Planning with Transaction Logic

A Dissertation presented

by

Reza Basseda

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

Stony Brook University

December 2015

Stony Brook University

The Graduate School

Reza Basseda

We, the dissertation committe for the above candidate for the
Doctor of Philosophy degree, hereby recommend
acceptance of this dissertation

Professor Michael Kifer - Dissertation Advisor
Department of Computer Science

Professor Scott D. Stoller - Chairperson of Defense
Department of Computer Science

Professor Emeritus David S. Warren
Department of Computer Science

Professor Yanhong A. Liu
Department of Computer Science

Professor Neng-Fa Zhou - External
Department of Computer and Information Science,
City University of New York

This dissertation is accepted by the Graduate School

Charles Taber
Dean of the Graduate School

Abstract of the Dissertation

Planning with Transaction Logic

by

Reza Basseda

Doctor of Philosophy

in

Computer Science

Stony Brook University

2015

Automated planning has been the subject of intensive research and is at the core of several areas of AI, including intelligent agents and robotics. In this dissertation, we argue that Transaction Logic is a natural specification, experimentation, prototyping, and potentially implementation language for planning algorithms. This enables one to see further afield and thus discover better and more general solutions than using one-of-a-kind formalisms that have been traditionally dominating this field. Specifically, we take the well-known *STRIPS* planning strategy and show that Transaction Logic lets one specify this strategy and several of its far-reaching extensions easily and concisely. In addition, we demonstrate the power of our approach by applying it to a fundamentally different (from *STRIPS*) planning strategy, called GraphPlan.

To summarize, this dissertation introduces a novel planning formalism based on Transaction Logic and validates it by applying it to some existing planning strategies and developing new ones as follows: (1) we propose a non-linear extension to *STRIPS* planning and prove its completeness; (2) we introduce *fast-STRIPS*—a modification of our non-linear *STRIPS* extension, which is also complete and yields speedups of orders of magnitude; (3) we extend *STRIPS* with regression analysis, which is again complete and significantly improves performance; (4) we extend *STRIPS* to enable it to solve planning problems with negative derived atoms; (5) to illustrate the versatility of our formalism, we also applied it to GraphPlan— a planning strategy that bears no resemblance to *STRIPS*.

Dedication Page

To my parents, Nasser Basseda and Esmat Hessan.
Without their patience, support, and most of all love,
the completion of this dissertation would not have been possible.

And to my sisters, Nargess and Zahra.
For their endless love and encouragement.

And to my teachers.
For the education they provided to me.

Contents

List of Figures

List of Tables

Preface

This Ph.D. dissertation contains the result of research undertaken at the Department of Computer Science of Stony Brook University.

My journey towards this dissertation started in December 2012, when my adviser recommended me to read a paper about *Analysis of State Modifying Access Control Policies* [12]. A few weeks afterward, in one of our department's events, I was fortunate to have a short conversation about that topic with Professor Scott Stoller, who also provided me with papers on *Analysis of Administrative Policies* [48], which is a related topic. After having a discussion about analysis of such policies, Professor Kifer advised me to generalize and reformulate this analysis as a classical planning problem. The generalization and reformulation, together with my background knowledge about Transaction Logic, led me to the current research direction, *Planning with Transaction Logic*.

The extended abstract of this dissertation has appeared in [8]. Part of Chapter 3 has been also published in [11]. A material from Chapters 3, 4, and 5 will be also submitted for publication in a journal paper next year. Part of my work on planning with Transaction Logic is not included in this dissertation. But it can be found in [10, 9].

Reza Basseda
Stony Brook, NY
November 2015

Acknowledgements

First of all, I would like to praise God the most merciful, for giving me patience, self-power and ability to accomplish this work. I praise God, the almighty for providing me this opportunity and granting me the capability to proceed successfully in achieving all my goals. I would also like to express my sincere thanks and appreciation to all who helped me during the past years.

The last six years have been a challenging trip, with both ups and downs. Fortunately, I was not alone on this road, but accompanied by a great adviser, Professor Michael Kifer, who was always willing to coach, help, and motivate me. For this, I would like to kindly thank him. The door to Professor Kifer's office was always open whenever I ran into a trouble spot or had a question about my research or writing. He was always there to listen and to give his best advice. He is responsible for involving me in this research topic in the first place. He taught me how to ask questions, express my ideas, and write a scientific statement. He showed me different ways to approach a research problem and the need to be persistent to accomplish any academic goals. I believe that I am tremendously fortunate and blessed to have worked with him. I will never find words to tell what I owe to him. But, I would sincerely like to say him *"thank you."*

I would also like to thank the rest of my dissertation committee: Professor Scott Stoller, who introduced *administrative policies* to me, and played an important role in directing me to this area of research, Professor David Warren, for everything I learned about Prolog from his classes and our research meetings, Professor Yanhong Liu, for invaluable lessons about research in programming languages, and Professor Neng-fa Zhou, who gave insightful comments and reviewed my work.

I also owe a special thank you to Dr. Paul Fodor, who was a constant help, source of encouragement, and positivity for me during the implementation phase of this research work. He continually reminded me to stay focused, and cheerfully supported me whenever I expressed feelings of doubt or uncertainty. I would also like to sincerely thank all my teachers at Stony Brook University, including Professor Radu Grosu for teaching a great and interesting course on algorithms, Professor Ker-I Ko for introducing me to the theory of computation, Professor Anita Wasilewska for great lessons into logic for computer science, Professor Samir Das for teaching a great course on computer networks, Professor Klaus Mueller for infusing his in-depth knowledge and interest in visualization into me, and Professor Himanshu Gupta for refreshing my knowledge in his database course. I am also thankful of Professor Rezaul Chowdhury, my teacher in supercomputing and parallel programming graduate courses, who also was my co-adviser for a short period of time. I also wish to thank Professor I.V. Ramakrishnan and C.R. Ramakrishnan, my teachers in artificial intelligence and programming languages graduate courses, who also sat on my Research Proficiency Exam committee. By working with all these top professional people, I have gained more knowledge to pursue my career and become more confident at my research.

Many thanks to Maryam Bahojb Imani and Najmehalsadat Miramirkhani whose editing suggestions and precise sense of language contributed to the final copy. Maryam provided decisive and energetic support during the write-up stage. Najmehalsadat also provided a lot of useful comments on my defense presentation and dissertation. The loyal assistance of Koosha Mirhosseini, Ali Aleyasin, and especially Alireza Abedi, enabled me to bypass the more persistent obstacles to dissertation completion. Also thanks to Tiantian Gao for his assistance and willingness to discuss ideas at their initial stage. I am really grateful for all support by the present and former members of the Web-Scale Knowledge Representation Lab: Senlin Liang, Polina Kuznetsova, and Hui Wan, for engendering an atmosphere of intellectual vitality, exchange, and collegiality in our lab. I am also grateful to Dr. Steven Greenspan, Steven Lee, and Michael Cohen from CA Inc., and Dr. Peng Xu, Dr. Klaus Macherey, and Dr. Wolfgang Macherey from Google Inc. for all of their support during the time when I was research assistant at CA and software engineering intern at Google. I would also like to thank all the student members of programming languages group: Kazim Tuncay Tekle, Jonathan Brandvein, Spyros Hadjichristodoulou, and Bo Lin. Listening to and participating in their discussions on various matters provided me with an opportunity to engage in intellectual discourse beyond my disciplinary boundaries.

