FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Planning in Graph Databases under Description Logic Constraints

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Master of Science (M.Sc.)

im Rahmen des Erasmus-Mundus-Studiums

Computational Logic

eingereicht von

Shqiponja Ahmetaj
Matrikelnummer 1228388

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: O.Univ.-Prof. Dr.techn. Thomas Eiter
Mitwirkung: Prof.Dr. Diego Calvanese, Univ.-Ass. Dr.techn. Mantas Šimkus

Wien, 16.09.2013

(Unterschrift Verfasserin) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Planning in Graph Databases under Description Logic Constraints

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science (M.Sc.)

in

Computational Logic

by

Shqiponja Ahmetaj
Registration Number 1228388

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: O.Univ.-Prof. Dr.techn. Thomas Eiter
Assistance: Prof.Dr. Diego Calvanese, Univ.-Ass. Dr.techn. Mantas Šimkus

Vienna, 16.09.2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Shqiponja Ahmetaj
Grornergasse 3, 1060 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasserin)

Acknowledgements

I would like to take this chance to express gratitude towards all those people who made this thesis possible.

I would like to thank my supervisor Prof. Diego Calvanese for his discussions, suggestions and ideas. I am very grateful to Prof. Thomas Eiter for his useful advices, support, guidance and for being ready to offer his help when is needed.

I also want to sincerely thank Mantas Simkus, for his feedback and comments. I am grateful for all he has taught me, for the time he has devoted for this thesis, for all his patience, encouragement and guidance. Without his support and trust, this thesis would have not been possible.

My thanks extend to my parents Mereme and Shemun for everything they have given me. To my brother and sisters, for being such an important and wonderful part of my life. My thanks also go to my family and especially to my uncle Isa and uncle Imer, for their support and for believing in me.

Finally, I would like to thank my love Bojken, for all his patience, love, support and for his effort for remaining close despite the distance.

Abstract

Graph databases are gaining increasing importance and they are fundamental for storing, for example, web data in RDF form, but also other forms of semi-structured data. Given the very large amount of data currently available in these stores, efficient management of these data becomes harder and harder. In addition, the development of automated management tools for them is becoming a pressing problem. As in traditional databases, integrity constraints on graph databases are important to capture the semantics of domain of interest. One of the tools to modify this data are transactions. A transaction encapsulates a sequence of modifications to the data which are executed and committed as a unit. Description Logics (DLs) are decidable languages that have been strongly advocated for managing data repositories, for expressing integrity constraints and they are particularly natural for talking about graph databases, as argued in some recent work. A challenging topic is planning in such context. Planning is a classical topic and has been an important problem in Artificial Intelligence for over five decades.

In this thesis, we propose a framework for planning in graph databases. To the best of our knowledge, this is the first attempt to formally define planning in such setting. We use an expressive action language that is already defined by an earlier work. This language is expressed through a DL that has the feature to introduce new values to the data. Graph databases are seen as finite DL interpretations. We identify some interesting reasoning tasks, relevant for the setting we consider. The standard one is plan-existence, which corresponds in trying to find a plan for a given instance of a planning problem. We investigate two variants of the problem. The difference between them is that in one of them we do not allow the transactions to introduce fresh constants. We prove that deciding plan-existence for the variation without fresh constants is PSpace-complete. After applying several syntactic restrictions, we are able to determine NlogSpace and NP-hard cases. We are also able to provide several polynomial algorithms for some other cases. In addition, to get some insights to the relationship between our formalism and STRIPS, which is a classical approach to planning, we investigate whether planning in our setting can be reduced to STRIPS. We show that such encoding is possible under some syntactical restrictions. In this way, STRIPS planners can be exploited to solve our planning problem.

Furthermore, we study the variation, where the transactions are allowed to introduce fresh constants. Intuitively, this makes planning more involved. We are able to single out some cases in this setting that can be reduced to planning without fresh constants. In this way we prove that those cases are in PSpace.

Kurzfassung

Graph-Datenbanken gewinnen zunehmends aufgrund ihrer Verwendung für RDF Daten an Bedeutung. Sie sind beispielsweise fundamental für die Speicherung von Web-Daten in RDF Form, aber auch für andere Arten von semi-strukturierten Daten. Anbetrachts der sehr großen Datenmenge, die in Speichern dieses Formats derzeit vorhanden ist, wird effizientes Management der Daten immer schwieriger. Zusätzlich wird die Entwicklung von automatisierten Management-Tools für die Daten ein dringliches Problem. Wie in traditionellen Datenbanken sind Integritätsbedingungen für Graph-Datenbanken wichtig um die Semantik einer Domäne zu erfassen. Eines der Werkzeuge zur Modifikation der Daten sind Transaktionen. Eine Transaktion kapselt eine Folge von Datenmodifikationen, die als Einheit ausgeführt und abgeschlossen werden. Beschreibungslogiken (engl. Description Logics, DLs) sind entscheidbare Sprachen, deren Verwendung für Datenmanagement nachdrücklich befoürwortet worden ist um Integritätsbedingungen auszudrücken; diese Sprachen eignen sich in natürlicher Weise für Aussagen über Graph-Datenbanken wie vor kurzem in einer Arbeit aufgezeigt wurde. Eine Herausforderung ist Planen in diesem Kontext. Planen ist ein klassisches Gebiet und seit mehr als 50 Jahren ein wichtiges Problem in der künstlichen Intelligenz.

In dieser Arbeit schlagen wir ein Rahmenwerk für Planen in Graph-Datenbanken vor. Nach unserem Wissen ist dies der erste Versuch, Planen in diesem Bereich formal zu definieren. Wir verwenden dazu eine ausdrucksstarke Aktionssprache, die bereits in einer vorliegenden Arbeit definiert worden ist. Sie wird in einer DL formuliert, die neue Werte für Daten einführen kann; Graph-Datenbanken werden als endliche Interpretationen dieser DL gesehen. Wir identifizieren einige Schluss-Probleme, die in diesem Zusammenhang relevant sind. Das Standardproblem ist Plan-Existenz, welches darin besteht, die Existenz eines Plans, d.h. einer Lösung für ein Planungsproblem, zu entscheiden. Wir untersuchen dabei zwei Varianten dieses Problems, wobei der Unterschied darin besteht, dass in der einen Variante Transaktionen keine neuen Datenwerte einführen dürfen. Wir zeigen dass Plan-Existenz für die Variante ohne neue Konstanten PSPACE-vollständig ist. Unter der Verwendung von einigen syntaktischen Einschränkungen können wir Fälle gewinnen, in denen einerseits das Problem in NLOGSPACE bzw. in polynomieller Zeit lösbar ist, wobei wir Algorithmen angeben, auf der anderen Seite aber auch NP-hart ist. Um zusätzlich eine besseres Verständnis des Verhältnisses von unserem Formalismus zum STRIPS Ansatz zu erhalten, der einen klassischer Planungsansatz darstellt, untersuchen wir, ob Planen in unserem Modell auf STRIPS-Planen reduziert werden kann. Wir zeigen dass eine solche Transformation unter bestimmten syntaktischen Einschränkungen möglich ist. Auf diesem Wege können STRIPS-Planner zur Lösung unseres Planungsproblems genutzt werden.

Wir untersuchen weiters die Variante des Problems in der es Transaktionen erlaubt ist, neue Werte einzuführen. Intuitiv gesehen wird macht dies das Planen schwieriger. Es gelingt uns, bei diesem Ausgang einige Fälle zu ermitteln, in denen das Planungsproblem auf den Fall des Planens ohne neue Werte reduziert werden kann. Auf diesem Wege zeigen wir, dass diese Fälle in PSPACE liegen; es bleibt offen, ob dieses Resultat verbessert werden kann. Wir zeigen dabei auch Merkmale auf, die nach unserer Meinung die Transaktionen dazu anhalten, neue Werte einzuführen.

Contents

CHAPTER 1

Introduction

Databases are collections of data that model relevant aspects of reality. Many different database models have been presented and used since they first emerged [45]. One of them, gaining popularity recently are graph databases.

There is an interesting history of development of graph databases, presented in the paper of Angles et al. [1], according to which graph databases took off in the eighties and flourished by the first half of the nineties. But then the attention moved toward semi-structured data which did not have links to the graph database work in the nineties, and the emergence of XML became the main focus of those working on hypertext. Moreover, since the tree-like structure became sufficient for most applications, people working on graph databases moved to particular applications like spatial data, Web, and documents. Thus, according to Angels et al. [1], the influence of graph databases gradually died and this topic almost disappeared. Only recently, the need to manage information with graph-like nature has reestablished the relevance of this area leading to the renaissance of graph data and graph thinking.

In contrast planning has been an important problem in Artificial Intelligence, ever. Several planning approaches have been developed in extensive work over the last decades. Planning can be considered as the process that involves thinking ahead for providing a sequence of actions, ordered in a way to satisfy some pre-defined requirements and to achieve some pre-stated objectives [40]. Intuitively, thinking in advance in planning would require not only action selection, but also action sequencing by anticipating their expected effects. In planning, actions are viewed as means to change and update states of the world. They are arranged and organized with the aim to satisfy some requirements on the evolution of these states. These requirements, in planning terminology, are called goals and the combinations of actions to achieve these goals are called plans. A planning problem is then to find a plan given actions and a goal [21]. An example of a requirement for a robot is that of carrying a container and move it from one location to another. A large variety of approaches to planning have been developed over the last decades. One of them, a prototypical logic-based planning approach is STRIPS-planning [18], which will also be considered in this thesis.

For more information about graph databases and planning, we refer the reader to Chapter 7, which reviews the 'State of the Art',.

1.1 Motivation

Currently, there exists a wide variety of formal representations of collections of data. They are typically organized to model relevant aspects of reality (for example, storing the project database of some research institute), in a way that supports processes requiring this information (for example, finding the employees that work for some specific project). One of several ways to organize these data is through relational databases, first proposed by Codd [10] in 1970. They are represented as collections of tables of data items, formally described and organized according to the relational model. Another form of organizing collections of data is through graph databases, which have gained a particular importance recently. The latter in comparison to relational databases has some differences and in some aspects also advantages. For example, as stated by Angles et al. [1], the relational model is geared towards simple record-type data, where the data structure is known in advance (airline reservations, accounting, inventories, etc.). Furthermore, it is difficult to extend these databases, since the schema is fixed and it is not easy to integrate different schemas. The query language cannot explore the underlying graph of relationships among the data, such as paths, neighborhoods, patterns [1].

Even though graph databases re-emerged only in the last few years, graph databases have helped to solve important problems in the areas of social networking, master data management, and more. They are gaining increasing importance and are fundamental for storing, for example, web data in the form of RDF [5], and other forms of semi-structured data.

In addition, given the very large amount and complexity of data currently available in the stores of this format, managing these data becomes harder and harder. Also, graph databases may evolve as a result of operations carried out by users or applications. Therefore, the development of automated management tools for them is becoming a pressing problem. A basic classical management tool are transactions. A transaction encapsulates a sequence of modifications to the data, which are executed and committed as a unit. The acronym ACID [25] describes some ideal properties of a database transaction: Atomicity, Consistency, Isolation, and Durability.

There is a recent work by Calvanese et al. [8], on transactions in graph databases using Description Logics (DLs). The transactions are expressed through Description Logics (DLs) [2]. They have the property of being decidable languages. Furthermore, they have been strongly advocated for managing data repositories [30], and they are particularly natural for talking about graph databases [8]. In turn, graph databases can be naturally seen as finite DL interpretations. In [8], Calvanese et al. prove that verifying whether the constraints are still satisfied in the database state resulting from the execution of a given action, for every possible initial state satisfying them, is decidable. This is essentially the *consistency* property of database transactions. A transaction is consistent if its execution over a legal database state never leads to an illegal one.

Inspired by the positive decidability results for transaction verification, in this thesis we explore planning in graph databases with sequences of transactions. Planning, which is considered a key ability for intelligent agents, has been an important problem in Artificial Intelligence for over five decades. A large variety of techniques has been developed during this period. Planning

consists in determining future sequences of actions for achieving certain goals and automated planning studies this process computationally. In addition, planning involves the representation of actions and world models, reasoning about the effects of actions, and evolving strategies for efficiently searching the space of possible sequences that lead to one of the goal states.

This thesis is, to our best knowledge, the first attempt to formally define and investigate the planning problem in graph databases. Not only, we will propose a framework for planning in graph databases, but we also will outline several reasoning tasks for this setting and provide a deep analysis of one of them.

In the following section, we will give a brief description of our contributions in this thesis.

1.2 Contributions

Our main contributions in this thesis are the following.

1. **Framework for Planning in Graph Databases.** We propose a framework for planning in graph databases as follows.

   - Graph databases will be seen as finite DL interpretations. When convenient, for representation reasons, we view an interpretation as a finite set I of atoms. An example of a (part of) project database of some research institute is:

     $$\begin{aligned} I = \{ & ActiveProject(P20840), ActiveProject(P24090), \\ & Project(P20840), Project(P24090), \\ & Employee(E01), Employee(E03), Employee(E04), Employee(E07), \\ & ProjectEmployee(E01), ProjectEmployee(E03), ProjectEmployee(E07), \\ & PermanentEmployee(E04), \\ & worksFor(E01, P20840), worksFor(E03, P20840), \\ & worksFor(E07, P24090)\}. \end{aligned}$$

   - To express the changes to the databases, we consider an action language suitable for our setting, adapted to compute insertions and deletions on these interpretations. It is expressed through a DL called $\mathcal{ALCHOIQ}br$. The action language consists of atomic actions and transactions. Transactions are sequences of atomic actions computed as a unit. We also define transactions with preconditions, expressed through $\mathcal{ALCHOIQ}br$ formulas. This offers high expressibility. In particular, it has the feature of allowing to introduce fresh constants through variables. We present an example for each type of actions allowed in our language.

     The following is an atomic action, which intuitively deletes from *ProjectEmployee* all the employees that occur to a *workfor* link with some object in *Project*.

     $$\rho_1^{tr} = ProjectEmployee \ominus \exists worksFor.Project$$

The following transaction is a sequence of three atomic actions. x is a variable that can be instantiated by any constant.

$$\begin{aligned}\rho_2^{tr} = \; & ActiveProject \ominus \{x\} \circ \\ & ConcludedProject \oplus \{x\} \circ \\ & ProjectEmployee \ominus \exists worksFor.\{x\}\end{aligned}$$

Intuitively, the grounded ρ_2^{tr}, removes some project from *ActiveProject*, adds it to *ConcludedProject* and removes from *ProjectEmployee* all employees that occur in a *workFor* link with this project.

The following transaction ρ_3^{tr} with variables x, y, z transfers the employee x from project y to project z:

$$\begin{aligned}\rho_3^{tr} = \; & (x : Employee \wedge y : Project \wedge z : Project \wedge (x, y) : worksFor) \; ? \\ & worksFor \ominus \{(x, y)\} \circ worksFor \oplus \{(x, z)\} : \epsilon\end{aligned}$$

It first checks whether x is an employee, y and z are projects, and x works for y. If yes, it removes the *worksFor* link between x and y and creates a *worksFor* link between x and z. If any of the checks fails, it does nothing, which is described by ϵ.

- We formally define the updates on databases through a mapping $M_{\rho^{tr}}$ from interpretations to interpretations. It captures the result of application of a finite sequence of transactions to an interpretation.
  For instance, the interpretation $I' = M_{\rho_1^{tr}}(I)$ that reflects the status of the database after transaction ρ_2^{tr}, where x is instantiated with $P20840$ looks as follows.

$$\begin{aligned}I' = M_{\rho_1^{tr}}(I) = \{ & ActiveProject(P24090), \\ & Project(P20840), Project(P24090), \\ & ConcludedProject(P20840), \\ & Employee(E01), Employee(E03), Employee(E04), \\ & Employee(E07), \\ & ProjectEmployee(E07), \\ & PermanentEmployee(E04), \\ & worksFor(E01, P20840), worksFor(E03, P20840), \\ & worksFor(E07, P24090)\},\end{aligned}$$

  Intuitively, ρ_2^{tr} applied to I, removes *ActiveProject*$(P20840)$, *ProjectEmployee*$(E01)$, *ProjectEmployee*$(E03)$, and adds *ConcludedProject*$(P20840)$. The resulting interpretation is I'.
- Finally, we formally define what is a planning problem for the setting of graph databases. An instance of a graph database planning problem is (I, Act, G), where I is the initial state (a graph database), Act is a set of predefined transactions and G is a goal formula. Intuitively, a plan for a planning problem is a sequence of actions

from the set of predefined transactions such that after applying to the initial state, a state that satisfies the goal is produced. States are finite DL interpretations represented as sets of atoms.

E.g. assume I is given as above, Act is $\langle \rho_1^{tr}, \rho_2^{tr}, \rho_3^{tr} \rangle$, where each of them are given as above and G is $ConcludedProject(P20840)$. This planning instance requires that the initial graph database is updated such that the new one contains the atom $ConcludedProject(P20840)$. Intuitively, a plan for this instance is $\langle \rho_2 \rangle$, where x is instantiated with $P20840$. It is obvious that the state produced by the plan is $I' = M_{\rho_1^{tr}}(I)$, and I' satisfies $ConcludedProject(P20840)$. In addition, notice that for $P20840$ to be a *ConcludedProject*, the ρ_2^{tr} takes care that $P20840$ is no longer an *ActiveProject* and no *ProjectEmployee* is working on this project any more.

2.
- We identify interesting reasoning problems that are suitable for our setting.
- We study one of them in depth, namely the plan-existence problem. Plan-existence is the decision problem of checking whether there exists a plan for a given instance. We formally state and analyze two variations of this problem for graph databases.
  - **Deciding Fixed Domain Plan-Existence.** We will study in depth the plan-existence decision problem for planning when only constants occurring the initial instances are allowed. We prove that it is decidable and it is **PSpace**-complete.
  - **Deciding General Plan-Existence.** We will analyze the variation where fresh constants are considered. We notice that the complexity increases. But is general plan-existence decidable?

To get a better intuition on the difference between a *fixed domain planning problem* and a *general planning problem*, we will illustrate with a simple example from the project database of some research institute.

Example 1.1. *Assume a graph database planning problem* (I, Act, G) *is given as follows.*

$$\begin{aligned} I = \{&Project(P20840),\\ &Employee(E01), Employee(E03),\\ &ProjectEmployee(E01),\\ &worksFor(E01, P20840)\}, \end{aligned}$$

$$Act = \{\rho_1^{tr}, \rho_2^{tr}, \rho_3^{tr}\},$$

where:
$\rho_1^{tr} = (x : Project \wedge \neg(x : Employee))\ ?\ worksFor \oplus \{(E03, x)\} : \epsilon$,
$\rho_2^{tr} = (x : Project \wedge \neg(x : Employee))\ ?\ ProjectEmployee \oplus \exists worksFor.\{x\} : \epsilon$,

$\rho_3^{tr} = Project \oplus \{x\}$,

$$G = ProjectEmployee(E03) \wedge \neg worksFor(E03, P20840).$$

This example of a graph database planning problem asks for computing a sequence of actions that reaches a state that contains ProjectEmployee$(E03)$. Meanwhile, it requires that no atom worksFor$(E03, P20840)$ is added. In addition, the set of predefined actions contains just two transactions with preconditions and a third atomic action. The precondition in both of them is that the instantiation of variable x should be a project and not an employee. Clearly, I satisfies $\neg worksFor(E03, P20840)$. To achieve a state that satisfies $ProjectEmployee(E03)$, in the plan there should be ρ_2^{tr}, since it is responsible for modifications to $ProjectEmployee$. By the precondition, x cannot be instantiated in ρ_2^{tr} neither with $E01$ and nor with $E03$. In addition, by instantiating x with $P20840$ in ρ_2^{tr}, then $worksFor(E03, P20840)$ would be added by ρ_1^{tr}. It follows that there is no plan for this instance considering the fixed domain planning problem.

But, by instantiating x with any fresh constant, there exists a plan. Indeed, a plan might be as follows.

$$\mathcal{P} = \langle (Project \oplus \{P11111\}),$$

$$(P11111 : Project \wedge \neg(P11111 : Employee)) \; ? \; worksFor \oplus \{(E03, P11111)\} : \epsilon,$$

$$(P11111 : Project \wedge \neg(P11111 : Employee)) \; ?$$

$$ProjectEmployee \oplus \exists worksFor.\{P11111\} : \epsilon \rangle.$$

A new project $P11111$ is initially added by the atomic action ρ_3^{tr}. Clearly, x instantiated with $P11111$ satisfies the preconditions in ρ_1^{tr} and ρ_2^{tr}. Then by applying ρ_1^{tr} and ρ_2^{tr}, a desired plan for the general planning problem, is obtained. Clearly, there are infinite possibilities of instantiations, hence infinite number of plans.

Intuitively, allowing new constants makes planning more difficult as this creates infinite search space.

3. **Complexity Results for Fixed Domain Plan-Existence.** We perform a thorough study of the complexity of plan-existence over a fixed domain for the setting of graph databases. The only constants allowed to be used by the actions, are the ones that occur in the initial instance.

   - By a reduction from a deterministic Turing machine, we are able to show that this problem is **PSpace**-hard for the general setting.
   - By giving a procedure that runs in non-deterministic polynomial space, we show that plan-existence is also in **PSpace**, and thus we obtain **PSpace**-completeness.
   - Next, we impose several syntactic restrictions on the type of transactions and goals allowed as input. We are able to find by a reduction from 3-colorability that a relatively basic case is already **NP**-hard. The case is defined as follows.

– The goal formula is a conjunction of ground positive and negative atoms expressed through atomic concepts, e.g.

$$ActiveProject(P20840) \wedge \neg Employee(E01) \wedge \neg ProjectEmployee(E03)$$

– The set of predefined actions is limited to atomic actions with atomic concepts only, e.g. $(ActiveProject \oplus Project)$.

In addition, we are able to encode this case to propositional STRIPS. When building the reduction, we noticed that more then one atom in the pre-condition and negative atoms in post-conditions are required.

- Further, we are interested in providing fine lines between tractable and intractable cases. We consider several cases with different type of goal formulas. Here, we will give an appetizer of some results.
  – If goal formula is a conjunction of only positive atoms e.g.

$$Project(P20840) \wedge \exists worksFor(E01, P20940) \wedge worksFor^{-}(E01, P20840),$$

and actions are limited to concepts, roles, variables, existential quantification and inversion, by reduction to reachability in graphs, we show that these cases belong in **NlogSpace**. We call this: *the positive case*.

  – If a goal requires the removal of several atoms over the same atomic concept, e.g.

$$\neg Project(P20840) \wedge \neg Project(P20040) \wedge \neg Project(P29840),$$

and actions are limited to atomic concepts only, we show by a reduction to the positive case, that the complexity remains in **NlogSpace**.

  – If goals are conjunctions of 2 atoms e.g. $\neg ActiveProject(E01) \wedge Project(E01)$, or $\neg ActiveProject(E01) \wedge \neg Project(E01)$, and actions are as above, we provide several algorithms that check plan-existence for these cases in **Ptime**. We illustrate with examples the intuitions for each of them.

  – Moreover, we investigate some relatively hard cases that seem to cause the **NP**-hardness.

4. We investigate whether planning in our setting can be reduced to STRIPS planning. Considering such possible encoding one can get some insights about the relationships between our formalism and the STRIPS formalism, which represents a classical approach to planning. In addition, in this way STRIPS planners can be exploited to solve our planning problem. Since planning STRIPS is **PSpace**-complete, then theoretically there exists a polynomial time reduction of fixed domain planning problem to STRIPS. In turn, when trying to encode to STRIPS, we encountered several obstacles. Since STRIPS operators are limited to only conjunctions of literals, it seems hard to express DL complex concepts and roles to STRIPS. In addition, also disjunction seems not well supported by the

STRIPS language. Furthermore, since STRIPS operates on a fixed domain, it seems difficult to encode the general planning problem.

Actually, we are able to encode to STRIPS the **NP**-hard case, which we described above. When translating the actions, we notice that negative atoms in post-conditions of operators in STRIPS seem necessary. Thus, we observe that our translation maps to a **PSpace**-complete STRIPS fragment. For the other low-complexity cases, we observe that a naive translation to STRIPS leads to planning instances with negative atoms in post-conditions. Such instances are **PSpace**-hard in STRIPS. For this reason, it seems difficult to infer optimal upper-bound complexity results from such an encoding.

5. **Complexity Results for General Plan-Existence.** We study also the general planning problem for the setting of graph databases, where variables hav e infinite possibilities of instantiations.

   - We aim at finding expressive settings, whose plans don't require fresh constants. Toward this aim, we are able to find a reduction to fixed domain plan-existence for some cases, showing that they belong in **PSpace**. For instance assume the actions are atomic and allow complex concepts and roles (except for universal restriction, qualified number restrictions and negation) like the action:

     $$ActiveProject \oplus \exists worksFor.(ConcludedProject \sqcup \exists worksFor^{-}.\{x\})$$

     and the goal formula is a conjunction of positive atoms e.g.

     $$ActiveProject(E01) \wedge Project(E01) \wedge Employee(E01).$$

     Then for every plan, there is a plan that reaches the goal using no fresh constants.
   - We provide some examples of triggers that enforce the use of fresh constants by the transactions. We illustrate with examples the intuition for each of them.

1.3 Structure of the Thesis

The rest of the thesis is structured as follows. We start with the formal definitions of the syntax and semantics of the expressive description logic $\mathcal{ALCHOIQ}$ in Chapter 2. Then, we give some general background on planning defining the notion of a plan and a trajectory, followed by some main reasoning tasks. In addition, in Chapter 2, we also provide an introduction to propositional STRIPS planning and recall some definitions for deterministic Turing machines. Then, in Chapter 3, we define an extension of $\mathcal{ALCHOIQ}$ called $\mathcal{ALCHOIQ}br$, which is a DL adopted to express updates in graph databases. Graph databases are seen as finite DL interpretations. We continue by introducing a transaction language followed by definitions describing the way updates of these interpretations occur. We conclude Chapter 3 by an example that illustrates these updates. In Chapter 4, we define the planning language we adopt for our setting of planning in graph databases. We instantiate the planning problem defined abstractly in Chapter 2 for a graph database planning problem and we specify two variations of this problem, namely the

general planning problem and the fixed domain planning problem. We illustrate each of them with an example and we outline some of the main reasoning tasks for them. One of these reasoning tasks is the decision problem that we will be analyzing during the rest of the thesis, called plan-existence. Finally, with the framework ready, we are prepared to embark to the complexity of fixed domain plan-existence in Chapter 6. We start by thoroughly studying the upper-bound and lower-bound complexity of this problem, followed by investigating several subcases along with their complexity. At the end of this chapter, for some cases, we also provide a reduction to STRIPS-planning. Chapter 6 continues in the same line but for the general plan-existence decision problem. We start this chapter by giving some intuition about the behavior of this variation along with several examples showing the main triggers that enforce the necessity for a fresh constant in a plan. Further, we impose some restrictions and study some particular cases. In Chapter 7, we present the state of the art by focusing on planning, graph databases and related works. Lastly, in Chapter 8, we draw our conclusions and suggest some directions for further research.

CHAPTER 2

Preliminaries

In this chapter, we recall the formal definitions of the syntax and semantics of the expressive description logic $\mathcal{ALCHOIQ}$, which is the basic description logic $\mathcal{ALC}$ extended with role hierarchies ($\mathcal{H}$), nominals ($\mathcal{O}$), inverse properties ($\mathcal{I}$) and qualified number restrictions ($\mathcal{Q}$). $\mathcal{ALCHOIQ}$ will serve as basis for defining, in Chapter 3, a new description logic, $\mathcal{ALCHOIQ}br$, extended with some properties, suitable for manipulation of graph databases. Further, we give some general background on planning, in particular we define planning and the notion of a trajectory in an abstract level. We instantiate it for a graph database planning problem in Chapter 4. Next, some main reasoning tasks related to planning are introduced an briefly discussed, since they represent very interesting decision problems for our setting. We will focus our analysis and study only on the Plan-Existence decision problem in Chapter 6. Further, we provide an introduction to propositional STRIPS planning, as the prototypical logic-based approach to planning, which will be used in Section 5.4. We recall its basic definitions, syntax, semantics followed by a brief overview of its complexity results and main extensions. In the last section, we define Turing machines, which will serve as basis for showing a PSPACE-hardness in Section 5.2.

2.1 The Description Logic $\mathcal{ALCHOIQ}$

In this section, we recall some basic definitions about one of the most expressive description logics, $\mathcal{ALCHOIQ}$. Description logics (DLs) are a family of knowledge representation formalisms, often used to reason about various application domains in a structured and formally well-understood way. The main ingredients of a DL are concepts, roles and individuals, which intuitively represent classes of objects that share some properties, relations between objects and specific domain objects respectively. Description logic knowledge bases usually consist of two components: a *TBox* and an *ABox*. The *TBox* contains intensional knowledge in a terminological form. It expresses general relations between the different concepts and roles in the knowledge base through the subsumption (and/or equivalence) relation. The *ABox* contains extensional

knowledge in the form of assertions (also called Assertional knowledge), which describes concrete individual objects of the domain of discourse, and the relation between these objects and the concepts and roles in the knowledge base.

The description logic $\mathcal{ALC}$ [2] is a prototypical logic which is at the basis of most expressive description logics. In fact, the expressive logic $\mathcal{ALCHOIQ}$, which is the one we consider in this thesis, is the extension of $\mathcal{ALC}$ with role hierarchy ($\mathcal{H}$), nominals ($\mathcal{O}$), inverse properties ($\mathcal{I}$) and qualified number restrictions ($\mathcal{Q}$). What makes description logics particularly useful for these purposes (especially $\mathcal{ALCHOIQ}$) is that they are very expressive decidable languages and particularly natural for talking about graph databases. We give a formal definition of the syntax and semantics of $\mathcal{ALCHOIQ}$.

2.1.1 Syntax

Definition 2.1. *($\mathcal{ALCHOIQ}$-syntax). Let N_C, N_R, N_I be pairwise-disjoint countably infinite sets of atomic concepts (concept names), atomic roles (role names) and individual names respectively.*

Roles $\Rightarrow$ Roles *are defined as follows:*

- *if $r \in N_R$, then r and r^- (the* inverse *of r) are $\mathcal{ALCHOIQ}$ roles.*

Concepts $\Rightarrow$ *The set of $\mathcal{ALCHOIQ}$* concepts *is the smallest set such that:*

- *every concept name $C \in N_C$ is an $\mathcal{ALCHOIQ}$ concept,*
- *$\top$, $\bot$ are $\mathcal{ALCHOIQ}$ concepts,*
- *if $o \in N_I$ then $\{o\}$ is an $\mathcal{ALCHOIQ}$ concept (called nominal),*
- *if C and D are $\mathcal{ALCHOIQ}$ concepts and r is a $\mathcal{ALCHOIQ}$ role, then $C \sqcap D$, $C \sqcup D$, $\neg C$, $\forall r.C$, and $\exists r.C$ are also $\mathcal{ALCHOIQ}$ concepts (the last two are called universal and existential restrictions, respectively), and*
- *if C is a $\mathcal{ALCHOIQ}$ concept, R an $\mathcal{ALCHOIQ}$ role and $n \in \mathbb{N}$, then ($\leq nr.C$) and ($\geq nr.C$) are also $\mathcal{ALCHOIQ}$ concepts (called at-most and at-leat number restrictions).*

The concepts $\bot$, ($C \sqcup D$), ($\forall r.C$), and ($\leq nr.C$) can be seen as abbreviations of $\neg\top$, $\neg(\neg C \sqcap \neg D)$, $\neg(\exists r.\neg C)$ and $\neg(\geq (n+1)r.C)$.

Inclusion Axioms $\Rightarrow$ *Let C and D be $\mathcal{ALCHOIQ}$ concepts, $A \in N_C$ and let r_1 and r_2 be $\mathcal{ALCHOIQ}$ roles.*

- *A* concept inclusion axiom *has form $C \sqsubseteq D$. A* concept equivalence *has form $C \equiv D$ which is an abbreviation for $C \sqsubseteq D$ and $D \sqsubseteq C$. A* concept definition *is a* concept equivalence *$A \equiv C$.*
- *A* role inclusion axiom *has form $r_1 \sqsubseteq r_2$.*

Assertions ⇒ *Let C be an $\mathcal{ALCHOIQ}$ concept, r an $\mathcal{ALCHOIQ}$ role, and $\{o, o_1, o_2\} \subseteq N_I$ be individual names. An assertion has form $o : C$ or $(o_1, o_2) : r$.*

TBoxes ⇒ *A* TBox axiom *is either a* concept inclusion axiom *or a* role inclusion axiom*. A* TBox *$\mathcal{T}$ is a finite set of* TBox *axioms.*

ABoxes ⇒ *An* ABox *is a finite set of assertions. An ABox is* normalized *if it contains only assertions of the form $o : A$ or $(o_1, o_2) : r$ where A and r are atomic concepts and atomic roles respectively.*

Axioms ⇒ *An* Axiom *is either an inclusion axiom or an assertion.*

Knowledge Bases ⇒ *A* knowledge base *$\mathcal{K}$ consists of a* TBox *and an* ABox.

2.1.2 Semantics

The semantics of $\mathcal{ALCHOIQ}$ is based on First-Order Logic interpretations. Intuitively, an interpretation is a function that assigns to every concept a set of objects from a given interpretation domain, to every role a binary relation over the domain, and to every individual a domain element. Moreover, for the rest of the thesis, we will adopt the *standard name assumption*. Hence, we may assume that $\triangle^{\mathcal{I}}$ contains the set of individuals, and that for each interpretation $\mathcal{I}$, we have that $o^{\mathcal{I}} = o$, where $o^{\mathcal{I}} \in \triangle^{\mathcal{I}}$.

Below, we define it formally.

Definition 2.2. *($\mathcal{ALCHOIQ}$ Semantics) An* interpretation *$\mathcal{I}$ is a pair $\mathcal{I} = (\triangle^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\triangle^{\mathcal{I}} \neq \emptyset$ is the* domain *and $\cdot^{\mathcal{I}}$ is the* interpretation function *that assigns:*

- *to every concept name $A \in N_C$ a set $A^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}}$,*
- *to every role name $r \in N_R$ a set of pairs $r^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$ and*
- *to every individual name $o \in N_I$ an element $o^{\mathcal{I}} = o \in \triangle^{\mathcal{I}}$,*

The function $\cdot^{\mathcal{I}}$ is extended to roles and concepts as follows:

- $(r^-)^{\mathcal{I}} = \{\langle o, o' \rangle \mid \langle o', o \rangle \in r^{\mathcal{I}}\}$,
- $\top^{\mathcal{I}} = \triangle^{\mathcal{I}}$, $\bot^{\mathcal{I}} = \emptyset$
- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$,
- $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$,
- $(\neg C)^{\mathcal{I}} = \triangle^{\mathcal{I}} \setminus C^{\mathcal{I}}$,
- $(\exists r.C)^{\mathcal{I}} = \{o \in \triangle^{\mathcal{I}} | \exists o'.(o, o') \in r^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\}$,
- $(\forall r.C)^{\mathcal{I}} = \{o \in \triangle^{\mathcal{I}} | \forall o'.(o, o') \in r^{\mathcal{I}} \rightarrow o' \in C^{\mathcal{I}}\}$,

- $(\geq nr.C)^{\mathcal{I}} = \{o \in \triangle^{\mathcal{I}} | \sharp\{o' | (o, o') \in r^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\} \geq n\}$,
- $(\leq nr.C)^{\mathcal{I}} = \{o \in \triangle^{\mathcal{I}} | \sharp\{o' | (o, o') \in r^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\} \leq n\}$ *and*
- *every nominal* $\{o\} \in N_C$ *is mapped to the singleton set* $\{o^{\mathcal{I}}\}$*, hence* $\{o\}^{\mathcal{I}} = \{o^{\mathcal{I}}\}$.

We have denoted the cardinality of a set M *by* $\sharp M$*. Also notice that* $\exists r.C$ *is semantically equivalent to* $(\geq 1r.C)$*.*

Satisfaction of axioms in an interpretation is defined as follows:

- $\mathcal{I} \models C \sqsubseteq D$ *iff* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$,
- $\mathcal{I} \models r_1 \sqsubseteq r_2$ *iff* $r_1^{\mathcal{I}} \subseteq r_2^{\mathcal{I}}$,
- $\mathcal{I} \models o : C$ *iff* $o^{\mathcal{I}} \in C^{\mathcal{I}}$,
- $\mathcal{I} \models (o_1, o_2) : r$ *iff* $(o_1^{\mathcal{I}}, o_2^{\mathcal{I}}) \in r^{\mathcal{I}}$.

