, it will find the plan a_3, a_2, a_1 with only one time step. However, bad orderings like $a_1 < a_2 < a_3$ would force the R^2Chained encoding to make three time steps.

Example 34. *To avoid this, consider the* rdfs *ordering, being the reverse of a depth-first search on the disabling graph. The rationale for this ordering is to try to minimize the number of possible interferences on actions appearing later in the ordering, and thus maximizing the number of actions potentially executed at the same time step.*

If we consider the previously presented task, the depth-first search would start on the node with least incoming edges (o_1) and continue then to o_2 and o_3. If we reverse the order in which nodes are explored, it would give $o_3 < o_2 < o_1$. The R^2C would then also be able to find a plan in one time step.

Note that with the $R^2\exists$-step semantics the one time step plan can sometimes become valid even if not all actions are applicable at the start or effects are not consistent. This could be because one action can enable the next one on the plan and non-commutative effects can now be sequentially applied.

	R^2C *dec*		R^2C informed *dec*		
	t. time	t. steps	t. time	t. steps	Δ_i
Depots(7)	7994.7	34	116.6	29	+2
Driverlog(11)	1734.3	49	2049.3	42	+2
Petrobras(49)	9546.3	150	57.1	57	-2
Planes(8)	1196.2	65	67.0	48	+2
Rovers(16)	375.5	72	257.4	58	+1
Zenotravel(15)	3735.6	52	631.6	42	+1
Total (96)	24582.6	422	3179.0	276	+6

Table 7.5: Total time and steps needed to solve the commonly solved instances with R^2Chained and the *dec* ordering, with and without informing. The number of commonly solved instances is shown between parentheses. Column Δ_i shows the difference in total instances solved.

Surprisingly, the *rdfs* ordering solved globally three fewer instances. Probably, this happens because the a-priori computed interferences are not a good indicator on how the actions should be ordered.

7.4.1 Informed Orders

Intuitively, a good ordering for the encoding at hand would be one inferred from a valid sequential plan, because the sequence of actions needed for a valid plan is strongly influenced by the objective and the initial state. Thus, finding an optimal ordering should be as hard as finding a plan itself.

To experimentally validate the previous assumption, we propose to *inform* a given total ordering using a sequential plan obtained from a relaxed version of the planning problem. This plan is obtained by using delete relaxation heuristics [BG01] on the original problem and removing the predicates belonging to the considered theory T. Once a plan is obtained by solving the relaxed problem, we extract an ordering by serializing this plan and removing duplicate occurrences of each action. Then, given a total ordering on the actions, we can *inform* it by only reordering the subset of actions appearing in the relaxed plan, according to the order in which they occur in the relaxed plan.

Table 7.5 shows that informing the previously used *dec* ordering, results in needing 146 steps fewer to solve the same number of instances than without informing it. This increase in parallelism is followed by a dramatic reduction on solving times (of about two orders of magnitude) in most of the families, resulting in 6 more instances solved.

	R^2C *dec*			R^2C informed *dec*		
	time	steps	actions	time	step	actions
depots4	3434,1	6	49/41	23,7	6	36/31
depots10	2379,1	4	39/39	7,9	3	37/27
driverlog11	146,0	4	32/26	2,2	3	29/27
driverlog14	-	-	-	178,2	4	51/48
petro-C15	372,9	3	78/57	0,9	1	62/62
petro-D15	428,8	3	80/58	0,9	1	62/62
planes6	690,7	9	28/28	37,6	8	38/32
planes11	-	-	-	871,8	8	42/41
rovers9	58,6	9	61/36	35,4	7	70/36
rovers11	7,0	5	56/35	4,7	3	62/42
zeno15	831,6	4	71/60	585,4	3	66/56
zeno16	-	-	-	3113,7	3	72/61

Table 7.6: Solving time in seconds, time steps and actions on a sample of instances, with and without informing.

Additionally to the *dec* ordering, we also informed the *rdfs* ordering and its inverted versions. Table 7.7 depicts how many instances each order solved in the given timeout and the difference without informing it. None of the orderings was clearly better. However, the total gains on the number of solved instances by informing them was always positive, ranging from 4 to 9 extra instances, experimentally supporting the intuition. For example in the Petrobras domain, informing the order allows it to capture in the right order the pattern $[load, undock, sail, dock, unload]$, reducing even more the needed time steps and reducing solving times by more than one order of magnitude.

Regarding plan quality, with respect to the number of actions, experiments with informed orderings show that no significant change can be seen. Note that fewer time steps imply less space for possible redundant actions, but more parallelism can also mean more selected actions per time step.

7.5 Relation with Macro-Actions

One could say that the Relaxed Chained Encoding has some similarities with some techniques regarding macro-actions. The main idea of a macro-action [Kor85b, Min85] is to express the combination of one or more actions. Some methods for automatically learning macro-actions have been

Δ_i	inf. *dec*	inf. *rdec*	inf. *dfs*	inf. *rdfs*
Depots	9/+2	7/+3	5/+1	6/+1
Driverlog	14/+2	14/0	12/+2	14/+1
Petrobras	49/-2	59/+4	55/+2	59/+2
Planes	10/+2	8/+1	8/0	9/0
Rovers	17/+1	17/+1	7/0	7/0
Zenotravel	16/+1	14/0	14/-1	15/0
Total	115/+6	119/+9	105/+4	110/+4

Table 7.7: Total number of instances solved for each domain and each informed order. After the slash the difference without informing the order is shown.

developed [BEMS05]. Also, macro-actions have been extended for numeric planning problems [Sca14].

The way actions are sequenced in a time step is de-facto a combination of regression and progression of precondition and action effects. This can be achieved via substitution, and it amounts to computing weakest precondition and cumulative effects, which is how numeric macro-actions can be built. However, a difference with macro-actions is that, in the presented encoding, the actions that are going to be sequenced are not fixed. In other words, the Relaxed Chained Encoding encoding benefits from letting the solver decide which subsequence is necessary to use.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

The Planning as SAT approach has suffered from the beginning with formula sizes. In fact, Kautz et al. [KMS96] transported the notion of parallel plans to the planning as SAT approach to overcome this problem. The main motivation for using parallel plans was that it could compactly represent all intermediate states of a sequential plan. The reduced number of explicitly represented states translated to smaller formulas and sometimes to a more easily solvable problem. Therefore, a compact encoding was proven to be essential. This idea has been present during all the thesis. The presented encodings help in generating formulas with smaller search spaces. This reduction of the search space (i.e. the increase of parallelism) has shown to be effective at solving more planning problems.