My acknowledgement will never be complete without the special mention of my fellows in Computer Science Graduate Student Council: Hau Chan, Abhishek Murthy, and William Jannen. They not only helped me run as the president of the council, but also have taught me the group culture and have lived by example to make me understand the hard facts of life. Abhishek was a wonderful friend throughout grad school and most of my dissertation endeavors. Not only did he provide me with intellectual support, but he also kept me encouraged along the way. William kept me entertained, inspired, and motivated to keep on plugging away at my goals even when our research workload seemed completely overwhelming. I would like to acknowledge the chairman of the computer science department, Distinguished Professor Arie Kaufman, for all his support and motivation during the time I was serving as the president of the council and the representative of computer science graduate students in Graduate Student Organization. By collaborating with all of these colleagues, I have obtained more invaluable experiences to enhance my career growth and skills.

I would like to acknowledge my other friends for their moral support and motivation, which drives me to give my best. Roozbeh Ebrahimi Soorchaei, Navid Hamed Azimi, Mohammadali Bankehsaz, Rami Al-Rfou', Kota Yamaguchi, Siming Li, Mohammad Tanvir Irfan, Arun Nampally, Muhammad Asiful Islam, Andrey Gorlin, Golnaz Ghasemiesfeh, Zhongyuan Xu, Puneet Gupta, Md. Ariful Islam, Behzad Farshid, Hossein Ebrahiminejad, Hoda Ehsan, Jesmin Jahan Tithi, Pramod Ganapathi, Sumeet Bajaj, Seyed Ahmad Javadi, Kazem Shirinzad, Saad Nadeem, Michael Ortega, Seyed Mohammad Halimi, Jianfu Chen, Mostafa Haghir Chehreghani, Noopur Motiyani, Rohit Mathew, Prankur Gupta, Zafar Ayyub Qazi, Wai-Kit Sze, Andrea Fodor, Niranjan Sudhir Hasabnis, Dilip Nijagal Simha, Riccardo Pelizzi, Prasad Narasimhan, Charilaos Papadopoulos, Ievgeniia Gutenko, Mariola Szenk, Travis Kruse, Max Katz,... the list is endless ... thanks to one and all. My special gratitude to all of them for being with me in thicks and thins of life. I find myself lucky to have friends like them in my life.

I would also like to extend my gratitude to the systems and administrative staff at the department including Brian Tria, Cynthia Scalzo, Kathy Germana, Stella Mannino, and Betty Knitweiss.

Above all, a special thanks to my family. Words cannot express how grateful I am to my father and mother for all of the sacrifices that they have made on my behalf. Their prayers for me were what sustained me this far. I thank my parents, Nasser Basseda, and Esmat Hessan, for giving me life in the first place, for educating me at early steps, for unconditional support and encouragement to pursue my interests, even when the interests went beyond boundaries of language, field and geography. My sister, Zahra, for sharing her research experience which helped me to write this dissertation. And also for listening to my complaints and frustrations, but always believing in me. My older sister, Nargess, for sharing her life experiences with me, which boosted my confidence to pursue my dreams. Hence, I would not have accomplished my goals without my family's endless support, encouragement, and cheers.

Chapter 1

Introduction

The classical automated planning refers generally to planning for restricted state-transition systems. Given a restricted state-transition system, an initial state, and a set of goal conditions, the classical planning problem is to find a sequence of actions whose execution started at initial state results in a state satisfying the goal conditions. The classical problem of automated planning has been used in a wide range of applications such as robotics, multi-agent systems, and more. Due to this wide range of applications, automated planning has become one of the most important research areas in Artificial Intelligence (AI).

The history of classical planning in AI dates back to the early 1970s when the first planning system, STanford Research Institute Problem Solver (STRIPS), was introduced by Fikes and Nilsson [32]. STRIPS problem solving system not only introduced the *STRIPS* planning algorithm, but also established a representation language called *classical* planning problem language [32, 78], extended further by Planning Domain Definition Language (PDDL) standards [34]. Based on the search structures and algorithms, planning techniques to solve classical planning problems can be grouped in two categories: classical and neoclassical [72].

In classical planning techniques, every node of the search space represents a partial plan. When the planning solutions are considered a total order sequence of actions, a divide-and-conquer planning technique, called *linear planning*, can be applied to solve a planning problem [80, 79]. However, this approach is not complete for non-serializable sub-goal problems [85, 65, 64, 51]. State space planners are instances of linear planning and are further developed using different heuristics [69, 18, 50, 49]. On the other hand, in *non-linear planning*, planning solutions are constructed from partial order solutions. Plan space planners are well-known instances of non-linear planning that are successful in addressing the incompleteness problem [80]. TWEAK is an example of partial order planners that also accounts for the proof of completeness and decidability of different planning problems. Partial order planning approach has also been used in several other planners [68, 75] as well.

In neoclassical planning techniques, every node of the search space can be viewed as a set of several partial plans. *GraphPlan* is an example of these techniques introducing a very powerful search space called *planning graph* [16]. The performance of GraphPlan is better than other neoclassical planning techniques. There are also several research works that study classical planning as satisfiability problem [58][44][30]. *BLACKBOX* is one the most recent planners based on both graph-based technique and propositional logical satisfiability. In the AIPS planning competition 2001, BLACKBOX was able to compete with other planners. Planning as a constraint satisfaction problem (CSP) is another example of neoclassical planning techniques [84][31].

Besides planning as satisfiability and CSP, a number of deductive planning frameworks have been proposed over the years. Situation calculus is one of the first logic based approaches applied for solving a classical planning problem [46]. Linear connection proof method [13][14][15], equational horn logic [52], and linear logic [56][26] are well-known examples of logic-based deduction methods applied for solving classical planning problems. Answer set programming is another, more recent logic based technique to solve planning problems [28][29][66][40][37]. The representation of planning domains and problems in answer set programs is similar to that of situation calculus [36]. Picat is another logic programming framework that simulates the *forward state space search* planning algorithm. Its novel customized tabling technique makes it an efficient logic-based system for solving planning problems [7][88][6][87].

There are several reasons that make logical deduction suitable for a classical planner: (1) Logic-based deduction used in planning can be cast as a formal framework that eases proving different planning properties such as completeness and termination. (2) Logic-based systems naturally provide a declarative language that simplifies the specification of planning problems. (3) Logical deduction is usually an essential component of intelligent knowledge representation systems. Therefore, applying logical deduction in classical planning makes the integration of planners with such systems simpler. Despite these benefits of using logical deduction in planning, many of the above mentioned deductive planning techniques are not getting as much attention as algorithms specifically provided for planning problems. One can enumerate a couple of reasons for this state of affairs:

- Many of the above approaches invent one-of-a-kind theories that are suitable only for the particular problem at hand. This makes such approaches not scalable as they require great intellectual effort for each new domain.

- These works generally show how they can represent and encode classical planning actions and rely on a theorem prover of some sort to find plans. Therefore, the planning techniques embedded in such planners are typically simple state space planning (e.g. forward state space search) that has a very large search space. Thus, they do not exploit heuristics and techniques developed for different classical and neoclassical planning technique with the purpose of reducing the search space.