Moreover, an interpretation $\mathcal{I}$ *satisfies an* ABox *if and only if* $\mathcal{I}$ *satisfies every axiom of the* ABox*.* $\mathcal{I}$ *satisfies a* TBox *if and only if* $\mathcal{I}$ *satisfies every axiom of the* TBox *and* $\mathcal{I}$ *satisfies a knowledge base* $\mathcal{K}$*, if and only if* $\mathcal{I}$ *satisfies it's* ABox *and* TBox.

Above, we described the main features of this description logic that we will be using in the thesis. We do not specify some of the main reasoning problems regarding different terminological knowledge that may be present (e.g., acyclic TBox, general TBox). Therefore, for more details about the expressive description logic $\mathcal{ALCHOIQ}$ see [2]. In most papers and books about DLs, one would notice the description logic $\mathcal{SHOIQ}$, which is $\mathcal{ALCHOIQ}$ extended with a constructor allowing transitivity of roles (cf. [26]).

Furthermore, for convenience, we will view an interpretation $\mathcal{I}$ as a set of atoms I as follows.

1. $o \in A^{\mathcal{I}}$ iff $A(o) \in I$, for each $o \in \triangle^{\mathcal{I}}$ and $A \in N_C$,
2. $(o_1, o_2) \in r^{\mathcal{I}}$ iff $r(o_1, o_2) \in I$, for each $(o_1, o_2) \in \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$ and $r \in N_R$.

We give a simple example for illustrating the idea.

Example 2.1. *Assume the following interpretation* $\mathcal{I}$*:*

$$\begin{aligned} ActiveProject^{\mathcal{I}} &= \{P20840\}, \\ Project^{\mathcal{I}} &= \{P20840, P24090\}, \\ ConcludedProject^{\mathcal{I}} &= \{\}, \\ Employee^{\mathcal{I}} &= \{E01, E03\}, \\ ProjectEmployee^{\mathcal{I}} &= \{E01\}, \\ PermanentEmployee^{\mathcal{I}} &= \{E04\}, \\ worksFor^{\mathcal{I}} &= \{(E01, P20840)\}. \end{aligned}$$

Analogously, we may view this interpretation as follows.

$$
\begin{aligned}
I = \{ & \\
& ActiveProject(P20840), \\
& Project(P20840), Project(P24090), \\
& Employee(E01), Employee(E03), \\
& ProjectEmployee(E01), \\
& PermanentEmployee(E04), \\
& worksFor(E01, P20840)\}.
\end{aligned}
$$

Note: When viewing an interpretation as a set of atoms, we assume that all atoms of form $\{c\}(c)$ and $\{c_1, c_2\}(c_1, c_2)$ are included, even though we do not specifically write them.

2.2 Automated Planning

In this section, we recall basic definitions of planning at an abstract level, which will be then used and instantiated in the next chapters for our purposes. Further, we discuss some of the main reasoning problems, involving some decision problems related to planing, which might be interesting also for the framework that we will introduce later in the thesis.

The material here is based on the existing literature; in particular, we follow the excellent book (cf. [21]).

2.2.1 Abstract Planning

Definition 2.3. *(***Planning Problem***) A* Planning Problem *is a 5-tuple* $AP = (S, i, G, Act, T)$*, where:*

1. S *is a* set of states*,*
2. $i \in S$ *represents the* initial state*,*
3. $G \subseteq S$ *represents a set of* goal states*,*
4. Act *is a* set of actions*,*
5. $T \subseteq S \times Act \times S$ *is a* transition relation*. Each triple* $t \in T$ *is called a* state transition*.*

As it is seen, we do not give specifications on the type of initial states, goals or actions allowed in the planning problem. The use of the planning problem, presented in such an abstract form, will become more clear later in the next chapters when instantiated for the setting that we consider.

Further, we specify the notion of a *trajectory* for the Abstract Planning Problem.

Definition 2.4. *(***Trajectory***) Assume a planning problem* $AP = (S, i, G, Act, T)$*. A* trajectory $\mathcal{T}$ *for* AP *is a sequence of state transitions*

$$\langle\langle s_0, act_1, s_1\rangle, \langle s_1, act_2, s_2\rangle, ...\langle s_{n-1}, act_n, s_n\rangle\rangle,$$

where $\{s_0, ..., s_n\} \subseteq S$*,* $\{act_1, ..., act_n\} \subseteq Act$ *and* $t_i = \langle s_{i-1}, act_i, s_i\rangle \in T$*,* $\forall i\ 1 \leq i \leq n$*.*

The above definition states that a *trajectory* is a sequence of state transitions, where actions play a crucial role in changing a state and producing a new state as effect. The new state serves as basis of further modification in the next state transition. This proceeds until some effect state is also the ending point of the trajectory (no new state transition is available).

Now, after we prepared the right environment, we can introduce the definition of a plan for an *AP* as suitable sequences of actions which lead from an initial state to some success state which satisfies a given goal.

Definition 2.5. *(***Plan***) Given a planning problem* $AP = (S, i, G, Act, T)$*, a* plan $\mathcal{P}$ *for* AP *is a finite sequence of actions* $\langle act_1, ..., act_n\rangle$*,*$\{act_1, ..., act_n\} \subseteq Act$*,* $0 \leq n$ *s.t. there exists a trajectory:*

$$\mathcal{T}_\mathcal{P} = \langle\langle s_0, act_1, s_1\rangle, \langle s_1, act_2, s_2\rangle, ...\langle s_{n-1}, act_n, s_n\rangle\rangle$$

for AP *where:*

- $\{s_0, ..., s_n\} \subseteq S$*,*
- $s_0 = i$ *(*s_0 *is the initial state),*
- $s_n \in G$ *(*s_n *is a goal state).*

In this definition, it can be noticed that in case for a given AP, $i \in G$, then the plan would be of length 0. In this case, it basically means that the initial state already reaches the goal before the application of any action, thus it is already a goal state. Hence, there would be an empty trajectory $\mathcal{T} = \langle\rangle$, and the plan would be of length 0.

Furthermore, it can be easily seen that we do not give any specifications regarding the first state in the trajectory that is a goal state. The main reason, is that in general for this thesis, our intention is determining the existence of a plan for a given instance along with the complexity of this problem, and the precise optimized length of a plan is not of interest for us.

W.l.o.g., for proof purposes, we will often assume that a plan stops as soon as, it reaches a goal state.

2.2.2 Reasoning Problems

Depending on the types of properties various planning problems have in various dimensions, a large variety of classes of problems can be identified. Changing the quantity and appearance of the parameters allowed as input, several reasoning problems may be distinguished.

In this section, we point out some of the main ones. Not all of them are in the focus of this thesis, but we state them as they represent interesting problems which might be basis for further research for the setting of graph databases.

2.2.2.1 Plan-Existence

Defining strategies and algorithms for achieving a certain goal in a given instance, is surely, the main focus of planners. In contrast, often it might happen that a plan does not exist and the planner might be subject to exhaustive search, with intention of generating an executable plan. Therefore, to prevent time and space exhaustion, several current analysis have focussed on the plan existence problem, which consists in determining if there exists a plan for a given instance. Next, we give a definition of this planing problem and we call it *Plan-Existence*.

Definition 2.6. ***(Plan-Existence)***

Instance*: A planning problem AP.*
Question*: Does there exist a plan $\mathcal{P}$ for AP?*

This is the planning problem that we will study most. Unfortunately, it is well known that deciding plan existence is intractable [9]. However, putting severe restrictions in the type of actions and goals allowed as input, one might find clear dividing lines between tractable and intractable cases.

2.2.2.2 Bounded Plan-Existence

In practice, generally planners are not interested in plans of arbitrary length, because in many cases it would require large amounts of computations, since the search would proceed until all alternatives have been exhausted (which often may be infinite). Instead, putting a bound on the length of the plan, would restrict the search space to only the possible paths not exceeding this bound.

Therefore, a very interesting and well known planning problem is deciding if there exists a plan for a given instance, whose length is bounded by an integer n. We call this planning problem *Bounded-Plan-Existence*, which we define as follows:

Definition 2.7. ***(Bounded-Plan-Existence)***

Instance*: (AP, n), where AP is a planning problem and n is a non-negative integer.*
Question*:Does AP have a plan $\mathcal{P}$ of length at most n?*

By this definition, the *Bounded-Plan-Existence* problem consists in deciding if there exists a plan of a bounded length for a given AP.

2.2.2.3 Conformant Planning

Sometimes one would need an initial state to be given as input for an the planning problem and sometimes one might want to have a plan that is independent of the initial state. Therefore, we consider the decision problem defined as follows:

Definition 2.8. ***(Conformant Planning)***

Instance*: (S, G, Act, T), where Act is a set of actions, G is a set of goal states, S is a set of states and T is a transition relation.*
Question*: Does (S, s, G, Act, T) have a plan for every $s \in S$?*

By the above definition, one would search for a universal plan which reaches the goal independently of the initial database.

Next we distinguish between two more decision problems of planning that have the same given instance, which contains a sequence of actions and a goal. It would be interesting to verify if there exists an initial state such that the given sequence of actions is a plan for the instance. The other decision problem is to verify if this sequence of actions is such that it is a plan for every possible initial states. We call them '**SynVerif**$\exists$' and '**SynVerif**$\forall$', respectively.

Below, we give a more detailed description of those two decision problems.

2.2.2.4 SynVerif$\exists$

Definition 2.9. ***(SynVerif$\exists$)***

***Instance**: $(S, \langle act_1, ..., act_n \rangle, G, T)$, where $\langle act_1, ..., act_n \rangle$ is a sequence of actions, G is a set of goal states, S is a set of states and T is a transition relation.*
***Question**: Does there exist an initial state $i \in S$ s.t. the sequence of actions $\langle act_1, ..., act_n \rangle$ is a plan for (S, i, Act, G, T), where $Act = \{act_1, ..., act_n\}$?*

It is not hard to see, that in this case one would need to check if there exists an initial state $i \in S$, s.t. there exists a trajectory

$$\mathcal{T} = \langle \langle i, act_1, s_1 \rangle, \langle s_1, act_2, s_2 \rangle, ... \langle s_{n-1}, act_n, s_n \rangle \rangle$$

with the properties:

- $\{s_1, ..., s_n\} \subseteq S$,
- $s_n \in G$.

2.2.2.5 SynVerif$\forall$

Definition 2.10. ***(SynVerif$\forall$)***

***Instance**: $(S, \langle act_1, ..., act_n \rangle, G, T)$, where $\langle act_1, ..., act_n \rangle$ is a sequence of actions, G is a set of goal states, S is a set of states and T is the transition relation.*
***Question**: Is $\langle act_1, ..., act_n \rangle$ a plan for (S, s, G, Act, T) for every $s \in S$?*

This problem is similar to the *Conformant Planning* problem, but the way the question is imposed in both of them and the answers required are different. In *Conformant Planning*, a set of predefined actions Act is given as input, and one is asked to arrange these actions (or a part of them) in a sequence that serves as a universal plan (if there exists one) for the abstract planning problem formed from the instance for every possible initial state. In contrast, a *SynVerif*$\forall$ instance consists of a specific sequence of actions (apart from the other three parameters, which *SynVerif*$\forall$ and *Conformant Planning* have in common). The reasoning problem consists in checking if this sequence of actions serves as a universal plan. Hence the plan does no depend

on the possible initial states. In other words, *Conformant Planning* is the reasoning problem of finding a universal plan for any initial state, and *SynVerif*$\forall$ is the reasoning problem of checking if a given sequence of actions is a universal plan independently of the initial state.

2.3 STRIPS Planning

STRIPS planning is a planning language, considered as a mixture between logic and procedural computation. It is one of the very few logic-related planning systems. One distinguishes between different versions of STRIPS planning. The prototypical one is the *Propositional* version. Meanwhile, different extensions such as *First Order STRIPS Planning* or *Extended Propositional STRIPS Planning* (cf. [40]) have been extensively studied.

In the following sections, we describe only the first one, which will be used further in the thesis. Later, we give some complexity results regarding the decision problem of checking if there exists a plan for *Propositional STRIPS Planning* problems.

The material here is based on the existing literature; in particular, we follow the excellent paper [6].

2.3.1 Propositional STRIPS Planning

We provide the basic definitions about *Propositional STRIPS Planning*. The components of a STRIPS language are:

1. an initial state,
2. a set of goal states,
3. a set of actions. For each action, the following are included:
   - *preconditions* which must hold before the action can be executed.
   - *postconditions* which describe how the state of the environment changes when the action is executed.

We introduce a simple example, which describes the action of moving a box from location l_1 to location l_2:

$$\begin{aligned} &Action : MoveBox(l_1, l_2), \\ &Precondition : LocAtBox(l_1), \\ &Postcondition : LocAtBox(l_2), \neg BoxAtLoc(l_1) \end{aligned}$$

To formalize the above description, we give the following definition:

Definition 2.11. ***(Propositional STRIPS Planning)*** *An instance of* Propositional STRIPS Planning *is given by a quadruple $SP = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, where:*

- *$\mathcal{P}$ represents a finite set of propositional variables, called the* conditions.
- *$\mathcal{O}$ is a finite set of* actions *(operators). Each* action *has form* **Pre**$\Rightarrow$**Post**.

  - ***Pre*** *(*precondition*) is a satisfiable conjunction of positive and negative atomic formulas (from $\mathcal{P}$). (o^+) and (o^-) are used to denote the* positive preconditions *and* negative preconditions*, respectively.*
  - ***Post*** *(*postcondition*) is a satisfiable conjunction of positive and negative atomic formulas (from $\mathcal{P}$). (o_+) and (o_-) are used to denote the* positive postconditions *and* negative postconditions*, respectively.*

- $\mathcal{I} \subseteq \mathcal{P}$ *is the* initial state

- $\mathcal{G}$*, is called the* goals*. It is a satisfiable conjunction of positive and negative* conditions*, respectively called the* positive goals $(\mathcal{G}_+)$ *and the* negative goals $(\mathcal{G}_-)$

The set $\mathcal{P}$ is the set of conditions that are relevant. A state S is a subset of $\mathcal{P}$. Hence $p \in \mathcal{P}$ is true at state S if $p \in S$, otherwise it is said to be false. $\mathcal{O}$ is the set of operators (actions), which change a state to another state if it satisfies some preconditions. $\mathcal{I}$ is the initial state. It contains exactly the conditions that are true in the initial state. Any operator, whose preconditions are in the initial state, can be applied to $\mathcal{I}$. Every $p \in \mathcal{P}$ that is in $\mathcal{I}$ is initially true. $\mathcal{G}$ is a formula, which specifies a set of states, called the goal states. That is, every $S \subseteq \mathcal{P}$ that $S \models G$ is a goal state. Intuitively, it can be understood as follows: $\mathcal{G}_+ \subseteq S$ and $\mathcal{G}_- \cap S = \emptyset$.

Next, we define the semantics of an application of an action to a state:

Definition 2.12. ***(Action Application)*** *Let o^+, o^-, o_+, o_- be defined as above. The result of applying a finite sequence of actions (operators) $(o_1, o_2, ..., o_n)$ on a state S is inductively defined as follows:*

- $Result(S, ()) = S$,
- $Result(S, (o))$=$\begin{cases} (S \cup o_+) \setminus o_-, & \text{if } o^+ \subseteq S \text{ and } o^- \cap S = \emptyset, \\ S, & \text{otherwise}, \end{cases}$
- $Result(S, (o_1, o_2, ..., o_n)) = Result(Result(S, (o_1)), (o_2, ..., o_n))$.

According to the above definition, any action can be applied at any state if the state satisfies its preconditions. If these preconditions are satisfied at a state $S \subseteq \mathcal{P}$, then the positive postconditions are added to S and the negative postconditions that occur in S are deleted from it. The new obtained state might contain the necessary preconditions for application of other actions. Also, an action might be applied several times in a sequence of actions.

We also define what is a plan for STRIPS planning:

Definition 2.13. ***(Plan)*** *Let $\langle o_1, o_2, ..., o_n \rangle$ be a finite sequence of actions. This sequence is a plan for an instance of propositional planning $SP = \langle \mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$ if $Result(\mathcal{I}, (o_1, o_2, ...o_n))$ is a goal state.*

2.3.2 Complexity results

In this section, we introduce some complexity results for the decision problem of checking if there exists a plan for a propositional STRIPS planning problem. As stated earlier, it is defined as a decision problem of determining whether there exists a plan or not for an instance of propositional STRIPS planning.

Deciding if there exists a plan for the unrestricted case of propositional STRIPS planning is **PSPACE**-*complete*. Depending on the number and type of preconditions, postconditions and goals, several interesting results are obtained. We introduce some of the results below. For proofs for each of the cases see [6].

- Checking if exists a plan for cases where actions are limited to only positive postconditions is **NP**-*complete*.
- Checking if exists a plan for cases where actions are limited to only one positive postcondition and one precondition is also **NP**-*complete*.
- Checking if exists a plan for cases where actions are limited to positive preconditions only and one postcondition is *polynomial*.
- Checking if exists a plan for cases limited to only a constant number of facts in the goal and one precondition is *polynomial*.

2.3.3 Extensions

There have been defined several extensions of STRIPS planning like First-Order STRIPS planning, Extended STRIPS planning and others. First-Order STRIPS planning can be reduced to propositional STRIPS planning [16] in polynomial time under some restrictions as follows:

- the initial state and goal state should be ground, preconditions and postcondition should be literals,
- each variable in an action should have only a polynomial number of values and
- each operator is limited to a constant number of values.

In turn, Extended STRIPS planning contains other parameters like Σ which is the *domain theory*. It puts constraints on the states and actions, which should be consistent with Σ. Furthermore, $\mathcal{D}$, called the *default preference ordering* is also a parameter of Extended STRIPS planning. It specifies an order to the literals that are preferred to be true after applying an action. Hence, the definition of application of an action for this case is different since it depends also one those two parameters.

2.4 Turing Machines

In this section we define *Turing machines*, which form the basic model of computation in Complexity Theory, since by the widely accepted Church-Turing thesis, they seem able to simulate all physically realizable computational methods. In particular, we consider *deterministic Turing machines* (abbreviated DTM).

The material here is based on the existing literature; in particular, we follow the excellent book [39], to which we refer the reader for a more extensive exposition.

Definition 2.14. ***(DTM)*** *A* deterministic Turing machine *is given by the 5-tuple* $T = (Q, \Sigma, s_0, \delta)$*, where*

- Q *is a* finite set of states *which contains the* accepting state s_{accept}*,*
- Σ *is a* finite set of symbols *(the tape alphabet) that contains the blank symbol* $\sqcup$*,*
- $s_0 \in Q$ *is the* initial state*,*
- $\delta : (Q \times \Sigma) \to (Q \times \Sigma \times \{-1, 0, +1\})$ *is the* transition function *(or* program*).*

Intuitively, a *deterministic Turing machine* (DTM) $T = (Q, \Sigma, s_0, \delta)$ works as follows. An *input* to T is simply a string W of symbols that is written on a *tape*, which consists of cells each storing one character. T has a *read/write (R/W) head* that can move along the tape, reading and modifying the contents of the cell it is currently on. In particular, -1 means one symbol to the left, $+1$ means one symbol to the right, while 0 means staying in the current position. At each time instant, T is in some state $s \in Q$, the tape contains some string $w_1, ..., w_n$, and the head is positioned at some cell $p \leq n$. This description is called a *configuration*, and is described by the triple $(s, w_1...w_{p-1}, w_p...w_n)$. A run of T starts in the initial state s_0 and with the head over the first character of W (if any) and the tape is all blank otherwise. Then it executes the program δ. In particular, if the current configuration is (q, v, u) and the first symbol of u is d, and $\delta(s, d) = (s', d', D)$, it overwrites d with d', changes its state to s' and moves the R/W head $+1$ or -1 or 0 based on D . T has a run over W (T accepts W), if T reaches the state s_{accept} on input W.

Next, we formalize the above intuitions:

Definition 2.15. ***(Configuration, yields)****. Assume a* DTM $T = (Q, \Sigma, s_0, \delta)$*. A* configuration *for* T *is a tuple* (s, v, u)*, where* $s \in Q$ *and* $v, u \in \Sigma^+$*. Assume a configuration* $C = (q, v \cdot c, d \cdot u)$*, where* $v, u \in \Sigma^+$ *and* $c, d \in \Sigma$*, and suppose* $\delta(s, d) = (s', d', D)$*. Then* C yields *the following configuration* C'*:*

1. *if* $D = 0$*, then* $C' = (s', v \cdot c, d' \cdot u)$*;*
2. *if* $D = +1$*, them* $C' = (s', v \cdot c \cdot d', u')$*, where* $u' = u$ *if* $u \neq \epsilon$ *and* $u' = \sqcup$ *otherwise:*
3. *if* $D = -1$*, them* $C' = (s', v', c \cdot d' \cdot u')$*, where* $v' = v$ *if* $v \neq \epsilon$ *and* $v' = \sqcup$ *otherwise;*

We can now formally define the computation of a DTM on a given input word.

Definition 2.16. ***(Computation, accepting a word).*** *Let $T = (Q, \Sigma, s_0, \delta)$ be a DTM and $v \in (\Sigma \backslash \{\sqcup\})^*$ be a word. The* computation *of T on v is the (possibly infinite) sequence $C_0, C_1, C_2, ...$ of configurations of T such that:*

1. *$C_0 = (s_0, \sqcup, w)$;*
2. *for each $i > 0$, C_{i-1} yields C_i;*
3. *for any $i > 0$, if $C_i = (s_{accept}, v', u')$, then C_i is the last element in the sequence.*

We say that T accepts *v if the computation of T on v if finite and the state in the last configuration is s_{accept}.*

CHAPTER 3

Description Logic For Database Manipulation

In this thesis, the environment considered for planning is the one of graph databases. The language adopted to express the updates on these databases, is an extension of $\mathcal{ALCHOIQ}$, defined in Chapter 2 (see Section 2.1) as a highly expressive description logic allowing nominal concepts, qualified number restrictions and several other constructors. The extension to $\mathcal{ALCHOIQ}br$ (cf. [8]), presented in Section 3.1 resulted to be important for fully capturing the changes in databases.

In Section 3.2, we introduce a simple transaction language, expressed through Description Logics (DLs) [2]. Transactions are finite sequences of insertions and deletions performed on unary and binary predicates (concepts and roles, in DL jargon). As usual in DLs, the semantics of the DL knowledge bases is defined in terms of interpretations. Further, we provide a descriptions of the way updates of these graph databases occur. At last, we illustrate with an example, to get a better intuition of what will follow in the thesis.

Note: Graph databases using concept names and role names can be naturally seen as finite DL interpretations.

3.1 $\mathcal{ALCHOIQ}br$ for Database Manipulation

As mentioned earlier, description logics are decidable languages which are particularly natural for talking about graph databases. In turn, even though $\mathcal{ALCHOIQ}$ is a very expressive description logic, it is still not very suitable for the planning in these graph databases. Instead, there is an extension of this DL, introduced in the paper *Evolving Graph Databases under Description Logic Constraints* (see [8]) that we use as a query language for manipulating graph databases. It is called $\mathcal{ALCHOIQ}br$, which is the standard $\mathcal{ALCHOIQ}$ (see Section 2.1) extended with Boolean combinations of axioms and a constructor for a singleton role. This particular descrip-

tion language, is able to almost fully capture the changes that might occur in a graph database. It offers high expressibility by allowing formulas as boolean combinations of assertions and inclusions, and permitting singleton concepts and roles through variables allowing different possible instantiations.

Below, we formalize the syntax and semantics of $\mathcal{ALCHOIQ}br$.

3.1.1 $\mathcal{ALCHOIQ}br$ Syntax

Intuitively, the syntax of $\mathcal{ALCHOIQ}br$ is similar to the $\mathcal{ALCHOIQ}$ language, but extended. Hence, we consider a countably infinite set N_V of *variables* apart from the usual countably infinite sets N_R of *role names*, N_C of *concept names*, N_I of *individual names*.

Definition 3.1. *($\mathcal{ALCHOIQ}br$ **Syntax**) Let N_V, N_R, N_C, N_I be countably infinite sets of* variables, role names, concept names, *and* individual names *respectively.*

$\mathcal{ALCHOIQ}br$ Roles *are defined inductively as follows:*

1. *if $r \in N_R$, then r and r^- (the* inverse *of r) are roles;*
2. *if $\{t, t'\} \subseteq N_I \cup N_V$, then $\{(t_1, t_2)\}$ is also a role;*
3. *if r_1, r_2 are roles, then $r_1 \cup r_2$, and $r_1 \backslash r_2$ are also roles.*

The set of $\mathcal{ALCHOIQ}br$ Concepts *is the smallest set such that:*

1. *if $A \in N_C$, then A is an $\mathcal{ALCHOIQ}br$ concept;*
2. *if $t \in N_I \cup N_V$, then $\{t\}$ is an $\mathcal{ALCHOIQ}br$ concept (called* nominal*);*
3. *if C_1, C_2 are concepts, then $C_1 \sqcap C_2$, $C_1 \sqcup C_2$, and $\neg C_1$ are also $\mathcal{ALCHOIQ}br$ concepts;*
4. *if r is a $\mathcal{ALCHOIQ}br$ role, C is an $\mathcal{ALCHOIQ}br$ concept, and n is a non-negative integer, then $\exists r.C$, $\forall r.C$, $\leqslant nr.C$, and $\geqslant nr.C$ are also $\mathcal{ALCHOIQ}br$ concepts.*

Expressions of the form $t : C$ and $(t, t') : r$, where $\{t, t'\} \subseteq N_I \cup N_V$, C is a concept, and r is a role, are called $\mathcal{ALCHOIQ}br$ concept assertions *and $\mathcal{ALCHOIQ}br$* role assertions*, respectively. Concepts, roles, inclusions and assertions that have no variables are called* ordinary *(corresponding to $\mathcal{ALCHOIQ}$ concepts, roles, inclusions and assertions respectively).*

*($\mathcal{ALCHOIQ}br$-)*formulae *are inductively defined as follow:*

1. *every inclusion and assertion is a formula;*
2. *if $\mathcal{K}_1, \mathcal{K}_2$ are formulas, then so are $\mathcal{K}_1 \wedge \mathcal{K}_2$, $\mathcal{K}_1 \vee \mathcal{K}_2$ and $\neg \mathcal{K}_1$.*

If a formula $\mathcal{K}$ has no variables, it is called a knowledge base (KB).

Next, we will define a new set which we call the set of *basic concepts* as follows.

Definition 3.2. *(**Basic Concepts**). The set of* basic concepts *is defined as follows:*

- *if $A \in N_C$, then A is a* basic concept*;*
- *if $t \in N_I \cup N_V$, then $\{t\}$ is a* basic concept*;*
- *if r is an $\mathcal{ALCHOIQ}br$ role then $\exists r$, $\exists r^-$ are* basic concepts.

3.1.2 $\mathcal{ALCHOIQ}br$ Semantics

As usual, semantics of a description logic is given through interpretations. An *interpretation* is a pair $\mathcal{I} = (\triangle^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\triangle^{\mathcal{I}} \neq \emptyset$ is the *domain*, $A^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}}$ for each $A \in N_C$, $r^{\mathcal{I}} \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$ for each $r \in N_R$, and $o^{\mathcal{I}} \in \triangle^{\mathcal{I}}$ for each $o \in N_I$. For the ordinary roles of the form $\{(o_1, o_2)\}$, we let $\{(o_1, o_2)\}^{\mathcal{I}} = \{(o_1^{\mathcal{I}}, o_2^{\mathcal{I}})\}$. The function $\cdot^{\mathcal{I}}$ is extended to the remaining ordinary concepts and roles in the usual way (see Section 2.1).

Assume an interpretation $\mathcal{I}$. For an ordinary inclusion $\alpha_1 \sqsubseteq \alpha_2$, $\mathcal{I}$ *satisfies* $\alpha_1 \sqsubseteq \alpha_2$ (in symbols, $\mathcal{I} \models \alpha_1 \sqsubseteq \alpha_2$) if $\alpha_1^{\mathcal{I}} \subseteq \alpha_2^{\mathcal{I}}$. For an ordinary assertion $\beta = o : C$ (resp., $\beta = (o_1, o_2) : r$), $\mathcal{I}$ *satisfies* β (in symbols, $\mathcal{I} \models \beta$) if $o^{\mathcal{I}} \in C^{\mathcal{I}}$ (resp., $(o_1^{\mathcal{I}}, o_2^{\mathcal{I}}) \in r^{\mathcal{I}}$). The notion of satisfaction is extended to knowledge bases as follows:

1. $\mathcal{I} \models \mathcal{K}_1 \wedge \mathcal{K}_2$ if $\mathcal{I} \models \mathcal{K}_1$ and $\mathcal{I} \models \mathcal{K}_2$;
2. $\mathcal{I} \models \mathcal{K}_1 \vee \mathcal{K}_2$ if $\mathcal{I} \models \mathcal{K}_1$ or $\mathcal{I} \models \mathcal{K}_2$;
3. $\mathcal{I} \models \neg\mathcal{K}$ if $\mathcal{I} \not\models \mathcal{K}$.

If $\mathcal{I} \models \mathcal{K}$, then $\mathcal{I}$ is a *model* of $\mathcal{K}$.

3.2 Action Language

We now define an action language $\mathcal{L}^{act}$ that allows us to express finite sequences of operations on interpretations (i.e. graph databases). $\mathcal{L}^{act}$ is expressed via operations on $\mathcal{ALCHOIQ}br$ roles, $\mathcal{ALCHOIQ}br$ concepts and $\mathcal{ALCHOIQ}br$ formulae as follows:

Definition 3.3. *(**Atomic action**). An* atomic action $\rho \in \mathcal{L}^{act}$ *is given by the following grammar:*

$$\rho \rightarrow (A \oplus C)|(A \ominus C)|(r \oplus p)|(r \ominus p)$$

where $A \in N_C$, $r \in N_R$, C is an $\mathcal{ALCHOIQ}br$ concept, p is an $\mathcal{ALCHOIQ}br$ role.

Next we define the notion of a *basic action*, which will be used often throughout the thesis.

Definition 3.4. *(**Basic action**). A* basic action *is given by the following grammar:*

$$(A \oplus C)|(A \ominus C)|(r \oplus p)|(r \ominus p)$$

where $A \in N_C$, $r \in N_R$ and C is an basic concept *and p is an $\mathcal{ALCHOIQ}br$ role.*

Note: It is easy to see from the above two definitions that every basic action is a atomic action.

Definition 3.5. *(**Transaction**)* Transactions *(*complex action*)* $\rho^{tr} \in \mathcal{L}^{act}$ *are defined inductively.*

- *An atomic action is a transaction* ρ^{tr}.
- *If* ρ_1^{tr} *and* ρ_2^{tr} *are transactions then*

$$(\rho \circ \rho_1^{tr})|(\mathcal{K}?\rho_1^{tr} : \rho_2^{tr})|\varepsilon,$$

where ρ *is an atomic action,* $\mathcal{K}$ *is an arbitrary* $\mathcal{ALCHOIQ}br$*- formula, are transactions.*

A transaction ρ^{tr} *is ground if it has no variables. A transaction* $\rho^{tr'}$ *is called a ground instance of a transaction* ρ^{tr} *if* $\rho^{tr'}$ *is ground and it can be obtained from* ρ^{tr} *by replacing each variable by an individual name from* N_I*. For a transaction* ρ^{tr}*, we will sometimes write* $\rho^{tr}(x)$*, where* x *is a tuple containing exactly the variables of* ρ^{tr}.

Intuitively, an application of an atomic action $(A \oplus C)$ on an interpretation $\mathcal{I}$ stands for the addition of the content of $C^{\mathcal{I}}$ to $A^{\mathcal{I}}$. The deletion of $C^{\mathcal{I}}$ from $A^{\mathcal{I}}$ can be done by applying $(A \ominus C)$ on $\mathcal{I}$. The two operations can be also performed on extensions of roles. In addition, transactions allow for composing basic actions and for conditional action execution.

In order to formally define the semantics of actions, we first introduce the notion of *interpretation updates*. Furthermore, we call *positive actions* the actions that perform addition and *negative actions* the ones that perform deletion.

3.3 Interpretation Updates

Below, we formally define how updates on interpretation occur.

Definition 3.6. *(**Interpretation Updates**) Assume an interpretation* $\mathcal{I}$ *and let* σ *be a concept or role name. If* σ *is a concept, let* $W \subseteq \triangle^{\mathcal{I}}$ *be a unary relation, otherwise, if* σ *is a role, let* $W \subseteq \triangle^{\mathcal{I}} \times \triangle^{\mathcal{I}}$ *be a binary relation. Then we let* $\mathcal{I} \oplus_\sigma W$ *(resp.,* $\mathcal{I} \ominus_\sigma W$*) denote the interpretation* $\mathcal{I}'$ *such that:*

- $\triangle^{\mathcal{I}'} = \triangle^{\mathcal{I}}$
- $\sigma_1^{\mathcal{I}'} = \sigma_1^{\mathcal{I}}$ *for all symbols* $\sigma_1 \neq \sigma$ *and*
- $\sigma^{\mathcal{I}'} = \sigma^{\mathcal{I}} \cup W$ *(resp.* $\sigma^{\mathcal{I}'} = \sigma^{\mathcal{I}} \backslash W$*)*

Now we can define the semantics of ground transactions inductively as follows.

Definition 3.7. *Given a ground transaction* ρ^{tr}*, we define a mapping* $M_{\rho^{tr}}$ *from interpretations to interpretations as follows:*

$M_{(A \oplus C)\cdot\rho^{tr}}(\mathcal{I}) = M_{\rho^{tr}}(\mathcal{I} \oplus_A C^{\mathcal{I}})$

$$M_{(A\oplus C)\cdot\rho^{tr}}(\mathcal{I}) = M_{\rho^{tr}}(\mathcal{I} \oplus_A C^{\mathcal{I}})$$

$$M_{(r\oplus p)\cdot\rho^{tr}}(\mathcal{I}) = M_{\rho^{tr}}(\mathcal{I} \oplus_r p^{\mathcal{I}})$$

$$M_{(r\ominus p)\cdot\rho^{tr}}(\mathcal{I}) = M_{\rho^{tr}}(\mathcal{I} \ominus_r p^{\mathcal{I}})$$

$$M_{\epsilon}(\mathcal{I}) = \mathcal{I}$$

$$M_{\mathcal{K}?\rho_1^{tr}:\rho_2^{tr}}(\mathcal{I}) = \begin{cases} M_{\rho_1^{tr}}(\mathcal{I}), \text{ if } \mathcal{I} \models \mathcal{K} \\ M_{\rho_2^{tr}}(\mathcal{I}), \text{ if } \mathcal{I} \not\models \mathcal{K} \end{cases}$$

Note: Throughout the thesis, when convenient, we will view an interpretation $\mathcal{I}$ as a set of literals I (see Example 2.1). The corresponding definition of updates on interpretations viewed as sets of atoms, would intuitively be as follows:

Given a ground transaction ρ^{tr}, we define a mapping $M_{\rho^{tr}}$ from interpretations as sets of atoms to sets of atoms as follows:

$$M_{(A\oplus C)\cdot\rho^{tr}}(I) = M_{\rho^{tr}}(I \cup \{A(c) \mid I \models C(c)\})$$

$$M_{(A\ominus C)\cdot\rho^{tr}}(I) = M_{\rho^{tr}}(I \setminus \{A(c) \mid I \models C(c)\})$$

$$M_{(r\oplus p)\cdot\rho^{tr}}(I) = M_{\rho^{tr}}(I \cup \{r(c_1, c_2) \mid I \models p(c_1, c_2)\})$$

$$M_{(r\ominus p)\cdot\rho^{tr}}(I) = M_{\rho^{tr}}(I \setminus \{r(c_1, c_2) \mid I \models p(c_1, c_2)\})$$

$$M_{\epsilon}(I) = I$$

$$M_{\mathcal{K}?\rho_1^{tr}:\rho_2^{tr}}(I) = \begin{cases} M_{\rho_1^{tr}}(I), \textit{ if } I \models \mathcal{K}, \\ M_{\rho_2^{tr}}(I), \textit{ if } I \not\models \mathcal{K}. \end{cases}$$

Intuitively, each time an update of an interpretation occurs, depending on the transaction applied, literals are added or deleted by making the set bigger or smaller, respectively. Surely, to perform the updates, the transactions should be ground. Hence, the variables (if there exists any) should be instantiated before application of the action. Furthermore, it is easy to see that if no action is applied, the interpretation will remain unchanged. The last mapping in the Definition 3.7 defines the semantics of application of a conditional transaction on an interpretation. Thus, if the current interpretation satisfies the $\mathcal{ALCHOIQ}br$ formulae $\mathcal{K}$, then the first transaction ρ_1^{tr} is applied, otherwise ρ_2^{tr} is applied.