The Direct Approximation

The first approach in this thesis has been to generalize the ideas of Planning as SAT to the SMT setting. The Generalized SMT Encoding proposed in Section 5.2 tightly integrates arithmetic into the problem, generalizing Rintanen's approach to Planning as SAT [RHN06]. The other considered approximations rely into making state-space exploration on the numerical variables, or loosely integrate external solvers for evaluating arithmetic constraints (and therefore not being able to infer anything from the numerical side). The Generalized SMT Encoding proved to be competitive with other exact and complete methods for planning with resources on the Petrobras problem, and also with some incomplete (heuristic) ones. In particular, we have obtained better results than NumReach [HGSK07] and similar results to the heuristic planner SGPlan [CWH06]. We have seen that the method of [HGSK07], which is based on approximating the reachable domains of numeric variables, is very sensitive to the number of distinct possible values, and it is not well-suited for this real real-life problem.

A Semantic Notion of Interference and its application

From here onwards, our main goal in this thesis has been to work in the direction of increasing the parallelism in the Planning as SMT approach. As we said, if we can increase the number of actions in a single time step, this generally leads to smaller formulas and easier problems.

Our objective was then to reduce the incompatibilities to a minimum. In this direction, the first contribution is the formalization of an elegant solution to the problem of determining interference between actions in Planning Modulo Theories. This notion can moreover be implemented as a set of satisfiability checks of SMT formulas. We introduced a new relaxed semantics for the parallel execution of actions, and formalized a semantic notion of interference, that are suitable for both $\forall$-step and $\exists$-step plans in the context of Planning Modulo Theories.

We also showed how this notion can be applied by proposing the Chained SMT Encoding in Section 6.3. This encoding resulted to be generally better than encodings with purely syntactic approaches to interference. We provided empirical evidence of its usefulness by showing a significant improvement in parallelism in some domains. The gains in parallelism were due to the semantic checks, as they notably reduced the number of useless interferences as a pre-processing step. It should be noted that the presented checks are not only useful, but can also be done independently of the underlying planning system. Therefore, any other planner could implement them as a preprocessing step. In fact, a recent work [IM17] has already benefited from this approach since presenting this idea to the community [BEV16a].

Relaxed Semantics for Planning as SMT

As we said, our main goal in this thesis has been to work in the direction of increasing the parallelism in the Planning as SMT approach. Based on the Relaxed Relaxed Exists-step semantics, our second effort has been able to change the paradigm by devising an encoding that does not need any mutex.

We proved its correctness and completeness and shown experimentally that it generally pays off.

We generalized the idea of a "chain of assignments" introduced with the previous Chained SMT Encoding to all kinds of variables. We later introduced in Section 7.2 the Relaxed Chained Encoding, which makes use of these chains. The experimental section empirically proves that this encoding allows for more parallelism than that of the rest of the considered planners. The approach lets the planner put more actions per time step, but it also makes the search space wider at each time step considered, and hence infeasible (too short) plans are more difficult to refute. However, in general a shorter time horizon pays off in terms of formula size and solving time, as it can be seen from the empirical results.

As we have seen, making encodings more parallel can also introduce redundant actions in the final plans. A post-processing step for eliminating redundant actions from resulting plans has also been proposed. It has proven to be cheap in terms of solving time, and useful for maintaining plan quality when considering highly parallel encodings. However, further work in the direction of finding good orderings for the Relaxed Chained Encoding is needed. In fact, finding an optimal ordering could be as hard as finding a plan itself.

Closing Remarks

In our contributions, we were able to gradually make encodings more parallel, reducing total formula size. We also enabled more actions per time step and therefore solved notably more problems. Therefore, we consider that the proposed objectives for the thesis have succeeded. All of our work has been implemented in RANTANPLAN, a planner that reformulates numeric planning problems into SMT instances and solve them using SMT solvers as black boxes. Thanks to these implementations, we have proven repeatedly to be competitive with the state of the art numeric exact planners.

Although our main efforts have been with numeric SMT theories, many of our efforts are independent of what theory the user may need, and therefore our ideas can be applied to other theories with little to no effort. We also explored the use of various SMT Theories for solving numeric planning problems, including the usage of QF_UFLIA in Section 5.5.

8.2 Future Work

Although good results have been obtained with the Relaxed Chained Encoding, We believe that there is still much room of improvement on selecting optimal orders for it. Many heuristic approaches explained in Section 2.3.1 simplify the problem beforehand to get useful information. As the main step to obtain an order for the Relaxed Chained Encoding is to solve a relaxed version of the problem, the aforementioned simplifications should be studied to see if they can be useful in this setting.

One type of domains where all the presented encodings struggle are domains where agents move in a given grid. This happens because when an agent has to move repeatedly in one direction, the same action has to be taken repeatedly. If the encoding only encodes each action once per time step, it does not matter how much parallel the encoding is, because the agent will only be able to take the move action once per step.

To solve this kind of problems, Springroll [SRHT16] implements a very interesting concept of "rolling up" an unbounded yet finite number of instances of an action into a single plan step. Some problems where "foldable" actions occur, the planner is able to greatly reduce the number of time steps. It should be studied how its concepts could be implemented in the Relaxed Chained Encoding, as it could greatly help in grid-like domains.

In [Rin12a] Rintanen modified a SAT solver to implement new heuristics specifically for solving planning problems. By doing this, this new heuristics replaced VSIDS. He also greatly reduced memory needs by using the fact that between time steps the structure of the problem does not change. A similar approach should be evaluated, as most SMT solvers use SAT solvers internally.