In this dissertation, we show that a general logical theory, specifically *Transaction Logic* (or $\mathcal{TR}$) [24, 23, 22], addresses the above mentioned issues and also provides multiple advantages for specifying, generalizing, and solving planning problems. Transaction Logic is an extension of classical logic with dedicated support for specifying and reasoning about actions, including sequential and parallel execution, atomicity of transactions, and more.

Our contention is that precisely because planning techniques are cast here as purely logical problems in a suitable general logic, a number of otherwise non-trivial extensions become low-hanging fruits and we get them almost for free. For instance, we will show that *STRIPS*-based planners can be naturally extended with intensional rules and derived predicates.

We validate approach by applying it to two classes of planning algorithms: *STRIPS*-based planners and GraphPlan, the former being classical and the latter neoclassical. Specifically, our contributions are as follows:

1. We use Concurrent Transaction Logic to introduce a novel non-linear *STRIPS*-based planning algorithm, which, unlike the original linear *STRIPS* planner [32], is proven to be complete.

2. After inspecting the logic rules of our non-linear *STRIPS*-based planning algorithm, we introduce certain heuristics that reduce the search space. Our experiments show that these heuristics yield significant speedups.

3. We further enhance our *STRIPS*-based planner with regression analysis and show that this also yields significant speedup.

4. We extend the *STRIPS* planning algorithm with negative derived literals. This can simplify the specification of certain planning problem and, more

interestingly, can reduce the number of states to be searched thus improving the performance.

5. Finally, we apply our techniques to a completely different class of planners, the neoclassical GraphPlan algorithm. The heuristic underlying GraphPlan differs from *STRIPS* significantly and the ability to handle this different class of algorithms testifies to the versatility of our approach.

6. We show that, our basic version of the non-linear *STRIPS* planner as well as our extensions of this algorithm are both sound complete. Our GraphPlan simulation is sound and, we believe, is also complete.

We believe that our GraphPlan-based planner can also be extended with intensional rules and negative derived predicates, which will be the part of future work. Likewise, in our future work we would like to explore incorporating other aspects found in the planning literature, like parallelization of plans and plans with loops [56, 55, 82].

This dissertation is organized as follows. Chapter 2 introduces the *STRIPS* planning framework and its extension in support of action ramification and provides the necessary background on Transaction Logic. Chapter 3 casts *STRIPS* as a problem of posing a transactional query in Transaction Logic and shows that executing this query using the $\mathcal{TR}$'s proof theory makes for a sound and complete *STRIPS* planning algorithm. It also proposes the *fSTRIPS* planning strategy and shows that the $\mathcal{TR}$'s proof theory also yields a complete planning algorithm. Chapter 4 introduces the regression of literals through *STRIPS* actions and proposes a regression analysis method for $\mathcal{TR}$. It then introduces *STRIPS*^{R}, an extension of our *STRIPS*-based planning algorithm with regression analysis, which has a "smarter" search strategy. Chapter 5 extends *STRIPS* with negative derived literals and proposes the *STRIPS*^{neg} planner for this planning domain. Chapter 6 provides a $\mathcal{TR}$-based representation of the *GraphPlan* planning algorithm. In Chapter 7, we present our experiments that compare the performance of the different algorithms and Chapter 8 concludes the dissertation.

Chapter 2

Overview of *STRIPS* and $\mathcal{TR}$

In this chapter, we first formally review the basics of *STRIPS* planning and show how this formalism is related to Planning Domain Definition Language, PDDL. Then we briefly overview some components of $\mathcal{TR}$ used in our planning technique.

2.1 Extended *STRIPS*-style Planning

To review the formal definitions of *STRIPS* planning, we first remind the reader a number of standard concepts in logic. Then we introduce the *STRIPS* planning problem.

We assume denumerable sets of variables $\mathcal{V}$, constants $\mathcal{C}$, and predicate symbols $\mathcal{P}$—all three sets being pairwise disjoint. The set of predicates, $\mathcal{P}$, is further partitioned into *extensional* ($\mathcal{P}_{ext}$) and *intensional* ($\mathcal{P}_{int}$) predicates. In *STRIPS*, actions update the state of a system by adding or deleting statements about predicates. In the original *STRIPS*, all predicates were extensional, and the addition of intentional predicates is a major enhancement, which allows us to deal with the so-called *ramification problem* [43], i.e., with indirect consequences of actions.

Atomic formulas (or just ***atoms***) have the form $p(t_1, ..., t_n)$, were $p \in \mathcal{P}$ and each t_i is either a constant or a variable. Extending the logical signature with function symbols is straightforward in our framework, but we avoid doing this here in order to save space.

An atom is ***extensional*** if $p \in \mathcal{P}_{ext}$ and ***intensional*** if $p \in \mathcal{P}_{int}$. A ***literal*** is either an atom or a negated extensional atom of the form $\neg p(t_1, ..., t_n)$. Negated intensional atoms are not supported by the formalisms in Chapters 3 and 4—only negated *extensional* literals were allowed. In Chapter 5, we extend our framework to allow negated intensional atoms and propose a novel extension of *STRIPS* planning for such domains.

Extensional predicates represent database facts: they can be directly manipulated (inserted or deleted) by actions. Intensional predicate symbols are used for atomic statements defined by *rules*—they are *not* affected by actions directly. Instead, actions make extensional facts true or false and this indirectly affects the dependent intensional atoms. These indirect effects are known as action *ramifications* in the literature.

A ***fact*** is a ***ground*** (i.e., variable-free) extensional atom. A set $\mathbf{S}$ of literals is ***consistent*** if there is no atom, atm, such that both atm and $\neg atm$ are in $\mathbf{S}$.

A ***rule*** is a statement of the form $head \leftarrow body$ where *head* is an intensional atom and body is a conjunction of literals. A ***ground instance*** of a rule, R, is any rule obtained from R by a substitution of variables with constants from $\mathcal{C}$ such that different occurrences of the same variable are always substituted with the same constant. Given a set $\mathbf{S}$ of literals and a ground rule of the form $atm \leftarrow \ell_1 \wedge \cdots \wedge \ell_m$, the rule is *true* in $\mathbf{S}$ if either $atm \in \mathbf{S}$ or $\{\ell_1, \ldots, \ell_m\} \not\subseteq \mathbf{S}$. A (possibly non-ground) rule is *true* in $\mathbf{S}$ if all of its ground instances are true in $\mathbf{S}$.