Next, we give some examples to better illustrate the notion of an interpretation update and transactions. Interpretations will be viewed as sets of literals.

3.3.1 Examples

Example 3.1. *The following interpretation I represents (part of) the project database of some research institute. There are two active projects, and there are four employees of which three work in the active projects.*

$$\begin{aligned} I = \{ & ActiveProject(P20840), ActiveProject(P24090), \\ & Project(P20840), Project(P24090), \\ & ConcludedProject(), \\ & Employee(E01), Employee(E03), Employee(E04), Employee(E07), \\ & ProjectEmployee(E01), ProjectEmployee(E03), ProjectEmployee(E07), \\ & PermanentEmployee(E04), \\ & worksFor(E01, P20840), worksFor(E03, P20840), worksFor(E07, P24090)\}, \end{aligned}$$

The following transaction captures the termination of project $P20840$, which is removed from the active projects and added to the concluded ones. The employees working for this project are removed from the project employees.

$$\begin{aligned} \rho_1^{tr} = \; & ActiveProject \ominus \{P20840\} \circ \\ & ConcludedProject \oplus \{P20840\} \circ \\ & ProjectEmployee \ominus \exists worksFor.\{P20840\} \end{aligned}$$

The interpretation $M_{\rho_1^{tr}}(\mathcal{I}_1)$ that reflects the status of the database after transaction ρ_1^{tr} looks as follows:

$$\begin{aligned} M_{\rho_1^{tr}}(I) = \{ & ActiveProject(P24090), \\ & Project(P20840), Project(P24090), \\ & ConcludedProject(P20840), \\ & Employee(E01), Employee(E03), Employee(E04), Employee(E07), \\ & ProjectEmployee(E07), \\ & PermanentEmployee(E04), \\ & worksFor(E01, P20840), worksFor(E03, P20840), \\ & worksFor(E07, P24090)\}, \end{aligned}$$

In our approach, all the individual variables of a transaction are seen as parameters, whose values are given before executing a transaction.

Example 3.2. *The following transaction ρ_2^{tr} with variables x, y, z transfers the employee x from project y to project z:*

$$\begin{aligned} \rho_2^{tr} = \; & (x : Employee \wedge y : Project \wedge z : Project \wedge (x, y) : worksFor)\,? \\ & worksFor \ominus \{(x, y)\} \cdot worksFor \oplus \{(x, z)\} : \epsilon \end{aligned}$$

The transaction ρ_2^{tr} first checks whether x is an employee, y and z are projects, and x works for y. If yes, it removes the worksFor *link between x and y and creates a* worksFor *link between x and z. If any of the checks fails, it does nothing.*

CHAPTER 4

Planning in Graph Databases

Planning as a reasoning task in Artificial Intelligence, has been studied extensively for different settings like robots, agents, etc. In turn, DLs are particularly natural for talking about graph databases, which may be updated and changed by users or applications. Therefore, different planning problems regarding the achievement of a certain goal (addition/deletion of information in the database) are important for the automated reasoning.

In this thesis, we view planning in the setting of graph-structured data where actions are expressed through the DL $\mathcal{ALCHOIQ}br$, which is already defined in Section 3.1. Below we give a simple example for illustrating the idea.

Example Consider the previous database of some research institute with one active project, four employees of which two work in this project and assume at a certain moment a new project needs to become active. Therefore, the database needs to be changed by adding this new project to active projects. But, this is not such an easy task to accomplish, since some 'external' constraints might be present. E.g. there might be a constraint that imposes a project to become active if there exists a certain amount of employees who can work on that, depending on the difficulty and amount of time required to finish the project. Moreover, these employees might need to be currently not working on other projects or it might be necessary for them to have a certain background compatible with the requirements of the project. All these constraints must be accomplished and before adding the intended project to active ones the database should be updated to this new state containing all the information needed in a suitable shape.

To sum up, the way information is added/deleted from a certain database, very much depends on the constraints imposed by the user and changes that the database needs to submit to, prior to the final intended update to happen.

In Chapter 3, we have already introduced a simple action language for insertions and deletions performed on literals represented as unary and binary atoms (concepts and roles, in DL jargon). This language is suitable for capturing changes on graph databases. Thus, in this thesis, we consider the planning problem for the setting of graph databases under DL constraints. It

consists in finding a suitable sequence of transactions so that the goal is achieved. As usual in DLs, the semantics of DL knowledge bases expressing the constraints is defined in terms of interpretations. In turn, graph databases can be naturally seen as finite DL interpretations. Furthermore, the updates provided by the transaction language are similar in spirit to the knowledge base (more concretely, ABox) updates studied in other works, e.g. [36], but are done directly to states of the world (which can be seen as states of complete knowledge) rather than on states of knowledge.

4.1 Planning language

In Preliminaries, we have already defined the notion of an *planning problem* as a tuple $AP = (S, i, g, Act, T)$, where *S* is a set of states, *i* is the initial state, *g* is a set of goal states, *Act* is a finite set of actions and *T* is ternary transition relation capturing all the possible transitions between states.

Furthermore, in the previous chapter, we introduced the environment we will be basing our work, which is the one of graph databases under description logic constrains. Later, we also defined the transaction language adopted for describing changes in this setting.

In this section we define the planning language we adopt for our setting. We instantiate the *planning problem* with the setting of graph databases and then we consider two different planning problems related to it: *the general planning problem* and *the fixed domain planning problem*.

4.1.1 Graph Database Planning Problem

Next, we will give a definition for a *graph database planning problem* and two problems corresponding to it.

Definition 4.1. ***(Graph Database Planning Problem (GDPP))*** *A* GDPP *is a tuple* $GP = (I', Act', G')$ *s.t.*

- I' *is an interpretation representing the initial database,*
- $Act' \subset \mathcal{L}^{act}$ *is a set of predefined transactions (*$\mathcal{L}^{act}$ *is the transaction language defined in the previous chapter), and*
- G' *is a satisfiable* $\mathcal{ALCHOIQ}br$ *formula, called the goal.*

The goal is an $\mathcal{ALCHOIQ}br$ formula, which needs to be satisfiable and the actions are the transactions that update the databases.

Below we will define two planning problems $p(GP)$ and $fp(GP)$ corresponding to GP. The difference between them is in the interpretation domain. While $p(GP)$ is the general planning problem, $fp(GP)$ is a planning problem under a fixed domain.

More formally we introduce the following definitions:

Definition 4.2. *(**General Planning Problem**) Assume a GDPP $GP = (I', Act', G')$. The* general planning problem *p corresponding to GP is:*

$$p(GP) = (S, I, G, \Gamma^{act}, T),$$

where S, I, G, Γ^{act}, T are defined as follows:

- *S is the set of all interpretations,*
- *$I = I'$,*
- *$G = \{s \in S \mid s \models G'\}$,*
- *$\Gamma^{act} = Act'$, and*
- *$T = \{(s, \rho^{tr}, s') \mid \{s, s'\} \subseteq S \wedge \rho^{tr} \in \Gamma^{act} \wedge s' = M_{\rho^{tr}}(s)\}$.*

This planning problem $p(GP)$ represents the general planning problem where interpretations are defined as usual (see 3.1.2).

Before we define the $fp(GP)$ planning problem, we need to formally define the notion of a *fixed domain* with respect to a graph database planning problem *GDPP*, which we call $dom(GP)$.

Definition 4.3. *(**dom(GP)**) Let $GP = (I', Act', G')$ be a GDPP. dom(GP) is defined as follows:*

$$(o \in dom(GP)) \text{ iff}$$

$$o \in N_I \text{ and } o \text{ occurs in some atom in} I' \text{ or } o \text{ occurs in } Act' \text{ or } o \text{ occurs in } G'$$

dom(GP) is the interpretation domain restricted to only the individuals, which occur in the given GP instance. Since we have adopted the *standard name assumption*, every individual, for all interpretations, will be interpreted by itself. Now, we are prepared to define the fixed domain planning problem which corresponds to GP.

Definition 4.4. *(**Fixed Domain Planning Problem**) Assume a GDPP $GP = (I', G', Act')$. The* fixed domain planning problem *corresponding to GP is:*

$$fp(GP) = (S, I, G, \Gamma^{act}, T),$$

where S, I, G, Γ^{act}, T are defined as follows.

- *S is the (finite) set of all interpretations $\mathcal{I}$ with domain $\triangle^{\mathcal{I}} = dom(GP)$,*
- *$I = I'$,*
- *$G = \{s \in S \mid s \models G'\}$,*
- *$\Gamma^{act} = Act'$, and*

- $T = \{(s, \rho^{tr}, s') \mid \{s, s'\} \subseteq\in S \wedge \rho^{tr} \in \Gamma^{act} \wedge s' = M_{\rho^{tr}}(s)\}$.

The *fixed domain planning problem* is defined over *dom(GP)*. Intuitively, no new constants (constants that do not appear in $dom(GP)$) can be introduced by the application of transactions. Only the constants that occur in the initial instance can happen in any possible constructible trajectory $\mathcal{T}$ for $fp(GP)$.

Definition 4.5. *Assume $AP = (S, i, G, Act, T)$. We denote $s_C \subseteq s \in S$, the set of atoms that satisfy C, where C is an $\mathcal{ALCHOIQ}br$ concept or role.*

Note: Sometimes, during the rest of the thesis, we will say that G is a conjunction of positive and negative literals expressed via $\mathcal{ALCHOIQ}br$ roles and basic concepts. This will be an abbreviation for assertions as follows.

- If $o : C$, $\neg(o : C)$, then $C(o)$, resp. $\neg C(o)$ are positive, resp. negative literal, where C is a basic concept.
- If $(o_1, o_2) : R$, $\neg((o_1, o_2) : R)$, then $R(o_1, o_2)$, resp. $\neg R(o_1, o_2)$ are positive, resp. negative literals, where R is an $\mathcal{ACLHOIQ}br$ role.

If C is a concept name and R is a role name, we will say that G is a conjunction of positive and negative literals expressed via role names and concept names.

4.1.2 Examples

Now that we have defined the planning language in the setting of graph databases under description logic constraints, we illustrate with the following example.

Example 4.1. *We consider again the previous '*project*' example. Let $GP = (I, Act, G)$ be as follows.*

$$
\begin{aligned}
I = \{&ActiveProject(P20840), ActiveProject(P24090),\\
&Project(P20840), Project(P24090),\\
&ConcludedProject(),\\
&Employee(E01), Employee(E03), Employee(E04), Employee(E07),\\
&ProjectEmployee(E01), ProjectEmployee(E03),\\
&ProjectEmployee(E07),\\
&PermanentEmployee(E04),\\
&worksFor(E01, P20840), worksFor(E03, P20840),\\
&worksFor(E07, P24090)\}.
\end{aligned}
$$

The initial database of this research institute contains two active projects, four employees of which three work in the active projects.

Assume, the goal G is a conjunction of one negative atom and a positive atom as follows.

$$G = \neg ActiveProject(P20840) \wedge PermanentEmployee(E03)$$

Thus, the initial database needs to be updated so that it reaches a state that does not contain ActiveProject*$(P20840)$ but contains* PermanentEmployee*$(E03)$.*
The set of predefined transactions Act consists of one transaction and one atomic action as follows:

$$Act = \{\rho_1^{tr}, \rho_2\},$$

where:
$\rho_1^{tr} =$ (ActiveProject $\ominus X$) $\circ$ (ConcludedProject $\oplus X$) $\circ$ (ProjectEmployee $\ominus \exists$worksFor.X),
and
$\rho_2 =$ PermanentEmployee $\oplus X$.

ρ_1^{tr} is a transaction consisting of a sequence of three atomic actions. It captures the termination of some project, which in the second action is added to ConcludedProjects *and the employees working in that project are taken out of* ProjectEmployee. *Thus, every time ρ_1^{tr} is applied, each of these three atomic actions will be applied in the order they appear in the transaction. The second transaction ρ_2 is an atomic action which captures the addition of some employee to concept* PermanentEmployee.

Example 4.2. *We view the general planning problem and the fixed domain planning problem corresponding to GP separately.*

- *Given GP as above, we extract $dom(GP) = \{P20840, P24090, E01, E03, E04, E07\}$. Now, we consider the planning problem over $dom(GP)$. Hence, the variable X can be instantiated only with elements from $dom(GP)$. After we ground ρ_1^{tr} and ρ_2 by instantiating the variable X in the ρ_1^{tr} with $P20840$ and X in ρ_2 with $E03$, a plan that would satisfy the goal G is:*

$$\mathcal{P} = \langle \rho_1^{tr}, \rho_2 \rangle =$$

$$\langle (\text{ActiveProject} \ominus \{P20840\}) \circ (\text{ConcludedProject} \oplus \{P20840\}) \circ$$

$$(\text{ProjectEmployee} \ominus \exists \text{worksFor.}\{P20840\}), (\text{PermanentEmployee} \oplus \{E03\}) \rangle$$

To show that $\mathcal{P}$ is a plan for this instance we need to show that the trajectory

$$\mathcal{T}_{\mathcal{P}} = \langle \langle s_1, \rho_1^{tr}, s_2 \rangle, \langle s_2, \rho_2, s_3 \rangle \rangle,$$

where $s_2 = M_{\rho_1^{tr}}(s_1)$ and $s_3 = M_{\rho_2}(s_2)$ is such that $s_3 \models G$.

We construct s_2 and s_3 as follows.

$$s_2 = M_{\rho_1^{tr}}(I) = \{ActiveProject(P24090),$$
$$Project(P20840), Project(P24090),$$
$$ConcludedProject(P20840),$$
$$Employee(E01), Employee(E03), Employee(E04),$$
$$Employee(E07),$$
$$ProjectEmployee(E01),$$
$$PermanentEmployee(E04)$$
$$worksFor(E01, P20840), worksFor(E03, P20840),$$
$$worksFor(E07, P24090)\}$$

Now, we construct s_3.

$$s_3 = M_{\rho_2}(s_2) = \{ActiveProject(P24090),$$
$$Project(P20840), Project(P24090),$$
$$ConcludedProject(P20840),$$
$$Employee(E01), Employee(E03), Employee(E04),$$
$$Employee(E07),$$
$$ProjectEmployee(E01),$$
$$PermanentEmployee(E04), PermanentEmployee(E03)$$
$$worksFor(E01, P20840), worksFor(E03, P20840),$$
$$worksFor(E07, P24090)\}$$

It is easily seen that $s_3 \models \neg ActiveProject(P20840)$ *and* $s_3 \models PermanentEmployee(E03)$. *Hence,* $\mathcal{P}$ *is a plan for this instance.*

- *Next, let's consider* $p(GP)$. *We are free to instantiate* X *with any fresh constant that does not appear in the given instance. Thus, we have infinite possibilities of instantiations for a single atomic action. An possibility of grounding the second action is by instantiating* X *in* ρ_2 *with a new employee that does not appear in the instance, i.e.* $\rho_2 = (PermanentEmployee \oplus E10)$.

  In turn, for this instance, it is easy to see that a plan does not require fresh constants to achieve the plan. Furthermore, any possible grounding of a transaction with a new constant is redundant. Intuitively, a sequence of transactions to be a plan for $p(GP)$, *requires* ρ_1^{tr} *to be grounded by instantiating* X *with* $P20840$ *and* ρ_2 *with* $E03$. *Any other transaction can be deleted.*

But, there are cases that enforce the use of a fresh individual so that a plan for a given instance exists. We show this in the example below:

Example 4.3. *Let $GP = (I, Act, G)$ be as follows:*

$$I = \{worksFor(E01, P20840)\}.$$

$$Act = \{\rho_1^{tr}\},$$

where:
$\rho_1^{tr} = (Project \oplus \{X\}) \circ (worksFor \oplus (E03, \{X\})) \circ (ProjectEmployee \oplus \exists worksFor.\{X\})$, *and*

$$G = \neg Project(E01) \wedge \neg Project(E03) \wedge$$

$$ProjectEmployee(E03) \wedge \neg worksFor(E03, P20840).$$

The goal formula to be satisfied, requires a state which does not contain $Project(P20840)$, $Project(E01$ and $Project(E03$. In addition, this state should contain $ProjectEmployee(E03)$ and does not contain $worksFor(E03, P20840)$. $dom(GP) = \{P20840, E01, E01\}$. Also, $I \models \neg worksFor(E03, P20840)$.

It is not hard to notice that there is no plan for $fp(GP)$, since X cannot be instantiated by any of the constants from $dom(GP)$.

Instead, $p(GP)$ has an infinite number of plans, since there can be infinite possible instantiations of X such that a goal state can be reached. E.g. grounding ρ_1^{tr} by instantiating X with $P10000$ and grounding ρ_2, instantiating X with $P201840$, the following sequence of ground transactions is indeed a plan for $p(GP)$.

$$\mathcal{P} = \langle (Project \oplus \{P10000\}) \circ (worksFor \oplus (E03, \{P10000\})) \circ$$

$$(ProjectEmployee \oplus \exists worksFor.\{P10000\}) \rangle$$

The final state of the trajectory $\mathcal{T}_\mathcal{P}$ is the following:

$$\begin{aligned} s = \{&Project(P10000), \\ &ProjectEmployee(E03) \\ &worksFor(E01, P20840), worksFor(E03, P10000)\}. \end{aligned}$$

Clearly, the following holds.

$$s \models G.$$

Thus, $s \in S$ is a goal state for $p(GP)$.

The above two examples show that it is not the case that whenever there is a plan for $p(GP)$, there will also be a plan for $fp(GP)$. We will study plan existence for those two problems as separate decision problems.

4.2 Reasoning Problems For GDPP

The space problem, in our setting, involves several dimensions, depending on the parameters given as input, whose different variations lead to different planning problems. We have already given an overview of different decision problems in Section 2.2.2. For the planning in the setting of graph databases under description logic constraints, some of them become very interesting.

One of the main reasoning tasks for planning, is determining the existence of a plan (that we call plan-existence). It consists in deciding if there exists a plan for a given planning problem. For the setting of graph databases, we have distinguishes between two cases in Section 4.1.1.

- The first case is deciding *plan-existence*, where the use of constants that do not appear in $dom(GP)$ is not allowed. We call it *fixed domain plan-existence*.
- The second case is deciding *plan-existence* where there is no specification of a $dom(GP)$. We call it the general *plan-existence*.

Deciding *fixed domain plan-existence* will be our main focus for the rest of the thesis. Allowing the use of constants that do not appear in $dom(GP)$, seems to increase the complexity for different cases. But, one might try answering questions like: Which is the most expressive setting that uses fresh constants s.t. we have the property that for all plans that use fresh constants, for the same given instance, there exists a plan that does not use fresh constants? This coincides with deciding the least expressive setting that enforces the use of fresh constants.

Another interesting problem for planning in graph databases is Conformant Planning. The idea is that given a sequence of transactions and a goal as input, one might ask if these sequence represents a universal plan for any planning problem containing this goal. In practice, these are the kind of procedures that consist in manipulation of databases. One would want to create a program that reaches the goal. The actions would be like small program building blocks and G would be a goal. The task would be to compose these modules into a big program which would achieve the goal for any initial database.

CHAPTER 5

Deciding Fixed Domain Plan-Existence

In this chapter, we study the *plan-existence* decision problem for our setting of graph databases under description logic constraints. *Plan-existence* is the decision problem which consists in determining the existence of a plan for a given planning instance (see 4.2), which in general is known to be intractable [9]. Fortunately, by enforcing some restrictions on different parameters allowed as input, several tractable cases can be distinguished. Therefore, in addition to analyzing the complexity for *plan-existence* in the general setting, we try to define the fine lines between complexity classes for different restricted cases of this problem.

In our framework, we distinguish between two types of cases of *plan-existence* depending on the allowance or not of instantiations of variables with fresh constants by the transactions: *the general plan-existence* and *fixed domain plan-existence* (see Section 4.1.1).

In Section 3.2, we have defined the action language $\mathcal{L}^{act}$ using the DL $\mathcal{ALCHOIQ}br$. The variables, complex concepts with existential quantifications and negative atoms in the goal formula seem to be the main triggers that might enforce the use of fresh constants in a transaction. We will give a proof of the **PSPACE**-*completeness* of deciding *fixed domain plan-existence*. Further, we study this case in detail, trying to reach tractable cases, by restricting the types of transactions, and goal formulas allowed as input. We are able to extract *nlogpsace* and *polynomial* cases when allowing basic actions combined with several types of goals. We also notice that even for cases with basic actions that contain only concept names and role names, deciding *fixed domain plan-existence* is already intractable. Indeed, it is already **NP**-*hard*.

Regarding the general case, intuitively even basic actions with variables performing additions or deletions on concepts or roles of the form $(Student \oplus X)$ have infinite number of possible instantiations. But, are these instantiations necessary or there might exist a plan also with constants from $dom(GP)$? Does this create any problem when studying the complexity? We will investigate this problem deeper in Chapter 6.

Now, we will illustrate with an example to clarify the distinction between the two cases stated above.

Example 5.1. *Let $GP = (I, Act, G)$ be given as follows.*

$$\begin{aligned} I = \{&ActiveProject(P24090),\\ &Project(P20840), Project(P24090),\\ &ConcludedProject(P20840),\\ &Employee(E01), worksFor(E07, P24090)\}. \end{aligned}$$

$$Act = \{\rho_1, \rho_2, \rho_3, \rho_4\},$$

where:

$$\begin{aligned} \rho_1 &= ActiveProject \oplus Project,\\ \rho_2 &= (ActiveProject \ominus X) \circ (ConcludedProject \oplus X),\\ \rho_3 &= ConcludedProject \ominus X,\\ \rho_4 &= ProjectEmployee \oplus \exists worksFor.X \end{aligned}$$

$$G = \neg ActiveProject(P24090) \wedge ConcludedProject(P24090) \wedge PermanentEmployee(E01).$$

If we restrict the interpretation domain to $dom(GP) = \{P24090, P20840, E01\}$ then for every possible trajectory over $fp(GP)$, X will be instantiated only with elements from $dom(GP)$. If this restriction is not imposed, then X has infinite possibilities of instantiations. This corresponds to the General Planning Problem.

In this chapter, we will study the fixed domain plan-existence decision problem. We prove that it is **PSPACE**-*complete*. Further, we analyze different cases with syntactic restrictions on the set of predefined actions and the goal. We prove **NP**-*hardness* for a basic case and outline several tractable cases.

5.1 PSPACE Upper-Bound

It is well-known that planning is intractable in general and that severe obstacles stand in the way [9]. In our case, *fixed domain plan-existence* is in **PSPACE**. In this section we give an algorithm for deciding plan existence over a fixed domain, which runs in polynomial space, but before we introduce the following three sets.

Assume a *GDPP* $GP = (I, G, Act)$. Let I_{GP}, C_{GP}, R_{GP} be finite sets of individuals, concept names and role names w.r.t. GP defined as follows:

- I_{GP} =*dom(GP)*,
- C_{GP} is a set of all concept names that occur in GP,
- R_{GP} is a set of all role names that occur in GP.

Next, we introduce a non-deterministic algorithm for checking if there exists a plan for a given planning problem over a fixed domain. We prove that the algorithm runs in polynomial space.

5.1.1 Intuition

Let $GP = (I, Act, G)$ be a graph database planning problem and let I_{GP}, C_{GP}, R_{GP} be the extracted sets defined as in 5.1. It is easily seen that $S(GP) = |R_{GP}| \times |I_{GP}|^2 + |C_{GP}| \times |I_{GP}|$ is the biggest number of different facts that can be in the largest database that can be built for an arbitrary planning problem over a fixed domain. Indeed, a set of atoms of this size would represent the largest state that can be reached from any constructible trajectory over $fp(GP)$.

In turn, $2^{S(GP)}$ is the largest number of different databases that can be constructed for a given planning problem over a fixed domain. It is easily seen that there is an exponential number of states that can be reached depending on the size of the given instance. If there exists a plan for a given $fp(GP)$, then there exists another plan s.t. the corresponding trajectory, would be less than $2^{S(GP)}$. Intuitively, any trajectory of length $2^{S(GP)}$ or larger must have *'loops'*, i.e. there must be some state that it visits at least twice. Therefore, we would end up in redundancies, which can be avoided by removing these transitions that produce the same state, resulting in a plan of length less than $2^{S(GP)}$.

To define this intuition more formally, we give the following theorem.

Theorem 5.1. *Let $GP = (I, Act, G)$ be a* GDPP. *For every plan $\mathcal{P}$ of $fp(GP)$ there exist a plan $\mathcal{P}'$ whose length is bounded by $2^{S(GP)}$.*

To prove this theorem, it is sufficient to prove the following lemma.

Lemma 5.1.1. *Let $\mathcal{P}$ be a plan for $fp(GP)$ and let $s \in S$ be a state which is repeated k times in the trajectory $\mathcal{T}_{\mathcal{P}}$. Then there exists plan $\mathcal{P}'$ with $k-1$ repetitions of $s \in S$ in $\mathcal{T}_{\mathcal{P}'}$.*

Proof. W.l.o.g assume that $\mathcal{P} = \langle \rho_1^{tr}, ..., \rho_n^{tr} \rangle$ has length n and the kth repetition of $s \in S$ occurs at position $l \leq n$ and the $(k-1)$th repetition occurs at position $m < l$. Hence, the trajectory has the following form:

$$\mathcal{T}_{\mathcal{P}} = \langle \langle s_0, \rho_1^{tr}, s_1 \rangle, ..., \langle s_{m-1}, \rho_m^{tr}, s \rangle, \langle s, \rho_{m+1}^{tr}, s_{m+1} \rangle, ...,$$

$$\langle s_{l-1}, \rho_l^{tr}, s \rangle, \langle s, \rho_{l+1}^{tr}, s_{l+1} \rangle, ..., \langle s_{n-1}, \rho_n^{tr}, s_n \rangle \rangle,$$

where $\{s_0, ..., s_n\} \subseteq S$, s_n is the goal state, $I = s_0$ is the initial database corresponding to the initial state.

It is easily seen that the sequence of transactions $\rho_{m+1}^{tr}, ..., \rho_l^{tr}$ is redundant since the goal state can still be reached even after deleting this sequence from $\mathcal{P}$. The new plan $\mathcal{P}'$ will still be a plan for $fp(GP)$. The new trajectory is as follows:

$$\mathcal{T}_{\mathcal{P}'} = \langle \langle s_0, \rho_1^{tr}, s_1 \rangle, .., \langle s_{m-1}, \rho_m^{tr}, s \rangle, \langle s, \rho_{l+1}^{tr}, s_{l+1} \rangle, .., \langle s_{n-1}, \rho_n^{tr}, s_n \rangle \rangle.$$

□

Intuitively, applying this lemma for every state and for every repetition, the resulting plan would be bounded by $2^{S(GP)}$.

5.1.2 *FDPE* Algorithm

Now, we give the pseudo-code describing a non-deterministic algorithm for checking the existence of a plan for a planning problem over a fixed domain. It takes as input an instance of a fixed domain planning problem. We call it *FDPE* (Fixed Domain Plan-Existence).

Input: An instance of a graph database planning problem $GP = (I, Act, G)$.
Output: The algorithm returns **true** if it finds a plan for $fp(GP)$, **false** otherwise.

```
1  if I ⊨ G then
2  |  return true
3  end
4  s ← I;
5  count ← 0 ;
6  while count < 2^S(GP) do
7  |  Select an arbitrary ρ^tr from Act;
8  |  s ← Mρ^tr(s);
9  |  if s ⊨ G then
10 |  |  return true;
11 |  |  break;
12 |  else
13 |  |  count ← count +1;
14 |  end
15 end
16 return false;
```

Algorithm 5.1: Fixed Domain Plan-Existence (*FDPE*)

The above *FDPE* algorithm uses a counter, which counts the number of non-deterministically chosen transactions. The algorithm starts by checking if the initial database given as input, is already a goal state. In that case, it stops and returns **true**. If not, it initializes by mapping s, which represents the states $s \in S$, to the initial database I, and by setting the counter to 0. The counter, which counts the length of the plan, is bounded in the algorithm by $2^{S(GP)}$ (see Theorem 5.1). Next, it non-deterministically selects a transaction from the set Act and applies it to the initial database. This is accomplished by applying the mapping function $M_{\rho^{tr}}$ (see Definition 3.7), whose application results in a new database. It checks if the new database is a goal state; if yes then it returns **true** and stops. Otherwise, it increases the counter by one and checks if the counter has reached the limit. In case it hasn't reached the limit, it continues by selecting a new transaction and repeating the same procedure again until it reaches **true**. If the counter has reached the limit $2^{S(GP)}$ then the algorithm returns **false**, which means that it didn't find a plan. Note that the algorithm does not store in memory all the transactions applied. In contrast, it stores only one transaction at a time, the current database, the counter and the goal formula.

Next, we introduce the following theorem, which states that the algorithm is sound and complete.

Theorem 5.2. *Let $GP = (I, G, Act)$ be a GDPP. Then $fp(GP)$ has a plan iff the **FDPE** algorithm has a run that returns **true** on $fp(GP)$.*

Proof. '$\Rightarrow$' Assume there exists a plan for $fp(GP)$. By Theorem 5.1, if there exists a plan then there is a plan $\mathcal{P}$ whose the length cannot exceed $2^{S(GP)}$. By Definition 2.5, there exists a finite sequence of transactions $\langle \rho_1^{tr}, ..., \rho^t r_n \rangle$, where $\{\rho_1^{tr}, ..., \rho^t r_n\} \subseteq Act$, $n \geq 1$ s.t. there exists a trajectory $\mathcal{T}_\mathcal{P}$:

$$\mathcal{T}_\mathcal{P} = \langle \langle s_0, \rho_1^{tr}, s_1 \rangle, \langle s_1, \rho_2^{tr}, s_2 \rangle, ... \langle s_{n-1}, \rho_n^{tr}, s_n \rangle \rangle, n \geq 0$$

for $fp(GP)$ where: $\{s_0, ..., s_n\} \subseteq S$, $s_0 = I$ and $s_n \models G$. Also by Definition 4.4, a transition relation for a *fixed domain planning problem* would be:

$$T = \{(s, \rho^{tr}, s') \mid \{s, s'\} \subseteq\in S \wedge \rho^{tr} \in \Gamma^{act} \wedge s' = M_{\rho^{tr}}(s)\}.$$

Combining those two definitions, it is easily seen that the above algorithm is correct. For each transaction in $\mathcal{P}$ and corresponding transition relation in the trajectory, the algorithm will compute the necessary checks in the *while loop*. Therefore, if $\mathcal{T}_\mathcal{P}$, which leads to a goal state exists, it means that there exists a sequence of transactions s.t. the algorithm *FDPE* reaches a state s s.t. $s \models G$ before reaching the limit of the counter. Hence, there exists a run of the algorithm, which returns **true**.

'$\Leftarrow$' Assume *FDPE* algorithm has a run over $fp(GP)$. One case would be when $I \models G$ which corresponds to a plan of length 0. If not, it follows that for a *count* $\leq 2^{S(GP)}$, after non-deterministically selecting transactions $\rho_1^{tr}, ..., \rho_n^{tr}$, the algorithm obtains states $I, s_1, ..., s_n$, reaches a state $s_n \models G$ and returns **true**. It is easily seen that this sequence of transactions is a plan for $fp(GP)$, whose trajectory starts from the initial database and each time the algorithm goes through the *while loop* a new transition in the trajectory is created until it reaches s_n which is a goal state. □

5.1.3 Complexity of FDPE

In this section, we will show that deciding *plan-existence* for a *fixed domain planning problem* is feasible in *polynomial space*.

Lemma 5.1. *Let $GP = (I, Act, G)$ be a* GDPP. *Deciding whether $fp(GP)$ has a plan, is in* **PSPACE**.

Proof. We just need to prove that the *FDPE* algorithm runs in *non-deterministic polynomial space*. This is the case, because a sequence of transactions can be non-deterministically chosen, and the size of a state is bounded by $S(GP) = |R_n| \times |I_n|^2 + |C_n| \times |I_n|$. That is, a state is bounded by the number of all possible literals that can be constructed from all possible combinations of role names and concept names with constants from $dom(GP)$. Also, the length of a

plan is bounded by $2^{S(GP)}$ (see Theorem 5.1). Hence, no more than $2^{S(GP)}$ non-deterministic choices are required. The algorithm requires non-deterministic polynomial space, because it chooses non-deterministically the transactions and at each step, the algorithm stores in memory only the current state, the goal formula, the counter and the current transaction. It does not store in memory the whole plan $\mathcal{P}$, but the plan can be extracted by the transactions stored at each step. Furthermore, all the checks it needs to compute throughout the algorithm do not exceed this bound. In particular, it checks if $s \models G$, where G is an $\mathcal{ALCHOIQ}br$ formula (hence, a First Order Logic formula). This check can be done in *polynomial space*. (Model Checking is **PSPACE**-*complete*, see [15]). Hence, this algorithm requires a polynomial amount of memory in the size of the input. Therefore, this is a non-deterministic algorithm which runs in *non-deterministic polynomial space*.

Using Savitch's theorem, **NPSPACE = PSPACE** (cf. [42]). Therefore, determining plan existence for a given planning instance using the above algorithm is in **PSPACE**. □

5.2 PSPACE-Hardness

We have proved that *deciding plan existence* for a *fixed domain planning problem* is in **PSPACE**. In this section we will prove that it is also **PSPACE**-*hard*.

We will prove the **PSPACE**-*hardness* by a *non-deterministic Turing machine* simulation. Let $\mathcal{L}$ be a language in **PSPACE** and let T be a *deterministic Turing machine* (DTM) $T = (Q, \Sigma, s_0, \delta)$ which decides whether a given finite word w is in $\mathcal{L}$ within space polynomial in w. We assume $|w| = m$ and w.l.o.g. we assume that the *R/W*-head does not move outside of the mth position. Now, we must construct a *GDPP* $GP_T = (I, Act, G)$ s.t. T accepts w if and only if $fp(GP_T)$ has a plan. Before we introduce the encoding, we will first state the symbols that we will be using, providing some intuition for each of them:

- There is a new concept name A_i^σ, $1 \leq i \leq m$, each of which encodes the symbol σ at position i.
- There is a new concept name B^{s_k}, $s_k \in Q$ for each state of the DTM T.
- There is a new concept name Pos_i, $1 \leq i \leq m$ for each position of the *R/W*-head.

5.2.1 DTM Encoding to Fixed Domain Planning Problem

Based on these symbols, we now define the encoding of T to a *fixed domain planning problem* as follows:

- The initial database encodes the initial configuration $(s_0, \sqcup, w_1 ... w_m)$ of T:

$$I = \{A_1^{\sigma_1}(c), ..., A_m^{\sigma_m}(c), Pos_1(c), B^{s_0}(c)\},$$

  where each $A_i^{\sigma_i}(c)$, for $1 \leq i \leq m$ encodes the symbol σ_i at position i of the word w s.t. $\sigma_1 ... \sigma_m = w$, $Pos_1(c)$ encodes the initial position of the *R/W*-head and $B^{s_0}(c)$ encodes the initial state.

- The goal formula is given as follows:

$$G = B^{s_{accept}}(c)$$

  where $B^{s_{accept}}(c)$ encodes the final (accepted) state.