With the considered encodings, some problems are too large to be managed by a SMT solver. Another and wider line of work could be to consider incorporating more powerful inference capabilities to existing heuristic numeric planners, by making calls to an external SMT solver.

Bibliography

[ABP$^+$11] Carlos Ansótegui, Miquel Bofill, Miquel Palahí, Josep Suy, and Mateu Villaret. Satisfiability Modulo Theories: An Efficient Approach for the Resource-Constrained Project Scheduling Problem. In *Proceedings of the 9th Symposium on Abstraction, Reformulation, and Approximation (SARA)*, pages 2–9, 2011.

[Ack54] W. Ackermann. *Solvable cases of the decision problem.* Studies in Logic and the Foundations of Mathematics. 1954.

[AN17] Johannes Aldinger and Bernhard Nebel. Interval based relaxation heuristics for numeric planning with action costs. In *Proceedings of the Tenth International Symposium on Combinatorial Search (SOCS)*, pages 155–156, 2017.

[Bäc92] Christer Bäckström. Equivalence and Tractability Results for SAS+ Planning. In *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 126–137, 1992.

[Bäc95] Christer Bäckström. Expressive Equivalence of Planning Formalisms. *Artificial Intelligence*, 76(1-2):17–34, 1995.

[Bal13] Tomas Balyo. Relaxing the Relaxed Exist-Step Parallel Planning Semantics. In *IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 865–871, 2013.

[Bat68] Kenneth E. Batcher. Sorting Networks and Their Applications. In *Proceedings of the spring joint computer conference*, pages 307–314. ACM, 1968.

[BB15] Tomas Balyo and Roman Barták. No One SATPlan Encoding To Rule Them All. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SOCS)*, pages 146–150, 2015.

[BBC+06] Marco Bozzano, Roberto Bruttomesso, Alessandro Cimatti, Tommi A. Junttila, Silvio Ranise, Peter van Rossum, and Roberto Sebastiani. Efficient Theory Combination via Boolean Search. *Information and Computation*, 204(10):1493–1525, 2006.

[BBT15] Tomáš Balyo, Roman Barták, and Otakar Trunda. Reinforced encoding for planning as SAT. *Acta Polytechnica CTU Proceedings*, 2(2):1–7, 2015.

[BCC+09] Keith Bell, Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. The role of AI planning as a decision support tool in power substation management. *AI Communications*, 22(1):37–57, 2009.

[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic Model Checking without BDDs. In *Tools and Algorithms for Construction and Analysis of Systems, (TACAS)*, pages 193–207, 1999.

[BCF+06] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. Delayed Theory Combination vs. Nelson-Oppen for Satisfiability Modulo Theories: A Comparative Analysis. In *Logic for Programming, Artificial Intelligence, and Reasoning, 13th International Conference (LPAR)*, pages 527–541, 2006.

[BCK14] Tomas Balyo, Lukás Chrpa, and Asma Kilani. On Different Strategies for Eliminating Redundant Actions from Plans. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search, (SOCS)*, 2014.

[BEG+14] Miquel Bofill, Joan Espasa, Marc Garcia, Miquel Palahí, Josep Suy, and Mateu Villaret. Scheduling B2B meetings. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP)*, volume 8656 of *LNCS*, pages 781–796, 2014.

[BEMS05] Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer. Macro-FF: Improving AI Planning with Automatically Learned Macro-Operators. *Journal of Artificial Intelligence Research (JAIR)*, 24:581–621, 2005.

[BEV14] Miquel Bofill, Joan Espasa, and Mateu Villaret. Efficient SMT Encodings for the Petrobras Domain. In *Proceedings of the 13th International Workshop on Constraint Modelling and Reformulation (ModRef 2014)*, pages 68–84, Lyon, France, 2014.

[BEV15] Miquel Bofill, Joan Espasa, and Mateu Villaret. The RANTANPLAN Planner: System Description. In *Proceedings of the ICAPS-15 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS)*, pages 1–10, 2015.

[BEV16a] Miquel Bofill, Joan Espasa, and Mateu Villaret. A Semantic Notion of Interference for Planning Modulo Theories. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS)*, pages 56–64, 2016.

[BEV16b] Miquel Bofill, Joan Espasa, and Mateu Villaret. The RANTANPLAN planner: System description. *Knowledge Engineering Review*, 31(5):452–464, 2016.

[BEV17] Miquel Bofill, Joan Espasa, and Mateu Villaret. Relaxed exists-step plans in planning as SMT. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, (IJCAI)*, pages 563–570, 2017.

[BF95] Avrim Blum and Merrick L Furst. Fast Planning Through Planning Graph Analysis. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, (IJCAI)*, pages 1636–1642, 1995.

[BF97] Avrim Blum and Merrick L Furst. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence*, 90(1-2):281–300, 1997.

[BG99] Blai Bonet and Hector Geffner. Planning as Heuristic Search: New Results. In *Recent Advances in AI Planning, 5th Eu-*

ropean Conference on Planning (ECP 1999), pages 360–372, 1999.

[BG01] Blai Bonet and Hector Geffner. Planning as Heuristic Search. *Artificial Intelligence*, 129(1-2):5–33, 2001.

[BHvMW09] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[BJMR05] John L. Bresina, Ari K. Jónsson, Paul H. Morris, and Kanna Rajan. Activity Planning for the Mars Exploration Rovers. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS)*, pages 40–49, 2005.

[BKO$^+$07] Randal E. Bryant, Daniel Kroening, Joël Ouaknine, Sanjit A. Seshia, Ofer Strichman, and Bryan A. Brady. Deciding Bit-Vector Arithmetic with Abstraction. In *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4424 of *LNCS*, pages 358–372, 2007.

[BKS04] Paul Beame, Henry A. Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research (JAIR)*, 22:319–351, 2004.

[BLO$^+$12] Cristina Borralleras, Salvador Lucas, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. SAT Modulo Linear Arithmetic for Solving Polynomial Constraints. *Journal of Automated Reasoning*, 48(1):107–131, 2012.

[BM10] Milan Bankovic and Filip Maric. An Alldifferent Constraint Solver in SMT. In *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (SMT)*, 2010.