Definition 1 (State)**.** *Given a set $\mathbb{R}$ of rules, a* ***state*** *is a consistent set $\mathbf{S} = \mathbf{S}_{ext} \cup \mathbf{S}_{int}$ of literals such that*

1. *For each fact atm, either $atm \in \mathbf{S}_{ext}$ or $\neg atm \in \mathbf{S}_{ext}$.*

2. *Every rule of $\mathbb{R}$ is true in $\mathbf{S}$.* □

Definition 2 (*STRIPS* action)**.** *A STRIPS* ***action*** *is a triple of the form $\alpha = \langle p_\alpha(X_1, ..., X_n), Pre_\alpha, E_\alpha \rangle$, where*

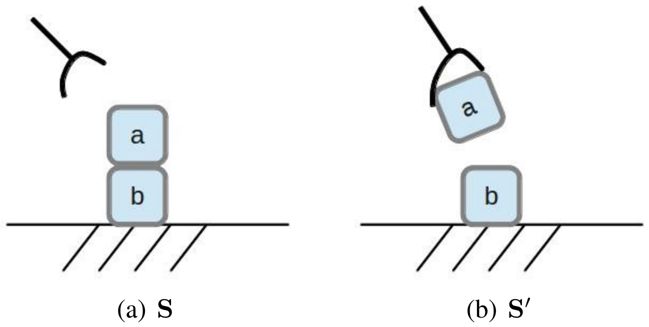
- *$p_\alpha(X_1, ..., X_n)$ is an intensional atom in which $X_1, ..., X_n$ are variables and $p_\alpha \in \mathcal{P}_{int}$ is a predicate that is reserved to represent the action α and can be used for no other purpose;*

- *Pre_α, called the* ***precondition*** *of α, is a set of literals that may include extensional as well as intensional literals;*

- *E_α, called the* ***effect*** *of α, is a consistent set that may contain extensional literals only;*

- *The variables in Pre_α and E_α must occur in $\{X_1, ..., X_n\}$.*[1] □

[1]Requiring the variables of Pre_α to occur in $\{X_1, ..., X_n\}$ is not essential for us: we can easily extend our framework and consider the extra variables to be existentially quantified.



(a) $\mathbf{S}$ (b) $\mathbf{S}'$

Figure 2.1: An example of state change.

Note that the literals in Pre_α can be both extensional and intensional, while the literals in E_α can be extensional only.

Definition 3 (Execution of a *STRIPS* action)**.** *A STRIPS action α is* ***executable*** *in a state* $\mathbf{S}$ *if there is a substitution $\theta : \mathcal{V} \longrightarrow \mathcal{C}$ such that $\theta(Pre_\alpha) \subseteq \mathbf{S}$. A* ***result of the execution*** *of α with respect to θ is the state, denoted $\theta(\alpha)(\mathbf{S})$, defined as $(\mathbf{S} \setminus \neg\theta(E_\alpha)) \cup \theta(E_\alpha)$, where $\neg E = \{\neg\ell \mid \ell \in E\}$. In other words, $\theta(\alpha)(\mathbf{S})$ is $\mathbf{S}$ with all effects of $\theta(\alpha)$ applied. When α is ground, we simply write $\alpha(\mathbf{S})$.* □

Note that $\mathbf{S}'$ is well-defined since $\theta(E_\alpha)$ is unique and consistent. Observe also that, if α has variables, the result of an execution, $\mathbf{S}'$, may depend on the chosen substitution θ.

The following simple example illustrates the above definition. We follow the standard logic programming convention whereby lowercase symbols represent constants and predicate symbols. The uppercase symbols denote variables that are implicitly universally quantified outside of the rules.

Example 1. *Consider a world consisting of just two blocks and the action $pickup = \langle pickup(X,Y), \{clear(X)\}, \{\neg on(X,Y), clear(Y)\}\rangle$. Consider also the state $\mathbf{S} = \{clear(a), \neg clear(b), on(a,b), \neg on(b,a)\}$. Then the result of the execution of $pickup$ at state $\mathbf{S}$ with respect to the substitution $\{X \to a, Y \to b\}$ is $\mathbf{S}' = \{clear(a), clear(b), \neg on(a,b), \neg on(b,a)\}$, shown in Figure 2.1. It is also easy to see that $pickup$ cannot be executed at $\mathbf{S}$ with respect to any substitution of the form $\{X \to b, Y \to ...\}$.* □

Definition 4 (Planning problem)**.** *A* ***planning problem*** *$\langle \mathbb{R}, \mathbb{A}, G, \mathbf{S}\rangle$ consists of a set of rules $\mathbb{R}$, a set of STRIPS actions $\mathbb{A}$, a set of literals G, called the* ***goal*** *of the planning problem, and an* ***initial state*** $\mathbf{S}$*. A sequence of actions $\sigma = \alpha_1, \ldots, \alpha_n$ is a* ***planning solution*** *(or simply a* ***plan****) for the planning problem if:*

- $\alpha_1, \ldots, \alpha_n \in \mathbb{A}$*; and*
- *there is a sequence of states* $\mathbf{S}_0, \mathbf{S}_1, \ldots, \mathbf{S}_n$ *such that*
  - $\mathbf{S} = \mathbf{S}_0$ *and* $G \subseteq \mathbf{S}_n$ *(i.e.,* G *is satisfied in the final state);*
  - *for each* $0 < i \leq n$*,* α_i *is executable in state* $\mathbf{S}_{i-1}$ *and the result of that execution (for some substitution) is the state* $\mathbf{S}_i$*.*

In this case we will also say that $\mathbf{S}_0, \mathbf{S}_1, \ldots, \mathbf{S}_n$ *is an execution of* σ*.* □

Definition 5 (Non-redundant plan)**.** *Given a planing problem* $\langle \mathbb{R}, \mathbb{A}, G, \mathbf{S} \rangle$ *and a sequence of actions* $\sigma = \alpha_1, \ldots, \alpha_n$*, we call* σ *a non-redundant plan for* $\langle \mathbb{R}, \mathbb{A}, G, \mathbf{S} \rangle$ *if and only if:*

- σ *is a planning solution for* $\langle \mathbb{R}, \mathbb{A}, G, \mathbf{S} \rangle$*;*
- *None of* σ*'s sub-sequences is a planning solution for the given planning problem.*

In other words, removing any action from σ *either makes the sequence non-executable at* $\mathbf{S}$ *or* G *is not satisfied after the execution.* □

2.2 Compatibility with PDDL

PDDL is a standard language intended to express planning domains [42] in AIPS planning competitions [4]. A planning domain consists of domain predicates, possible actions, the structure of compound actions, and the effects of actions. To express a planning domain and a planning problems, PDDL supports several syntactic features such as basic *STRIPS*-style actions, conditional effects, universal quantifications in the effects, ADL features [74], domain axioms, safety constraints, hierarchical actions, and more. Clearly, the formalism in Section 2.1 also supports the basic *STRIPS*-style actions and domain axioms. Although, it is simple to extend this formalism to include other features, we intend to keep our formalism as simple as possible. Such simplicity makes our later discussed soundness and completeness proofs straightforward and understandable. We will also extend this formalism in the following chapters due to the requirements of further developments. For instance, we will add equality built-ins in Chapter 4.

It is easy to show that $\mathcal{TR}$ is expressive enough to represent and encode most of the features provided by different extensions of PDDL. The following list briefly shows the ability of $\mathcal{TR}$ to express some of these main features.

- ADL features [70, 42]: in PDDL, actions can have a first order formula in their precondition. The effect of an action can also include universal quantifications over fluents. $\mathcal{TR}$ can use Lloyd-Topor transformation to support first order formula (including universal and existential quantifiers and disjunction) in the precondition of actions. It also can simulate universal quantifications over fluents in the effects of actions.