- The set of predefined transactions Act is given by a unique transaction as follows:

$$\rho^{tr} = \alpha_1 \circ ... \circ \alpha_m$$

  - where each α_i, $1 \leq i \leq m$, encodes all possible configurations at position i as follows:

$$\alpha_i = \beta^i_{\sigma_1,s_1} \circ ... \circ \beta^i_{\sigma_m,s_n},$$

    where $\{(\sigma_1, s_1), ..., (\sigma_m, s_n)\} = \Sigma \times Q$.
  - Each $\beta^i_{\sigma_j,s_k}$ is a conditional action defined as follows:

$$\beta^i_{\sigma_j,s_k} = c : A_i^{\sigma_j} \wedge c : B^{s_k}?$$

$$(B^{s_k} \ominus \{c\}) \circ (B^{s_l} \oplus \{c\}) \circ (A_i^{\sigma_j} \ominus \{c\}) \circ (A_i^{\sigma_f} \oplus \{c\}) \circ (Pos_i \ominus \{c\}) \circ (Pos_{(i+d)} \oplus \{c\}) : \epsilon.$$

    where $\beta^i_{\sigma_j,s_k}$ encodes the transition function $\delta(\sigma_j, s_k) = (\sigma_f, s_l, d)$ at position i, for $\{s_k, s_l\} \subseteq Q$, $d \in \{-1, 0, +1\}$ and σ_j, σ_f are symbols of w.

    The intuition of an application of a transaction $\beta^i_{\sigma_j,s_k}$ is the following: if the *R/W*-head at position i is reading the symbol σ_j of the word w and is at state $s_k \in Q$ then delete from the current state $B^{s_k}(c)$, $A_i^{\sigma_j}$ and $Pos_i(c)$. In turn, add $B^{s_l}(c)$, $A_i^{\sigma_l}(c)$ and $Pos_{(i+d)}(c)$ to the new state.

It is easy to see that the encoding is feasible in *polynomial* time, $\mathcal{O}(n \times m^2)$, where n is the number of states and m is the length of w.

5.2.2 Correctness of the Encoding

We are now prepared to show the correctness of the above encoding through the following theorem:

Theorem 5.3. *For every DTM T, one can compute in* polynomial *time a $GP_T = (I, Act, G)$ such that: T accepts w if and only if $fp(GP_T)$ has a plan.*

Proof. We prove both directions separately:

'$\Rightarrow$' Assume T accepts w. Then, by Definition 2.16, there is a finite sequence of configurations $C_0, C_1, ...C_n$ of T s.t. the initial configuration is $C_0 = (s_0, \sqcup, w)$, for each $i > 0$, C_{i-1} yields C_i and $C_n = (s_{accept}, w', u')$ is the last element of the sequence (accepting state). By construction, it is easily seen that ρ^{tr} encodes all possible configurations for a given w. Furthermore, for each transition between these configurations there is exactly one transaction in ρ^{tr} that satisfies

the precondition $c : A_i^{\sigma_j} \wedge c : B^{s_k}$ and computes the necessary changes to the states of the trajectory. For each configuration $C_i = (s, w_0...w_n, w_{n+1}...w_m)$, $C_i \in \{C_0, ..., C_n\}$ the corresponding state in the trajectory is $I_i = \{A_1^{\sigma_1}(c), ..., A_m^{\sigma_m}(c), Pos_{n+1}(c), B^s(c)\}$. Hence, a plan for $fp(GP_T)$ would be $\mathcal{P} = \rho^{tr} \circ_1 ... \circ_n \rho^{tr}$ and the trajectory is

$$\mathcal{T}_{\mathcal{P}} = \langle I, \rho^{tr}, I_1\rangle, \langle I_1, \rho^{tr}, I_2\rangle, ..., \langle I_{n-1}, \rho^{tr}, I_n\rangle,$$

where I is the initial database, representing the initial state, $\{I, I_1, ..., I_n\} \in S$. It is easy to see that the last repetition of the transaction ρ^{tr} produces a state I_n that adds $B^{s_{accept}}(c)$ to $I_n \in S$.

'$\Leftarrow$' Assume there is a plan $\mathcal{P}$ for $fp(GP_T)$. Hence, the trajectory over $\mathcal{P}$ has the following form: $\mathcal{T}_{\mathcal{P}} = \langle I, \rho^{tr}, I_1\rangle, \langle I_1, \rho^{tr}, I_2\rangle, ..., \langle I_{n-1}, \rho^{tr}, I_n\rangle$, where $I_n \in S$ is a goal state for $fp(I, Act, B^{s_{accept}}(c))$. It follows that $B^{s_{accept}}(c) \in I_n$. By construction, it is easily seen that the initial database corresponds to the initial configuration $C_0 = (s_0, \sqcup, w_0...w_n)$. Furthermore, there is exactly one transaction $\beta^1_{\sigma_1,s_0}$ occurring in ρ^{tr} whose precondition satisfies the initial state corresponding to the initial configuration of T. The idea is that each state will satisfy exactly one conditional transaction as well. It corresponds to exactly one transition which yields a new configuration. Hence, every state produced in the trajectory corresponds to exactly the respective configuration in the sequence. The sequence is as follows:

$$C_0, C_1, ..., C_n,$$

where C_0 corresponds to the initial state, C_i to I_i, $1 \leq i \leq n$. Using the same idea, the last repetition of ρ^{tr} which adds $B^{s_{accept}}(c)$ to I_n, would yield the configuration $C_n = (s_{accept}, w, \sqcup)$ where s_{accept} is the accepting state.

□

From the above reduction from *DTM*, one can notice that the goal formula is the most basic one (consists only of one positive fact). Hence, **PSPACE**-*hardness* is caused by the complex action allowed as input. This is a very nice result since later in the thesis, we will show **NP**-*hardness* for the case where actions are atomic with only concept names and role names and the goal formula is the typical conjunction of arbitrary positive and negative facts.

Lemma 5.2. *Deciding whether a $fp(GP)$ has a plan, is* **PSPACE**-*hard.*

In the previous sections we introduced the *FPDE* algorithm 5.1.2 for checking *plan-existence* for a *fixed domain planning problem* and we showed that it runs in **PSPACE** in the Lemma 5.1. By a **PSPACE**-*complete deterministic Turing machine* simulation in Section 5.2, we also showed the **PSPACE**-*hardness*. Thus, now we can formally define the following theorem:

Theorem 5.4. *Let $GP = (I, Act, G)$ be a GDPP. Deciding whether a $fp(GP)$ has a plan, is* ***PSPACE***-complete.

5.3 Fixed Domain Plan Existence with Syntactic Restrictions

We have shown above that *fixed domain plan-existence* is PSPACE-complete. Applying severe restrictions on the type of goal formulas and transactions allowed as input, one might extract different cases to be analyzed, which offer better complexity results.

We will start by analyzing plan-existence over a *fixed domain planning problem* with atomic actions that consist only of concept names. Unfortunately, for this basic case, we will prove by reduction from *3-colorability*, that deciding plan existence is already intractable. Further, we are interested in identifying the largest setting, whose plan-existence can be decided in polynomial time. In addition, we identify the cases that 'cause' the above problem to be NP-hard. We will increase the syntactic restrictions on the type of goal formulas allowed as input, and we try to define a fine line between tractable and intractable planning for these cases.

To get a better intuition about the relation between the complexity and the parameters of the planning problem, we initiate our search by investigating *the case of positive goals*, where the goal formula is a conjunction of only positive atoms. We will show that for this interesting case, for every plan, there exists a plan with only positive actions. The intuition is that for the setting we will consider, where no universal restriction is allowed neither in the actions, neither in the goal formula, the goal states are always a superset of the initial database. Since there is no action that performs deletion, the states monotonically grow throughout each trajectory that can be constructed. This is a very nice property which ensures nice complexity results. Indeed, we will prove that it is in **Nlogspace**.

Next, attempting to find the 'causes' of the NP-hardness, we start by considering the case where goal formulas are conjunctions of only negative literals. We call it *the case of negative goals*. One would expect the same behavior in the opposite direction as for the positive case, meaning that a plan for this type of goal formulas would require only actions with negation. But surprisingly, some of the cases are even reduced to the positive one.

We continue by considering several possible combinations of positive and negative facts in the goal formula. We provide algorithms for checking the existence of a plan for some of these cases. We distinguish some specific types of goal formulas, that seem to be the triggers that increase the complexity to NP.

Note: For the rest of the thesis, for a GDPP $GP = (I, Act, G)$, instead of viewing the goal formula G as a general $\mathcal{ALCHOIQ}br$-formulae (see Definition 4.1 and Section 3.1.1), we restrict it by only considering G as a conjunction of positive and negative literals (as usual literals are represented by ground concept names and role names in DL-jargon). Also, we call a *positive goal*, a goal formula which is a conjunction of only positive literals and a *negative goal* a goal formula which is a conjunction of only negative literals.

5.3.1 NP-Hardness for the Case of Atomic Actions with Concept Names

Let $GP = (I, Act, G)$ be a GDPP, where Act is a set of atomic actions with only concept names. We prove that *deciding plan existence* for $fp(GP)$ under these conditions is already **NP**-*hard*. The proof is done by reduction from *3-colorability*, which is known to be an **NP**-*complete* problem.

We start by giving a definition for the problem of 3-*colorability* of graphs.

Definition 5.1. ***(3-Colorability of Graphs)***

Instance: *A graph $\mathcal{G} = (V, E)$, where V is a set of* vertices *and E is a set of* edges*.*

Question: *Can $\mathcal{G}$ be colored with the three colors $\{r, g, b\}$ in such a way that two adjacent vertices have a distinct color?*

The intuition is the following: given a graph and three colors *red*, *green, blue*, the idea is to check if it is possible to color this graph in such a way that each two neighbor vertices that are connected by an edge have different colors. If this is the case, the graph is said to be 3-*colorable*.

Below, we built the reduction, but before we list the concept names that we will use along with their intuitive meaning.

- There are new concept names $A \in N_C$, for each vertice of the graph G. Each of these concept names is expected to simulate the color of the corresponding vertice in the 3-*colored* graph.
- There are three new concept names $F_1, F_2, F_3 \in N_C$. Each of them will encode exactly two different colors and an auxiliary constant f. They will be used as auxiliary atoms so that each edge is forced to have exactly two different colors and that the vertices, expressed via concept name A do not have more than one color.
- For each edge $e_{ij} = (v_i, v_j)$ of the graph G there will be a new concept name E_{ij}.

5.3.1.1 Reduction from 3-Colorability of Graphs

We, now can introduce the reduction. Assume a graph $\mathcal{G} = (V, E)$, where :

- $V = \{v_1, ..., v_n\}$ and
- $E = \{e_{ij} = (v_i, v_j) \mid e_{ij} \in E\}$

We construct an instance of $GP_{3col} = (I, Act, G)$ as follows:

- The initial database is given as follows:

$$\begin{aligned} I = \{&A_1(r), A_1(b), A_1(g), A_1(f), \\ &..., \\ &A_n(r), A_n(b), A_n(g), A_n(f), \\ &F_1(r), F_1(b), F_1(f), \\ &F_2(b), F_2(g), F_2(f), \\ &F_3(r), F_3(g), F_3(f)\}, \end{aligned}$$

  where $r, b, g \in N_I$ are new constants for *red, blue, green*, respectively. Each $A_i \in \{A_1, ..., A_n\}$ has four atoms of the form $A_i(r), A_i(b), A_i(g), A_i(f)$ where each of the first

three encode the three possibilities of coloring of vertices v_i and the last is an auxiliary atom. Also, as stated earlier, each of the three concept names F_l, $1 \leq l \leq 3$, represents the different possibilities of coloring of the edges.

I contains $(4n + 9)$ atoms, where n is the number of vertices in G.

- The set of predefined actions for each edge $e_{ij} \in E$ is given as follows:

$$Act = \{\rho_1^1, \rho_1^2, \rho_1^3, ..., \rho_n^1, \rho_n^2, \rho_n^3\} \cup \bigcup_{e_{ij} \in E} Act_{ij},$$

where:
$\rho_1^1 = A_1 \ominus F_1, \rho_1^2 = A_1 \ominus F_2, \rho_1^3 = A_1 \ominus F_3,$
...,
$\rho_n^1 = A_n \ominus F_1, \rho_n^2 = A_n \ominus F_2, \rho_n^3 = A_n \ominus F_3,$

and each Act_{ij} is as follows:

$$Act_{ij} = \{\rho_{ij}^1, \rho_{ij}^2, \rho_{ij}^3, \rho_{ij}^4, \rho_{ij}^5\},$$

where:

$$\begin{aligned}
\rho_{ij}^1 &= E_{ij} \oplus F_1, \\
\rho_{ij}^2 &= E_{ij} \oplus F_2, \\
\rho_{ij}^3 &= E_{ij} \oplus F_3, \\
\rho_{ij}^4 &= E_{ij} \ominus A_i, \\
\rho_{ij}^5 &= E_{ij} \ominus A_j.
\end{aligned}$$

Each E_{ij} encodes the edge $e_{ij} \in E$. Each application of an action ρ_i^l, $1 \leq l \leq 3$, $1 \leq i \leq n$ at a state ensures a new state that contains at most one of the atoms $A_i(g), A_i(r), A_i(b)$. Furthermore, each application of an action ρ_{ij}^l, $1 \leq i, j \leq n$, $1 \leq l \leq 3$ at a state ensures that the new state obtained, contains at most two of the atoms $E_{ij}(r), E_{ij}(b), E_{ij}(g)$. The last two types of actions are necessary for checking that the edges contain the same colors as the respective vertices.

Hence, there are $3n$ actions for the n vertices and $5 \times m$ actions for the edges where m is the number of edges in the graph.

- The goal formula is given as follows:

$$G = \bigwedge_{k=1}^{n} \neg A_k(f) \wedge \bigwedge_{e_{ij} \in E} (E_{ij}(f) \wedge \neg E_{ij}(r) \wedge \neg E_{ij}(b) \wedge \neg E_{ij}(g)).$$

So that the goal formula can be satisfied, it requires a sequence of actions, s.t. the trajectory over these sequence reaches a state $s \in S$ where each $E_{ij}(f) \in s$, for all concept names representing the vertices $A_k(f) \notin s$. Finally for all edges represented by E_{ij} it

should hold that $E_{ij}(r) \notin s$, $E_{ij}(b) \notin s$, $E_{ij}(g) \notin s$. It is easy to see that to satisfy the goal, a sequence of actions to be a plan should contain for each $1 \leq k \leq n$ an action $A_k \ominus F_l$ for some $l \in \{1, 2, 3\}$, for each $e_{ij} \in E$ an action $(E_{ij} \oplus F_l)$, $l \in \{1, 2, 3\}$ and we will also prove that it needs actions $(E_{ij} \ominus A_i)$ and $(E_{ij} \ominus A_j)$.

The goal formula contains $(n + 4m)$ atoms.

It is easy to see that the reduction can be built in linear time, $\mathcal{O}(8n+9m+9)$, where n is the number of vertices and m is the number of edges in $\mathcal{G}$. Next, we need to prove that this reduction from 3-*colorability* of graphs to an instance of *fixed domain planning problem* is correct.

5.3.1.2 Correctness of Reduction

We introduce the following theorem that states that if there is a 3-*colorable* graph, then there exists a plan for the corresponding $fp(GP_{3col})$. The other direction also holds. More formally:

Theorem 5.5. *For every finite graph G one can compute in linear time a graph database planning problem GP_{3col} such that:*

G *is 3-colorable if and only if there exists a plan for* $fp(GP_{3col})$.

Proof. We prove both directions as follows:

'$\Longrightarrow$' Assume a graph $\mathcal{G} = (V, E)$ with n vertices and m edges and let G be *3-colorable*. We need to prove that $fp(GP_{3col})$ has a plan.

Since $\mathcal{G}$ is *3-colorable*, it means that each of the vertices of the graph can be labeled by one of the colors *red, blue, green* s.t. every edge of the graph is labeled by two different colors. We construct three sequences of actions for $fp(GP_{3col})$ as follows.

1. For each $e_{ij} \in E$, $1 \leq i, j \leq m$, there is exactly one action of form $\rho^l_{ij} = (E_{ij} \oplus F_l)$, $1 \leq l \leq 3$, s.t. if e_{ij} is labeled with the colors $\{x, y\} \subset \{r, b, g\}$ then $F_l(x) \in I$ and $F_l(y) \in I$. Intuitively, since the initial database contains for each $l \in \{1, 2, 3\}$ exactly three atoms, one of which is $F_l(f)$ and the two others1 represent two different colors, we add only atoms E_{ij} with the constants occurring in F_l that correspond to the colors of the edge e_{ij}. Let s' be this sequence.

2. For each $A'_k = v_k \in V$, $1 \leq k \leq n$, there is exactly one action of the form $\rho^l_k = A_k \ominus F_l$ for $1 \leq l \leq 3$, such that if A'_k is labeled with color $x \in \{r, b, g\}$ then $F_l(x) \notin I$. Since the initial database contains for each A_k, $1 \leq k \leq n$ contains atoms for each color and an atom $A_k(f)$, the intuition is that we for each A_k we chose an action that leaves exactly the atom that has the constant corresponding to the color of the node $v_k \in V$. This is done by deleting the F_l that has the constants that correspond to the other two colors. Let s'' be this sequence.

3. For each edge $e_{ij} \in E$ there are two actions of the form $\rho^1_{ij} = E_{ij} \ominus A_i$ and $\rho^2_{ij} = E_{ij} \ominus A_j$. We populated the atoms E_{ij} with constants corresponding to the colors of nodes of the corresponding edges in the graph and the same with the atoms A_k. Hence, these two

actions will produce a state where for each E_{ij} there will be just one atom $E_{ij}(f)$. Let s''' be this sequence.

Now, it just remains to prove that applying 1, 2, 3 in a sequence results in a plan $\mathcal{P}$ for $fp(GP_{3col})$. Thus we claim that:

$$\mathcal{P} = \langle s's''s''' \rangle$$

is a plan for $fp(GP_{3col})$. The proof is almost straightforward. The last state $s_n \in S$ of the trajectory $\mathcal{T}_\mathcal{P}$ has the following properties:

- By s', for each E_{ij} that corresponds to an edge $e_{ij} \in E$, $E_{ij}(f) \in s_n$.
- By s'', for each A_k that corresponds to a node $v_k \in V$, $A_k(f) \notin s_n$.
- By 1 and 2, the states will be updated such that the colors of the atoms of E_{ij} will be compatible to the colors of A_i and A_j. Hence applying the sequence s''' of actions, enforces that for each E_{ij} the following holds: $E_{ij}(r) \notin s_n$, $E_{ij}(b) \notin s_n$, $E_{ij}(g) \notin s_n$. By 2 no $A_k(f) \in s_n$, hence $E_{ij}(f) \in s_n$ for each E_{ij}.

Hence, $\mathcal{P}$ is indeed a plan for $fp(GP_{3col})$.

'$\Longleftarrow$' Assume $\mathcal{P}$ is a plan for $fp(GP_{3col})$. We need to prove that G is 3-*colorable*.

By assumption there exists a plan $\mathcal{P}$ s.t. $\mathcal{T}_\mathcal{P}$ is a trajectory that reaches a goal state $s_n \in S$, i.e. $s_n \models G$.

We claim that for each A_k corresponding to a vertice in G, s_n contains exactly one atom $A_k(x)$, $x \in \{r, g, b\}$, where x corresponds to the color of the respective vertice in $\mathcal{G}$, s.t. each edge of $\mathcal{G}$ is labeled by two different colors.

We prove the claim considering the properties the trajectory $\mathcal{T}_\mathcal{P}$ should have so that a final goal state $s_n \models G$ is achieved.

1. For each E_{ij} s.t. $e_{ij} \in E$, it holds that $E_{ij}(f) \in s_n$. It follows that in $\mathcal{P}$ there are actions of the form $(E_{ij} \oplus F_l)$ for each E_{ij} . Hence, for each of them at least 2 different atoms of the form $E_{ij}(x)$, $E_{ij}(y)$ (in addition to adding $E_{ij}(f)$), where $x, y \in \{r, b, g\}$ are added by these actions to some state in the trajectory. Hence, at this point, each edge in the graph has at least two colors.

2. For each A_k, s.t. $v_k \in V$, it holds that $A_k(f) \notin s_n$. It follows that, for each of them, there are actions in $\mathcal{P}$ of the form $(A_k \ominus F_l)$. Hence, there remains at most 1 atom of the form $A_k(x)$, $x \in \{r, b, g\}$. At this point, each vertice in the graph has at most one color.

3. For each E_{ij}, s.t. $e_{ij} \in E$, it should hold that $E_{ij}(r) \notin s_n$, $E_{ij}(b) \notin s_n$, $E_{ij}(g) \notin s_n$. The latter, and by 1 and 2 (of this page) it is easily seen the following.

    - For each E_{ij}, exactly one action $(E_{ij} \oplus F_l)$, for some $F_l \in \{F_1, F_2, F_3\}$ and for each A_k exactly one action $(A_k \ominus F_l)$, for some $F_l \in \{F_1, F_2, F_3\}$ should occur in $\mathcal{T}_\mathcal{P}$. We show this claim by contradiction.

– To show the first, assume for some E_{ij} actions $(E_{ij} \oplus F_{l'})$ and $(E_{ij} \oplus F_{l''})$, $\{F_{l'}, F_{l''}\} \subseteq \{F_1, F_2, F_3\}$ occur in $\mathcal{T}_\mathcal{P}$. Hence, there is a state $s' \in S$ s.t. $\{E_{ij}(r), E_{ij}(g), E_{ij}(b), E_{ij}(f)\} \subseteq s'$. In this case, so that a goal state is reached, in $\mathcal{T}_\mathcal{P}$ should be actions that delete these three atoms. The only actions responsible for deleting from E_{ij} are $(E_{ij} \ominus A_i)$ and $(E_{ij} \ominus A_j)$. To preserve that $E_{ij}(f) \in s_n$, some action $(A_i \ominus F_l)$ and $(A_j \ominus F_l)$ should occur. But then, by 2 at most one atom will be remained for each of them. It follows, that applying $(E_{ij} \ominus A_i)$ and $(E_{ij} \ominus A_j)$ there is no possibility to delete the three desired atoms. Hence, exactly one action $(E_{ij} \oplus F_l)$ should be applied for each E_{ij}, so that exactly two atoms representing two different colors are added.

– Next, we claim that for each A_k, there is exactly one action $(A_k \ominus F_l)$ for some $F_l \in \{F_1, F_2, F_3\}$ occurs in $\mathcal{T}_\mathcal{P}$. Considering that for each E_{ij}, exactly one action $(E_{ij} \oplus F_l)$ exists in $\mathcal{T}_\mathcal{P}$, using the same argument as in previous case, it is easy to see that so that a state that satisfies the goal formula is reached, the claim should hold.

Thus, until now we have proved that there is a state s'_n in $\mathcal{T}_\mathcal{P}$, where for each A_k there is exactly one atom $A_k(z)$, where $z \in \{r, b, g\}$. Also for each E_{ij}, s'_n contains exactly three atoms. One of these three atoms is $E_{ij}(f)$ and the two others are $E_{ij}(x)$, $E_{ij}(y)$, s.t. $\{x, y\} \subset \{(r, b), (b, g), (r, g)\}$.

- Next, the goal state of $\mathcal{T}_\mathcal{P}$ should be s.t. $E_{ij}(r) \notin s_n$, $E_{ij}(b) \notin s_n$, $E_{ij}(g) \notin s_n$. The only actions in Act responsible for deleting atoms E_{ij} from a state are $(E_{ij} \ominus A_i)$ and $(E_{ij} \ominus A_j)$. So that s_n is reached, s'_n should contain exactly $A_i(x)$ and $A_j(y)$ for $E_{ij}(x)$, $E_{ij}(y)$, for $\{x, y\} \subseteq \{r, b, , g\}$. Thus s_n will contain atoms $A_k(z)$ for each vertice of the graph with colors s.t. the corresponding edge in the graph is labeled by two different colors.

Thus the claim is shown.

□

It is known that the *3-colorability* problem is an **NP**-*complete* problem. Thus, we can introduce the following theorem.

Theorem 5.6. *Deciding whether there exists a plan for $fp(I, Act, G)$, where Act is a set of atomic actions with only concept names, is* **NP**-hard.

Next, we analyze different subcases, by restricting the syntax of the goal formula.

5.3.2 The Case of Positive Goals

We start by considering a graph database planning problem $GP = (I, Act, G)$, where Act is a set of basic actions and G is a conjunction of positive atoms expressed via $\mathcal{ALCHOIQ}br$ basic concepts and roles. We do not put any restrictions on the initial database.

In this section, we will prove that for each plan of $fp(GP)$, there exists a plan that contains no negative actions. Thus deleting all the negative actions from a plan $\mathcal{P}$, it will still be a plan

for $fp(GP)$. The latter entails that only positive actions can be responsible for the necessary changes to the states of the trajectory $\mathcal{T}_\mathcal{P}$ so that a goal state is reached. Indeed, this is a very nice property which ensures nice complexity results. The intuition behind this interesting case is that considering only positive actions, states produced after each step will monotonically grow. Thus, for each possible trajectory, the states obtained after each application of an action, are a superset of the previous one. To get a better intuition, we give an example to illustrate this case:

Example 5.2. *We consider the usual example with the project database of some research institute. Assume the following graph database planning problem* $GP = (I, Act, G)$.

$$\begin{aligned} I = \{&ActiveProject(P20840),\\ &Project(P20840),,\\ &ConcludedProject(),\\ &Employee(E01), Employee(E03),\\ &ProjectEmployee(E01),\\ &PermanentEmployee(E04),\\ &worksFor(E01, P20840)\}. \end{aligned}$$

$$Act = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5, \rho_6, \rho_7\},$$

where:

$$\begin{aligned} \rho_1 &= ActiveProject \oplus Project,\\ \rho_2 &= ActiveProject \ominus Project,\\ \rho_3 &= Project \oplus X,\\ \rho_4 &= PermanentEmployee \oplus ProjectEmployee,\\ \rho_5 &= PermanentEmployee \ominus ProjectEmployee,\\ \rho_6 &= worksFor \oplus (X, Y),\\ \rho_7 &= worksFor \ominus (X, Y). \end{aligned}$$

$$G = ActiveProject(P24090) \wedge PermanentEmployee(E01) \wedge worksFor(E03, P24090).$$

In this example, Act *consists of a set of seven predefined basic actions and the goal* G *is a conjunction of three positive literals, which intends the addition of some new project* $P24090$ *to ActiveProject,* $E03$ *to work for this specific project and lastly, it requires* $E01$ *to be a PermanentEmployee. A state that satisfies the goal* $s \in S$ *should be such that* $\{$*ActiveProject(*$P24090$*), PermanentEmployee(*$E01$*),* $worksFor(E03, P24090)\} \subseteq s$. *It is easily seen that, by instantiating* X *in* ρ_3 *with* $P24090$ *and* (X, Y) *in* ρ_7 *with* $(E03, P24090)$, *a plan* $\mathcal{P}$ *for* $fp(GP)$ *can be the following:*

$$\mathcal{P} = \langle (Project \oplus P24090), (ActiveProject \oplus Project),$$

$$(PermanentEmployee \oplus ProjectEmployee), (worksFor \oplus (E03, P24090))\rangle.$$

Now, let's consider the following sequence:

$$\mathcal{P}' = \langle (Project \oplus P24090), (ActiveProject \ominus Project), (ActiveProject \oplus Project),$$

$$(PermanentEmployee \oplus ProjectEmployee), (worksFor \oplus (E03, P24090))\rangle.$$

$\mathcal{P}'$ is still a plan for the above instance, since ρ_2 is applied before ρ_1. Thus considering the first three state transitions of the trajectory $\mathcal{T}_{\mathcal{P}'}$,

$$\langle I, (Project \oplus P24090), s_1\rangle\langle s_1, (ActiveProject \ominus Project), s_2\rangle$$

$$\langle s_2, (ActiveProject \oplus Project), s_3\rangle,$$

it is easily seen that $s_2 \not\models ActiveProject(P24090)$, but $s_3 \models ActiveProject(P24090)$. Furthermore, no action with deletion is there further in the sequence, hence the last state of the trajectory models $ActiveProject(P24090)$ as well. So ρ_2 can be deleted from the sequence and the sequence will still be a plan. But, if ρ_2 is performed after ρ_1, then $\mathcal{P}'$ will not be a plan anymore. Hence, every negative action that can be added to this sequence either is redundant, since it would only delete facts from some state, or it will cause the sequence not to be a plan anymore. Thus, each of the actions ρ_2, ρ_5 and ρ_7 can be forgotten and not considered for every goal formula with only positive literals for the above given I and Act.

Hence, for every plan of a given $fp(GP)$ with a positive goal formula, there exists a plan that contains only positive actions. Surely, some of them might still be redundant, but intuitively no positive actions can harm the property of being a plan, for some sequence of actions. Indeed, applying to a state an action that performs addition, only produces a state, which is a superset of the state it was applied to. It follows that, the cardinality of the states after each application of a positive action only grows (or stays the same). We will prove that these nice properties of *fixed domain planning problems* with positive goals, ensure nice complexity results when deciding *plan-existence*.

To formally define this intuition, we introduce the following theorem and lemma.

Theorem 5.7. *Let $GP = (I, Act, G)$ be a* GDPP*, where G is a conjunction of positive literals, Act is a set of basic actions. For every plan $\mathcal{P}$ of $fp(GP)$, there exist a plan $\mathcal{P}'$ of $fp(GP)$ with only positive actions.*

To prove this theorem it is sufficient to prove the following lemma:

Lemma 5.7.1. *If $\mathcal{P}$ is a plan for $fp(GP)$ with k negative actions, then there exists a plan $\mathcal{P}'$ for $fp(GP)$ with $(k-1)$ negative actions.*

Proof. Assume $\mathcal{P}$ is a plan of $fp(GP)$ which contains k negative actions. By Definition 2.5, there exists a finite sequence of actions $\langle\rho_1, ..., \rho_n\rangle$ where $\{\rho_1, ..., \rho_n\} \subseteq Act$, $n \geq 1$ s.t. for some $\{I, s_1, ..., s_n\} \subseteq S$, there exists a trajectory $\mathcal{T}_{\mathcal{P}}$:

$$\mathcal{T}_{\mathcal{P}} = \langle\langle I, \rho_1, s_1\rangle, \langle s_1, \rho_2, s_2\rangle, ...\langle s_{n-1}, \rho_n, s_n\rangle\rangle, n \geq 0$$

where I is the initial database, and $s_n \models G$. W.l.o.g assume that the last negative action appears at position $l \leq n$ and has the form $\rho_l = (A \ominus C) \in Act$, where A is a concept name or a role name and C is an $\mathcal{ALCHOIQ}br$ basic concept or an $\mathcal{ALCHOIQ}br$ role. Let the state transition at position l be $t_l = \langle s_{l-1}, (A \ominus C), s_l\rangle$ s.t. $s_l = M_{(A\ominus C)}(s_{l-1})$ (see Definition 4.4).

By Definition 3.7, $M_{(A \ominus C)}(s_{l-1}) = s_{l-1} \setminus \{A(c) \mid s_{l-1} \models C(c)\}$. Hence, $s_l \subseteq s_{l-1}$. Since C can be a basic concept or an $\mathcal{ALCHOIQ}br$ role, it is easily seen that, if $s_{l-1} \not\models G$ then $s_l \not\models G$ as well. It follows, that eliminating the action ρ_l from the plan, does not have any impact on the plan.

Note: For a more extended proof see the proof of Lemma 6.6. □

Next, we will investigate the complexity of the positive case. To get a better intuition about its complexity, we start by analyzing the complexity of the least expressive subcase by enforcing even more severe restrictions on the goal formula, where in the beginning we will consider a goal formula with a single positive literal. By a reduction to *reachability in graphs*, we will show that deciding *plan-existence* for this case can be done in **Nlogspace**. Further, we view the goal formula as a conjunction of two or more positive literals and analyze its complexity.

Note: In the following we consider only basic actions and the goal formula as a conjunction of literals expressed via $\mathcal{ALCHOIQ}br$ basic concepts and $\mathcal{ALCHOIQ}br$ roles. Furthermore, w.l.o.g. due to Theorem 5.7, for cases with goals with only positive literals, we will consider only plans with positive actions.

5.3.2.1 Encoding to Reachability in Graphs

Now, we consider again a GDPP $GP = (I, Act, G)$, where Act is a finite set of basic actions and G is a conjunction positive literals expressed via $\mathcal{ALCHOIQ}br$ basic concepts and $\mathcal{ALCHOIQ}br$ roles. We prove that deciding plan existence for $fp(GP)$, under this conditions, can be done in *non-deterministic logarithmic space*. We start by encoding the $fp(GP)$ planning problem for the case with G as a single positive literal to reachability in graphs.

For the case of the general positive goal formula of the form $C_1(c) \wedge ... \wedge C_n(c)$ where C is an $\mathcal{ALCHOIQ}br$ basic concept or role and c is a constant or a tuple, it is not hard to see that the complexity does not change. One can find a plan for each literal $C_j(c)$, $1 \leq j \leq n$, in a sequence, as if they were goals with a single literal, and then concatenate the plans. This would mean that after finding a plan for the first literal in the goal, the resulting goal state is an initial state for the plan of the second literal, and so on. This holds based on the fact that the trajectory of a plan for $fp(I, Act, C_1(c))$ results in a goal state $s \in S$ s.t. $s \models C_1$. Moreover, since w.l.o.g. we consider only plans with positive actions, $s \supseteq I$. Hence, if there exists a plan for $fp(I, Act, C_2(c))$, then there is a plan also for $fp(s, Act, C_2(c))$. The argument is the same for the case of a goal with more than two positive literals. Further in this section, we will formally prove this intuition.

Below we give the reduction for $fp(GP)$ where the goal is a ground atomic formula, but before we introduce the following lemma:

Lemma 5.3. *Let $GP = (I, Act, G)$ be a GDPP and $\mathcal{T}$ be a trajectory for $fp(GP)$ of the form $\mathcal{T} = \langle\langle s_0, act_1, s_1\rangle, \langle s_1, act_2, s_2\rangle, ...\langle s_{n-1}, act_n, s_n\rangle\rangle, n \geq 0$. If $\{act_1, ..., act_n\}$ are positive actions then $s_0 \subseteq s_1 \subseteq ... \subseteq s_n$.*

Proof. It is easily seen that if $\{act_1, ..., act_n\}$ are positive actions, thus they perform only additions, then a fact can never be deleted from some state. Hence, each application of one of these actions to a state cannot result in a state with smaller cardinality. □

Next, we will provide the encoding to reachability of graphs.
Assume a *GDPP* $GP = (I, Act, G)$, where Act is a set of basic actions and G is a single positive literal that has one of the following forms:

- $C(c)$, where C is a basic concept and $c \in N_I$,
- $R(c, d)$, where R is an $\mathcal{ALCHOIQ}br$ role and $\{c, d\} \subseteq N_I$.

Let $dom(G) \subseteq dom(GP)$ be the set of all constants that appear in G. Corresponding to $fp(GP)$ we construct a graph $g(GP) = (V, E)$, where V is a set of vertices and E is a set of edges defined as follows:

- $V = \{C(c) \mid C$ is a basic concept that occurs in I, Act or G, where $c \in dom(G)\} \cup \{R(c, d) \mid R$ is an $\mathcal{ALCHOIQ}br$ role that occurs in I, Act or G s.t. $\{c, d\} \subseteq dom(G) \cup \{b\}$, where $b \in dom(GP)\}$.
- $(v_1, v_2) \in E$ iff $v_1, v_2 \in V$ and one of the following holds:
  1. $v_1 = C(c), v_2 = A(c)$ and $(A \oplus C) \in Act$, where $A \in N_C$ and C is a concept name or a concept $\exists R$, or $\exists R^-$, where $R \in N_R$,
  2. $v_1 = t, v_2 = A(c)$ and $(A \oplus t) \in Act$ where $A \in N_C$ and $\{t\} \in N_v \cup \{c\}$,
  3. $v_1 = P(c, d), v_2 = R(c, d)$ and $(R \oplus P) \in Act$, where $R \in N_R$ and $P \in N_R$,
  4. $v_1 = (t_1, t_2), v_2 = R(c, d)$ and $(R \oplus (t_1, t_2)) \in Act$, where $R \in N_R$ and $\{t_1, t_2\} \subseteq N_V \cup \{c, d\}$,
  5. $v_1 = P(d, c), v_2 = R(c, d)$ and $(R \oplus P^-) \in Act$,
  6. $v_1 = R(c, d), v_2 = P^-(d, c)$ and $P = R$,
  7. $v_1 = R(c, d), v_2 = \exists P(c)$ and $P = R$,
  8. $v_1 = R(d, c), v_2 = \exists P^-(c)$ and $P = R$.