[BM12] Lamia Belouaer and Frédéric Maris. SMT Spatio-Temporal Planning. In *ICAPS Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS)*, pages 6–15, 2012.

[BNO$^+$08] Miquel Bofill, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. A Write-Based Solver for SAT Modulo the Theory of Arrays. In *Proceedings of the 8th Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pages 1–8, 2008.

[BRBT16] Kshitij Bansal, Andrew Reynolds, Clark W. Barrett, and Cesare Tinelli. A New Decision Procedure for Finite Sets and Cardinality Constraints in SMT. In *Automated Reasoning - 8th International Joint Conference (IJCAR)*, pages 82–98, 2016.

[BST10] Clark Barrett, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2010.

[BSV10] Miquel Bofill, Josep Suy, and Mateu Villaret. A System for Solving Constraint Satisfaction Problems with SMT. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 6175 of *LNCS*, pages 300–305, 2010.

[BvdB14] Blai Bonet and Menkes van den Briel. Flow-Based Heuristics for Optimal Planning: Landmarks and Merges. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, (ICAPS)*, 2014.

[Byl91] Tom Bylander. Complexity results for planning. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 274–279, 1991.

[BZ13] Roman Barták and Neng-Fa Zhou. On Modeling Planning Problems: Experience from the Petrobras Challenge. In *Advances in Soft Computing and Its Applications - 12th Mexican International Conference on Artificial Intelligence (MICAI)*, pages 466–477, 2013.

[CFG$^+$10] Alessandro Cimatti, Anders Franzén, Alberto Griggio, Roberto Sebastiani, and Cristian Stenico. Satisfiability Modulo the Theory of Costs: Foundations and Applications. In *Proceedings of the 16h International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 6015 of *LNCS*, pages 99–113, 2010.

[CFLM16] Michael Cashmore, Maria Fox, Derek Long, and Daniele Magazzeni. A compilation of the full PDDL+ language into SMT. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS)*, pages 79–87, 2016.

[CFLS08] Andrew Coles, Maria Fox, Derek Long, and Amanda Smith. A Hybrid Relaxed Planning Graph-LP Heuristic for Numeric Planning Domains. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS*, pages 52–59, 2008.

[CFM12] Lucas C. Cordeiro, Bernd Fischer, and João Marques-Silva. SMT-Based Bounded Model Checking for Embedded ANSI-C Software. *IEEE Transactions on Software Engineering*, 38(4):957–974, 2012.

[CWH06] Yixin Chen, Benjamin W. Wah, and Chih-Wei Hsu. Temporal Planning using Subgoal Partitioning and Resolution in SGPlan. *Journal of Artificial Intelligence Research (JAIR)*, 26:323–369, 2006.

[CXZ07] Yixin Chen, Zhao Xing, and Weixiong Zhang. Long-distance mutual exclusion for propositional planning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, (IJCAI)*, pages 1840–1845, 2007.

[DDM06a] Bruno Dutertre and Leonardo De Moura. The Yices SMT Solver. Technical report, Computer Science Laboratory, SRI International, 2006. `http://yices.csl.sri.com`.

[DdM06b] Bruno Dutertre and Leonardo Mendonça de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *Proceedings of the 18th International Conference Computer Aided Verification (CAV)*, volume 4144 of *LNCS*, pages 81–94, 2006.

[DIMM10] Giuseppe Della Penna, Benedetto Intrigila, Daniele Magazzeni, and Fabio Mercorio. Planning for Autonomous Planetary Vehicles. In *Sixth International Conference on Autonomic and Autonomous Systems, (ICAS)*, pages 131–136, 2010.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A Machine Program for Theorem-Proving. *Communications of the ACM*, 5(7):394–397, 1962.

[dMB08a] Leonardo Mendonça de Moura and Nikolaj Bjørner. Model-based Theory Combination. *Electronic Notes in Theoretical Computer Science*, 198(2):37–49, 2008.

[dMB08b] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems TACAS*, pages 337–340, 2008.

[dMB08c] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 4963 of *LNCS*, pages 337–340, 2008.

[dMR04] L. de Moura and H. Ruess. An Experimental Evaluation of Ground Decision Procedures. In *Proceedings of the 16th International Conference Computer Aided Verification (CAV)*, volume 3114 of *LNCS*, pages 162–174, 2004.

[DNK97] Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding Planning Problems in Nonmonotonic Logic Programs. In *Recent Advances in AI Planning, 4th European Conference on Planning (ECP)*, pages 169–181, 1997.

[DP60] Martin Davis and Hilary Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7(3):201–215, 1960.

[EMW97] Michael D. Ernst, Todd D. Millstein, and Daniel S. Weld. Automatic SAT-Compilation of Planning Problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, (IJCAI)*, pages 1169–1177, 1997.

[ENS92] Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian. On the Complexity of Domain-Independent Planning. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI)*, pages 381–386, 1992.

[ES03] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In *Theory and Applications of Satisfiability Testing, 6th International Conference (SAT)*, pages 502–518, 2003.

[FG10] Alan M. Frisch and Paul A. Giannaros. SAT Encodings of the At-Most-k Constraint. Some Old, Some New, Some Fast, Some Slow. In *10th International Workshop on Constraint Modelling and Reformulation (ModRef)*, 2010.

[FL02] Maria Fox and Derek Long. PDDL+: Modeling continuous time dependent effects. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, volume 4, page 34, 2002.

[FL03] Maria Fox and Derek Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research (JAIR)*, 20:61–124, 2003.

[FN71] Richard Fikes and Nils J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3/4):189–208, 1971.

[Gab00] Harold N. Gabow. Path-Based Depth-First Search for Strong and Biconnected Components. *Information Processing Letters*, 74(3-4):107–114, 2000.

[Gef00] Héctor Geffner. Functional STRIPS: a more flexible language for planning and problem solving. In *Logic-based artificial intelligence*, pages 187–209. Springer, 2000.

[GKSS08] Carla P. Gomes, Henry A. Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability Solvers. In *Handbook of Knowledge Representation*, pages 89–134. 2008.