- Numerical extensions [34]: in PDDL, one can associate actions, objects, and plans with numerical values and use these values in numerical expressions to compute different planning metrics. This syntactical feature also needs PDDL to include numerical operators. Since $\mathcal{TR}$ can express and encode numerical operators and expressions as a part of its model theory, it can easily have this feature.

- Temporal extensions and durative actions [34, 38, 71, 81]: PDDL is able to express discretised and continuous actions [34]. $\mathcal{TR}$ is also able to represent discretised and continuous actions because the notion of *time* can be encoded in $\mathcal{TR}$'s transactions.

- Plan and solution preferences and constraints [47, 41]: In a planning problem, it is possible that only a subset of goals can be achieved because of the conflict between goals. In this situation, the ability to distinguish the importance of different goals is essential. PDDL provides such ability to express such preferences among goals and planning solutions. $\mathcal{TR}$ augmented with defeasible reasoning also can easily provide this feature.

The compatibility of our formalism in Section 2.1 with PDDL makes it easy to migrate standard AIPS planning competition benchmarks to our $\mathcal{TR}$ frameworks. Such compatibility allows us to develop a simple translator that maps *STRIPS* planning problems specified in PDDL and planning algorithms to $\mathcal{TR}$ [10].

2.3 Overview of Transaction Logic

To make this dissertation self-contained, we provide a brief introduction to the parts of Transaction Logic that are needed for the understanding of this dissertation. For further details, the reader is referred to [21, 23, 24, 19, 22].

$\mathcal{TR}$ is a faithful extension of the first-order predicate calculus and so all of that syntax carries over. In this dissertation, we focus on rules, however, so we will be dealing exclusively with that subset of the syntax from now on. The most important new connectives that Transaction Logic brings in are:

1. ***Serial conjunction ($\otimes$)***: Like the classical conjunction, serial conjunction is a binary associative connective, but it is not commutative. Informally, the formula $\phi \otimes \psi$ is understood as a composite action that denotes an *execution* of ϕ followed by an execution of ψ.

2. ***Concurrent conjunction ($|$)***: Concurrent conjunction is a binary associative *and commutative* connective. Informally, $\phi | \psi$ says that ϕ and ψ can execute in an *interleaved* fashion. For instance, $(\alpha_1 \otimes \alpha_2)|(\beta_1 \otimes \beta_2)$ can execute as $\alpha_1, \beta_1, \alpha_2, \beta_2$, or as $\alpha_1, \beta_1, \beta_2, \alpha_2$, or as $\alpha_1, \alpha_2, \beta_1, \beta_2$, while $(\alpha_1 \otimes \alpha_2) \otimes (\beta_1 \otimes \beta_2)$ can execute only as $\alpha_1, \alpha_2, \beta_1, \beta_2$.

3. ***Isolation operator ($\odot$)***: It specifies that the formula in its scope must be executed without interleaving with other, concurrent subgoals. For instance, $(\alpha_1 \otimes \alpha_2)|\odot(\gamma_1 \otimes \gamma_2)|(\beta_1 \otimes \beta_2)$ means that the middle part of the transaction cannot interleave with the α-part of β-part. For instance, $\alpha_1, \beta_1, \alpha_2, \gamma_1, \gamma_2, \beta_2$ is a valid execution, but $\alpha_1, \beta_1, \gamma_1, \alpha_2, \gamma_2, \beta_2$ is not.

When ϕ and ψ are regular first-order formulas, both $\phi \otimes \psi$ and $\phi|\psi$ reduce to the usual first-order conjunction, $\phi \wedge \psi$. The logic also has other connectives but they are beyond the scope of this dissertation.

In addition, $\mathcal{TR}$ has a general, extensible mechanism of ***elementary updates*** or elementary ***actions*,** which have the important effect of taking the infamous *frame problem* out of many considerations in this logic (see [23, 24, 21, 77, 20]). Here we will use only the following two types of elementary actions, which are specifically designed on complete *STRIPS* states (Definition 1): $+p(t_1, \ldots, t_n)$ and $-p(t_1, \ldots, t_n)$, where p denotes an *extensional* predicate symbol of appropriate arity and t_1, ..., t_n are terms.

Given a state $\mathbf{S}$ and a *ground* elementary action $\alpha = +p(a_1, \ldots, a_n)$, an execution of α at state $\mathbf{S}$ deletes the literal $\neg p(a_1, \ldots, a_n)$ and adds the literal $p(a_1, \ldots, a_n)$. Similarly, executing $-p(a_1, \ldots, a_n)$ results in a state that is exactly like $\mathbf{S}$, but $p(a_1, \ldots, a_n)$ is deleted and $\neg p(a_1, \ldots, a_n)$ added. In some cases (e.g., if $p(a_1, \ldots, a_n) \in \mathbf{S}$), the action $+p(a_1, \ldots, a_n)$ has no effect, and similarly for $-p(a_1, \ldots, a_n)$.

A ***concurrent Horn rule*** is a statement of the form $h \leftarrow \phi$ where h is an atomic formula and ϕ is a *concurrent serial goal*, which is defined as follows:

Definition 6 (Concurrent serial goal)**.** *A* **concurrent serial goal** *is any formula of the form:*

- *An atomic formula; or*
- $\phi_1 \otimes \cdots \otimes \phi_n$ *where each* ϕ_i *is a concurrent serial goal, and* $n \geq 1$*; or*
- $\phi_1 | \ldots | \phi_n$ *where each* ϕ_i *is a concurrent serial goal, and* $n \geq 1$*; or*
- $\odot\phi$ *where* ϕ *is a concurrent serial goal.*

□

The informal meaning of such a rule is that h is a complex action and one way to execute h is to execute ϕ.

Thus, we now have regular first-order as well as concurrent-Horn rules. For simplicity (although this is not required by $\mathcal{TR}$), we assume that the sets of intentional predicates that can appear in the heads of regular rules and those in the heads of concurrent Horn rules are disjoint. Thus, we now have the following types of atomic statements:

- *Extensional atoms*.
- *Intentional atoms*: The atoms that appear in the heads of regular rules. These two categories of atoms populate database states and will be collectively called***fluents***.
  We will now allow any fluent (extensional or intensional) to be negated in the body of a concurrent Horn rule of the form $h \leftarrow \phi$, extending Definition 6.
- *Elementary actions*: $+p$, $-p$, where p is an extensional atom.
- *Complex actions*: These are the atoms that appear at the head of the concurrent Horn rules. Complex and elementary actions will be collectively called ***actions***.

As remarked earlier, for fluents $f \otimes g$ and $f|g$ are equivalent to $f \wedge g$ and, to emphasize this point, we will often write $f \wedge g$ for fluents even if they occur in the bodies of concurrent Horn rules. Note that a concurrent Horn rule all of whose body literals are fluents is essentially a regular rule, since all the $\otimes$-connectives and $|$-connectives can be replaced with $\wedge$. Therefore, one can view the regular rules as a special case of concurrent Horn rules.