Note: W.l.o.g. we can assume that for any plan, its last actions is the first one that reaches the goal.

Next, for convenience, we introduce a new set of atoms as follows.

Definition 5.2. *Let Σ_V be a new set $\Sigma_V \subseteq V$ defined as follows:*

- *if $t \in V$ and t is a variable or a constant from $dom(G)$, then $(t) \in \Sigma_V$,*
- *if $(t_1, t_2) \in V$ is a tuple, and t_1, t_2 are variables or constants from $dom(G) \cup b$, where $b \in dom(GP)$, then $(t_1, t_2) \in \Sigma_V$, and*
- $I \subseteq \Sigma_V$.

Now, we are ready to introduce the following theorem.

Theorem 5.8. *Assume $GP = (I, Act, G)$, where Act is a finite set of basic actions and G is a single literal. $fp(GP)$ has a plan iff there exists a path from some L s.t. $L \in \Sigma_V$ to G.*

Proof. '$\Longleftarrow$'(*Soundness*) Let $P = \langle (v_1, v_2), (v_2, v_3), ..., (v_{n-1}, v_n) \rangle$ be a path from some $v_1 = L$ to $v_n = G$, where $L \in \Sigma_V$. We need to prove that there exists a plan for $fp(GP)$.

To prove this claim it suffices to prove the following lemma.

Lemma 5.8.1. *If P is a path from some $L \in \Sigma_V$ to G, then each $fp(I, Act, v_i)$ has a plan, for all v_i that occur in P.*

Proof. We prove the claim by induction on i.

Base case. $i = 1$. Thus, $v_1 = L \in \Sigma_V$. We distinguish between two cases:

1. L is a variable t, or a tuple of variables (t_1, t_2) or constants from $dom(G) \cup b$, where $b \in dom(GP)$. Then it is trivial that $fp(I, Act, L)$, viewing I as an interpretation, for every instantiation of variables with elements of $dom(G)$, has a plan of length 0.

2. $L \in I$ then $I \models L$, thus I is already a goal state for $fp(I, Act, L)$.

Induction Hypothesis. $\forall i, i \leq k$, $fp(I, Act, v_i)$ has a plan.

Induction Step. Now we prove the claim for $fp(I, Act, v_{k+1})$. Let (v_k, v_{k+1}) be the corresponding edge in the path P that leads to v_{k+1}. By hypothesis, there exists a plan for $fp(I, Act, v_k)$. Let $\mathcal{P}$ be this plan and $s \in S$ the final state of the trajectory $\mathcal{T}_{\mathcal{P}}$. Then, one of the following cases holds:

- $v_k = C(c)$ and $v_{k+1} = A(c)$ and $(A \oplus C) \in Act$, where C is a concept name or a concept $\exists R$, or $\exists R^-$, where $R \in N_R$. We claim that $\langle \mathcal{P}, (A \oplus C) \rangle$ is a plan for $fp(I, Act, A(c))$. The last state trajectory is $\langle s, (A \oplus C), s' \rangle$. By Definition 3.7, $s' = s \cup \{A(c) \mid s \models C(c)\}$. By the hypothesis, it is easy to see that $s' \models A(c)$.

- $v_k = P(c, d), v_{k+1} = R(c, d)$ and $(R \oplus P) \in Act$, where $R \in N_R$ and $P \in N_R$. Reasoning as in the previous case $\langle \mathcal{P}, (R \oplus P) \rangle$ is the required plan for $fp(I, Act, v_{k+1})$, .

- $v_k = t, v_{k+1} = A(c)$ and $(A \oplus t) \in Act$ where $A \in N_C$ and $\{t\} \in N_v \cup c$. It is easy to see that $A(c)$ can be reached immediately by substituting t with c or just adding $A(c)$ if $v_k = c$.

- $v_k = (t_1, t_2), v_{k+1} = R(c, d)$ and $(R \oplus (t_1, t_2)) \in Act$, where $R \in N_R$. This is identical to the previous case considering $\{t_1, t_2\} \subseteq N_V \cup \{c, d\}$.

- $v_k = P(d, c), v_{k+1} = R(c, d)$ and $(R \oplus P^-) \in Act$. By the hypothesis $fp(I, A, P(d, c))$ has plan $\mathcal{P}$ with final state $s \in S$. Applying $(R \oplus P^-)$ to s, a new state s' is obtained s.t. by Definition 3.7, $s' = s \cup \{R(c, d) \mid P(d, c) \in s\}$. Thus, $s' \models R(c, d)$. Hence $\langle \mathcal{P}, (R \oplus P^-) \rangle$ is a plan for $fp(I, Act, R(c, d))$.

- $v_k = R(c,d), v_{k+1} = P^-(d,c)$ and $P = R$. It is not hard to see that if $s \models R(d,c)$, then $s \models P^-(d,c)$, since $R = P$ and $(P^-(d,c))^{\mathcal{I}} = R(c,d)$. It follows that $\mathcal{P}$ is a plan for $fp(I, Act, R(c,d))$.

- $v_k = R(c,d), v_{k+1} = \exists P(c)$ and $P = R$. By the semantics of $\exists R(c)$ and by the hypothesis, it is easy to see that $\mathcal{P}$ is also a plan for $fp(I, Act, \exists P(c))$.

- $v_k = R(d,c), v_{k+1} = \exists P^-(c)$ and $P = R$. By the semantics of $\exists R^-(c)$ and the hypothesis, it is easy to see that $\mathcal{P}$ is also a plan for $fp(I, Act, \exists P^-(c))$.

□

Thus, by induction we showed that for all nodes of the path P, $fp(I, Act, v_i)$ has a plan. Since, $v_n = G$ is also a node of this path, it also holds that $fp(I, Act, G)$ has a plan.

'$\Longrightarrow$' (*Completeness*) Let $\mathcal{P}$ be a plan for $fp(I, Act, G)$. We need to prove that there exists a path from some $L \in \Sigma_V$ to G.
We prove the claim by induction on the length l of a plan. W.l.o.g we assume that the last action of the plan reaches the goal.

Base Case. The length of the plan is *l*=0. By definition of a plan it means that $I \models G$. We distinguish between two cases as follows:

- G is a literal of the form $A(c)$ or $r(c_1, c_2)$, $A \in N_C$, $r \in N_R$ or a constant c or a tuple of constants (c_1, c_2). It follows that they should be in I. By construction, they should be in Σ_V. Hence, $G \in \Sigma_V$, thus the path is of length 0.

- G is of form $\exists R(c)$ or $\exists R^-(c)$ or $P^-(c,d)$. We analyze each case separately.

  – $I \models \exists R(c)$ or $I \models \exists R^-(c)$. By Definition 2.2, there exists some atom of the form $R(c,f) \in I$, (respectively $R(f,c) \in I$). By Definition 5.2, $R(c,b) \in I \subseteq \Sigma_V$. Thus, there are nodes $\exists R(c) \in V$ and $R(c,b) \in V$. Hence by construction, 7 (resp. 8) in encoding of edges, there is an edge between those two nodes.

  – Also for the case when $G = P^-(c,d)$, since $I \models P^-(c,d)$ it means that $P(d,c) \in I$. It follows that $P(d,c) \in \Sigma_V$. Hence, by 8 in the encoding of edges, there is a path of length 1 between those two nodes.

Induction Hypothesis. For any plan of length $\leq l$ there exists a path P from $L \in \Sigma_V$ to G.

Induction Step. We prove the claim for plans of length $l + 1$. We distinguish between different cases of the last (hence $(l+1)$th) basic action that appears in the plan.

1. $G = A(c)$ and $A \in N_R$. The $(l+1)$th action can have one of the following forms.

   a) $(A \oplus C)$ is the $(l+1)th$ action in the plan where C is a concept name or a concept of the form $\exists R$ or $\exists R^-$. Let the last state transition in $\mathcal{T}_{\mathcal{P}}$ be $\langle s', (A \oplus C), s \rangle$. Under the assumption that $(A \oplus C)$ is the first action that reaches the goal state, then by Definition 3.7, it holds that $s' \models A(c)$ and $s \models C(c)$. By the hypothesis, there

exists a path from some $L \in \Sigma_V$ to $C(c)$. Since $C(c) \in V$, also $A(c) \in V$ being a goal literal, and $(A \oplus C) \in Act$, by 1 in the encoding of edges, there exists an edge between $C(c), A(c)$. Hence $\langle P, (C(c), A(c))\rangle$ is the desired path.

b) $(A \oplus t)$, where $t \in N_V \cup \{c\}$. By 2 in the encoding of edges, it is easy to see that there is a path of length 1.

2. $G = r(c, d)$. Then one of the following actions that updates R can happen as the last and $(l + 1)$th action in $\mathcal{P}$.

   a) $(r \oplus P)$ where $r, P \in N_R$. Let the last state transition in $\mathcal{T}_\mathcal{P}$ be of the form $\langle s, (r \oplus P), s'\rangle$. By assumption and by Definition 3.7, $s' \models r(c, d)$ and $s \models P(c, d)$. Hence, there is a plan of length $\leq l$ for $fp(I, Act, P(c, d))$. By hypothesis and by 3 in the encoding of edges, there is an edge between some $L \in \Sigma_V$ to $r(c, d)$.

   b) $(r \oplus (t_1, t_2))$, where $\{t_1, t_2\} \in N_V \cup \{c, d\}$. By 4 in the construction of edges, it is easy to see that there is a path of length 1.

   c) $(r \oplus P^-)$ where $r, P \in N_R$. Let the last state transition in $\mathcal{T}_\mathcal{P}$ be of the form $\langle s, (r \oplus P), s'\rangle$. By assumption and Definition 3.7, and by Definition 2.2 $s' \models r(c, d)$ and $s \models P(d, c)$. Hence, there is a plan of length $\leq l$ for $fp(I, Act, P(d, c))$. By hypothesis and by 5 in the encoding of edges, there is an edge between some $L \in \Sigma_V$ to $r(c, d)$.

3. $G = r^-(c, d)$, where $r \in N_R$. Clearly a plan for $fp(I, Act, r^-(c, d))$ is a plan for $fp(I, Act, r(d, c))$. By 6 in the encoding of edges there is an edge $(r(d, c), r^-(c, d)$. To show that there is a path from some $L \in \Sigma_V$ to $r(c, d)$, the argument then is the same as in 2.

4. $G = \exists r(c)$, where $r \in N_R$. Let $s \in S$ be the last state of $\mathcal{T}_\mathcal{P}$, s.t. $s \models \exists r(c)$. By Definition 2.2, $s \models r(c, d)$, where $d \in dom(GP)$. By 7 in the construction of edges there is an edge $(r(c, d), R(c))$. To show that there is a path from some $L \in \Sigma_V$ to $r(c, d)$, the argument then is the same as in 2.

5. $G = \exists r^-(c)$, where $r \in N_R$. Let $s \in S$ be the last state of $\mathcal{T}_\mathcal{P}$, s.t. $s \models \exists r(c)$. By Definition 2.2, $s \models r(d, c)$, where $d \in dom(GP)$. By 8 in the construction of edges there is an edge $(r(d, c), R(c))$. To show that there is a path from some $L \in \Sigma_V$ to $r(c, d)$, the argument then is the same as in 2.

□

Thus, by encoding plan existence for $fp(GP)$ into reachability in graphs, which can be decided in **Nlogspace**, we showed that also deciding plan existence for a fixed domain planning problem with basic actions and goal formulas with one positive literal is also **Nlogspace**. More formally:

Theorem 5.9. *Let $GP = (I, Act, G)$ be a GDPP, where Act is a finite set of basic actions and G is a positive atom expressed via $\mathcal{ALCHOIQ}br$ basic concepts and $\mathcal{ALCHOIQ}br$ roles. Deciding plan existence for $fp(GP)$ can be done in* **Nlogspace**.

Proof. The above encoding to reachability, whose correctness is proved by Theorem 5.8, proves that plan existence for $fp(GP)$ can be decided in **Nlogspace**. □

Thus, we have shown that the case where the goal formula is a single positive literal, deciding *plan-existence* for a *fixed domain planning problem* with basic actions is **Nlogspace**. Next, we will show that the complexity of the case where the goal formula is a conjunction of an arbitrary number of positive literals is still the same.

5.3.2.2 Complexity of the Case with Positive Goals

Below, we show that the case where the goal formula is a conjunction of positive literals expressed via $\mathcal{ALCHOIQ}br$ basic concepts and $\mathcal{ALCHOIQ}br$ roles does not increase the complexity.

Firstly, we give the following lemma, which states that one can decide plan-existence, considering each conjunct of the goal formula separately.

Lemma 5.9.1. *Let $GP = (I, Act, G)$ be a GDPP, where $G = g_1 \wedge ... \wedge g_n$ s.t. each g_i is a positive literal as above. $fp(GP)$ has a plan iff each $fp(I, Act, g_i)$, $1 \leq i \leq n$ has a plan.*

Proof. We prove both directions separately:

'$\Longrightarrow$' This directions is straightforward. Let $\mathcal{P}$ be a plan for $fp(GP)$ and let $s \in S$ be the final state of the trajectory $\mathcal{T}_{\mathcal{P}}$ s.t. $s \models G$. It follows that $s \models g_i, \forall 1 \leq i \leq n$. Hence, $\mathcal{P}$ is a plan for each $fp(I, Act, g_i)$, $1 \leq i \leq n$.

'$\Longleftarrow$' Let $\mathcal{P}_i$ be a plan for $fp(I, Act, g_i), \forall 1 \leq i \leq n$, and $s_i \in S$ be the final states (resp. goal states, $s_i \models g_i$) of the trajectories $\mathcal{T}_{\mathcal{P}_i}$. We claim that

$$\mathcal{P} = \langle \mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_n \rangle$$

is a plan for $fp(GP)$. The proof is also straightforward. Let $s'_i \in S$ be the new states obtained after applying sequence $\mathcal{P}_i$ in the trajectory $\mathcal{T}_{\mathcal{P}}$. We need to show that $s'_n \models G$. By Lemma 5.3 $I \subseteq s'_1 \subseteq s'_2 \subseteq ... \subseteq s'_n$. It is also easy to notice that if $s_i \models g_i$ then $s'_i \models g_i$ since intuitively, each $s'_i \supseteq s_i$. Hence, the final state will be a superset of all previous states and will satisfy each goal formula that the previous states do. It follows that $s'_n \models g_1 \wedge ... \wedge g_n$. Thus, $\mathcal{P}$ is a plan for $fp(GP)$. □

Now, we are prepared to introduce the following theorem.

Theorem 5.10. *Let $GP = (I, Act, G)$ be a GDPP, where Act is a finite set of basic actions and G is a conjunction of positive literals expressed via $\mathcal{ALCHOIQ}br$ basic concepts and $\mathcal{ALCHOIQ}br$ roles. Deciding plan existence for $fp(GP)$ can be done in* **Nlogspace**.

Proof. By Theorem 5.9, each check can be done in **Nlogspace**. □

To sum up, in this section, we showed that deciding *fixed domain plan-existence* for a finite set of basic actions and a positive goal formula with literals expressed via basic concepts and $\mathcal{ALCHOIQ}br$ roles is **Nlogspace**.

Next, we study the case where the atoms in the goal formula are negated. We study their impact to the complexity of deciding *plan-existence*.

5.3.3 The Case of Negative Goals

Now, we assume a graph database planning problem $GP = (I, Act, G)$ which consists only of negative goals G and Act is a finite set of basic actions. As before, we do not put any restrictions on the initial database.

Intuitively, being the opposite of the positive case, one expects that a plan for this case needs only actions that perform deletion, and that positive actions are redundant since they add atoms. But, surprisingly this is not how it works.

To get a better intuition, we give an example to illustrate this case:

Example 5.3. *Assume $GP = (I, Act, G)$ if given as follows.*

$$I = \{Project(P20840), Project(P24090)\}.$$

$$Act = \{\rho_1, \rho_2, \rho_3\}.$$

where:

$$\begin{aligned}\rho_1 &= ConcludedProject \oplus X,\\ \rho_2 &= ConcludedProject \ominus X,\\ \rho_3 &= Project \ominus ConcludedProject.\end{aligned}$$

$$G = \neg Project(P24090)).$$

It can be easily seen from this example that there doesn't exist a plan with only negative actions for $fp(GP)$. Therefore, for achieving a state with no atom Project($P24090$) there is just one action, namely ρ_3, but it can be easily noticed that there is no atom ConcludedProject($P20490$) in the initial database. Hence, another earlier action to the initial database needs to be performed so that ConcludedProject($P24090$) is added. This can be done by the action ρ_1, by substituting X with ($P24090$). Thus, a plan for this instance is:

$$\langle \rho_1, \rho_3 \rangle.$$

By the very basic Example 5.3, it was shown that for the case with negative goals one may need plans with not only negative actions. In turn, positive actions are often needed to capture the necessary changes to the states s.t. a goal state is reached. Indeed, we will show that the case where the goal formula is a single negative literal can be treated as the case of positive goals.

To sum up, the negative case with only one negative atom, can be reduced to the positive case. In the next section, we formally define and prove the reduction.

5.3.3.1 Reducing Singleton Negative Goals Case to Positive Goals Case

To get a better intuition, we start by analyzing the basic subcase with a single negative literal in the goal formula. Thus, we will consider a $GP = (I, Act, G)$ with G as one of the following forms: $\neg A(c)$, $\neg R(c_1, c_2)$ or $\neg R^-(c_1, c_2)$, where $A \in N_C$ and $R \in N_R$. As in the positive case, Act is a finite set of basic actions. Surprisingly, if the initial database is not already a goal state and if there exists a plan, then there is always a plan with a (possibly empty) sequence of positive actions followed by a single negative action. E.g. to delete $A(c)$, where $A \in N_C$, from the initial database there should exist some action of the form $(A \ominus C_j)$, where C_j is an $\mathcal{ALCHOIQ}br$ basic concept. There are two cases to be distinguished; $(A \ominus C_j)$ reaches the goal state or it doesn't. If it's the first case, then there is a plan with a single action, namely $(A \ominus C_j)$. The latter means that $C_j(c)$ is not satisfied by the initial database. Hence, the next goal is $fp(I, Act, C_j(c))$, which is a positive goal and checking plan existence for this case is done as in the positive case (see Theorem 5.8).

Note: It is obvious that if $I \models G$ then I is already a goal state, but for this section, w.l.o.g. we assume that $I \not\models G$ and that the length of the plan is *n*, where the *n*-th action is the first one that reaches the goal.

Theorem 5.11. *Assume $GP = (I, Act, G)$, where Act is a finite set of basic actions and G is a single negative atom of the form $\neg A(c)$, $\neg R(c_1, c_2)$ or $\neg R^-(c_1, c_2)$, where $A \in N_C$ and $R \in N_R$. For every plan of $fp(GP)$, there exist a plan with only positive actions and a last negative action.*

To prove this theorem it is sufficient to prove the following lemma:

Lemma 5.11.1. *$fp(I, Act, G)$ has a plan iff one of the following cases holds.*

- *$G = \neg A(c)$, where $A \in N_C$ and $c \in N_I$ and there is an action $\rho_j = (A \ominus C_j) \in Act$, where C is an $\mathcal{ALCHOIQ}br$ basic concept, s.t. $fp(I, Act, C_j(c))$ has a plan.*
- *$G = \neg R(c_1, c_2)$, where $R \in N_R$ and $\{c_1, c_2\} \subseteq N_I$ and there is an action $\rho_j = (R \ominus P) \in Act$, where P is an $\mathcal{ALCHOIQ}br$ role s.t. $fp(I, Act, P(c_1, c_2))$ has a plan.*
- *$G = \neg R^-(c_2, c_1)$, where $R \in N_R$ and $\{c_1, c_2\} \subseteq N_I$ and there is an action $\rho_j = (R \ominus P) \in Act$, where P is an $\mathcal{ALCHOIQ}br$ role s.t. $fp(I, Act, P(c_1, c_2))$ has a plan.*

Proof. We prove both directions as follows:
'$\Longleftarrow$' (*Soundness.*) This direction is straightforward. We prove each case separately.

- Let $G = \neg A(c)$, $\rho_j = (A \ominus C_j) \in Act$ and $\mathcal{P}$ be a plan for $fp(I, Act, C_j(c))$, where $A \in N_C$, $c \in N_I$ and C an $\mathcal{ALCHOIQ}br$ basic concept. If C_j is a variable t or a constant c, then it is easy to see that the plan is of length 0. Otherwise, there exists a trajectory $\mathcal{T}_\mathcal{P}$, whose last state $s \in S$ is such that $s \models C_j(c)$. Next, applying $\langle s, (A \ominus C_j), s' \rangle$, by Definition 4.4 and by Definition 3.7, $s' = M_{(A \ominus C_j)}(s) = s \setminus \{A(c) \mid s \models C_j(c)\}$, $\{s', s\} \subseteq S$. The latter and by assumption that $s \models C_j(c)$, it follows that $s' \models \neg A(c)$. Hence, $\langle \mathcal{P}, (A \ominus C_j) \rangle$ is a plan for $fp(GP)$.

- Let $G = \neg R(c_1, c_2)$, $\rho_j = (R \ominus P) \in Act$, and $\mathcal{P}$ be a plan for $fp(I, Act, P(c_1, c_2))$, where $R \in N_R$, $\{c_1, c_2\} \subseteq N_I$ and P a $\mathcal{ALCHOIQ}br$ role. If P is a tuple of variables (t_1, t_2) or a tuple of constants (c_1, c_2), then it is easy to see that the plan is of length 0. Otherwise, using the same arguments as in the previous case, the last state $s \in S$ of the trajectory $\mathcal{T}_\mathcal{P}$, is s.t. $s \models P(c_1, c_2)$. Applying $(R \ominus P)$ to s, results in a state $s' \in S$ s.t. $s' \models \neg R(c_1, c_2)$. Thus $\langle \mathcal{P}, (R \ominus P) \rangle$ is a plan for $fp(GP)$.

- Let $G = \neg R^-(c_2, c_1)$, $\rho_j = (R \ominus P) \in Act$, and $\mathcal{P}$ be a plan for $fp(I, Act, P(c_1, c_2))$, where $R \in N_R$, $\{c_1, c_2\} \subseteq N_I$ and P is an $\mathcal{ALCHOIQ}br$ role. If P is a tuple of variables (t_1, t_2) or a tuple of constants (c_1, c_2), then the plan is of length 0. Otherwise, the last state $s \in S$ of $\mathcal{T}_\mathcal{P}$ is s.t. $s \models P(c_1, c_2)$. Applying $(R \ominus P)$ to s results in a state $s' \in S$, s.t. $s' \models \neg R(c_1, c_2)$. It follows that $s' \models \neg R^-(c_2, c_1)$. Thus $\langle \mathcal{P}, (R \ominus P) \rangle$ is a plan for $fp(GP)$.

'$\Longrightarrow$' (*Completeness.*) We distinguish between different types of the goal formula.

- Let $G = \neg A(c)$ and $\mathcal{P}$ be a plan for $fp(I, Act, \neg A(c))$, where $A \in N_C$. By assumption the last action is the first that reaches the goal and has the form $(A \ominus C_j)$ where C_j is a basic concept. If C_j is a variable t or a constant c, then it is obvious that there is a plan of length 0, since by assumption avery state models a constant. Otherwise, let $s \in S$ be the last state of $\mathcal{T}_\mathcal{P}$ s.t. $s \models \neg A(c)$. Let the last state transition have form $\langle s', (A \ominus C_j), s \rangle$. By Definition 4.4 and by Definition 3.7, $s = M_{(A \ominus C_j)}(s') = s' \setminus \{A(c) \mid s' \models C_j(c)\}$. The latter and by assumption that $I \not\models \neg A(c)$ (hence, $A(c) \in I$) it follows that $s' \models C_j(c)$. Hence, there exists a plan for $fp(I, Act, C_j(c))$, namely $\mathcal{P}$ without the last action is a plan for $fp(I, Act, C_j(c))$.

- Let $G = \neg R(c_1, c_2)$ and $\mathcal{P}$ be a plan for $fp(I, Act, \neg R(c_1, c_2))$, where $R \in N_R$. Reasoning as in the previous case the last action has form $(R \ominus P)$, where P is an $\mathcal{ALCHOIQ}br$ role. If P has form (t_1, t_2), $\{t_1, t_2\} \subseteq N_V$ or (c_1, c_2) constants, then obviously the claim holds. Otherwise, let $s \in S$ be the last state of $\mathcal{T}_\mathcal{P}$ s.t. $s \models \neg R(c_1, c_2)$, and the last state transition $\langle s', (R \ominus P), s \rangle$. By Definition 3.7, by Definition 4.4, and by assumption that $I \not\models \neg R(c_1, c_2)$, it holds that $s' \models P(c_1, c_2)$. Hence, there exists a plan for $fp(I, Act, P(c_1, c_2))$, namely $\mathcal{P}$ without the last action.

- Let $G = \neg R^-(c_1, c_2)$ and $\mathcal{P}$ be a plan for $fp(I, Act, \neg R^-(c_2, c_1))$, where $R \in N_R$. The last action has form $(R \ominus P)$, where P is an $\mathcal{ALCHOIQ}br$ role. If P has form (t_1, t_2), $\{t_1, t_2\} \subseteq N_V$ or (c_1, c_2) constants, then the claim holds. Otherwise, let $s \in S$ be the last state of $\mathcal{T}_\mathcal{P}$ s.t. $s \models \neg R(c_1, c_2)$, and the last state transition $\langle s', (R \ominus P), s \rangle$. By Definition 3.7 and by Definition 4.4, and by assumption that $I \not\models \neg R^-(c_2, c_1)$, hence $I \not\models \neg R(c_1, c_2)$, it holds that $s' \models P(c_1, c_2)$. Hence, there exists a plan for $fp(I, Act, P(c_1, c_2))$, namely $\mathcal{P}$ without the last action.

□

The above theorem and proof shows that the complexity bound is still **Nlogspace**.

Theorem 5.12. *Assume $GP = (I, Act, G)$, where Act is a finite set of basic actions and G is a single atom of form $\neg A(c)$, $\neg R(c_1, c_2)$ or $\neg R^-(c_1, c_2)$, where $A \in N_C$ and $R \in N_R$. Then, deciding plan existence for $fp(GP)$ is in* **Nlogspace**.

Thus, we showed that one can turn a planning instance with an atomic negative goal of the form $\neg A(c)$, $\neg R(c_1, c_2)$ or $\neg R^-(c_1, c_2)$, where $A \in N_C$ and $R \in N_R$ into a positive planning instance, preserving the same complexity. It can be noticed that goal atoms of the form $\neg\exists R(c)$ and $\neg\exists R^-(c)$ are not considered. Indeed, Theorem 5.11 does not apply for cases where the goal atom is existentially quantified. The intuition is that a state that satisfies $\neg\exists R(c)$, should contain no atom of the form $R(c, d)$, where d might be any constant from $dom(GP)$. If the initial database contains two atoms $R(c, d)$ and $R(c, f)$, it might be the case that a plan for $fp(I, Act, \neg\exists R(c)$ might require more than just one negative action. We illustrate this intuition with a simple example.

Example 5.4. *Let $GP = (I, Act, G)$ be given as follows.*

$$\begin{aligned} I = \{&Project(P20840), Project(P24090) \\ &Employee(E01), Employee(E03), \\ &ProjectEmployee(E01), \\ &worksFor(E01, P20840), \\ &worksFor(E01, P24090)\}. \end{aligned}$$

$$Act = \{\rho = worksFor \ominus (t_1, t_2)\},$$

$$G = \neg\exists worksFor(E01).$$

This is a very simple example, where given an initial database and a single action, it is required that E01 should not work for any project. Thus, there should exist a state where no relation worksFor with E01 appears. A plan for this instance is

$$\mathcal{P} = \langle (worksFor \ominus (E01, P20840)), (worksFor \ominus (E01, P24090))\rangle.$$

Hence, applying ρ twice, by instantiating (t_1, t_2), the desired state is reached.

Until now we have considered only cases where the goal formula is a single negative literal. The same idea is expected to apply also to instances with negative goals, expressed as conjunction of two ore more negative literals. Thus, plans for these cases would be expected to be a sequence of positive actions followed by negative actions. The cardinality of these negative actions would be equal to the cardinality of the goal formula. Unfortunately, even for a goal formula with two negative literals, checking *plan-existence* becomes more complicated. A simple counterexample is the following:

Example 5.5. *Assume $GP = (I, Act, G)$ is given as follows.*

$$\begin{aligned} I = \{&ActiveProject(P20840), \\ &OldProject(P20840), OldProject(P24090), \\ &ReviewProject(P20840), ReviewProject(P24090)\}. \end{aligned}$$

$$Act = \{\rho_1, \rho_2, \rho_3, \rho_4\},$$

where:

$$\begin{aligned}\rho_1 &= ActiveProject \ominus ConcludedProject,\\ \rho_2 &= ConcludedProject \oplus OldProject,\\ \rho_3 &= ConcludedProject \ominus UpdateProject,\\ \rho_4 &= UpdateProject \oplus ReviewProject.\end{aligned}$$

$$G = \neg ActiveProject(P20840) \wedge \neg ConcludedProject(P24090).$$

Thus, it is easy to see that I is already a goal state for $fp(I, Act, \neg ConcludedProject(P24090)$, but the same does not hold for $fp(I, Act, \neg ActiveProject((P20840))$. The only way to achieve a goal state for the latter is by applying ρ_2 to I and then ρ_1. But, then this state is not any more a goal state for $fp(I, Act, \neg ConcludedProject(P24090)$. It is not hard to notice that adding ρ_4 and then ρ_3 to the sequence, the desired goal state for $fp(I, Act, negActiveProject(P20840) \wedge \neg ConcludedProject(P24090))$ can be achieved. The plan is the following:

$$\mathcal{P} = \langle \rho_2, \rho_1, \rho_4, \rho_3 \rangle.$$

We showed that for the case with a goal formula with two negative literals, deciding *plan-existence* cannot be encoded to a positive case. Basically, through Example 6.6, we just showed that it cannot be the case that always in a plan we would have a sequence of positive actions followed by negative actions. Actually, carefully observing it, we can deduce that treating the goal formula as two atomic negative goals, two plans as for the basic case can be concatenated, and deciding *plan-existence* can be still done in *Nlogspace*. But, making the previous example a little more complicated and less intuitive, one could show that the above claim does not hold.

Example 5.6. *Assume a GDPP $GP = (I, Act, G)$ given as follows.*

$$\begin{aligned}I = \{&ConcludedProject(P24090),\\ &OldProject(P20840), OldProject(P24090),\\ &ReviewProject(P20840), ReviewProject(P24090)\\ &NewProject(P20840)\}.\end{aligned}$$

$$Act = \{\rho_1, \rho_2, \rho_3, \rho_4\},$$

where:

$$\begin{aligned}\rho_1 &= ConcludedProject \ominus UpdateProject,\\ \rho_2 &= UpdateProject \oplus ActiveProject,\\ \rho_3 &= ActiveProject \oplus ReviewProject,\\ \rho_4 &= ReviewProject \ominus NewProject.\end{aligned}$$

$$G = \neg ActiveProject(P20840) \wedge \neg ConcludedProject(P24090).$$

Thus, $I \models \neg ActiveProject(P20840)$, but $I \not\models \neg ConcludedProject(P24090)$. Intuitively, we must look for a plan for $fp(I, Act, \neg ConcludedProject(P24090))$. Now, in order to

achieve a state that doesn't contain ConcludedProject($P24090$), it is sufficient to apply the sequence

$$l = \langle \rho_3, \rho_2, \rho_1 \rangle.$$

The last state $s \in S$ of $\mathcal{T}_l$ is surely a goal state for $fp(I, Act, \neg ConcludedProject(P24090))$. But it is easily seen that $s \not\models \neg ActiveProject(P20840)$. It follows that applying the ρ_4 before l is a solution. Hence,

$$\mathcal{P} = \langle \rho_4, \rho_3, \rho_2, \rho_1 \rangle$$

is the required plan for the corresponding $fp(GP)$.

But does this already increase the complexity? Does this already make the problem harder? Well, we will show that some cases with a goal formula with two or more negative literals, deciding plan existence is still *tractable*. For other cases with negative literals or even with combination of negative and positive literals in the goal formula, the complexity seems to increase. Next, we provide a deeper analysis of there different combinations.

5.3.4 Analysis of Subcases

In this section, we consider a GDPP $GP = (I, Act, G)$, where Act is a finite set of atomic actions expressed using concept names only. We will analyze different subcases by imposing syntactic restrictions on the goal formula. Most of these cases will be still tractable. Our purpose is to reach the most expressive setting, whose complexity does not exceed *polynomial* time.

5.3.4.1 Subcase of Negative Goals $\neg$ A($\mathbf{c}_1$) $\wedge ... \wedge$ $\neg$A($\mathbf{c}_n$)

Firstly, we consider the case where there is given a GDPP $GP = (I, Act, \neg A(c_1) \wedge ... \wedge \neg A(c_n))$, where $A \in N_C$ and $\{c_1, ..., c_n\} \subseteq N_I$. We show that checking if there exists a plan for this case can be done still in **Nlogspace**. This can be achieved by reduction to the *the case of atomic negative goals*. To formally define this intuition, we introduce the following theorem.

Theorem 5.13. *Assume a GDPP $GP = (I, Act, \neg A(c_1) \wedge ... \wedge \neg A(c_n))$ and let $GP_j = (I, Act, \neg A(c_j))$, $1 \leq j \leq n$. Then $fp(GP)$ has a plan iff each $fp(GP_j)$ $1 \leq j \leq n$ has a plan.*

Proof. We prove both directions separately:

'$\Rightarrow$' This direction is trivial. Assume there exists a plan $\mathcal{P}$ for $fp(GP)$. This would mean that there is a sequence of actions and set of states s.t. the trajectory obtained reaches a state $s \in S$ that satisfies the goal, $s \models G$. Hence, $s \models \neg A(c_j)$ for each $\neg A(c_j)$ that occurs in G. $\mathcal{P}$ is the desired plan for each $fp(GP_j)$.

'$\Leftarrow$' Now, let's assume that there exist a plan for each $fp(GP_j)$. By Theorem 5.11 and Lemma 5.11.1, for each of these plans there exists a plan $\mathcal{P}_j$, whose last action has form $\rho_j = A \ominus C_j$, such that there is a plan $\mathcal{P}'_j$ for $fp(I, Act, C_j(c_j))$. Hence, $\mathcal{P}_j = \langle \mathcal{P}'_j, \rangle \rho_j$, $1 \leq j \leq m$. We argue that the following sequence of actions:

$$\mathcal{P} = \langle \mathcal{P}'_1, ..., \mathcal{P}'_n, \rho_1, ..., \rho_n \rangle$$

is a plan for $fp(GP)$.

Actually the proof is almost straightforward. One would need to show that the last state obtained by the trajectory over these sequence of actions would be a a goal state $s_{2n} \in S$ s.t. $s_{2n} \models \neg A(c_1) \wedge ... \wedge \neg A(c_n)$. By theorem 5.7, each $\mathcal{P}'_j$ consists only of positive actions. Hence, the states obtained after each application of each action in each P'_j will only grow. The state $s_n \supseteq s_j$, where s_n is the last state in the trajectory over $\mathcal{P}'_1, ...\mathcal{P}'_n$, and s_j is the state of the trajectory of each $\mathcal{P}'_j$. By assumption, each $C_j(c_j)$ is in the last state of the trajectory of each $\mathcal{P}'_j$, hence each $C_j(c_j) \in s_n, 1 \leq j \leq m$. Using this and the above assumption, the last sequence of actions $\langle \rho_1, ..., \rho_n \rangle$ would delete each $C_j(c_j)$ from s_n and would produce a state $s_{2n} \models G$.

□

We proved that the negative case, where the goal formula is a conjunction of literals, (represented as the same concept over different individuals in DL- jargon) can be reduced to the negative case with the singleton goal formula. This shows that the complexity is still in **Nlogspace**.

Thus, we can now introduce the following theorem.

Theorem 5.14. *Deciding whether there exists a plan for* $fp(I, Act, \neg A(c_1) \wedge ... \wedge \neg A(c_n))$, *where* $A \in N_C$ *and* $\{c_1, ..., c_n\} \subseteq N_I$, *can be done in* **Nlogspace**.