[GLFB12] Peter Gregory, Derek Long, Maria Fox, and J. Christopher Beck. Planning Modulo Theories: Extending the Planning Paradigm. In *Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI, 2012.

[GNT04] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004.

[GNT16] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning and Acting*. 2016.

[GSC97] Carla P. Gomes, Bart Selman, and Nuno Crato. Heavy-Tailed Distributions in Combinatorial Search. In *Principles and Practice of Constraint Programming (CP)*, pages 121–135, 1997.

[GSS05] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. Automated planning with numerical variables in LPG. *Intelligenza Artificiale*, 2(4):51–57, 2005.

[GSS08] Alfonso E Gerevini, Alessandro Saetti, and Ivan Serina. An approach to efficient planning with numerical fluents and multi-criteria plan quality. *Artificial Intelligence*, 172(8):899–944, 2008.

[HBG05] Patrik Haslum, Blai Bonet, and Hector Geffner. New Admissible Heuristics for Domain-Independent Planning. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI)*, pages 1163–1168, 2005.

[HBH$^+$07] Patrik Haslum, Adi Botea, Malte Helmert, Blai Bonet, and Sven Koenig. Domain-Independent Construction of Pattern Database Heuristics for Cost-Optimal Planning. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 1007–1012, 2007.

[Hel02] Malte Helmert. Decidability and Undecidability Results for Planning with Numerical State Variables. In *Sixth International Conference on Artificial Intelligence, Planning and Scheduling (AIPS)*, pages 303–312, 2002.

[HG00] Patrik Haslum and Hector Geffner. Admissible Heuristics for Optimal Planning. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 140–149, 2000.

[HGSK07] Jörg Hoffmann, Carla P. Gomes, Bart Selman, and Henry A. Kautz. SAT Encodings of State-Space Reachability Problems in Numeric Domains. In *20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1918–1923, 2007.

[HLF04] Richard Howey, Derek Long, and Maria Fox. VAL: Automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 294–301, 2004.

[HN01] Jörg Hoffmann and Bernhard Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research (JAIR)*, 14:253–302, 2001.

[HNR72] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *SIGART Newsletter*, 37:28–29, 1972.

[Hof02] Jörg Hoffmann. Local Search Topology in Planning Benchmarks: A Theoretical Analysis. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 92–100, 2002.

[Hof03] Jörg Hoffmann. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *Journal of Artificial Intelligence Research (JAIR)*, 20:291–341, 2003.

[HPS04] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. Ordered landmarks in planning. *Journal of Artificial Intelligence Research (JAIR)*, 22:215–278, 2004.

[Hüt16] Christian Hütter. More Shuttles, Less Cost: Energy Efficient Planning for Scalable High-Density Warehouse Environments. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS*, pages 403–411, 2016.

[IM17] Leon Illanes and Sheila A. McIlraith. Numeric planning via abstraction and policy guided search. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4338–4345, 2017.

[Int70] SRI International. Shakey the robot. "goo.gl/2fzffX", 1970. [Online; accessed 19-July-2016].

[JHFK12] Maximilian Junker, Ralf Huuck, Ansgar Fehnker, and Alexander Knapp. SMT-Based False Positive Elimination in Static Program Analysis. In *14th International Conference on Formal Engineering Methods, (ICFEM)*, pages 316–331, 2012.

[JS97] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP Look-Back Techniques to Solve Real-World SAT Instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, (AAAI) 97, (IAAI) 97*, pages 203–208, 1997.

[Kah62] A. B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5(11):558–562, 1962.

[Kau06] Henry A. Kautz. Deconstructing Planning as Satisfiability. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference*, pages 1524–1526, 2006.

[KD08] Michael Katz and Carmel Domshlak. Optimal additive composition of abstraction-based admissible heuristics. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS)*, pages 174–181, 2008.

[KD09] Erez Karpas and Carmel Domshlak. Cost-Optimal Planning with Landmarks. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1728–1733, 2009.

[KK07] Will Klieber and Gihwon Kwon. Efficient CNF encoding for selecting 1 from n objects. In *Proccedings of the International Workshop on Constraints in Formal Verification (CFV)*, 2007.

[KMS96] Henry A. Kautz, David A. McAllester, and Bart Selman. Encoding Plans in Propositional Logic. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 374–384, 1996.

[KN05] Shin-ichiro Kawano and Shin-Ichi Nakano. Constant Time Generation of Set Partitions. *IEICE Transactions*, 88-A(4):930–934, 2005.

[Kno94] Craig A. Knoblock. Generating parallel execution plans with a partial-order planner. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems (AIPS)*, pages 98–103, 1994.

[Kor85a] Richard E. Korf. Iterative-Deepening-A*: An Optimal Admissible Tree Search. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1034–1036, 1985.

[Kor85b] Richard E. Korf. Macro-Operators: A Weak Method for Learning. *Artificial Intelligence*, 26(1):35–77, 1985.

[KS92] Henry A. Kautz and Bart Selman. Planning as Satisfiability. In *European Conference on Artificial Intelligence (ECAI)*, pages 359–363, 1992.

[KS98] Henry Kautz and Bart Selman. BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving. In *Workshop on Planning as Combinatorial Search (AIPS)*, volume 58260, pages 58–60, 1998.

[KS99] Henry A. Kautz and Bart Selman. Unifying sat-based and graph-based planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, (IJCAI)*, pages 318–325, 1999.

[KSH06] Henry A. Kautz, Bart Selman, and Jörg Hoffmann. SatPlan: Planning as Satisfiability. Abstracts of the International Planning Competition (IPC-05), 2006.

[KW99a] Henry Kautz and Joachim P Walser. State-Space Planning by Integer Optimization. In *Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI)*, pages 526–533, 1999.

[KW99b] Henry A. Kautz and Joachim P. Walser. State-space Planning by Integer Optimization. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence (AAAI/IAAI)*, pages 526–533, 1999.

[LCLK16] Kwan Hui Lim, Jeffrey Chan, Christopher Leckie, and Shanika Karunasekera. Towards Next Generation Touring: Personalized Group Tours. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, (ICAPS)*, pages 412–420, 2016.