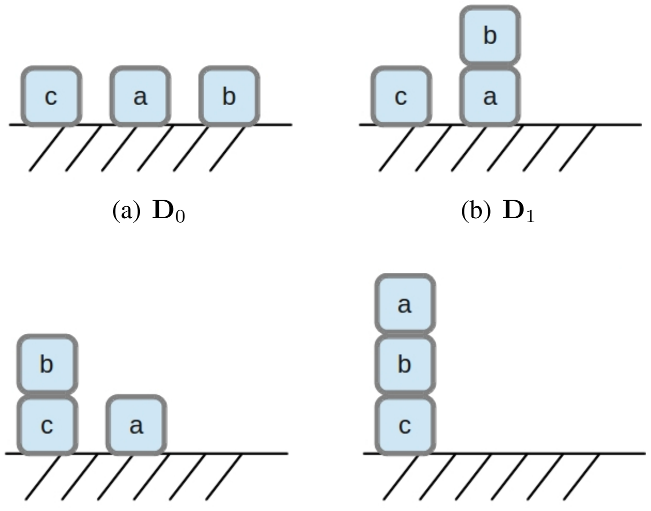

(a) $\mathbf{D}_0$ (b) $\mathbf{D}_1$ (c) $\mathbf{D}_2$ (d) $\mathbf{D}_3$

Figure 2.2: State change in BlocksWorld example.

The following example illustrates the above concepts.

$$\begin{array}{rcl} move(X,Y) & \leftarrow & (on(X,Z) \wedge clear(X) \wedge clear(Y) \wedge \neg tooHeavy(X)) \\ & & \otimes - on(X,Z) \otimes +on(X,Y) \otimes -clear(Y). \\ tooHeavy(X) & \leftarrow & weight(X,W) \wedge limit(L) \wedge W > L. \\ ?- \ move(b,X) \otimes move(Y,b). & & \end{array}$$

Here *on*, *clear*, *tooHeavy*, *weight*, and *limit* are fluents and the rest of atoms represent actions. The predicate *tooHeavy* is an intentional fluent, while *on*, *clear*, and *weight* are extensional fluents. The actions $+on(...)$, $-clear(...)$, $-on(...)$, and $+clear(...)$ are elementary and the intentional predicate $move$ represents a complex action. This example illustrates several features of Transaction Logic. The first rule is a concurrent Horn rule defining of a complex action of moving a block from one place to another. The second rule defines the intensional fluent $tooHeavy$, which is used in the definition of $move$ (under the scope of default negation). As the second rule does not include any action, it is a regular rule.

The last statement above is a *request to execute* a composite action, which is analogous to a query in logic programming. The request is to find some block, X, and move block b from its current position to the top of X and then find some other block, Y, and move it on top of b. Traditional logic programming offers no logical semantics for updates, so if after placing b on top of X the second operation ($move(Y, b)$) fails (say, all available blocks are too heavy), the effects of the first operation will persist and the underlying database becomes corrupted. In contrast, Transaction Logic gives update operators the logical semantics of an *atomic database transaction*. This means that if any part of the transaction fails, the effect is as if nothing was done at all. For example, if the second action in our example fails, all actions are "backtracked over" and the underlying database state remains unchanged.

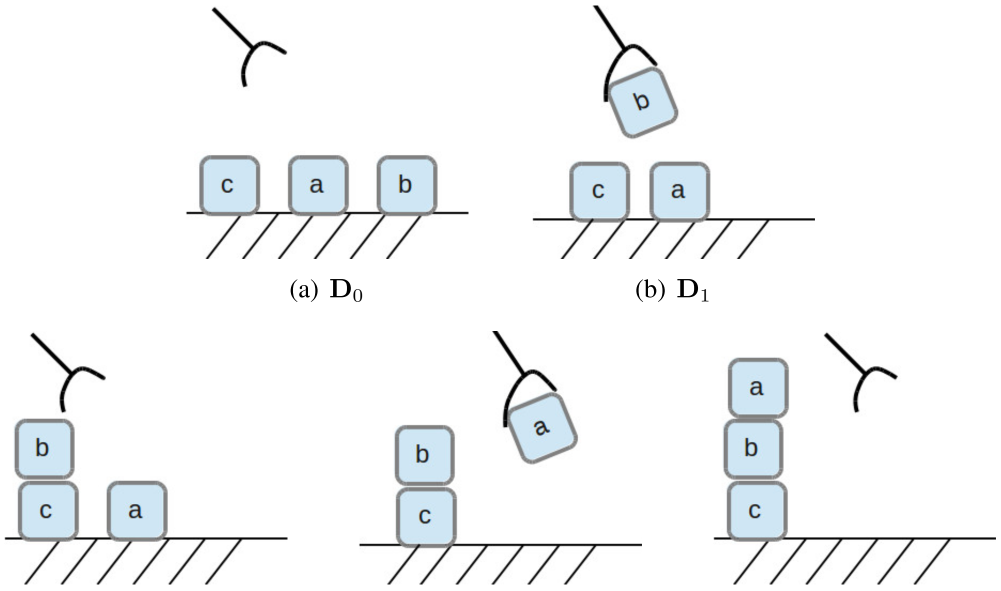
Consider the execution of $move(b, X) \otimes move(Y, b)$ in $\mathbf{D}_0$, a state shown in Figure 2.2(a), where $\{weight(a, 10), weight(b, 20), weight(c, 30), limit(25)\} \subseteq \mathbf{D}_0$. In traditional logic programming, substitution of X with a causes a state change from $\mathbf{D}_0$ to $\mathbf{D}_1$, at which point $move(Y, b)$ fails since block c is too heavy to be moved on top of b. Then, the underlying database remains in $\mathbf{D}_1$ and a wrong answer will be returned. In contrast, in Transaction Logic, if the state of database changes from $\mathbf{D}_0$ to $\mathbf{D}_1$ due to the substitution of X with a, the failure of $move(Y, b)$ causes backtracking through the update $move(b, a)$. On a retry, the right answer will be returned through the execution of $move(b, c) \otimes move(a, b)$, which causes state changes from $\mathbf{D}_0$ to $\mathbf{D}_2$ and from $\mathbf{D}_2$ to $\mathbf{D}_3$.

This semantics is given in purely model-theoretic terms and here we will only give an informal overview. The truth of any action in $\mathcal{TR}$ is determined over sequences of states—***execution paths***—which makes it possible to think of truth assignments in $\mathcal{TR}$'s models as executions. If an action, ϕ, defined by a set of serial rules, $\mathbb{P}$, evaluates to true over a sequence of states $\mathbf{D}_0, \ldots, \mathbf{D}_n$, we say that it can *execute* at state $\mathbf{D}_0$ by passing through the states $\mathbf{D}_1$, ..., $\mathbf{D}_{n-1}$, ending in the final state $\mathbf{D}_n$. This is captured by the notion of ***executional entailment***, which is written as follows:

$$\mathbb{P}, \mathbf{D}_0 \ldots \mathbf{D}_n \models \phi \tag{2.1}$$

The next example further illustrates $\mathcal{TR}$ by showing a definition of a recursive action.