5.3.4.2 Subcase of Goal A(c) $\wedge\neg$ B(c)

Next, we consider the *GDPP* $GP = (I, Act, A(c) \wedge \neg B(c))$, where $\{A, B\} \subseteq N_C$ and $A \neq B$. We will introduce an algorithm for checking plan-existence for this case. To get a better intuition, let's consider the following example.

Example 5.7. *Assume a* GDPP $GP = (I, Act, G)$ *is given as follows.*

$$I = \{C(c), C(d)\}.$$

$$Act = \{\rho_1, \rho_2, \rho_3, \rho_4\},$$

where:

$$\begin{aligned} \rho_1 &= A \oplus B, \\ \rho_2 &= B \oplus C, \\ \rho_3 &= B \ominus D, \\ \rho_4 &= D \oplus A. \end{aligned}$$

$$G = \mathrm{A}(c) \wedge \neg \mathrm{B}(c).$$

This example is given in an abstract level so that one can better understanding the intuition behind it. $I \not\models A(c)$ *and* $I \models \neg B(c)$. *Hence, we look firstly for a plan for* $fp(I, Act, A(c))$. *The only possible plan is* $\mathcal{P}' = \langle (B \oplus C), (A \oplus B) \rangle$. *It is not hard to notice that the last state* $s \in S$ *of* $\mathcal{T}_{\mathcal{P}'}$ *is s.t.* $s \not\models \neg B(c)$. *But only after applying* ρ_4 *and* ρ_3, *the result is a goal state for* $fp(GP)$. *The plan is as follows:*

$$\mathcal{P} = \langle \rho_2, \rho_1, \rho_4, \rho_3 \rangle.$$

In general for this type of instances, where $I \models \neg B(c)$ and $I \not\models A(c)$, it is best to search for a plan not considering actions with B on the left side. If there is a plan of this form, then we are done. If not, then a plan $\mathcal{P}$ for $fp(I, Act, A(c))$ would necessarily add $B(c)$ to the goal state. The intuition is that by Lemma 5.8.1, $B(c)$ will be a vertice of the path from some $F(c) \in I$ to with $A(c)$. Next, we need to get rid of $B(c)$. It is not hard to see that if there is a plan for $fp(I \cup B(c), Act, \neg B(c))$, by Theorem 5.11 there is a plan $\mathcal{P}'$ which consists of a sequence of positive actions followed by a single negative action with B in the right side. Clearly, adding $\mathcal{P}'$ after $\mathcal{P}$, the result would still be a plan for $fp(I, Act, A(c) \wedge \neg B(c))$. The idea is that the resulting plan is still a sequence of positive actions with a last negative action deleting atoms of the form $B(d)$, where $d \in dom(GP)$. On the other hand, if $I \models A(c)$, one only needs to check if there exists a plan for $fp(I, Act, \neg B(c))$. It will certainly not do any 'harm' to the positive literal, since for every plan there exists a plan with only positive actions and one last negative action.

The algorithm below should make everything more clear, but before we introduce the following two lemmas:

Lemma 5.4. *Let $GP = (I, Act, A(c) \wedge \neg B(c))$ be a graph database planning problem where $A \neq B$ and $B(c) \in I$. There exists a plan for $fp(I, Act, A(c))$ and there exists a plan for $fp(I, Act, \neg B(c))$ if and only if there exists a plan for $fp(I, Act, A(c) \wedge \neg B(c))$.*

Proof. '$\Longrightarrow$'Assume $\mathcal{P}_1$ is a plan for $fp(I, Act, A(c))$ and $\mathcal{P}_2$ is a plan for $fp(I, Act, \neg B(c))$. Since $B(c) \in I$ then the plan should have at least one action of the form $(B \ominus C_i)$, thus $|\mathcal{P}_2| \geq 1$. We need to show that there exists a plan $\mathcal{P}$ for $fp(I, Act, A(c) \wedge \neg B(c))$.

By Theorem 5.11 and Lemma 5.11.1 there exists a plan $\langle \mathcal{P}'_2, (B \ominus C_j) \rangle$ for $fp(I, Act, \neg B(c))$, where $\mathcal{P}'_2$ has only positive actions and a final negative action of form $(B \ominus C_i)$. Assume, $s_2 \in S$ is a goal state and the final state of the trajectory $\mathcal{T}_{\langle \mathcal{P}'_2, (B \ominus C_i) \rangle}$. The last state transition has form $\langle s_{2'}, (B \ominus C_i), s_2 \rangle$. Also, by Theorem 5.7 there exist a plan $\mathcal{P}'_1$ for $fp(I, Act, A(c))$ with only positive actions. Assume, $s_1 \in S$ is a goal state and the final state of the trajectory $\mathcal{T}_{\mathcal{P}'_1}$.

We claim that $\mathcal{P} = \langle \mathcal{P}'_1, \mathcal{P}'_2, (B \ominus C_i) \rangle$ is a plan for $fp(I, Act, A(c) \wedge \neg B(c))$. Assume, $s \in S$ is the final state of the trajectory $\mathcal{T}_{\mathcal{P}}$, where the last state transition has form $\langle s', (B \ominus C_i), s \rangle$.

By assumption, $s_1 \models A(c)$. By Lemma 5.3, $s' \models A(c)$ as well. Since the last action would just delete facts of form $B(c)$, then $s \models A(c)$ as well. Furthermore, since $B(c) \in I$, again using Lemma 5.3, $s' \models B(c)$. Also since $s_2 \models \neg B(c)$ then $s_{2'} \models C_i(c)$. Using this and Lemma 5.3, $s' \supseteq s_{2'}$, hence $s' \models C_i(c)$. It follows that $s \models \neg B(c)$ as well. Thus, $s \models A(c) \wedge \neg B(c)$, which shows the claim.

'$\Longleftarrow$' This direction is straightforward. Let $\mathcal{P}$ be a plan for $fp(I, Act, A(c) \wedge \neg B(c))$ and let $s \in S$ the be the goal state produced by $\mathcal{T}_{\mathcal{P}}$. By assumption $s \models A(c) \wedge \neg B(c)$. By entailment properties, it holds that $s \models A(c)$ and $s \models \neg B(c)$. Thus, $\mathcal{P}$ is a plan for $fp(I, Act, A(c))$ and a plan for $fp(I, Act, \neg B(c))$. □

Now we are ready to introduce the following algorithm, which we call *posneg*Goal.

Input: $GP = (I, Act, A(c) \wedge \neg B(c))$.
Output: The algorithm returns **true** iff $fp(GP)$ has a plan.
begin
case 1:
if *B(c)* $\in I$ *and A(c)* $\in I$ **then**
  if $\exists$ *a plan for* $fp(I, Act, \neg B(c))$ **then**
    return **true**
  else
    return **false**
  end
end

case 2:
if *B(c)* $\in I$ *and A(c)* $\notin I$ **then**
  if $\exists$ *a plan* $\mathcal{P}$ *for* $fp(I, Act, A(c))$ **then**
    if $\exists$ *a plan for* $fp(I, Act, \neg B(c))$ **then**
      return **true**
    end
  else
    return **false**
  end
end

case 3:
if *B(c)* $\notin I$ *and A(c)* $\in I$ **then**
  return **true**
end

case 4:
if *B(c)* $\notin I$ *and A(c)* $\notin I$ **then**
  if $\exists$ *a plan for* $fp(I, Act', A(c))$ *where*
  $Act' = Act \setminus \{(B \oplus C_i) \mid \exists C_i : (B \oplus C_i) \in Act\}$ **then**
    return **true**
  else
    if $\exists$ *a plan* $\mathcal{P}$ *for* $fp(I, Act, A(c))$ **then**
      if $\exists$ *a plan for* $fp(I \cup B(c), Act, \neg B(c))$ **then**
        return **true**
      else
        return **false**
      end
    end
  end
end

Algorithm 5.2: *posneg*Goal

We will give an intuition of the *posneg*Goal algorithm. It takes as input a graph database

planning instance $GP = (I, Act, A(c) \wedge \neg B(c))$. It does four main checks.

- At first, if $B(c) \in I$ and if $A(c) \in I$, it checks if there is a plan for $fp(I, Act, \neg B(c))$. If yes, then by a plan for $fp(I, Act, A(c) \wedge (c))$ exists. The argument is that by Theorem 5.11, there exists a plan consisting of a sequence of positive actions followed by only one negative action of the form $(B \ominus C_i) \in Act$. Hence, $A(c)$ would surely still be in the goal state of $fp(I, Act, \neg B(c))$.

- Secondly, the algorithm checks if $B(c) \in I$ and if $A(c) \notin I$, hence none of these facts is satisfied by the initial database. In this case, we start by checking if there exists a plan for $fp(I, Act, A(c))$. If yes, then it checks if there is a plan for $GP = (s_j, Act, \neg B(c))$. If this is the case than the algorithm returns **true**. This case is proven in Lemma 5.4.

- The third case is if $B(c) \notin I$ and if $A(c) \in I$. Clearly, the initial database is already the goal state for this case.

- Lastly, if it is the case that $B(c) \notin I$ and $A(c) \notin I$, is a bit more complicated. Since $B(c) \notin I$, we would want not to add this fact for all states until a goal state is reached for $fp(I, Act, A(c))$. Hence for this case we do two sub-checks.

  - Firstly, the algorithm checks if there exists a plan for $fp(I, Act, A(c))$, not considering any action from Act, where B appears in the right hand side i.e. no action of form $(B \oplus C_i)$. Hence, there is no chance of adding $B(c)$ to some state. If this is the case, then the algorithm returns **true**.
  - Otherwise, intuitively, it means that $B(c)$ is in the goal state (see proof of Theorem 5.18). Next, we need to search for a plan considering for $fp(I \cup B(c), Act, \neg B(c))$. If a plan exists the algorithm returns **true**.

Next, we prove the correctness of the algorithm.

Theorem 5.15. *Assume $GP = (I, Act, A(c) \wedge \neg B(c))$. There exists a plan for $fp(GP)$ if and only if the algorithm* PosNeg*Goal returns* **true**.

Proof. We prove both directions as follows:

'$\Longrightarrow$'(*Completeness.*) Assume $\mathcal{P}$ is a plan for $fp(I, Act, A(c) \wedge \neg B(c))$. It follows that the final state $s_n \in S$ of the trajectory $\mathcal{T}_{\mathcal{P}}$ is a goal state $s_n \models A(c) \wedge \neg B(c)$. Hence, since $s_n \models A(c)$ and $s_n \models \neg B(c)$ then $\mathcal{P}$ is a plan for $fp(I, Act, A(c))$ and also a plan for $fp(I, Act, \neg B(c))$. Thus, depending on the initial database, the algorithm for the first 3 cases, will return **true**. We consider the fourth case separately. Thus, for the case when $B(c) \notin I$ and $A(c) \notin I$ then one distinguishes between two cases.

- $\mathcal{P}$ does not contain any action $(B \oplus C_i)$ or it contains, but after deleting them, it is still a plan for $fp(I, Act, A(c) \wedge \neg B(c))$. Then our algorithm will return **true** through the first check of case 4 of the PosNeg*Goal* algorithm .

- $\mathcal{P}$ contains an action $(B \oplus C_i)$, that cannot be removed without losing the property of being a plan for $fp(I, Act, A(c) \wedge \neg B(c))$. Clearly, $B(c)$ will be added at some state in the trajectory $\mathcal{T}_{\mathcal{P}}$. By assumption $\mathcal{P}$ is a plan for $fp(I, Act, A(c))$ and also a plan for $fp(I, Act, \neg B(c))$. It is easy to see that $\mathcal{P}$ is also a plan for $fp(I \cup B(c), Act, A(c))$. By checks $4, 5, 6$ of the PosNeg*Goal* algorithm, it will return **true**.

'$\Longleftarrow$'(*Soundness.*) This direction is also straightforward by construction and by Lemma 5.3 and Lemma 5.4. Assume the algorithm PosNeg*Goal* returns **true**. Hence one of the following 4 cases might happen.

1. $B(c) \in I$ and $A(c) \in I$ and there exists a plan $\mathcal{P}$ for $fp(I, Act, \neg B(c))$. Clearly, there is a plan of length 0 for $fp(I, Act, A(c))$. Hence, by Lemma 5.3, there exists a plan for $fp(I, Act, A(c) \wedge \neg B(c))$.

2. $B(c) \in I$ and $A(c) \notin I$ and there $\exists$ a plan for $fp(I, Act, A(c_j))$ and there $\exists$ a plan $\mathcal{P}''$, by Lemma 5.3, there exists a plan for $fp(I, Act, A(c) \wedge \neg B(c))$.

3. $B(c_i) \notin I$ and $A(c_j) \in I$, then I is automatically a goal state.

4. $B(c_i) \notin I$ and $A(c_j) \notin I$ and if one of the following subcases happens.

   - There exists a plan $\mathcal{P}$ for $fp(I, Act', A(c))$ where $Act' = Act \backslash \rho_i$ for all actions $\rho_i = (B \oplus C_i) \in Act$. It is easy to see that $\mathcal{P}$ is a plan also for $fp(I, Act, A(c_j) \wedge \neg B(c_i))$.
   - All plans for $fp(I, Act', A(c))$ contain at least one action of form $\rho_i = (B \oplus C_i)$. Let $\mathcal{P}'$ be one of them whose trajectory $\mathcal{T}_{\mathcal{P}'}$ contains $s \in S$ as final state. By Lemma 5.3, $s \models B(c)$. Now, assume there exists a plan for $fp(I \cup B(c), Act, \neg B(c))$ and let $\mathcal{P}''$ be this plan. Using the same reasoning as in proof of Lemma 5.4 and by Lemma 5.3, it is not hard to see that at $\langle \mathcal{P}', \mathcal{P}'' \rangle$ is a plan for $fp(I, Act, A(c_j) \wedge \neg B(c_i))$.

□

The above algorithm runs in *polynomial* time. We are now ready to formally define the following theorem.

Theorem 5.16. *Deciding whether there exists a plan for* $fp(I, Act, A(c) \wedge \neg B(c)$*, where* $A, B \in N_C$ *and* $c \in N_I$*, can be done in* polynomial *time.*

5.3.4.3 The Subcase of the Goal ¬A(c) ∧¬B(c)

Now, we will consider the case where there is given a GDPP of the form $GP = (I, Act, \neg A(c) \wedge \neg B(c))$, where $A \neq B$. This case seems even more complicated. To get an intuition about the complexity of this case we introduce the following examples.

Example 5.8. *Assume a* GDPP $GP = (I, Act, G)$ *where:*

$$I = \{A(c), D(c)\},$$

$$Act = \{\rho_1, \rho_2, \rho_3, \rho_4\},$$

where:

$$\rho_1 = A \ominus B,$$
$$\rho_2 = B \oplus D,$$
$$\rho_3 = B \ominus A,$$
$$\rho_4 = A \oplus D.$$

$$G = \neg A(c) \wedge \neg B(c).$$

The example given is in an abstract level so that one can better understanding the intuition behind it. It is not hard to notice that there is no plan for this $fp(GP)$. Every possible combination of these actions would fail to achieve a plan, since finding a plan for deleting $A(c)$, necessarily will add $B(c)$ too. Trying to delete $B(c)$, $A(c)$ will be added again. A sequence for achieving the goal would look as follows:

$$\mathcal{P} = \langle \rho_2, \rho_1, \rho_4, \rho_3, \rho_2, \rho_1, \rho_4, \rho_3, ... \rangle$$

Therefore every machine would run forever to find a plan for this case. But by definition the plan must be finite. Hence, there is no plan.

Now, we introduce another similar example, which has a plan.

Example 5.9. *Assume a* GDPP $GP = (I, Act, G)$ *is given as follows.*

$$I = \{A(c), D(c)\},$$

$$\boldsymbol{Act} = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5\}.$$

where:

$$\rho_1 = A \ominus C,$$
$$\rho_2 = C \oplus B,$$
$$\rho_3 = B \oplus D,$$
$$\rho_4 = B \ominus A,$$
$$\rho_5 = A \oplus D.$$

$$G = \neg \mathrm{A}(c) \wedge \neg \mathrm{B}(c).$$

In this example, we changed the first action and added a new action by adding a new intermediate concept C. A plan would be as follows:

$$\mathcal{P} = \langle \rho_3, \rho_2, \rho_1, \rho_5, \rho_4, \rho_1 \rangle$$

It is not hard to see that at first we tried to reach a state that does not contain $A(c)$. The only way to achieve this is by adding to this state $B(c)$ and $C(c)$. Next, we tried to achieve a new state without $B(c)$, but inevitably $A(c)$ was added again to this state. Notice that $C(c)$ was not deleted. Hence, after applying again ρ_1, finally a plan is reached.

But as it is easily seen that collecting the positive actions and then applying the two negative actions would still result in a shorter plan

$$\mathcal{P}' = \langle \rho_3, \rho_2, \rho_4, \rho_1 \rangle.$$

ρ_5 is not used in this case, and there is no need to apply ρ_1 twice. ρ_4 also needs to be applied earlier, since it depends on A which at the respective state contains $A(c)$. At last, applying ρ_1 would delete $A(c)$ and obtain a state that models both $\neg A(c) \wedge \neg B(c)$.

Below, we will introduce an algorithm for checking plan existence for this case of fixed domain planning problem, but before we introduce the following lemmas and theorem.

Lemma 5.5. *Assume a GDPP $GP = (I, Act, \neg A(c) \wedge \neg B(c))$ s.t. $\{A(c), B(c)\} \subseteq I$ and $A \neq B$. $fp(GP)$ has a plan if and only if $fp(I, Act, \neg A(c))$ has a plan and $fp(I, Act, \neg B(c))$ has a plan s.t. at least for one of them, there exist a plan that does not contain an action $(A \ominus B)$, $(B \ominus A)$, respectively.*

Proof. '$\Longrightarrow$' *(Completeness.)* Let $\mathcal{P}$ be a plan for $fp(GP)$. $\mathcal{P}$ is also a plan for $fp(I, Act, \neg A(c))$ and a plan for $fp(I, Act, \neg B(c))$. Now, assume all plans for $fp(I, Act, \neg A(c))$ and all plans for $fp(I, Act, \neg B(c))$ contain $(A \ominus B)$ and $(B \ominus A)$. It follows that $\mathcal{P}$ should contain both actions $(A \ominus B)$ and $(B \ominus A)$. W.l.o.g. assume that an occurrence of $(A \ominus B)$ is at length l in the trajectory $\mathcal{T}_{\mathcal{P}}$, and assume that the last occurrence of one of those two actions is $(B \ominus A)$ at position $n > l$. Let the state transition at position n be $t_n = \langle s_{n-1}, (B \ominus A), s_n \rangle$. By Definition 3.7, $\{A(c), B(c)\} \subseteq s_{n-1}$, otherwise this action would be redundant and can be deleted without affecting the plan. Now let the state transition at position l be $t_l = \langle s_{l-1}, (A \ominus B), s_l \rangle$. Using the same reasoning and by Definition 3.7, $A(c) \notin s_l$, where $l < (n-1)$. It follows that $A(c)$ was added again at some state between s_l and s_{n-1}. Hence, $(A \ominus B)$ can be deleted without affecting the plan. Using the same argument, all actions of form $(A \ominus B)$ are redundant. This is a contradiction to our assumption that $\mathcal{P}$ contains both actions $(A \ominus B)$ and $(B \ominus A)$.

'$\Longleftarrow$' *(Soundness.)* Let $\mathcal{P}'$ be a plan for $fp(I, Act, \neg A(c))$ and $\mathcal{P}''$ be a plan for $fp(I, Act, \neg B(c))$. We distinguish between two cases.

1. $\mathcal{P}'$ and $\mathcal{P}''$ do not contain $(A \ominus B)$, $(B \ominus A)$. We can consistently assume by Theorem 5.7 that $\mathcal{P}'$ is a sequence of positive actions (possibly empty) $\mathcal{P}'_1$ followed by a single negative action $(A \ominus C_i)$, where $C_i \in N_C$. Also $\mathcal{P}''$, is as a sequence of positive actions $\mathcal{P}''_1$ with last negative action $(B \ominus C_j)$, where $C_j \in N_C$. Also, let s' and s'' be the final states of the $\mathcal{T}_{\mathcal{P}'_1}$ and $\mathcal{T}_{\mathcal{P}''_1}$, respectively. We claim that

   $$\mathcal{P} = \langle \mathcal{P}'_1, \mathcal{P}''_1, (A \ominus C_i), (B \ominus C_j) \rangle$$

   is a plan for $fp(GP)$. Let s''_1 be the final state of $\mathcal{T}_{\mathcal{P}'_1 \circ \mathcal{P}''_1}$ and s the final state of $\mathcal{T}_{\mathcal{P}}$.

   By assumption, by Definition 3.7 and by Definition 4.4, $\mathcal{P}'_1$ is a plan for $fp(I, Act, C_i(c))$ and $\mathcal{P}''_1$ is a plan for $fp(I, Act, C_j(c))$. Notice that there is no restriction that C_i and C_j should be different. By Lemma 5.3 $s' \subseteq s''_1$, so $s''_1 \models C_i(c)$. In addition, $s''_1 \models C_j(c)$ since

$s'' \models C_j(c)$ and by Lemma 5.3 $s'' \subseteq s''_1$. Hence, $s''_1 \models C_j(c) \wedge C_i(c)$. Applying the last two actions, by Definition 3.7, it is easy to see that $s \models \neg A(c) \wedge \neg B(c)$.

Note: If $\langle \mathcal{P}'_1, \mathcal{P}''_1 \rangle$ is empty, then it means that $\{C_i(c), C_j(c)\} \subseteq I$. Also, notice that since $\{A(c), B(c)\} \subseteq I$ and by Theorem 5.7, $\mathcal{P}$ will contain exactly two negative actions. A plan for this case is as follows.

$$\langle (A \ominus C_i), (B \ominus C_j) \rangle,$$

where C_i, C_j are possibly the same concept name.

2. W.l.o.g. assume that $\mathcal{P}'' = \mathcal{P}''_1 \circ (B \ominus A)$ and $\mathcal{P}'$ is defined as above. We claim that a plan for $fp(GP)$ is as follows:

$$\mathcal{P} = \langle \mathcal{P}'_1, \mathcal{P}''_1, (B \ominus A), (A \ominus C_i) \rangle$$

It is easy to notice that the negative action $(B \ominus A)$ is put before deleting $A(c)$ by the last action. Using the same arguments as above, $\mathcal{P}$ is a plan for $fp(GP)$.

□

Lemma 5.6. *Assume $GP = (I, Act, \neg A(c) \wedge \neg B(c))$ s.t. $A(c) \in I$, $B(c) \notin I$. $fp(GP)$ has a plan if and only if one of the following holds.*

1. *There exists a plan for $fp(I, Act, \neg A(c))$ where Act does not contain any action with B in the right hand side.*
2. *There exists a plan for $fp(I, Act, \neg A(c))$ and exists a plan for $fp(I \cup B(c), Act, \neg B(c))$ s.t. at least for one of them, there exist a plan that does not contain an action $(A \ominus B)$, $(B \ominus A)$, respectively.*

Note: W.l.o.g we assumed that $A(c) \in I$ and $B(c) \notin I$.

Proof. '$\Longrightarrow$' Let $\mathcal{P}$ be a plan for $fp(GP)$. Thus $\mathcal{P}$ is a plan for $fp(I, Act, \neg A(c))$ and is also a plan for $fp(I, Act, \neg B(c))$. We distinguish between two cases.

- $\mathcal{P}$ is still a plan for $fp(GP)$ after deleting (if there exist) from $\mathcal{P}$ all positive actions with B in the right hand side. The latter and the fact that $\mathcal{P}$ is also a plan for $fp(I, Act, \neg A(c))$ proves 1.
- $\mathcal{P}$ contains at least an action of the form $B \oplus C_i$ that can not be deleted without harming the property of $\mathcal{P}$ of being a plan for $fp(GP)$. It is easily seen that $B(c)$ will be added to some state (otherwise the action is redundant and can be deleted). So that $\mathcal{P}$ is also a plan for $fp(I, Act, \neg B(c))$, there should exist an action $(B \ominus C_j)$ in the plan as well computed after $B \oplus C_i$. Thus, $\mathcal{P}$ is also a plan for $fp(I \cup B(c)), Act, \neg B(c)$. Using the same arguments as in the proof (completeness) of Lemma 5.5, $\mathcal{P}$ needs at most one action of the form $(A \ominus B)$, $(B \ominus A)$.

'$\Longleftarrow$' We consider each case separately.

- Assume $\mathcal{P}$ is a plan for $fp(I, Act, \neg A(c))$ where Act does not contain any action with B in the right hand side. Let the final state of $\mathcal{T}_{\mathcal{P}}$ be $s \in S$, s.t. $s \models \neg A(c)$. Clearly, $B(c)$ is not added to any state of the trajectory $\mathcal{T}_{\mathcal{P}}$. Thus, $s \models \neg B(c)$ as well. It follows that $\mathcal{P}$ is a plan for $fp(GP)$.

- The proof for the second case is identical to the proof (soundness) of Lemma 5.5.

□

Now that we defined the above two lemmas, we are prepared to introduce the following theorem.

Theorem 5.17. *Let $GP = (I, Act, \neg A(c) \wedge \neg B(c))$. $fp(GP)$ has a plan if and only if $fp(GP)$ has a plan with a (possibly empty) sequence of positive actions followed by at most two negative actions.*

Proof. We prove by cases.

- Assume $A(c) \notin I$ and $B(c) \notin I$. It is trivial. The plan will be of length 0 and the theorem holds immediately.

- Assume $A(c) \in I$ and $B(c) \in I$. Using Lemma 5.5 and in particular the soundness, it is shown that $fp(GP)$ for this case has a plan if and only if $fp(GP)$ has a plan, which contains a (possibly empty) sequence of positive actions, followed by at most two negative actions.

- Assume $A(c) \notin I$ and $B(c) \in I$ ($A(c) \in I$ and $B(c) \notin I$). Using Lemma 5.6, the theorem follows immediately. □

Now, we have all the means to introduce an algorithm that checks whether there exists a plan for $fp(I, Act, \neg A(c) \wedge \neg B(c))$. The algorithm runs in *polynomial* time. By means of the above

lemmas and theorem, we will show the correctness of the algorithm. We call it negneg*Goal*.

Input: Given $GP = (I, Act, \neg A(c) \wedge \neg B(c))$.
Output: The algorithm returns **true** if it finds a plan for $fp(GP)$, **false** otherwise.
begin
case 1: **if** *B(c)* $\in I$ *and A(c)* $\in I$ **then**
    if $\exists$ *a plan for* $fp(I, Act', \neg A(c))$, *where* $Act' = Act \setminus \{(A \ominus B)\}$ *and* $\exists$ *a plan for* $fp(I, Act, \neg B(c))$ **then**
        return **true**
    else
        if $\exists$ *a plan for* $fp(I, Act, \neg A(c))$ *and* $\exists$ *a plan for* $fp(I, Act', \neg B(c))$, *where* $Act' = Act \setminus \{(B \ominus A)\}$ **then**
            return **true**
        else
            return **false**
        end
    end
end

case 2:
if *B(c)* $\notin I$ *and A(c)* $\notin I$ **then**
    return **true**
end

case 3:
if *B(c)* $\in I$ *and A(c)* $\notin I$ **then**
    if $\exists$ *a plan for* $fp(I, Act', \neg B(c))$, *where* $Act' = Act \setminus \{(A \oplus C_j) \mid \exists C_j : (A \oplus C_j) \in Act\}$ **then**
        return **true**
    else
        if $\exists$ *a plan for* $fp(I, Act', \neg B(c))$, *where* $Act' = Act \setminus \{(B \ominus A)\}$ *and* $\exists$ *a plan for* $fp(I \cup A(c), Act, \neg A(c))$ **then**
            return **true**
        else
            if $\exists$ *a plan for* $fp(I, Act, \neg B(c))$ *and* $\exists$ *a plan for* $fp(I, Act', \neg A(c))$, *where* $Act' = Act \setminus \{(A \ominus B)\}$ **then**
                return **true**
            else
                return **false**
            end
        end
    end
end

case 4:

if *B(c)* $\notin I$ *and A(c)* $\in I$ **then**
  if $\exists$ *a plan for* $fp(I, Act', \neg A(c))$, *where* $Act' = Act \setminus \{(B \oplus C_j) \mid \exists C_j : (B \oplus C_j) \in Act\}$ **then**
    return **true**
  else
    if $\exists$ *a plan for* $fp(I, Act', \neg A(c))$, *where* $Act' = Act \setminus \{(A \ominus B)\}$ *and* $\exists$ *a plan for* $fp(I \cup A(c), Act, \neg B(c))$ **then**
      return **true**
    else
      if $\exists$ *a plan for* $fp(I, Act, \neg A(c))$ *and* $\exists$ *a plan for* $fp(I, Act', \neg B(c))$, *where* $Act' = Act \setminus \{(B \ominus A)\}$ **then**
        return **true**
      else
        return **false**
      end
    end
  end
end

Algorithm 5.3: *negneg*Goal

The *negneg*Goal algorithm takes as input $GP = (I, Act, \neg A(c) \wedge \neg B(c))$. It does four main checks depending whether $A(c)$ and/or $B(c)$ occur in the initial database. The algorithm returns **true** if there exists a plan, and **false** otherwise. It is easy to see that the algorithm runs in *polynomial* time, since basically the main checks that it does, we have proved to be in **Nlogspace**.

Theorem 5.18. *Assume* $GP = (I, Act, \neg A(c) \wedge \neg B(c))$. *There exists a plan for* $fp(GP)$ *if and only if the algorithm* negneg*Goal returns* **true**.

Proof. Depending on whether $A(c)$ and $B(c)$ are in I, we consider each case separately.

- Assume $A(c) \notin I$ and $B(c) \notin I$. The plan for $fp(GP)$ in this case, is of length 0. The algorithm, through *case 2*, returns **true**.
- Assume $A(c) \in I$ and $B(c) \in I$. By Lemma 5.5 the above theorem holds.
- Assume $A(c) \in I$ and $B(c) \notin I$ (or $A(c) \notin I$ and $B(c) \in I$). By Lemma 5.6, this theorem holds.

□

Now, we are prepared to formally introduce the following theorem:

Theorem 5.19. *Deciding whether there exists a plan for* $fp(I, Act, \neg A(c) \wedge \neg B(c))$, *where* $\{A, B\} \subseteq N_C$ *and* $c \in N_I$, *can be done in* polynomial *time.*

5.3.4.4 Other Subcases

For all the cases that we have analyzed until now, we were able to conclude that if there exists a plan, then there is one with at most two negative actions. Furthermore, these deletions are performed only on the concept name that occurs in the negative literal of the goal. For each of the above problems, checking plan existence for the respective cases, was basically reduced to checking plan existence, considering each literal in a goal as a separate goal formula.

Next, we will investigate three problematic cases which seem not to have an algorithm that runs in polynomial time. The cases are the following.

- $GP = (I, Act, A(c) \wedge \neg B(d))$ where $A \neq B$ and $c \neq d$.
- $GP = (I, Act, A(c) \wedge \neg A(d))$ where $A \neq B$ and $c \neq d$.
- $GP = (I, Act, \neg A(c) \wedge \neg B(d))$ where $A \neq B$ and $c \neq d$.

These three cases of fixed domain plan existence with basic actions, often require plans with more than just two negative actions. Moreover, their plans might require nested negative actions performed on atoms, other than the ones in the goal formula. Hence, checking plan existence considering each literal of the goal formula as a separate goal formula, does not ensure completeness. To get a better intuition, we give some examples:

- For $GP = (I, Act, A(c) \wedge \neg B(d))$, the problematic case is when $A(c) \notin I$ and $B(c) \notin I$. The intuition is that trying checking plan existence for $fp(I, Act, A(c) \wedge \neg B(d))$ does not necessarily mean to just check plan existence for $fp(I, Act, A(c))$, since the latter might have a plan, but it might add $B(d)$ and there might be no plan for $fp(I, Act, \neg B(d))$. Hence, one might be forced to check for a plan for the whole $fp(GP)$. The other cases can be easily solves as in the *posneg*Goal algorithm.

  Example 5.10. *Assume a* GDPP $GP = (I, Act, A(c) \wedge \neg B(d))$ *is given as follows.*

  $$I = \{D(c), D(d), C(d)\},$$

  $$Act = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5\},$$

  where:

  $$\begin{aligned} \rho_1 &= A \oplus B, \\ \rho_2 &= B \oplus D, \\ \rho_3 &= D \ominus C, \\ \rho_4 &= B \ominus F, \\ \rho_5 &= F \oplus C. \end{aligned}$$

  In this example, $I \models \neg B(c)$ *and* $I \not\models A(c)$*. It is not hard to notice, that a plan for* $fp(I, Act, A(c))$ *is:*

  $$\mathcal{P} = \langle \rho_2, \rho_1 \rangle$$

  The final state of the trajectory $\mathcal{T}_\mathcal{P}$ *contains* $B(d)$*. A way to transform* $\mathcal{P}$ *to a plan for* $fp(GP)$ *is by adding the sequence* $\langle \rho_5, \rho_4 \rangle$*. But, if* ρ_4 *and* ρ_5 *are not given in the set* Act*, then the only way to achieve a plan is the following:*

  $$\mathcal{P} = \langle \rho_3, \rho_2, \rho_1 \rangle.$$

Hence, while checking plan existence for $fp(I, Act, A(c))$, there needs to be a check at each step, if it is possible to avoid adding the constant d.

Even though, considering the plan-existence decision problem for this case, seems harder than for the cases analyzed until now, we are able to extract a subcase that can still be checked in *polynomial* time.

Lemma 5.7. *Let $GP = (I, Act, A(c) \wedge \neg B(d))$ be a graph database planning problem where $B(d) \in I$. $fp(GP)$ has a plan if and only if $fp(I, Act, A(c))$ has a plan and $fp(I, Act, \neg B(d))$ has a plan.*

Proof. The proof to this lemma is identical to the proof of Lemma 5.4. □

The intuition is that when $B(d) \in I$, there is no need to do any checks when finding a plan for $fp(I, Act, A(c))$. Thus, if there is a plan for $fp(I, Act, A(c))$ and a plan for $fp(I, Act, \neg B(d))$, a concatenation of the two plans (in this order) is a plan also for $fp(GP)$.

- The second case is when $GP = (I, Act, A(c) \wedge \neg A(d))$. Except for the case where the initial database is already a goal state, plans for $fp(GP)$ might contain several negative actions, even though a single negative atoms is in the goal formula.

  Example 5.11. *Assume a* GDPP *$GP = (I, Act, A(c) \wedge \neg A(d))$ is given as follows.*

  $$I = \{D(c), D(d), C(c)\}.$$

  and

  $$Act = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5\},$$

  where:

  $$\begin{aligned} \rho_1 &= A \oplus B, \\ \rho_2 &= B \oplus D, \\ \rho_3 &= A \ominus F, \\ \rho_4 &= F \oplus A, \\ \rho_5 &= F \ominus C. \end{aligned}$$

  In this instance, $A(c) \notin I$ and $A(d) \notin I$. This is a very basic example with just 5 actions and 3 atoms in the initial database. Indeed, it is not hard to notice that the only plan for $fp(I, Act, A(c) \wedge \neg A(d))$ involves all actions and is as follows:

  $$\mathcal{P} = \langle \rho_2, \rho_1, \rho_4, \rho_5, \rho_3 \rangle$$

  The first two actions add $A(c)$, but also $A(d)$ to the respective state. Adding ρ_4 and then ρ_5 would delete both of these atoms and it would not be a plan still. But, applying $(F \ominus C)$ after action $(F \oplus A)$, a state that contains $F(d)$ doesn't contain $F(c)$. The last action $(A \ominus F)$ deletes $A(d)$, producing a goal state for $fp(I, Act, A(c) \wedge \neg A(d))$.

As the above example shows, for a very simple example with a goal formula that contains a positive and a negative literal, a plan might require more than one negative action.

Now, consider $Act = \{\rho_1^{tr}, \rho_2^{tr}\}$, where ρ_1^{tr} and ρ_2^{tr} are as above. Clearly, there exists a plan for $fp(I, Act, A(c))$, namely $\langle \rho_2^{tr}, \rho_1^{tr} \rangle$. In addition, there is a plan of length 0 for $fp(I, Act, \neg A(d))$ since $I \models \neg A(d)$. In contrast there is no plan for $fp(I, Act, A(c) \wedge \neg A(d))$. Thus, checking if there is a plan considering each conjunct of G separately can not ensure the existence of a plan for G.