[LEKN12] Johannes Löhr, Patrick Eyerich, Thomas Keller, and Bernhard Nebel. A Planning Based Framework for Controlling Hybrid Systems. In *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS)*, 2012.

[LF03] Derek Long and Maria Fox. The 3rd International Planning Competition: Results and Analysis. *Journal of Artificial Intelligence Research (JAIR)*, 20:1–59, 2003.

[MBL09] Jean-Philippe Métivier, Patrice Boizumault, and Samir Loudni. Solving Nurse Rostering Problems Using Soft Global Constraints. In *International Conference on Principles and Practice of Constraint Programming (CP)*, volume 5732 of *LNCS*, pages 73–87, 2009.

[McC69] McCarthy, J. and Hayes, P.J. Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intelligence*, 4:463–502, 1969.

[Min85] Steven Minton. Selectively Generalizing Plans for Problem-Solving. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 596–599, 1985.

[MMZ$^+$01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC)*, pages 530–535, 2001.

[MR91] David A. McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the 9th National Conference on Artificial Intelligence (AAAI)*, pages 634–639, 1991.

[NII02] Hidetomo Nabeshima, Koji Iwanuma, and Katsumi Inoue. Effective SAT Planning by Speculative Computation. In *15th Australian Joint Conference on Artificial Intelligence (AI)*, pages 726–728, 2002.

[NO79] G. Nelson and D. C. Oppen. Simplification by Cooperating Decision Procedures. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1(2):245–257, 1979.

[NO05] Robert Nieuwenhuis and Albert Oliveras. DPLL(T) with exhaustive theory propagation and its application to difference logic. In *Proceedings of the 17th International Conference Computer Aided Verification (CAV)*, volume 3576 of *LNCS*, pages 321–334, 2005.

[NOT06] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.

[PLF$^+$17] Simon Parkinson, Andrew Longstaff, Simon Fletcher, Mauro Vallati, and Lukás Chrpa. On the Exploitation of Automated Planning for Reducing Machine Tools Energy Consumption between Manufacturing Operations. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, (ICAPS)*, pages 400–408, 2017.

[RG15] Masood Feyzbakhsh Rankooh and Gholamreza Ghassem-Sani. ITSAT: An Efficient SAT-Based Temporal Planner. *Journal of Artificial Intelligence Research (JAIR)*, 53:541–632, 2015.

[RGPS08] Nathan Robinson, Charles Gretton, Duc Nghia Pham, and Abdul Sattar. A Compact and Efficient SAT Encoding for Planning. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, (ICAPS)*, pages 296–303, 2008.

[RGPS09] Nathan Robinson, Charles Gretton, Duc Nghia Pham, and Abdul Sattar. SAT-Based Parallel Planning Using a Split Representation of Actions. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, (ICAPS)*, 2009.

[RGPS10] Nathan Robinson, Charles Gretton, Duc Nghia Pham, and Abdul Sattar. Partial Weighted MaxSAT for Optimal Planning. In *11th Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, pages 231–243, 2010.

[RHN04] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Parallel Encodings of Classical Planning as Satisfiability. In *Logics* *in Artificial Intelligence, 9th European Conference (JELIA)*, pages 307–319, 2004.

[RHN06] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Planning as Satisfiability: Parallel Plans and Algorithms for Plan Search. *Artificial Intelligence*, 170(12-13):1031–1080, 2006.

[RHW08] Silvia Richter, Malte Helmert, and Matthias Westphal. Landmarks Revisited. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, (AAAI)*, pages 975–982, 2008.

[Rin09] Jussi Rintanen. Planning and SAT. In *Handbook of Satisfiability*, pages 483–504. 2009.

[Rin12a] Jussi Rintanen. Engineering Efficient Planners with SAT. In *20th European Conference on Artificial Intelligence (ECAI)*, pages 684–689, 2012.

[Rin12b] Jussi Rintanen. Planning as satisfiability: Heuristics. *Artificial Intelligence*, 193:45–86, 2012.

[Rin15] Jussi Rintanen. Discretization of Temporal Models with Application to Planning with SMT. In *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence (AAAI)*, pages 3349–3355, 2015.

[Rin18] Jussi Rintanen. Jussi Rintanen: Planning as Satisfiability. "https://users.aalto.fi/ rintanj1/satplan.html", 2018. [Online; accessed 22-July-2018].

[RN10] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach.* Pearson Education, 2010.

[Rob65] J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM*, 12(1):23–41, 1965.

[Rya04] Lawrence Ryan. *Efficient algorithms for clause-learning SAT solvers.* PhD thesis, Master Thesis (School of Computing Science)/Simon Fraser University, 2004.

[SBDL01] A. Stump, C. W. Barrett, D. L. Dill, and J. R. Levitt. A Decision Procedure for an Extensional Theory of Arrays. In *Proceedings of the 16th Annual Symposium on Logic in Computer Science (LICS)*, pages 29–37, 2001.

[Sca14] Enrico Scala. Plan Repair for Resource Constrained Tasks via Numeric Macro Actions. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling (ICAPS)*, 2014.

[SD05] Ji-Ae Shin and Ernest Davis. Processes and continuous change in a SAT-based planner. *Artifical Intelligence*, 166(1-2):194–253, 2005.

[Seb07] Roberto Sebastiani. Lazy Satisability Modulo Theories. *JSAT*, 3(3-4):141–224, 2007.

[Sho84] Robert E. Shostak. Deciding Combinations of Theories. *Journal of the ACM*, 31(1):1–12, 1984.

[SNT98] Stephen J. J. Smith, Dana S. Nau, and Thomas A. Throop. Success in Spades: Using AI Planning Techniques to Win the World Championship of Computer Bridge. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, (AAAI)*, pages 1079–1086, 1998.

[SRHT16] Enrico Scala, Miquel Ramírez, Patrik Haslum, and Sylvie Thiébaux. Numeric Planning with Disjunctive Global Constraints via SMT. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS)*, pages 276–284, 2016.