Example 2 (Pyramid building)**.** *The following rules define a complex operation of stacking blocks to build a pyramid. It uses some of the already familiar fluents and actions from the previous example. In addition, it defines the actions* pickup,



(a) $\mathbf{D}_0$ (b) $\mathbf{D}_1$

(c) $\mathbf{D}_2$ (d) $\mathbf{D}_3$ (e) $\mathbf{D}_4$

Figure 2.3: State changes by $\mathcal{TR}$ rules in Example 2.

putdown*, and a recursive action* $stack$*.*

$$\begin{array}{l}
stack(0, AnyBlock) \leftarrow . \\
stack(N, X) \leftarrow N > 0 \otimes move(Y, X) \otimes stack(N-1, Y) \otimes on(Y, X). \\
move(X, Y) \leftarrow X \neq Y \otimes pickup(X) \otimes putdown(X, Y). \\
pickup(X) \leftarrow clear(X) \otimes on(X, Y) \otimes -on(X, Y) \otimes +clear(Y). \\
pickup(X) \leftarrow clear(X) \otimes on(X, table) \otimes -on(X, table). \\
putdown(X, Y) \leftarrow clear(Y) \otimes \neg on(X, Z1) \otimes \neg on(Z2, X) \otimes \\
\qquad\qquad\qquad -clear(Y) \otimes +on(X, Y).
\end{array} \tag{2.2}$$

The first rule says that stacking zero blocks on top of X *is a no-op. The second rule says that, for bigger pyramids, stacking* N *blocks on top of* X *involves moving some other block,* Y*, on* X *and then stacking* $N-1$ *blocks on* Y*. To make sure that the planner did not remove* Y *from* X *while building the pyramid on* Y*, we are verifying that* $on(Y, X)$ *continues to hold at the end. The remaining rules are self-explanatory. Indeed,* $\mathbb{P}, \mathbf{D}_0\mathbf{D}_1\mathbf{D}_2\mathbf{D}_3\mathbf{D}_4 \models stack(3, table)$*, where* $\mathbf{D}_0, \ldots, \mathbf{D}_4$ *are states shown in Figure 2.3.* □

An inference system for concurrent-Horn $\mathcal{TR}$ is described in [22]. The inference system is analogous to the well-known SLD resolution proof strategy for

Horn clauses plus some $\mathcal{TR}$-specific inference rules and axioms. The aim of such inference system is to prove statements of the form $\mathbb{P}, \mathbf{D} \cdots \vdash \phi$, called ***sequents***. Here $\mathbb{P}$ is a set of concurrent Horn rules and ϕ is a *concurrent serial goal* (see Definition 6). A proof of a sequent of this form is interpreted as a proof that action ϕ defined by the rules in $\mathbb{P}$ can be successfully executed starting at state $\mathbf{D}$.

An inference succeeds if it finds an ***execution*** for the transaction ϕ, i.e., a sequence of database states $\mathbf{D}_1, \ldots, \mathbf{D}_n$ such that $\mathbb{P}, \mathbf{D}\,\mathbf{D}_1 \ldots \mathbf{D}_n \vDash \phi$. Here we will use the following inference system, which we present in a simplified form—only the version for ground facts and rules. The inference rules can be read either top-to-bottom (if *top* is proved then *bottom* is proved) or bottom-to-top (to prove *bottom* first prove *top*).

Definition 7 ($\mathcal{TR}$ inference System)**.** *Let $\mathbb{P}$ be a set of concurrent Horn rules and $\mathbf{D}$, $\mathbf{D}_1$, $\mathbf{D}_2$ denote states.*

- *Axiom: $\mathbb{P}, \mathbf{D} \cdots \vdash ()$, where $()$ is an empty clause (which is true at every state).*
- *Inference Rules*
  1. *Applying transaction definition: Suppose $t \leftarrow body$ is a rule in $\mathbb{P}$.*
  $$\frac{\mathbb{P}, \mathbf{D} \cdots \vdash \phi[body]}{\mathbb{P}, \mathbf{D} \cdots \vdash \phi[t]} \tag{I1}$$
  where $\phi[body]$ is a concurrent serial goal that contains $body$ as a subformula and $\phi[t]$ is $\phi[body]$ with $body$ replaced with t.
  2. *Querying the database: If $\mathbf{D} \models t$ then*
  $$\frac{\mathbb{P}, \mathbf{D} \cdots \vdash left \mid middle \mid right}{\mathbb{P}, \mathbf{D} \cdots \vdash left \mid t \otimes middle \mid right} \tag{I2}$$
  where $left$, $right$, $middle$ are concurrent serial goals each of which can be empty.
  3. *Performing elementary updates: If the elementary update t changes the state $\mathbf{D}_1$ into the state $\mathbf{D}_2$ then*
  $$\frac{\mathbb{P}, \mathbf{D}_2 \cdots \vdash left \mid middle \mid right}{\mathbb{P}, \mathbf{D}_1 \cdots \vdash left \mid t \otimes middle \mid right} \tag{I3}$$
  where $left$, $right$, $middle$ are concurrent serial goals each of which can be empty.

4. *Executing atomic transactions:*

$$\frac{\mathbb{P}, \mathbf{D} \cdots \vdash \alpha \otimes (left \mid right)}{\mathbb{P}, \mathbf{D} \cdots \vdash left \mid \odot\alpha \mid right} \tag{I4}$$

where $left$, $right$, and α are concurrent serial goals and $left$, $right$ can be empty. □

A ***proof*** of a sequent, seq_n, is a series of sequents, seq_1, seq_2, $\dots$, seq_{n-1}, seq_n, where each seq_i is either an axiom-sequent or is derived from earlier sequents by one of the above inference rules. This inference system has been proven sound and complete with respect to the model theory of $\mathcal{TR}$ [22]. This means that if ϕ is a concurrent serial goal, the executional entailment $\mathbb{P}, \mathbf{D}_0\, \mathbf{D}_1 \dots \mathbf{D}_n \models \phi$ holds if and only if there is a proof of $\mathbb{P}, \mathbf{D}_0 \cdots \vdash \phi$ over the execution path $\mathbf{D}_0, \mathbf{D}_1, \dots, \mathbf{D}_n$. In this case, we will also say that such a proof derives $\mathbb{P}, \mathbf{D}_0\, \mathbf{D}_1 \dots \mathbf{D}_n \vdash \phi$.

Let the sequence of sequents seq_1, seq_2, $\dots$, seq_{n-1}, seq_n be a proof of seq_n. It is ***non-redundant*** if none of its sub-sequences is a proof of seq_n.

Lemma 1 (Linearity of proofs)**.** *Let $seq_1, \dots, seq_n$ be a non-redundant proof of seq_n. Then each seq_i is either an axiom-sequent or is derived from seq_{i-1} by one of the inference rules in Definition 7.*

Proof. Assume, to the contrary, that some sequent seq_i is derived from some seq_j, where $j < i - 1$. Suppose that i is the *largest* sequent number with such a property. Then seq_{i-1} can be removed from $seq_1, \dots, seq_n$ and the shorter sequence will still be a proof of seq_n. This is because seq_{i-1} is not needed to derive seq_i (since seq_i is derivable from seq_j), and all the sequents after seq_i are derivable from their immediately preceding sequents. Thus, the above proof is redundant, a contradiction. □

The following argument will be used multiple times in the proofs, so we make a separate lemma out of it.