Next, we consider a case when a single negative action is needed, but it should be embedded in the plan for $fp(I, Act, A(c))$. Let us give the following example.

Example 5.12. *Assume a* GDPP $GP = (I, Act, A(c) \wedge \neg A(d))$ *is given as follows.*

$$I = \{D(c), D(d), C(c)\}.$$

and

$$Act = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5\},$$

where:

$$\begin{aligned} \rho_1 &= A \oplus B, \\ \rho_2 &= B \oplus D, \\ \rho_3 &= D \ominus C. \end{aligned}$$

Comparing this example to the previous one, it is easily seen that there is an action $(D \ominus C)$. *Clearly, applying* ρ_3^{tr} *before* $\langle \rho_2^{tr}, \rho_1^{tr} \rangle$ *to* I *results in a state that does not contain* $A(d)$ *and contains* $A(c)$. *Hence, instance.*

$$\mathcal{P} = \langle \rho_3, \rho_2, \rho_1 \rangle$$

is a plan for $fp(GP)$.

- The last case we consider is when $GP = (I, Act, \neg A(c) \wedge \neg B(d))$. The intuition is similar to the previous cases. We will illustrate this case with an example.

  Example 5.13. *Assume a* GDPP $GP = (I, Act, \neg A(c) \wedge \neg B(d))$ *is given as follows.*

  $$I = \{D(c), D(d), C(d), A(c), B(d), F(c)\}.$$

  $$Act = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5, \rho_6, \rho_7\},$$

  where:

  $$\begin{aligned} \rho_1 &= A \ominus C, \\ \rho_2 &= C \oplus B, \\ \rho_3 &= B \oplus D, \\ \rho_4 &= D \ominus C, \\ \rho_5 &= B \ominus A, \\ \rho_6 &= A \oplus C, \\ \rho_7 &= C \ominus F. \end{aligned}$$

A plan for this very simple example is:

$$\mathcal{P} = \langle \rho_3, \rho_2, \rho_1, \rho_7, \rho_6, \rho_5 \rangle.$$

For this example, it can be seen that one can not construct a plan for $fp(I, Act, \neg A(c))$ and a plan for $fp(I, Act, \neg B(d))$ and combine those two. The intuition is that trying to find a plan for one of them, other negative actions are necessary for preserving the other. Now, assume that $B(d) \notin I$. $\mathcal{P}$ is still a plan for $fp(I, Act, \neg A(c) \wedge \neg B(d))$. But, it is easy to see that a shorter plan exists. A simple plan involves ρ_4, which needs to be applied first so that $B(d)$ is not added to some state. The plan is as follows:

$$\mathcal{P} = \langle \rho_4, \rho_3, \rho_2, \rho_1 \rangle.$$

These nested negations have in their right hand side atoms, other than the ones that appear in the goal formula. To assure completeness, we would need to come up with an algorithm that keeps track at each step for constant c and d and checks for plans to delete one or the other depending on the goal.

In Section 5.3.1, we proved that deciding if there exists a plan for a fixed domain planning problem with atomic actions that contain only concept names is **NP**-*hard*. Each of the three above problematic cases occurs in the goal formula of the instance built by the reduction from 3-colorability.

5.4 Encoding to Propositional STRIPS Planning

In this section, we encode a planning problem over a fixed domain $fp(I, Act, G)$ where Act is a finite set of atomic actions with only concept names to an instance of propositional STRIPS planning. We encode each action as a set of *Pre⇒Post* operators. There are two of these operators for each constant in $dom(GP)$ and for each action in Act. Each of these operators checks if an atom is at the respective state, and if yes, it adds or deletes an atom to the next state and jumps to the next operator. Otherwise, it jumps to the next operator without changing the state. When encoding to STRIPS, new literals for each action, with indexes that are bound by the size of $dom(GP)$ are added. These literals enforce the jumps to all possible operators of an action. The intuition behind it is that we want each action to be encoded such that the state that is obtained after applying an action for $fp(GP)$ satisfies the same formulas that the state that is obtained by applying the respective translated action to STRIPS.

Below, we build the reduction and we show its correctness.

5.4.1 Reduction

Let $GP = (I, Act, G)$ be a graph database planning problem, where Act is a set of atomic actions with only concept names. Let $dom(GP) = \{c_1, c_2, ..., c_n\}$. For simplicity, assume that $\{A, B, r, p\}$ are the only concept names and role names occurring in GP, where $A, B \in N_I$ and $r, p \in N_R$. Finally, let Act be a set of 2 actions as follows: $Act = \{\rho_1 = (A \oplus B), \rho_2 = (A \ominus B)\}$.

Next we built the reduction to *propositional STRIPS planning* $SP_{GP} = (\mathcal{P}, \mathcal{O}, \mathcal{I}, \mathcal{G})$, but before we state the atoms that we will be using in this reduction as follows:

- There are new atoms $in(c_i, A)$, $in(c_i, B)$, $1 \leq i \leq n$ for each $A(c_i)$, $B(c_i)$ that might occur at some state $s \in S$ of $fp(GP)$.
- There are new auxiliary atoms $N_1, N_2^{\rho_1}, ..., N_n^{\rho_1}, N_2^{\rho_2}, ..., N_n^{\rho_2}$ which will be used to capture the whole result of actions ρ_1 and ρ_2 in STRIPS.

Now, we are prepared to introduce the following reduction.

- The set of *conditions* is defined as follows.

$$\begin{aligned}\mathcal{P} =\{&in(c_1, A), in(c_2, A), ..., in(c_n, A),\\ &in(c_1, B), in(c_2, B), ..., in(c_n, B),\\ &N_1, N_2^{\rho_1}, ..., N_n^{\rho_1},\\ &N_2^{\rho_2}, ..., N_n^{\rho_2}\},\end{aligned}$$

  where facts over all possible combinations of concept names and individual names are contained. Moreover, the auxiliary literals $N_i^{\rho^j}$ for $2 \leq i \leq n$ and $1 \leq j \leq 2$ and N_1 will be used when constructing actions.

  The size of it is $|\mathcal{P}| = 4 \times n - 1$.

- The set of actions $\{\rho_1, \rho_2\}$ is encoded into STRIPS as follows.

$$\begin{aligned}\rho_1' =\{&N_1 \wedge in(c_1, B) \Rightarrow in(c_1, A) \wedge N_2^{\rho_1} \wedge \neg N_1\\ &N_1 \wedge \neg in(c_1, B) \Rightarrow N_2^{\rho_1} \wedge \neg N_1,\\ &N_2^{\rho_1} \wedge in(c_2, B) \Rightarrow in(c_2, A) \wedge N_3^{\rho_1} \wedge \neg N_2^{\rho_1},\\ &N_2^{\rho_1} \wedge \neg in(c_2, B) \Rightarrow N_3^{\rho_1} \wedge \neg N_2^{\rho_1},\\ &...,\\ &N_n^{\rho_1} \wedge in(c_n, B) \Rightarrow in(c_n, A) \wedge N_1 \wedge \neg N_n^{\rho_1},\\ &N_n \wedge \neg in(c_n, B) \Rightarrow N_1 \wedge \neg N_n^{\rho_1}\},\end{aligned}$$

$$\begin{aligned}\rho_2' =\{&N_1 \wedge in(c_1, B) \Rightarrow \neg in(c_1, A) \wedge N_2^{\rho_2} \wedge \neg N_1,\\ &N_1 \wedge \neg in(c_1, B) \Rightarrow N_2^{\rho_2} \wedge \neg N_1,\\ &N_2^{\rho_2} \wedge in(c_2, B) \Rightarrow \neg in(c_2, A) \wedge N_3^{\rho_2} \wedge \neg N_2^{\rho_2},\\ &N_2^{\rho_2} \wedge \neg in(c_2, B) \Rightarrow N_3^{\rho_2} \wedge \neg N_2^{\rho_2},\\ &...,\\ &N_n^{\rho_2} \wedge in(c_n, B) \Rightarrow \neg in(c_n, A) \wedge N_1 \wedge \neg N_n^{\rho_2},\\ &N_n \wedge \neg in(c_n, B) \Rightarrow N_1 \wedge \neg N_n^{\rho_2}\}.\end{aligned}$$

  Each action has $2n$ Pre⇒Post operators, where n is the number of constants that occur in $dom(GP)$. For each action, for each constant c_j, there are two operators. Each of them checks if the state $s \in S$, it is applied to, is such that $s \models B(c_j)$ or $s \not\models B(c_j)$. In each case, it makes the necessary updates to the states.

Notice that there are $4n$ Pre⇒Post operators.

- The initial state $\mathcal{I}$ is encoded as follows.

$$in(c_i, A) \in \mathcal{I} \textit{ iff } A(c_i) \in I, \textit{ and } in(c_i, B) \in \mathcal{I} \textit{ iff } B(c_i) \in I, \textit{ and } N_1 \in \mathcal{I}.$$

  The initial state contains all literals $in(c_i, A)$, resp. $in(c_i, B)$ s.t. the respective $A(c_i)$, $B(c_i)$ is in the initial database of $fp(GP)$. Moreover, to initialize the checks through the *Pre⇒Post* operators, we add N_1 to $\mathcal{I}$.

  The size of the initial state is $|\mathcal{I}| = |I| + 1$.

- The goal formula $\mathcal{G}$ is such that

$$A(c_j) \in G \textit{ iff } in(c_j, A) \in \mathcal{G}$$

  for all positive facts in G and

$$\neg A(c_j) \in G \textit{ iff } \neg in(c_j, A) \in \mathcal{G}$$

  for all negative facts in G.

  The size is $|\mathcal{G}| = G$.

It is easy to see that the reduction can be built in polynomial time in the size of *constants* that belong to $dom(GP)$ and concepts that occur in GP. For n constants and n concepts and 2 actions, it is $\mathcal{O}(4n^2)$.

5.4.2 Correctness of the Encoding

Now we prove that the above encoding is correct. We introduce the following lemma and theorem.

Lemma 5.8. *Let $P = \langle \rho_1, ..., \rho_n \rangle$ be a finite sequence of actions from Act.*
Let $\mathcal{T}_P = \langle \langle I, \rho_1, s_1 \rangle, ..., \langle s_{n-1}, \rho_n, s_n \rangle \rangle$, $\{I, s_1, ..., s_n\} \subseteq S$ and let $Result(\mathcal{I}, (\rho'_1, ..., \rho'_n)) = s'_n$, where $\{\mathcal{I}, s'_n\} \subseteq \mathcal{P}$. $A(c_i) \in s_n$ if and only if $in(c_i, A) \in s'_n$.

Proof. We prove the claim by induction on the length of a sequence of actions:

Induction base.
n=0. $Result(I, ()) = \mathcal{I}$ and $I = \mathcal{T}_{\langle\rangle}$. By construction the claim holds for I and $\mathcal{I}$.

n=1. We distinguish between two cases.

- $\rho_1 = A \oplus B$.
  $\mathcal{T}_{\rho_1} = \langle I, \rho_1, s_1 \rangle$, where $s_1 = I \cup \{A(c_i) \mid B(c_i) \in I\}$.
  By construction and by definition using $N_1, N_2^{\rho_1}, ..., N_n^{\rho_1}, ..., N_n^{\rho_1}, ..., N_n^{\rho_n}$,, there will be checked all *Pre⇒Post* of ρ_1. Only the ones such that $in(c_i, B) \in \mathcal{I}$ will add an atom $in(c_i, A)$ to I and also add $N_i^{\rho_1}$. For each $N_i^{rho_1}$ there will be two *Pre⇒Post*. Either $in(c_i, A) \in I$, or it is not. In the first case add $in(c_i, B) \in \mathcal{I}$, add $N_{i+1}^{rho_1} \in \mathcal{I}$ and delete

$N_i^{rho_1}$, $1 < i \leq n$. The idea will be repeated until all individuals are checked (if they belong or do not belong in B). In the last step N_1 is added to the set, so that other actions can be initialized. Thus, $Result(I, (A \oplus B)) = I \cup \{in(c_i, A) \mid in(c_i, B) \in I\} \cup N_1$. Hence, the claim is true.

- $\rho_1 = A \ominus B$.
  $\mathcal{T}_{\rho_1} = \langle I, \rho_1, s_1 \rangle$, where $s_1 = I \setminus \{A(c_i) \mid B(c_i) \in I\}$.
  Using the same arguments as above all literals $in(c_i, A)$ such that $in(c_i, B) \in \mathcal{I}$ will be deleted from $\mathcal{I}$. That is, $Result(I, (A \ominus B)) = I \cup N_I \setminus \{in(c_i, A) \mid in(c_i, B) \in I\}$. Hence, the claim is true.

Induction Hypothesis. The claim holds for a sequence of length $\leq n$.
Induction Step. We prove the claim for length $n + 1$. By hypothesis up to length n the claim is true, hence $A(c_i) \in s_n$ if and only if $in(c_i, A) \in s'_n$. Using the same arguments as in the *induction base*, the claim is true also for state $s_{n+1} \in S$ and $s'_{n+1} \in \mathcal{P}$. □

Theorem 5.20. *Assume $GP = (I, Act, G)$, where Act is a set of atomic actions with only concept names. One can build in polynomial time an instance of propositional STRIPS planning SP_{GP} s.t. $fp(GP)$ has a plan if and only if SP_{GP} has a plan.*

Proof. By Lemma 5.8, we proved that applying a sequence of actions to a database and applying the corresponding STRIPS sequence to the corresponding STRIPS state, results in states s.t. $s_n \models A(c_i)$ iff $s'_n \models in(c_i, A)$. The latter and by construction, immediately follows that there is a plan for $fp(GP)$ with a goal state $s_n \models G$ produced by the trajectory if and only it is a plan for SP_{GP} with a goal state $s'_n \models \mathcal{G}$. □

By the above theorem, we showed that the presented encoding into STRIPS is correct. Next, we perform a short analysis of some of the problems we encountered, when trying to encode more expressive actions.

5.4.3 Analysis

We investigated whether planning in our setting can be reduced to STRIPS planning. Considering such possible encoding one can get some insights about the relationships between our formalism and the STRIPS formalism, which represents a classical approach to planning. In addition, in this way STRIPS planners can be exploited to solve our planning problem.

We have encoded into propositional STRIPS only the already proven **NP**-hard case with a general goal formula and atomic actions with only concept names. By construction, it is obvious that negative actions require negated literals in the post-conditions of the operators. Deciding if there exists a plan for a planning problem where operators have negative post-conditions, it is known to be **PSPACE**-complete in propositional STRIPS. Therefore, the above reduction shows a **PSPACE** upper-bound complexity, which was already shown in Section 5.1.

In turn, it seems not so possible to encode arbitrary concepts and roles, which correspond to first order formulas. The latter are often with a complex structure with disjunction. It follows that the limited expressibility of *Pre⇒Post* operators as only conjunction of literals can not capture the actions with more complex concepts and roles. Variables, on the other hand, can be encoded

using literals $in(c_i, c_i)$, for all constants of $dom(GP)$. There would be n operators for an action of the form $(A \oplus X)$, if $|dom(GP)| = n$. The operators would have form $in(c_i, c_i) \Rightarrow in(c_i, A)$ and there would be n such operators for the n constants. Since we assume that the actions are grounded when a performing a plan, it follows that if X is instantiated by c_i, then exactly one of the operators will satisfy the pre-condition, namely only the operator having $in(c_i, c_i)$ in *Pre*. Then the postcondition is $in(c_i, A)$, hence the respective state is updated by adding this literal. Roles can also be encoded similarly to concept names, as described above. The only difference is that for a role and n constants, there would be $n \times n$ tuples in $\mathcal{P}$. Hence, for each action with a role, there would be exactly n^2 *Pre*⇒*Post* corresponding operators.

In addition, STRIPS operates on a closed domain. The set of conditions should be specified in advance. Thus, fresh constants cannot be introduced by the operators. This excludes the possibility of an encoding of the general planning problem to propositional STRIPS.

CHAPTER 6

Deciding General Plan-Existence

Until now, we have considered only planning problems over a fixed domain. It was shown in Chapter 4 that deciding fixed domain plan-existence is **PSPACE**-*complete*. We showed the upper-bound by introducing an algorithm that runs in non-deterministic polynomial space. At each step, it non-deterministically chooses the transactions and its memory is polynomially bounded by the size of the maximal state that can be constructed over the signature of GP.

Unfortunately, for the general planning problem where the interpretation domain is infinite, the deciding the plan-existence problem seems to be more complicated. A plan for a given instance might require the use of constants that do not occur in $dom(GP)$ (see Example 6.4). Furthermore, it seems hardly possible that one needs a polynomial amount of fresh constants, i.e. constants that do not occur in $dom(GP)$. Only if this is the case, a polynomial bound on the maximal state can be ensured. Then deciding plan-existence is in **PSPACE**. In turn, our intuition is that it already is **EXPTIME**-*hard*.

Instead, under some syntactic restrictions on the set of predefined actions and the goal formula allowed as input, we could find cases whose plan-existence can be still decided in **PSPACE**. We will formally introduce and prove this intuition in Section 6.2.

We will start this chapter by providing the main features that enforce the use of constants that do not occur in $dom(GP)$. We call these constants, *new (fresh) constants*, and present them in Section 6.1.

Note: For any given GDPP, GP, the set of plans for $fp(GP)$ is contained in the set of plans for $p(GP)$. The intuition is that $fp(GP)$ is a special case of $p(GP)$ where no fresh constants are considered. Hence, if $fp(GP)$ has a plan, then $p(GP)$ has a plan. Since this is trivial, we will not formally prove this.

6.1 Triggers For New Constants

Let $p(I, Act, G)$ be a general planning problem corresponding to a graph database planning problem. Following our intuition, the main triggers that might enforce the use of new constants

in a plan, are transactions with preconditions, transactions with occurrences of negation in concepts/roles, the existence of negative atoms in the goal formula combined with concepts that contain existential/universal restrictions.

In this section we will consider each case separately, by illustrating the respective intuition with an example.

- Firstly we will give a simple example from the database of some research institute. We will see through this example that having a negative literal in the goal formula and a concept with existential quantification in a plan along with positive preconditions, might require a plan that uses some fresh constant.

  Example 6.1. *Let $GP = (I, Act, G)$ be as follows:*

$$\begin{aligned} I = \{&Project(P20840),\\ &Employee(E01), Employee(E03),\\ &ProjectEmployee(E01),\\ &worksFor(E01, P20840)\}, \end{aligned}$$

$$Act = \{\rho_1^{tr}, \rho_2^{tr}, \rho_3^{tr}\},$$

where:
$\rho_1^{tr} = (X : \textbf{Project} \ \wedge \neg(X : Employee))\ ?\ worksFor \oplus \{(E03, X)\} : \epsilon$,
$\rho_2^{tr} = (X : \textbf{Project}\ \neg(X : Employee))\ ?\ ProjectEmployee \oplus \exists worksFor.X : \epsilon$,
$\rho_3^{tr} = (Project \oplus \{X\})$,

$$G = ProjectEmployee(E03) \wedge \neg worksFor(E03, P20840).$$

This example of a graph database planning problem asks for computing a sequence of actions that reaches a state that contains ProjectEmployee$(E03)$. Meanwhile, it requires that no atom worksFor$(E03, P20840)$ is added. Furthermore, the set of predefined transactions contains just two transactions with preconditions. The precondition in both of them is that the instantiation of variable X should be a Project *and not an* Employee*. Obviously, by instantiating X ins ρ_2^{tr} with $P20840$, one obtains immediately a plan for $p(I, Act, ProjectEmployee(E03))$. But, it is easily seen that this can not be a plan for for $p(I, Act, \neg worksFor(E03, P20840))$, since automatically $worksFor(E03, P20840)$ will be added.*

Observe that a plan for $p(I, Act, G)$ is as follows.

$$\mathcal{P} = \langle (Project \oplus P11111), (P11111 : \textbf{Project} \wedge \neg(P11111 : Employee))\ ?$$

$$worksFor \oplus \{(E03, P11111)\} : \epsilon, (P11111 : \textbf{Project} \wedge \neg(P11111 : Employee))\ ?$$

$$ProjectEmployee \oplus \exists worksFor.P11111 : \epsilon\rangle$$

Thus, a new project is initially added by the atomic action ρ_3^{tr}. Then by applying ρ_1^{tr} and ρ_2^{tr} where X is instantiated with this new project, a desired plan is obtained.

Observe that X can be instantiated with any project constant other then $P20840$. It follows that there are infinite possibilities of instantiations, hence infinitely many plans for this instance.

- Next, we give an example, where the initial database is empty and the set of predefined transactions contains a single transaction with no preconditions. The goal formula has a negative and a positive literal. We will see that in order to have a plan for this instance, a new constant will be needed.

  Example 6.2. *Assume $GP = (I, Act, G)$ is given as follows.*

  $$I = \{\},$$

  $$Act = \{\rho_1^{tr}\},$$

  where:
  $\rho_1^{tr} = Project \oplus \{X\} \circ worksFor \oplus \{(E03, X)\} \circ ProjectEmployee \oplus \exists worksFor.X.$

  $$G = \neg Project(P20840) \wedge \neg Project(E03) \wedge ProjectEmployee(E03).$$

  For this instance, it can be seen that X cannot be instantiated neither with $P20840$, nor with $E03$. A new constant is needed for a plan to exist. E.g. by instantiating X with $P11111$, a plan for this instance is:

  $$\mathcal{P} = \langle Project \oplus P11111 \circ worksFor \oplus \{(E03, P11111)\} \circ$$

  $$ProjectEmployee \oplus \exists worksFor.P11111 \rangle.$$

  Note that the role of precondition here was played by having a transaction as a sequence of atomic actions. Thus, X, which is the same variable repeated in the transaction, should be a project.

- Now, we will consider the case where the set of predefined transactions contain transactions with negative preconditions. It seems that the main triggers that enforce using fresh constants are exactly preconditions with negation. Even, for an instance with a positive goal formula and transactions that compute only addition, it seems hardly possible.

  Example 6.3. *Assume a $GP = (I, Act, G)$ is given as follows.*

  $$\begin{aligned} I = \{&Project(P20840), \\ &Employee(E01), Employee(E03), \\ &ProjectEmployee(E01), \\ &PermanentEmployee(E04), \\ &worksFor(E01, P20840)\}, \end{aligned}$$

  $$Act = \{\rho^{tr}\},$$

  where:
  $\rho^{tr} = (\neg(X : Project) \wedge \neg((Y, X) : worksFor) \wedge \neg(X : PermanentEmployee) \wedge$

$\neg(X : Employee))$? $worksFor \oplus \{(E03, X)\} \circ ProjectEmployee \oplus \exists worksFor.X : \epsilon$.

$$G = ProjectEmployee(E03).$$

The precondition of the transaction ρ^{tr} does not allow X to be neither $P20840$, nor $E01$, nor $E03$, and nor $E04$. No element from $dom(GP)$ can be used to instantiate X, since none of them satisfies the precondition. We are forced to introduce a fresh constants. Let's use the same constant $P1111$ as before. It is easy to see that this constant satisfies the precondition. Hence, a plan for this planning instance is the following:

$$\mathcal{P} = \langle(\neg(P1111 : Project) \wedge \neg((E03, P1111) : worksFor) \wedge$$

$$\neg(P1111 : PermanentEmployee) \wedge \neg(X : Employee))\ ?$$

$$worksFor \oplus \{(E03, P1111)\} \circ ProjectEmployee \oplus \exists worksFor.P1111 : \epsilon\rangle.$$

Note that this example was a basic example with a single positive literal in the goal formula and a transaction with two positive atomic actions. Thus, negative preconditions might negate all constants that belong to a concept name or role name in a state. This way they might enforce the use of a fresh constant each time they are applied. This might be the case, since when the same transaction is reapplied at some point in the plan, the fresh constant previously used, will be possibly negated by the precondition.

To understand better this intuition, let's consider again Example 6.3. Assume G is such that it requires a plan where ρ^{tr} will be reused and assume $worksFor(E03, P1111)$ will not be deleted further in the plan. Now, it is not hard to see that the second time ρ^{tr} will be applied, $P1111$ will not satisfy the precondition because of $worksFor(E03, P1111)$. Hence, the second conjunct of the precondition will not allow the first fresh constant to be reused. This means that another new constant should be introduced.

Observe that if one cannot find a bound on the length of the plan for the general planning problem, and transactions with preconditions that contain negation are allowed, it is hardly possible that one can find a polynomial bound on the number of fresh constants that will be added. Basically, it seems that allowing negative preconditions, increases the complexity for the decision problem of checking plan-existence, making it *unlikely* to be in **PSPACE**.

- At last, negative concepts in the transactions can also be viewed as an important trigger for fresh constants, because they behave in a similar way to negative preconditions. This will be seen clearly in the following example.

  Example 6.4. *Let $GP = (I, Act, G)$ be as follows:*

$$\begin{aligned} I = \{&Project(P20840),\\ &Employee(E01), Employee(E03),\\ &ProjectEmployee(E01),\\ &PermanentEmployee(E04),\\ &worksFor(E01, P20840)\}. \end{aligned}$$

$$Act = \{\rho_1^{tr}, \rho_2^{tr}\},$$

where:
$\rho_1^{tr} = ProjectEmployee \oplus \exists worksFor.(\neg Employee \sqcap \neg PermanentEmployee \sqcap \neg Project)$,
$\rho_2^{tr} = worksFor \oplus \{(X, Y)\}$.

$$G = ProjectEmployee(E03).$$

No transaction with precondition is contained in Act, but the negative concept names in the concept $(\neg Employee \sqcap \neg PermanentEmployee \sqcap \neg Project)$ in ρ^{tr} act like one. They do not allow the second element of $worksFor$ to be a constant that is used in the concept names $Employee$, $PermanentEmployee$, $Project$. It follows that none of the constants from $dom(GP)$ can instantiate Y. Only a fresh constants e.g. $P1111$ can be used so that a plan exists. The following sequence

$$\mathcal{P} = \langle (worksFor \oplus \{(E03, P1111)\}),$$

$$(ProjectEmployee \oplus \exists worksFor.(\neg Employee \sqcap \neg PermanentEmployee \sqcap \neg Project))\rangle$$

is a plan for this instance.

The above examples show that if some features are allowed, then deciding plan-existence might be more complex than **PSPACE**. But, fortunately, in the previous chapter we showed that fixed domain plan-existence can be decided in **PSPACE**. An interesting direction to consider is searching for the most expressive setting that uses fresh constants s.t. we have the property that for all plans that use fresh constants, for the same given instance, there exists a plan that does not use fresh constants. Hence, in the next section we impose some syntactic restrictions, and analyze the use of fresh constants, trying to find the most expressive setting that can be reduced to fixed domain plan-existence.

6.2 General Plan Existence With Syntactic Restrictions

In this section we will consider a special case of a graph database planning problem by imposing several syntactic restrictions on Act and G. We will be able to prove that deciding general plan-existence for this case can be turned into deciding fixed domain plan-existence. Since the latter, by Theorem 5.4, is shown to be **PSPACE**-*complete*, we will show that deciding plan-existence for the case under these restrictions is in **PSPACE**.

Note: During this section, we will view G as a conjunction of atoms expressed via concept names and role names.

We will start by showing that having fresh constants in a state will not increase its expressivity w.r.t to modeling the goal formula. In turn, we will show that a state is a goal state if and only if the state obtained by deleting the atoms, where some fresh constant occurs, is a goal state.

Lemma 6.1. *Assume $p(I, Act, G)$ is a GDPP. For all $s \in S$ the following holds: $s \models G$ iff $s' \models G$, where s' is obtained by removing from s all atoms that contain a constant c s.t. $c \notin dom(GP)$.*

Proof. We prove both directions separately:

$\Leftarrow$ Assume $s' \models G$. By assumption $s' \subseteq s$. In addition, by assumption G is a conjunction of positive and negative literals expressed via role names and concept names. Also s, s' are interpretations viewed as a set of atoms expressed the same way. Let G^+ be the set of positive literals in G and G^- the set of negative literals in G. Clearly, if $s' \models G$ then it holds that $G^- \cap s' = \emptyset$ and $G^+ \subseteq s'$. The latter holds also for s since $s' \subseteq s$, hence $G^+ \subseteq s$. Now we need to prove that $G^- \cap s = \emptyset$. Since $G^- \cap s' = \emptyset$, and by assumption s extends s' only with atoms that contain fresh constants, it is easy to see that $(s \setminus s') \cap G^- = \emptyset$. The latter shows that $G^- \cap s = \emptyset$. It follows that if $s' \models G$ then $s \models G$.

$\Rightarrow$ Assume $s \models G$. If $s \models G$ then it holds that $G^+ \subseteq s$ and $G^- \cap s = \emptyset$. The latter holds also for s', thus $G^- \cap s' = \emptyset$ since $s' \subseteq s$. Now we need to prove that $G^+ \subseteq s'$. By Definition 4.3, it is not hard to see that $(s \setminus s') \cap G^+ = \emptyset$. The latter and by the assumption that $s' \subseteq s$ shows $G^+ \subseteq s'$. The latter and $G^- \cap s' = \emptyset$ prove that $s' \models G$. □

Next, we will show that for every plan of some $p(GP)$ with only positive actions with no negation, no universal restrictions and no qualified number restrictions, there exists a plan with only constants from $dom(GP)$. We will star by showing that for any plan with k fresh constants, there is a plan with $k - 1$ fresh constants. Intuitively, if this holds then a plan with no fresh constants exists.

But before we must prove an auxiliary lemma that will be needed in the proof.

Definition 6.1. *Let $GP = (I, Act, G)$. We denote $s_{a_k \to a}$ to express that every a_k that occurs in s is substituted by a, where $a_k \notin dom(GP)$, a is an arbitrary constant from $dom(GP)$ and s is a state or an atom.*

Lemma 6.2. *Assume $p(I, Act, G)$. Let $\{s, s'\} \subseteq S$ be s.t. $s_{a_k \to a} \subseteq s'$, where $a_k \in N_I$, $a_k \notin dom(GP)$, a_k doesn't occur in s', and $a \in dom(GP)$. For all $o \in N_I$ that occur in s, it holds that: if $s \models C(o)$ then $s' \models C(o)_{a_k \to a}$, where C is an $\mathcal{ALCHOIQ}br$ role or an $\mathcal{ALCHOIQ}br$ concept with no negation, universal quantification and qualified number restrictions.*

Proof. We prove the claim by induction on the structure of C.

Induction base. As base case, we consider the cases where C is a concept name A, a variable X, a role name r, a tuple of variables (X, Y) and an inverse role r^-.

- Assume that for some o that occurs in s, $s \models A(o)$, where A is a concept name. It means that $A(o) \in s$. We consider two cases.
  - Assume $o \neq a_k$. Then by assumption $A(o) \in s'$. It follows that $s' \models A(o)$
  - Assume $o = a_k$. By assumption $A(a) \in s'$, hence $s' \models A(a_k)_{a_k \to a}$.

– Assume C is a variable. Clearly for every instantiation, the claim holds automatically. The argument is that we have assumed that states are interpretations, viewed as sets of atoms, where all constants and tuples are included, even though we do not specifically write them.
– Assume C is a tuple of variables expressing an $\mathcal{ALCHOIQ}br$ role. Also here, for every instantiation the claim holds.
– Assume that for some tuple of constants c that occur in s, $s \models r(c)$, where r is a role name. It means that $r(c) \in s$. We consider two cases.
  * Assume a_k does not occur in c. Then by assumption $r(c) \in s'$. Hence, $s' \models r(c)$
  * Assume a_k occurs in c. By assumption $r(c)_{a_k \to a} \in s'$, hence $s' \models r(c)_{a_k \to a}$.
– Assume that for some tuple of constants (c_1, c_2) that occur in s, $s \models p(c_1, c_2)$, where p is an inverse role of the form r^-, and r is a role name. It means that $r(c_2, c_1) \in s$. Arguing as above, $r(c_2, c_1)_{a_k \to a} \in s'$. Hence, $s' \models p(c_1, c_2)_{a_k \to a}$.

Induction hypothesis. The claim holds for R, R_1, C, C_1, where R, R_1 are $\mathcal{ALCHOIQ}br$ roles and C, C_1 are $\mathcal{ACLHOIQ}br$ concepts, where no negation, universal restrictions and qualified number restrictions occur.

Induction step. We prove that the claim holds for the following concept constructors: $\exists R$, $\exists R.C$, $R \sqcap R_1$, $R \sqcup R_1$, $C \sqcap C_1$, $C \sqcup C_1$.

– Assume that for some o that occurs in s, $s \models \exists R(o)$. By semantics of $\exists R$ and by Definition 2.2, it means that there exists a constant b that occurs in s, s.t. $s \models R(o, b)$ or $s \models R(b, o)$, depending on whether R is a role name or an inverse role, respectively. By hypothesis $s' \models R(o, b)_{a_k \to a}$, resp. $s' \models R(b, o)_{a_k \to a}$. It follows that $s' \models \exists R(o)_{a_k \to a}$.
– Assume that $s \models \exists R.C(o)$, for some o that occurs in s, . By semantics of $\exists R.C$ and by Definition 2.2, it means that there exists a constant b that occurs in s, s.t. $s \models R(o, b) \wedge C(b)$ or $s \models R(b, o) \wedge C(o)$, depending on whether R is a role name or an inverse role, respectively. By hypothesis $s' \models R(o, b)_{a_k \to a} \wedge C(b)_{a_k \to a}$, resp. $s' \models R(b, o)_{a_k \to a} \wedge C(o)_{a_k \to a}$. It follows that $s' \models \exists R.C(o)_{a_k \to a}$.
– Assume that $s \models (R \sqcap R_1)(c)$, for some tuple of constants c that occur in s. By semantics of entailment, $s \models R(c)$ and $s \models R_1(c)$. Now, by hypothesis $s' \models R(c)_{a_k \to a}$ and $s' \models R_1(c)_{a_k \to a}$. It follows that $s' \models (R \sqcap R_1)(c)_{a_k \to a}$.
– Assume that $s \models (R \sqcup R_1)(c)$, for some tuple of constants c that occur in s. By semantics of entailment, $s \models R(c)$ or $s \models R_1(c)$. W.l.o.g. assume that $s \models R(c)$. Now, by hypothesis $s' \models R(c)_{a_k \to a}$. It follows that $s' \models (R \sqcup R_1)(c)_{a_k \to a}$.
– Assume that $s \models (C \sqcap C_1)(c)$ and $s \models (C \sqcup C_1)(c)$. The argument for these two cases is the same as for $\mathcal{ALCHOIQ}br$ roles.

□

Now, we are prepared to introduce the following lemma, which makes use of Lemma 6.2.

Note: During the rest of the section, instead of writing $\mathcal{ALCHOIQ}br$ concept, resp. role with no negation, universal restrictions and qualified number restrictions, we will just write $\mathcal{ALCHOIQ}br$ concept, resp. role. We will implicitly assume the above, but not explicitly write the above restrictions. The same holds for Act.

Lemma 6.3. *Assume $GP = (I, Act, G)$, where Act is a finite set of positive atomic actions and G is a conjunction of positive atoms. For every plan of $p(GP)$ with k new constants, there exists a plan with $(k-1)$ new constants.*

Proof. Let $\mathcal{P}$ be a plan for $p(GP)$ and let $\mathcal{T}_\mathcal{P}$ be the corresponding trajectory with last state $s_n \in S$, s.t $s_n \models G$ as follows:

$$\mathcal{T}_\mathcal{P} = \langle\langle I, \rho_1, s_1\rangle\langle s_1, \rho_2, s_2\rangle ...\langle s_{l-1}, \rho_l, s_l\rangle\langle s_l, \rho_{l+1}, s_{l+1}\rangle ...\langle s_{n-1}, \rho_n, s_n\rangle\rangle,$$

where $\{s_1, ..., s_n\} \subseteq S$. Hence, the plan is of length n. W.l.o.g. assume that s_n is the first goal state in $\mathcal{T}_\mathcal{P}$. Consider the first action of the plan that introduces the k-th new constant. It will have one of the following forms:

1. $(A \oplus \{X\})$, or
2. $(r \oplus \{(X, Y)\})$.

Clearly, if actions of the form $(A \ominus \{X\})$, $(r \ominus \{(X, Y)\})$ introduce new constants they are redundant w.r.t the state they are applied to, since the state will not change. Also actions $(A \oplus \exists r.X)$ or $(A \oplus \forall r.X)$ cannot introduce new constants, because they do not change the state they are applied at. We will analyze each case separately.