[SS77] Richard M. Stallman and Gerald J. Sussman. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artificial Intelligence*, 9(2):135–196, 1977.

[SS99] João P. Marques Silva and Karem A. Sakallah. GRASP: A Search Algorithm for Propositional Satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[TDT$^+$12] Daniel Toropila, Filip Dvorak, Otakar Trunda, Martin Hanes, and Roman Barták. Three Approaches to Solve the Petrobras Challenge: Exploiting Planning Techniques for Solving Real-Life Logistics Problems. In *IEEE 24th International Conference on Tools with Artificial Intelligence, (ICTAI)*, pages 191–198, 2012.

[vdBBKV07] Menkes van den Briel, J. Benton, Subbarao Kambhampati, and Thomas Vossen. An LP-Based Heuristic for Optimal Planning. In *Principles and Practice of Constraint Programming (CP)*, pages 651–665, 2007.

[VIV13] Simon Vernhes, Guillaume Infantes, and Vincent Vidal. Problem Splitting Using Heuristic Search in Landmark Orderings. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2401–2407, 2013.

[WR07] Martin Wehrle and Jussi Rintanen. Planning as Satisfiability with Relaxed Exists-Step Plans. In *Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence (AI)*, pages 244–253, 2007.

[WW99] Steven A. Wolfman and Daniel S. Weld. The LPSAT engine & its application to resource planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 310–317, 1999.

[WW01] Steven A. Wolfman and Daniel S. Weld. Combining Linear Programming and Satisfiability Solving for Resource Planning. *Knowledge Engineering Review*, 16(1):85–99, 2001.

[Zha09] Hantao Zhang. Combinatorial designs by SAT solvers. In *Handbook of Satisfiability*, pages 533–568. 2009.

[ZM02] Lintao Zhang and Sharad Malik. The Quest for Efficient Boolean Satisfiability Solvers. In *Proceedings of the 14th International Conference Computer Aided Verification (CAV)*, volume 2404 of *LNCS*, pages 17–36, 2002.

[ZMMM01] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient Conflict Driven Learning in Boolean Satisfiability Solver. In *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design, (ICCAD)*, pages 279–285, 2001.

Alphabetical Index

Appendices

The Rantanplan Planner

Rantanplan is a planning system that implements the main ideas described in the previous chapters. It has been developed to empirically test the ideas presented in this thesis.

Rantanplan supports a fragment of PDDL which is close to general numeric PDDL 2.1, excluding the temporal extensions and metric optimizations. With respect to numeric effects, we consider $assign(x, exp)$, $increase(x, exp)$ and $decrease(x, exp)$, where exp is any expression over linear integer (or real) arithmetic. With respect to preconditions and conditions of numeric effects, we assume that the restrictions imposed on numeric fluents take the form of any SMT formula over linear integer (or real) arithmetic.

The structure of the Rantanplan planning system is represented in Figure 1. The first step is to parse and do some preprocessing on the PDDL instance. For example, arithmetic operations that can be solved at compile time are simplified and forall expressions are flattened.

Then, the PDDL instance is encoded to SMT. To encode the formulas ϕ_0, ϕ_1, ϕ_2, $\ldots$, various encodings can be used, transforming the PDDL problem to a pure SMT problem. Then the problem is iteratively solved, using the chosen SMT Solver as a black box.

A key aspect of the planner is the detection of interferences between parallel actions at compile time, by means of calls to a SMT Solver. In case the user demands a parallel plan, a disabling graph is computed. Broadly speaking, by *disabling graph* we refer to a directed graph, where nodes are the grounded actions from the planning problem and an edge exists from action a to action a' if the execution of a can affect a' (forbid its execution or change its active effects). Section 5.3 covers the disabling graph in detail. This graph is used, depending on the notion of parallelism chosen, to encode the necessary constraints restricting which actions can be carried out at the same time step.

The system supports solving via API or plain text file using the Yices

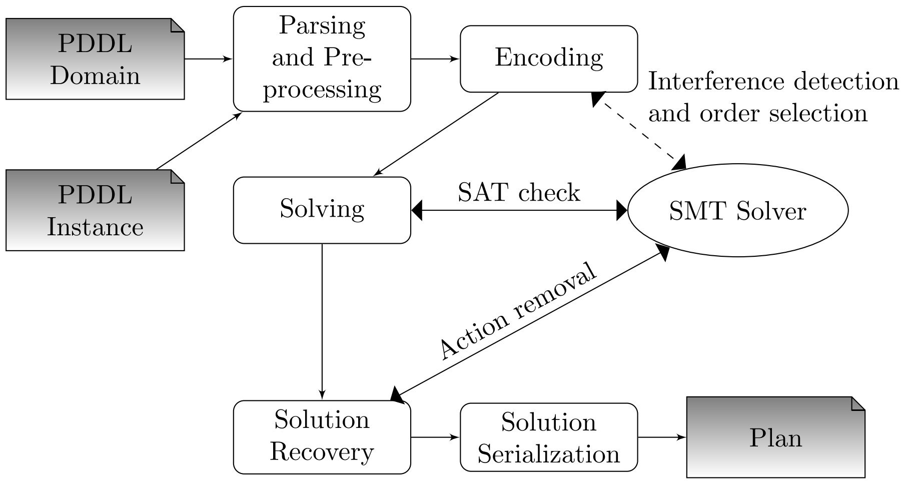

Figure 1: Architecture and solving process of the Rantanplan solver

and Z3 SMT solvers. Once a solution has been found, then it is retrieved. If there are redundant actions in the plan, they are removed as explained in Section 7.2.2, and finally the plan is serialized as explained in Section 5.4.1. In the following section, the relevant aspects for using the Rantanplan solver are explained in more detail.

Usage and Command-Line Options

As the system has been used to test many hypothesis and approaches, it has many command line switches, which enable different functionalities. This section explains in detail what each switch do:

`-v` makes Rantanplan go verbose.

`-d <domain.pddl>` specifies the domain file in PDDL format.

`-p <instance.pddl>` specifies the instance file in PDDL format.

`-s <output.pddl>` specifies the output file, where the planner will write the final serialized plan. The syntax is human-readable and compatible with VAL [HLF04], a plan validator.