Let $(A \oplus \{X\})$, or $(r \oplus \{(X, Y)\})$ occur at position $l < n$ in the plan s.t. they introduce a_k, the k-th fresh constant. Let a be an arbitrary constant from $dom(GP)$. We substitute each occurrence of a_k in the plan with some $a \in dom(GP)$, hence everywhere in the actions from length l to length n. Let $\mathcal{T}'_\mathcal{P}$ be the new trajectory

$$\mathcal{T}'_\mathcal{P} = \langle\langle I, \rho_1, s_1\rangle\langle s_1, \rho_2, s_2\rangle ...\langle s_{l-1}, \rho'_l, s'_l\rangle\langle s'_l, \rho'_{l+1}, s'_{l+1}\rangle ...\langle s'_{n-1}, \rho'_n, s'_n\rangle\rangle,$$

with the last state s'_n.

Claim 6.3.1. *We claim that $s'_n \models G$.*

To prove the above claim we first prove the following lemma.

Lemma 6.4. *$\forall l', l \leq l' \leq n$ it holds that: $(s_{l'})_{a_k \to a} \subseteq (s'_{l'})$, where $(s_{l'})_{a_k \to a}$ means that every a_k that occurs in atoms in $s_{l'}$ is substituted by a.*

Proof. We prove the Lemma 6.4 by induction on l'.

Induction base. Assume $l' = l$. We distinguish between two cases:

- The corresponding state transition in $\mathcal{T}_\mathcal{P}$ is $\langle s_{l-1}, (A \oplus \{a_k\}), s_l\rangle$ and the corresponding state transition in $\mathcal{T}'_\mathcal{P}$ is $\langle s_{l-1}, (A \oplus \{a\}), s'_l\rangle$. By Definition 3.7, $s_l = s_{l-1} \cup A(a_k)$ and $s'_l = s_{l-1} \cup A(a)$. Clearly, $(s_l)_{a_k \to a} = s'_l$.

– The corresponding state transition in $\mathcal{T}_{\mathcal{P}}$ is $\langle s_{l-1}, (r \oplus c), s_l \rangle$, where c is a tuple of constants where a_k might occur. The corresponding state transition in $\mathcal{T}'_{\mathcal{P}}$ is $\langle s_{l-1}, (r \oplus c_{a_k \to a}), s'_l \rangle$. By Definition 3.7, $s_l = s_{l-1} \cup r(c)$ and $s'_l = s_{l-1} \cup r(c)_{a_k \to a}$. Clearly, $(s_l)_{a_k \to a} = s'_l$.

Induction hypothesis. The Lemma 6.4 holds for $\forall l'', l \leq l'' \leq l'$. Hence, $(s_{l''})_{a_k \to a} \subseteq (s'_{l''})$.

Induction step. We prove now that the claim holds for length $(l' + 1)$. We consider all possible actions that might happen at length $(l' + 1)$. The corresponding state transition in $\mathcal{T}_{\mathcal{P}}$, resp. $\mathcal{T}'_{\mathcal{P}}$ is $\langle s_{l'}, \rho_{l'+1}, s_{l'+1} \rangle$, resp. $\langle s'_{l'}, \rho'_{l'+1}, s'_{l'+1} \rangle$. We need to prove that the claim holds for $s_{l'+1}$ and $s'_{l'+1}$.

– Assume $\rho_{l'+1} = \rho'_{l'+1} = (A \oplus C)$, where $A \in N_C$, C is an $\mathcal{ALCHOIQ}br$ concept. By Definition 3.7, $s_{l'+1} = s_{l'} \cup \{A(c) \mid s_{l'} \models C(c)\}$, where c is a constant that occurs in $s_{l'}$ and $s'_{l'+1} = s'_{l'} \cup \{A(c') \mid s'_{l'} \models C(c')\}$, where c' is a constant that occurs in $s'_{l'}$. W.l.o.g. assume $A(c)$ is added to $s_{l'}$. Hence, $s_{l'} \models C(c)$. By hypothesis the claim holds for $s_{l'}$ and $s'_{l'}$. The latter and by Lemma 6.2 it follows that, if $s_{l'} \models C(c)$ then $s'_{l'} \models C(c)_{a_k \to a}$. Hence, $A(c)_{a_k \to a}$ will be added to $s'_{l'}$. It follows that $(s_{l'+1})_{a_k \to a} \subseteq s'_{l'+1}$.
– Assume $\rho_{l'+1} = \rho'_{l'+1} = (r \oplus R)$, where $r \in N_R$, R is an $\mathcal{ALCHOIQ}br$ role. By Definition 3.7, $s_{l'+1} = s_{l'} \cup \{r(c) \mid s_{l'} \models R(c)\}$, where c is a tuple of constants that occurs in $s_{l'}$ and $s'_{l'+1} = s'_{l'} \cup \{r(c') \mid s'_{l'} \models R(c')\}$, where c' is a tuple of constants that occur in $s'_{l'}$. W.l.o.g. assume $r(c)$ is added to $s_{l'}$. Hence, $s_{l'} \models R(c)$. By hypothesis the claim holds for $s_{l'}$ and $s'_{l'}$. The latter and by Lemma 6.2 it follows that, if $s_{l'} \models R(c)$ then $s'_{l'} \models R(c)_{a_k \to a}$. Hence, $r(c)_{a_k \to a}$ will be added to $s'_{l'}$. It follows that $(s_{l'+1})_{a_k \to a} \subseteq s'_{l'+1}$.

□

We proved Lemma 6.4, which describes that substituting the last fresh constant by an arbitrary constant from $dom(GP)$, will never decrease the states produced after each state transition. By Lemma 6.1 (it is obvious that this lemma works also when deleting all atoms where a fixed new constant occurs), and by assumption that $s_n \models G$, it holds that the state s''_n obtained by deleting from s_n all atoms where a_k occurs, is s.t. $s''_n \models G$. By Lemma 6.4, $(s_n)_{a_k \to a} \subseteq s'_n$. Hence, it is obvious that $s''_n \subseteq s'_n$. Since G is a conjunction of positive atoms, it is easy to see that if $s''_n \models G$ and $s''_n \subset s'_n$, then $s'_n \models G$, which is what we wanted to show. □

Now, we are prepared to introduce the following theorem.

Theorem 6.1. *Assume $GP = (I, Act, G)$, where Act is a finite set of positive atomic actions with no negation, universal restrictions, qualified number restrictions and G is a conjunction of positive literals. If $p(GP)$ has a plan, then $fp(GP)$ has a plan.*

Proof. By Lemma 6.3, we proved that for every plan for $p(GP)$ with k fresh constants, there is a plan with $k - 1$ fresh constants. Let $\mathcal{P}$ be a plan for $p(GP)$. Hence, it is easy to

see that repeating the same procedure k times, a plan without fresh constants can be found. It follows, that also a plan with only constants from $dom(GP)$ exists. Thus, a plan $\mathcal{P}'$ for $fp(GP)$ exists. □

In turn, Theorem 6.1 does not apply for G with some negative literal. The following example is a contradiction that shows this claim.

Example 6.5. *Assume a* GDPP $GP = (I, Act, G)$ *where:*

$$I = \{\}.$$

$$\begin{aligned} Act = \{&\rho_1 = r \oplus \{(X, Y)\},\\ &\rho_2 = C \oplus \{X\},\\ &\rho_3 = A \oplus \exists r.C\}. \end{aligned}$$

$$G = A(c) \wedge \neg C(c).$$

A plan for $p(GP)$ *is the following:*

$$\mathcal{P} = \langle (r \oplus \{(c, a)\}), (C \oplus \{a\}), (A \oplus \exists r.C)\rangle.$$

It is easy to see that there is no plan that uses only constants from $dom(GP)$*. Hence, the new constant* a *is necessary for having a plan.*

Next, assume a GDPP $GP = (I, Act, G)$, where Act is a finite set of atomic actions and G is a conjunction of positive literals. We prove that for every plan of $p(GP)$, there exists a plan with no negative actions. But, before showing this, we introduce another lemma, identical to Lemma 6.2 without the restriction on some fresh constant a_k.

Lemma 6.5. *Assume* $p(I, Act, G)$*, where* $\{s, s'\} \subseteq S$ *and* $s \subseteq s'$*. It holds that: if* $s \models C(o)$ *then* $s' \models C(o)$*, where* C *is an* $\mathcal{ALCHOIQ}br$ *role or an* $\mathcal{ALCHOIQ}br$ *concept and* o *is a constant occurring in* s*.*

Proof. The proof is by induction on C, C is an $\mathcal{ALCHOIQ}br$ role or an $\mathcal{ALCHOIQ}br$ concept with no negation, universal quantification and qualified number restrictions. It is identical to parts of the proof of Lemma 6.2, when not considering the restriction on a_k. □

Now, we are prepared to introduce the following lemma, which states that from a plan for $p(I, Act, G)$ with a positive goal and some restrictions on the types of actions allowed, one can delete the negative actions, without affecting the reachability of the goal state.

Lemma 6.6. *Assume* $GP = (I, Act, G)$*,* Act *is a finite set of atomic actions with no negation, universal restrictions, qualified number restrictions and* G *is a conjunction of positive literals. If* $p(GP)$ *has a plan with* k *negative actions, then* $p(GP)$ *has a plan with* $k - 1$ *negative actions.*

Proof. Let $\mathcal{P}$ be a plan for $p(GP)$ and let $\mathcal{T}_\mathcal{P}$ be its corresponding trajectory

$$\mathcal{T}_\mathcal{P} = \langle\langle I, \rho_1, s_1\rangle\langle s_1, \rho_2, s_2\rangle ... \langle s_{l-1}, \rho_l, s_l\rangle\langle s_l, \rho_{l+1}, s_{l+1}\rangle ... \langle s_{n-1}, \rho_n, s_n\rangle\rangle,$$

where $\{s_1, ..., s_n\} \subseteq S$ and $s_n \models G$. Since G is a conjunction of only positive atoms expressed via concept names and role names, we can equivalently write the following $g_i \in s_n$, $\forall g_i$ that occur in G. Furthermore, let the k-th negative action be ρ_l and occur at length l in $\mathcal{T}_\mathcal{P}$. Now, let $\mathcal{T}_{\mathcal{P}'}$ be the new trajectory without ρ_l as follows

$$\mathcal{T}_{\mathcal{P}'} = \langle\langle I, \rho_1, s_1\rangle\langle s_1, \rho_2, s_2\rangle ... \langle s'_l, \rho_{l+1}, s'_{l+1}\rangle ... \langle s'_{n-1}, \rho_n, s'_n\rangle\rangle,$$

where $s'_l = s_{l-1}$. Thus, we need to prove the following claim:

Claim 6.6.1. *$s'_n \models G$, hence $g_i \in s'_n$, for all g_i that occur in G.*

To prove this, it is sufficient to prove that $s_n \subseteq s'_n$. Hence we introduce the following lemma.

Lemma 6.7. *For all l', $l \leq l' \leq n$ it holds that $s_{l'} \subseteq s'_{l'}$.*

Proof. We prove this lemma, by induction on l'.

Induction base. $l' = l$ Since ρ_l is a negative action, by Definition 3.7, it is easy to see that $s_l \subseteq s_{l-1} = s'_l$.

Induction hypothesis. For all l', $l \leq l' \leq n$, $s_{l'} \subseteq s'_{l'}$.

Induction step. We prove that the claim holds for states at length $l' + 1$. Thus, we need to prove that $s_{l'+1} \subseteq s'_{l'+1}$. We distinguish between different actions that might occur at length $l' + 1$ in $\mathcal{T}_\mathcal{P}$.

Note: By assumption, the last negative action occurs at length $l - 1$. Hence, all actions from length l to n are positive actions.

- Assume $\rho_{l'+1} = (A \oplus C)$, where $A \in N_C$ and C is an $\mathcal{ALCHOIQ}br$ concept. By Definition 3.7, $s_{l'+1} = s_{l'} \cup \{A(c) \mid s_{l'} \models C(c)\}$, where c is a constant that occurs in s. Let $A(c)$ be added to $s_{l'}$. It means that $s_{l'} \models C(c)$. By hypothesis and by Lemma 6.5, if $s_{l'} \models C(c)$ then $s'_{l'} \models C(c)$. Hence, $A(c)$ will also be added to $s'_{l'}$. It follows that $s_{l'+1} \subseteq s'_{l'+1}$.
- Assume $\rho_{l'+1} = (r \oplus R)$, where $r \in N_R$ and R is an $\mathcal{ALCHOIQ}br$ role. By Definition 3.7, $s_{l'+1} = s_{l'} \cup \{r(c) \mid s_{l'} \models R(c)\}$, where c is a tuple of constants that occurs in s. Let $r(c)$ be added to $s_{l'}$. It means that $s_{l'} \models R(c)$. By hypothesis and by Lemma 6.5, if $s_{l'} \models R(c)$ then $s'_{l'} \models R(c)$. Hence, $r(c)$ will also be added to $s'_{l'}$. It follows that $s_{l'+1} \subseteq s'_{l'+1}$.

□

Thus, we proved Lemma 6.7, which states that for all states of $\mathcal{T}_\mathcal{P}$ from length l, each state of $\mathcal{T}_\mathcal{P}$ is contained in the respective states in $\mathcal{T}_{\mathcal{P}'}$. Hence, $s_n \subseteq s'_n$. Remember that G is a conjunction of positive literals only. It follows that $s'_n \models G$. □

Now, we generalize Lemma 6.6 with the following theorem.

Theorem 6.2. *Assume $GP = (I, Act, G)$, where Act is a finite set of atomic actions with no negation, universal restrictions, qualified number restrictions and G is a conjunction of positive literals. If $p(GP)$ has a plan, then $p(GP)$ has a plan with only positive actions.*

Proof. Let $\mathcal{P}$ be a plan for $p(GP)$ with k negative actions. By Lemma 6.6, there is a plan with $k-1$ negative actions. Repeating the same argument k times, it is easy to see that there exists a plan $\mathcal{P}'$ for $p(GP)$ with only positive actions. Namely, the plan $\mathcal{P}'$ can be obtained by deleting from $\mathcal{P}$ all negative actions. □

The above theorem does not hold for actions that allow universal restrictions. Instead, often a negative action with concepts that use universal restriction might be necessary for reaching a goal state. The intuition is that the semantics of $\forall r.A$ w.r.t to some interpretation is monotonic w.r.t to A and anti-monotonic w.r.t r. Thus, deleting some tuple of r from some state, and then applying some action $(B \oplus \forall r.A)$ to some state s', might result in a bigger state then if the deletion would not have happen.
We illustrate this intuition with a simple example.

Example 6.6. *Assume a* GDPP, *$GP = (I, Act, G)$ where:*

$$I = \{r(c,a), r(c,b), B(b)\}.$$

$$Act = \{\rho_1 = r \ominus \{(X,Y)\}, \rho_2 = A \oplus \forall r.B\}.$$

$$G = A(c).$$

The plan for $p(GP)$ is the following:

$$\mathcal{P} = \langle (r \ominus \{(c,a)\}), (A \oplus \forall r.B) \rangle.$$

It is easy to see that without considering ρ_1, one could never find a plan for $p(I, Act, A(c))$.

Indeed, c has two r-successors and only one of them is in an atom $A(c) \in I$. The only two possibilities for having a plan, would be to increase I by adding $A(a)$, but there is no action in Act that can compute this. The other possibility is to delete $r(c,a)$ from I, and this can be easily done by applying ρ_1 before ρ_2.
The same idea applies to actions that have in their right side a concept with qualified number restrictions.

Now we are prepared to introduce the following theorem.

Theorem 6.3. *Assume $GP = (I, Act, G)$, where Act is a finite set of atomic actions with no negation, universal restrictions, and qualified number restrictions, and G is a conjunction of positive literals. $p(GP)$ has a plan if and only if $fp(GP)$ has a plan.*

Proof. We prove both directions separately:

$\Rightarrow$ Let $\mathcal{P}$ be a plan for $p(GP)$. By Theorem 6.2, there exists a plan $\mathcal{P}'$ for $p(GP)$ with only positive actions. By Theorem 6.1, there exists a plan $\mathcal{P}''$ for $fp(GP)$. Hence, $\mathcal{P}''$ contains no negative actions and no constants that do not belong to $dom(GP)$.

$\Leftarrow$ This direction is straight forward. If $\mathcal{P}$ is a plan for $fp(GP)$, it is obvious that $\mathcal{P}$ is also a plan for $p(GP)$. □

Theorem 6.4. *Assume $GP = (I, Act, G)$, where Act is a finite set of atomic actions with no negation, universal restrictions, and qualified number restrictions, and G is a conjunction of positive literals. Deciding whether $p(GP)$ has a plan can be done in* **PSPACE***.*

Proof. By Theorem 6.3, it can be seen that under the above restrictions on Act and G, deciding plan-existence for a general planning problem, can be turned into deciding plan-existence for a fixed domain planning problem. The latter, by Theorem 5.4 is proved to be **PSPACE**-complete. Hence, deciding plan-existence for $p(GP)$ can be achieved in **PSPACE**. □

Actually, we even think that this case in fixed domain plan-existence has a lower complexity. It might even be polynomial.

CHAPTER 7

State of the Art

To our best knowledge, this is the first attempt to formally define the planning problem in graph databases. However, there is a huge body of literature on planning and a considerable one on graph databases, a part of which will be presented in this chapter. Furthermore, we discuss the relationship between planning in graph databases expressed using DL as investigated in this thesis and existing literature on ABox updates and planning in DLs.

7.1 Automated Planning

An important problem in the main focus of current planning researchers is improving efficiency by reducing the size of the search space. As described in Artificial Intelligence [40], the perfect planning language would be one that offers high expressivity as to describe a large variety of problems, but, at the same time, is restrictive enough to allow efficient algorithms to operate over it.

As a problem in logic, planning was first introduced in the 1950s by J. McCarthy and then different methods have been developed over the last decades. One approach to planning was deductive planning as in the well-known situation calculus (cf. [37]), which through time was left aside because of defects such as the frame problem. Later on, planners would require representation of plans in a less rigid way, focusing more on flexible plans, able to be adapted to changing and unforeseen circumstances. An early attempt in providing a planning language toward this aim is STRIPS planning, which can be considered as a mixture between logic and procedural computation. For a long period then, fairly no other logic-related planning systems were explored [13].

STRIPS (Stanford Research Institute Problem Solver) was developed in 1971 by R. Fikes and N. Nilsson [18] and also used to name the language providing input to the planner. It is an automated planner, which still remains a prototypical approach, and provided the foundational basis of many modern day planning representation techniques. One distinguishes between different versions of STRIPS planning. The prototypical one is the propositional

version. Meanwhile, different extensions have been extensively studied, such as first order STRIPS Planning [31] or extended propositional STRIPS Planning, which is most closely related to Ginsberg and Smith's [24] possible world approach to reasoning about actions. Determining if a given planning instance has any solution for propositional and extended propositional STRIPS planning, restricted to ground formulas is PSPACE-complete [6]. Under several restrictions, first order STRIPS planning can be reduced to propositional STRIPS planning [16].

In 1997, Blum and Furst [4] introduced Graphplan, a sound and complete partial order planner that plans in STRIPS-like domains by constructing and analyzing a special data structure called a planning graph. Graphplan always returns a shortest possible partial order plan if one exists, given a planning problem. Planning graphs work only for propositional planning problems. Once they are constructed, they represent a rich source of information about the problem. Furthermore, they can be constructed in polynomial time and they can also help organize and maintain search information so as to guide the search for a plan [4]. As a tool for generating accurate heuristics, the planning graph is viewed as a relaxed problem that is efficiently solvable [40]. Because of its backward constraint-directed search and important features like soundness, completeness and termination, Graphplan brought a significant speed-up and contributed to the scalability of planning. Evidently, Graphplan does not change the PSPACE-complete complexity of planning in the set-theoretic representation [21].

Another classical planning approach is SATPLAN, a satisfiability based planner, firstly introduced by Kautz and Selman in 1992 [27] and subsequently incorporated into a planner called BlackBox [28], which unites SAT-based and Graph-based planning. SATPLAN models planning as Boolean satisfiability, exploiting advances made in SAT solvers. In this approach, a planning problem is encoded into a set of axioms in propositional logic such that any model of the axioms amounts to a valid plan. The sat-based planners have dominated several optimal tracks of International Planning Competitions. Graphplan has also been used to efficiently generate the SAT encoding [29].

A relatively new approach to planning, was proposed by Lifschitz in influential papers [32], [33]. It was considered even earlier by Dimopoulos et al. [11] and others. This approach consists in mapping planning problems, formulated in a domain-independent planning language, into an extended logic program whose answer sets give the solutions of the planning problem (cf. [19, 34]). Following this approach, a new and effective action description language, for planning under incomplete knowledge, was introduced by Eiter et al. in the paper 'A logic programming approach to knowledge-state planning: semantics and complexity' [13]. It was named $\mathcal{K}$ with the purpose to emphasize that it describes transitions between *states of knowledge* rather than between *states of the world*. Reasoning under incomplete knowledge, planning agents usually don't have a complete view of the world. Therefore, language $\mathcal{K}$ adopts a three-valued view of fluents, in which fluents might be true, false or unknown. This language resulted to be very flexible and optimized for capturing transitions between states of complete knowledge. Furthermore, it allows the use of default negation since it is closer in spirit to answer set semantics, which has the ability to deal with incomplete knowledge [20].

In a later paper, Eiter et al. [12] present a declarative logic-programming based planning system $DLV^{\mathcal{K}}$, which implements $\mathcal{K}$ on top of the DLV answer set programming system [14, 17]. It allows solving Σ_2^P- hard planning problems, like planning under incomplete initial states.

7.2 Graph Databases

Graph databases can be defined as databases in which data structures for the schema and instances are presented as graphs or generalizations of them. Also data manipulation is expressed by graph-oriented operations and type constructors. A graph bases approach was proposed already in the 1960s, when the first language specifications for the network database model which became generally known as the CODASYL Data Model [44] was published. It builds on a generalized graph, which comprises both the data (records) and the schema (class model). With the emergence of the relational database model in practice, graph oriented databases lost their attention for a long time. In turn, recently, the need to manage information with graph-like nature has led to the development of several approaches. Some of them have to do with metadata representation models like RDF [5].

The Resource Description Framework (RDF) is a World Wide Web Consortium (W3C) recommended standard, originally designed to represent metadata. It models information with graph-like structure, where basic notions of graph theory like node, edge, path, neighborhood, connectivity, distance, degree play a very crucial role. The nodes are called resources that are interlinked. One of the main advantages of the RDF model is its ability to interconnect resources in an extensible way, using a variety of syntax notations and data serialization formats [1]. Statements about resources (in particular web resources) are subject-predicate-object (the resource being described, the traits or aspects of the resource and the property value, respectively) expressions, which are known as triples in RDF terminology. In addition, several languages for querying RDF data have been proposed and implemented. One of them is SPARQL [43], which is a W3C recommended standard query language in a SQL-like style designed for easy access to RDF stores.

For more information about the work that has been conducted in the area of graph database modeling, we refer the reader to [1].

7.3 ABox Updates and Planning in Description Logics

In this thesis, we express a graph database through atomic concepts and roles, which seem suitable to capture integrity constraints. We have considered the planning problem for this setting.

The update, revision and evolution of DL Aboxes has recently received considerable attention. Calvanese et al. [7, 22, 46] investigated the problem of updating a DL-Lite ABox by ground literals. Milicic et al. in [35] and also in a later paper [36] consider the problem of updating ABoxes in a DL context, where no compound concepts are admitted in the ABox and no TBoxes are present. The update information is described at an atomic level, i.e.,

in terms of possibly negated ABox assertions that involve only atomic concepts and roles. Even though, they don't consider planning, the similarity to our work is that we both do not consider TBoxes and that our actions perform updates on DL interpretations. The crucial difference is that we are updating DL interpretations, which are complete graph databases and they update ABoxes, which can be viewed as incomplete graph databases.

There is also existing literature on planning in DLs. An action formalisms based in DLs has been proposed by Baader et al. in [3]. Then the work is carried over by Milicic in [38]. Planning with DLs and syntactic updates has been investigated in [41]. Finally, a nice survey on DLs and planning can be found in [23].

The relationship between planning in graph databases as investigated in this thesis, and the above works is the use of DLs. However, a central property of DLs is that they work with the Open World Assumption (OWA) and ABoxes store an incomplete view of the world. In contrast, since graph databases store complete knowledge, we consider the closed world perspective. Beside the fact that we do not consider TBoxes, this states the crucial difference that clearly separates their approach from ours. Thus, even though these approaches are similar in spirit to ours, they are very different.

CHAPTER 8

Conclusions

8.1 Results

In this thesis, we have presented a new framework for planning, namely planning for graph databases. Graph databases are seen as finite Description Logic (DL) interpretations. To manipulate the databases, we have introduced an action language. This language is constructed over a new DL called $\mathcal{ALCHOIQ}br$, which is the standard $\mathcal{ALCHOIQ}$ extended with Boolean combinations of axioms and a constructor for a singleton role. This particular DL, is able to capture the changes that might occur in a graph database. It offers nice expressive features by allowing formulas as boolean combinations of assertions and inclusions, and permitting singleton concepts and roles through variables. The latter allow to introduce fresh constants in a plan. Furthermore, we have outlined the main reasoning tasks that are suitable for the setting we have considered. We then have thoroughly analyzed the computational complexity of the plan-existence decision problem for two planning problems corresponding to a graph database planning problem.

As we have seen, operating on a fixed domain offers better complexity results. We proved that fixed domain plan-existence is **PSpace**-complete. Under various restrictions on the type of goal formulas and transactions allowed as input, we have shown that different cases range in complexity from **NlogSpace** for the positive case with only positive goals to **NP**-hard for a goal formula with positive and negative atoms. In addition, we have introduced several polynomial algorithms for several fragments. We have also analyzed the features of the goal formula that seem to cause the **NP**-hardness. The intuition for each of these cases is illustrated by examples.

We have related our work to a classical planning approach, namely propositional STRIPS-planning. Planning in STRIPS is **PSpace**-complete. That means that our fixed domain planning problem can be polynomially encoded to STRIPS. We are able to encode to STRIPS the already proven **NP**-hard case with a general goal formula and atomic actions with only concept names. When translating the actions, we notice that negative atoms in

post-conditions of operators in STRIPS seem necessary. Thus, we observe that our translation maps to a **PSpace**-complete STRIPS fragment. For the other low-complexity cases, we observe that a naive translation to STRIPS leads to planning instances with negative atoms in post-conditions. Such instances are **PSpace**-hard in STRIPS. For this reason, it seems difficult to infer optimal upper-bound complexity results from such an encoding. One needs to further investigate whether there is a more suitable translation.

In addition, since arbitrary concepts and roles correspond to first order formulas, which have a complex structure and allow disjunction, it seems hard to encode atomic actions without any restrictions into STRIPS. The latter is argued by the limited expressibility of *Pre*⇒*Post* operators, which are expressed in STRIPS as only conjunction of literals. In addition, STRIPS operates on a closed domain. The set of conditions should be specified in advance. Thus, fresh constants cannot be introduced by the operators. This makes it difficult to have an encoding of the general planning problem to propositional STRIPS.

Finally, we have investigated the general plan-existence problem. The question if it is decidable, still remains a problem to be solved. We have thoroughly studied the features that enforce the use of fresh constants and have carefully outlined each of them, by illustrating the corresponding intuitions with several examples. Then, we were interested in finding expressive settings that have the following property: for every plan there is a plan that does not require fresh constants. To achieve this we have imposed several restrictions on the goal and actions that are allowed to be given as input. We were finally able to show that for the case where the goal formula is a conjunction of positive atoms and the actions are atomic (where no qualified number restriction, universal restriction and negation appears) can be encoded to the fixed domain case. Thus, we showed a **PSpace** upper bound for this setting, which we think might not be tight.

8.2 Further Research

We propose several directions for further work.

Possibly the most important direction of further research is to investigate to what extent the theoretical results obtained in this thesis can be used in practical settings. We have shown that some cases can be encoded into propositional STRIPS in Section 5.4. It would be interesting to provide an encoding into the DLV$^{\mathcal{K}}$ planning system.

Additional further research includes filling the complexity gaps that are left. For instance, in Section 5.3.1, we proved only the **NP**-hardness for the fixed domain case of the general goal with atomic actions that use only concept names. An **NP** upper bound, if it exists, is missing. Also, an important open question is whether the general plan-existence is decidable. If yes, it would be interesting to define the upper bound complexity. We suspect that the problem is already **EXPTime**-hard, which can be shown by a reduction from an alternating Turing machine with an exponentially bounded space.

During the thesis, we have also proposed several interesting reasoning tasks for planning in graph databases such as *Plan-Existence*, *Bounded Plan-Existence*, *Conformant Planning*, *SynVerif*$\exists$, *SynVerif*$\forall$ (see Section 2.2.2). We have investigated only one of them, namely *Plan-Existence*. Another line of further research would be to study the remaining ones. In addition one can try to to define new reasoning problems, that might be relevant for practical reasons.

Bibliography

[1] Renzo Angles and Claudio Gutiérrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1), 2008.

[2] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[3] Franz Baader, Carsten Lutz, Maja Milicic, Ulrike Sattler, and Frank Wolter. Integrating description logics and action formalisms: First results. In Manuela M. Veloso and Subbarao Kambhampati, editors, *AAAI*, pages 572–577. Association for the Advancement of Artificial Intelligence (AAAI) Press / The MIT Press, 2005.

[4] Avrim Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artif. Intell.*, 90(1-2):281–300, 1997.

[5] Tim Bray, Jean Paoli, C. Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible markup language (xml) 1.0 (fifth edition). World Wide Web Consortium, Recommendation REC-xml-20081126, November 2008.

[6] Tom Bylander. The computational complexity of propositional strips planning. *Artif. Intell.*, 69(1-2):165–204, 1994.

[7] Diego Calvanese, Evgeny Kharlamov, Werner Nutt, and Dmitriy Zheleznyakov. Updating aboxes in DL-Lite. In Alberto H. F. Laender and Laks V. S. Lakshmanan, editors, *AMW*, volume 619 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[8] Diego Calvanese, Magdalena Ortiz, and Mantas Simkus. Evolving graph databases under description logic constraints. In Thomas Eiter, Birte Glimm, Yevgeny Kazakov, and Markus Krötzsch, editors, *Description Logics*, volume 1014 of *CEUR Workshop Proceedings*, pages 120–131. CEUR-WS.org, 2013.

[9] David Chapman. Planning for conjunctive goals. *Artif. Intell.*, 32(3):333–377, 1987.

[10] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[11] Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding planning problems in nonmonotonic logic programs. In Sam Steel and Rachid Alami, editors, *ECP*, volume 1348 of *Lecture Notes in Computer Science*, pages 169–181. Springer, 1997.

[12] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. A logic programming approach to knowledge-state planning, II: The dlv$^{\mathrm{K}}$ system. *Artif. Intell.*, 144(1-2):157–211, 2003.

[13] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. A logic programming approach to knowledge-state planning: Semantics and complexity. *ACM Trans. Comput. Log.*, 5(2):206–263, 2004.

[14] Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. The kr system dlv: Progress report, comparisons and benchmarks. In Anthony G. Cohn, Lenhart K. Schubert, and Stuart C. Shapiro, editors, *KR*, pages 406–417. Morgan Kaufmann, 1998.

[15] E. Allen Emerson and Edmund M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In J. W. de Bakker and Jan van Leeuwen, editors, *ICALP*, volume 85 of *Lecture Notes in Computer Science*, pages 169–181. Springer, 1980.

[16] Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artif. Intell.*, 76(1-2):75–88, 1995.

[17] Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Pushing goal derivation in dlp computations. In Gelfond et al. [19], pages 177–191.

[18] Richard Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artif. Intell.*, 2(3/4):189–208, 1971.

[19] Michael Gelfond, Nicola Leone, and Gerald Pfeifer, editors. *Logic Programming and Nonmonotonic Reasoning, 5th International Conference, LPNMR'99, El Paso, Texas, USA, December 2-4, 1999, Proceedings*, volume 1730 of *Lecture Notes in Computer Science*. Springer, 1999.

[20] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[21] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated planning - theory and practice*. Elsevier, 2004.

[22] Giuseppe De Giacomo, Maurizio Lenzerini, Antonella Poggi, and Riccardo Rosati. On instance-level update and erasure in description logic ontologies. *J. Log. Comput.*, 19(5):745–770, 2009.

[23] Yolanda Gil. Description logics and planning. *AI Magazine*, 26(2):73–84, 2005.

[24] Matthew L. Ginsberg. Computational considerations in reasoning about action. In James F. Allen, Richard Fikes, and Erik Sandewall, editors, *KR*, pages 250–261. Morgan Kaufmann, 1991.

[25] Jim Gray. The transaction concept: Virtues and limitations (invited paper). In *VLDB*, pages 144–154. The Institute of Electrical and Electronics Engineers (IEEE) Computer Society, 1981.

[26] Ian Horrocks and Ulrike Sattler. A tableaux decision procedure for SHOIQ. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 448–453, 2005.

[27] Henry A. Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In William J. Clancey and Daniel S. Weld, editors, *Association for the Advancement of Artificial Intelligence (AAAI/IAAI), Vol. 2*, pages 1194–1201. AAAI Press / The MIT Press, 1996.

[28] Henry A. Kautz and Bart Selman. The role of domain-specific knowledge in the planning as satisfiability framework. In Reid G. Simmons, Manuela M. Veloso, and Stephen F. Smith, editors, *AIPS*, pages 181–189. AAAI, 1998.

[29] Henry A. Kautz and Bart Selman. Unifying sat-based and graph-based planning. In Thomas Dean, editor, *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 318–325. Morgan Kaufmann, 1999.

[30] Maurizio Lenzerini. Ontology-based data management. pages 5–6, 2011.

[31] V. Lifschitz. On the semantics of STRIPS. In *Reasoning About Actions and Plans — Proceedings of the 1986 Workshop*, pages 1–9, San Mateo, CA, 1987. Morgan Kaufmann Publishers.

[32] Vladimir Lifschitz. On the logic of causal explanation. *Artificial Intelligence*, 96:96–451, 1997.

[33] Vladimir Lifschitz. Action languages, answer sets and planning. In *In The Logic Programming Paradigm: a 25-Year Perspective*, pages 357–373. Springer Verlag, 1999.

[34] Vladimir Lifschitz and Hudson Turner. Representing transition systems by logic programs. In Gelfond et al. [19], pages 92–106.

[35] Hongkai Liu, Carsten Lutz, Maja Milicic, and Frank Wolter. Updating description logic aboxes. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *KR*, pages 46–56. Association for the Advancement of Artificial Intelligence (AAAI) Press, 2006.

[36] Hongkai Liu, Carsten Lutz, Maja Milicic, and Frank Wolter. Foundations of instance level updates in expressive description logics. *Artif. Intell.*, 175(18):2170–2197, 2011.

[37] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, pages 463–502. Edinburgh University Press, 1969.

[38] Maja Milicic. Complexity of planning in action formalisms based on description logics. In Nachum Dershowitz and Andrei Voronkov, editors, *LPAR*, volume 4790 of *Lecture Notes in Computer Science*, pages 408–422. Springer, 2007.

[39] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[40] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[41] Antonio A. Sánchez-Ruiz, Pedro A. González-Calero, and Belén Díaz-Agudo. Planning with description logics and syntactic updates. In M. A. Salido and J. Fdez-Olivares, editors, *Planning, Scheduling and Constraint Satisfaction (CAEPIA 2007 Workshop)*, pages 140–150. Universidad de Salamanca, 2007.

[42] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[43] Andy Seaborne and Eric Prud'hommeaux. SPARQL query language for RDF. W3C recommendation, W3C, January 2008. http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[44] Robert W. Taylor and Randall L. Frank. Codasyl data-base management systems. *ACM Comput. Surv.*, 8(1):67–103, 1976.

[45] Jeffrey D. Ullman and Jennifer Widom. *A first course in database systems (2. ed.)*. Prentice Hall, 2002.

[46] Dmitriy Zheleznyakov, Diego Calvanese, Evgeny Kharlamov, and Werner Nutt. Updating tboxes in DL-Lite. In Volker Haarslev, David Toman, and Grant E. Weddell, editors, *Description Logics*, volume 573 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.