`-z <order.txt>` The Relaxed Chained Encoding from Section 7.2, needs an order between actions. This switch lets Rantanplan read a given order from a file.

`-e <y | f | g | z | c>` This switch specifies what SMT solver should be used as a black box to solve the problem. Solvers like Yices are implemented via API and plain text file, as some solvers have a hardcoded maximum file length and some problems can easily exceed those limits. Most solvers also treat differently problems ingested via API or via file.

- `y` The Yices SMT solver, via API. As the namespaces collide, only one version of Yices can be linked to the RANTANPLAN solver at compile time.
- `f` It generates SMTLIB files and calls yices to solve them. It supposes that Yices v1 is in the path.
- `g` The same as before, but generating files with the SMTLIB2 format and expecting Yices v2 in the path.
- `z` Z3 SMT solver via API.
- `c` CVC4 solver via API (deprecated).

`-m <l | f | e>` For encodings that rely on a parallelism semantic, this switch chooses what semantics should be used: sequential (l), ∀-step (f) or ∃-step (e).

`-i <o | s | c>` For encodings that need an interference notion, this switch chooses what notion should be used.

- `o` Syntactic notion of interference, as in Section 5.4.
- `s` Semantic notion of interference, presented in Chapter 6.
- `c` Consider no interferences. It should be used with the Relaxed Chained Encoding presented in Section 7.2.

`-t <f | p | c | u | a>` This switch defines the encoding that should be used.

- `f` The Generalized SMT Encoding from Chapter 5.
- `p` The Relaxed Encoding from Chapter 6.
- `c` The Relaxed Chained Encoding from Chapter 7.
- `u` The UF SMT encoding from Chapter 5, hardcoding the Z3 solver via files.
- `a` The same as above, but via API. The Z3 solver accepts QF_UFLIA or QF_UFLRA logics via file or API, but Yices do not.

`-a <0 | 1>` Activate a strategy where a NO-OP action is added, and each time step is given a maximum time to be solved. The idea is that with a high number of time steps, the problem should be easy to solve. In practice it does not work as expected.

`-o <dfs | lex | random | relaxed | comm | file>` In encodings that need an order, this switch defines what order should be used. All orders can be prepended with an “r” character to reverse it.

- `dfs` The Path-based SCC Order, from Section 6.3.2.
- `lex` Lexicographical order of the grounded action names.
- `random` A random order.
- `relaxed` Generate a total order between actions using a relaxed version of the problem. Write it to the file specified with the `z` switch, according to the method explained in Section 7.4.1.
- `comm` Generate an order using a SMT solver such that it minimizes interferences to actions later in the ordering.
- `file` Use the total order between actions that was read from the file specified in the `z` switch.

`-x <quadratic | lineal | dfscuts>` This switch chooses, given a disabling graph, how mutexes are added to the problem.

- `quadratic` The simple quadratic encoding from Section 5.2.2.
- `lineal` A lineal encoding by Rintanen [RHN06].
- `dfscuts` The mutexes generated by Sort and Cut order from Section 5.3.1.

`-f <or | range | none>` This switch governs how the domains of the numeric variables should be explained to the SMT solver:

- `or` If the domain is finite, a or clause determines its domain. i.e. $x = 3 \vee x = 4 \vee x = 5$.
- `range` The same as before, but expressed as a range: $x > 2 \wedge x < 6$.
- `none` No explicit description of the domain.

`-c` Regardless of the order chosen, if any two actions in the disabling graph form a cycle by themselves, before any preprocessing remove those edges and add a mutex.

`-n <int>` To help the SMT solver refute some hard time steps, add a sorting network [Bat68] to limit how many actions can be used in any time step.

`-u` Inspired by the ideas in Chapter 6 and using the SMT solver, every pair of grounded actions are checked if their preconditions contradict (i.e. its impossible that those two actions can be executed in the same time step). If they contradict, a mutex is added in addition to any other mutex generation steps.

`-l` The Yices variable selection heuristic was modified to be able to give more priority to any given variable. This switch gives priority to variables that express if an action is executed or not. It requires a modified Yices to be linked.

`-k` This switch is needed if the Relaxed Chained Encoding is used. It makes the planner stop after generating an order to the file specified in the `z` switch.

Planes Domain PDDL Model

```
(define (domain planes)
(:requirements :typing :fluents)
(:types  city locatable - object
         aircraft person - locatable)
(:functions
 (at       ?x - locatable) - city
 (in       ?p - person) - aircraft
 (fuel     ?a - aircraft) - number
 (seats    ?a - aircraft) - number
 (capacity ?a - aircraft) - number
 (onboard  ?a - aircraft) - number
 (distance ?c1 - city ?c2 - city) - number)

(:action board
 :parameters (?p - person
             ?a - aircraft
             ?c - city)
 :precondition (and (= (at ?p) ?c)
                    (= (at ?a) ?c)
                    (> (seats ?a) (onboard ?a)))
 :effect (and (assign (at ?p) undefined)
             (assign (in ?p) ?a)
             (increase (onboard ?a) 1)))

(:action debark
 :parameters (?p - person
             ?a - aircraft
             ?c - city)
 :precondition (and (= (in ?p) ?a)
                (= (at ?a) ?c))
 :effect (and (assign (in ?p) undefined)
          (assign (at ?p) ?c)
          (decrease (onboard ?a) 1)))

(:action fly
 :parameters (?a - aircraft ?c1 ?c2 - city)
 :precondition (and (= (at ?a) ?c1)
                (> (onboard ?a) 0)
                (>= (fuel ?a)
                    (distance ?c1 ?c2)))
 :effect (and (assign (at ?a) ?c2)
          (decrease (fuel ?a)
                    (distance ?c1 ?c2)))
 )

(:action refuel
 :parameters (?a - aircraft)
 :precondition (and
                (< (* (fuel ?a) 2) (capacity ?a))
                (= (onboard ?a) 0))
 :effect (and (assign (fuel ?a) (capacity ?a)))))
```

Figure 2: PDDL model of the Planes domain