Lemma 2 (Concurrent attachment of queries)**.** *Suppose there is a derivation for $\mathbb{P}, \mathbf{D}_0 \cdots \mathbf{D}_n \vdash \phi_1$, and also derivations for $\mathbb{P}, \mathbf{D}_n \vdash \phi_i$, where $2 \le i \le n$, such that the latter use only the $\mathcal{TR}$ inference rules (I1) and (I2). Then there is a derivation of $\mathbb{P}, \mathbf{D}_0 \cdots \mathbf{D}_n \vdash \phi_1 | \cdots | \phi_n$.*

Proof. Suppose $\mathbb{P}, \mathbf{D}_n \vdash (), \ldots, \mathbb{P}, \mathbf{D}_n \vdash \psi'$ is a derivation for $\mathbb{P}, \mathbf{D}_n \vdash \psi'$ that uses only the rules (I1) and (I2), i.e., ψ' is a query and the database state does not change during the derivation. Let also $\mathbb{P}, \mathbf{D}_0 \cdots \vdash (), \ldots, \mathbb{P}, \mathbf{D}_n \cdots \vdash \psi$ be a derivation for $\mathbb{P}, \mathbf{D}_0 \cdots \mathbf{D}_n \vdash \psi$. Then the following is a derivation for $\mathbb{P}, \mathbf{D}_0 \cdots \mathbf{D}_n \vdash \psi|\psi'$:

$$\begin{array}{l}\mathbb{P}, \mathbf{D}_0 \cdots \vdash (),\ \ldots\ ,\ \mathbb{P}, \mathbf{D}_n \cdots \vdash \psi, \\ \qquad \mathbb{P}, \mathbf{D}_n \cdots \vdash \psi|(),\ \ldots\ ,\ \mathbb{P}, \mathbf{D}_n \cdots \vdash \psi|\psi'\end{array}$$

The lemma now follows by applying the above step repeatedly, where first $\psi = \phi_1$ and $\psi' = \phi_2$; then $\psi = \phi_1|\phi_2$ and $\psi' = \phi_3$; then $\psi = \phi_1|\phi_2|\phi_3$ and $\psi' = \phi_4$; etc. $\square$

Chapter 3

STRIPS Planning Using $\mathcal{TR}$

In this chapter, we first introduce our $\mathcal{TR}$-*STRIPS* planner. Using and example, we also briefly illustrate how $\mathcal{TR}$ inference system, introduced in Section 2.3, uses $\mathcal{TR}$ planning rules to synthesize a plan. Then we will show that how some slight changes in the proposed planner can result in a much faster planner called *fSTRIPS*.

3.1 The $\mathcal{TR}$-*STRIPS* Planner

The informal idea of using $\mathcal{TR}$ as a planning formalism and an encoding of *STRIPS* as a set of $\mathcal{TR}$ rules first appeared in an unpublished report [21]. The encoding was incomplete and it did not include ramification and intensional predicates. We extend the original method with intentional predicates, make it complete, and formulate and prove the completeness of the resulting planner.

Definition 8 (Enforcement operator)**.** *Let G be a set of extensional literals. We define Enf*$(G) = \{+p \mid p \in G\} \cup \{-p \mid \neg p \in G\}$*. In other words, Enf*$(G)$ *is the set of elementary updates that makes G true.* □

Next we introduce a natural correspondence between *STRIPS* actions and $\mathcal{TR}$ rules.

Definition 9 (Actions as $\mathcal{TR}$ rules)**.** *Let* $\alpha = \langle p_\alpha(\overline{X}), Pre_\alpha, E_\alpha \rangle$ *be a STRIPS action. We define its* ***corresponding*** $\mathcal{TR}$ ***rule****,* $tr(\alpha)$*, to be a rule of the form*

$$p_\alpha(\overline{X}) \leftarrow (\wedge_{\ell \in Pre_\alpha} \ell) \ \otimes \ (\otimes_{u \in \mathit{Enf}(E_\alpha)} u). \tag{3.1}$$

□

Note that in (3.1) the actual order of action execution in the last component, $\otimes_{u \in \mathit{Enf}(E_\alpha)} u$, is immaterial, since all such executions happen to lead to the same state.

We now define a set of $\mathcal{TR}$ clauses that simulate the well-known *STRIPS* planning algorithm and extend this algorithm to handle intentional predicates and rules. The reader familiar with the *STRIPS* planner should not fail to notice that, in essence, these rules are a natural (and much more concise and general) verbalization of the classical *STRIPS* algorithm [32]. However—importantly—unlike the original *STRIPS*, these rules constitute a *complete* planner when evaluated with the $\mathcal{TR}$ proof theory.

Definition 10 ($\mathcal{TR}$ planning rules)**.** *Let* $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{S} \rangle$ *be a planning problem (see Definition 4). We define a set of* $\mathcal{TR}$ *rules,* $\mathbb{P}(\Pi)$*, which provides a sound and complete solution to the planning problem.* $\mathbb{P}(\Pi)$ *has three disjoint parts,* $\mathbb{P}_{\mathbb{R}}$*,* $\mathbb{P}_{\mathbb{A}}$*, and* $\mathbb{P}_G$*, described below.*

- *The* $\mathbb{P}_{\mathbb{R}}$ *part: for each rule* $p(\overline{X}) \leftarrow p_1(\overline{X}_1) \wedge \cdots \wedge p_n(\overline{X}_n)$ *in* $\mathbb{R}$*,* $\mathbb{P}_{\mathbb{R}}$ *has a rule of the form*

$$achieve_p(\overline{X}) \leftarrow \|_{i=1}^{n} achieve_{p_i}(\overline{X}_i). \tag{P1}$$

  Rule (P1) is an extension to the classical STRIPS planning algorithm and is intended to capture intentional predicates and ramification of actions; it is the only major aspect of our $\mathcal{TR}$*-based rendering of STRIPS that was not present in the original in one way or another.*

- *The part* $\mathbb{P}_{\mathbb{A}} = \mathbb{P}_{actions} \cup \mathbb{P}_{atoms} \cup \mathbb{P}_{achieves}$ *is constructed out of the actions in* $\mathbb{A}$ *as follows:*

  - $\mathbb{P}_{actions}$*: for each* $\alpha \in \mathbb{A}$*,* $\mathbb{P}_{actions}$ *has a rule of the form*

$$p_\alpha(\overline{X}) \leftarrow (\wedge_{\ell \in Pre_\alpha} \ell) \ \otimes \ (\otimes_{u \in \mathit{Enf}(E_\alpha)} u). \tag{P2}$$

    This is the $\mathcal{TR}$ *rule that corresponds to the action* α*, introduced in Definition 9.*

  - $\mathbb{P}_{atoms} = \mathbb{P}_{achieved} \cup \mathbb{P}_{enforced}$ *has two disjoint parts as follows:*

    - $\mathbb{P}_{achieved}$*: for each extensional predicate* $p \in \mathcal{P}_{ext}$*,* $\mathbb{P}_{achieved}$ *has the rules*

$$\begin{array}{l} achieve_p(\overline{X}) \leftarrow p(\overline{X}). \\ achieve_{\neg p}(\overline{X}) \leftarrow \neg p(\overline{X}). \end{array} \tag{P3}$$

These rules say that if an extensional literal is true in a state then that literal has already been achieved as a goal.

– $\mathbb{P}_{enforced}$*: for each action* $\alpha = \langle p_\alpha(\overline{X}), Pre_\alpha, E_\alpha \rangle$ *in* $\mathbb{A}$ *and each* $e(\overline{Y}) \in E_\alpha$*,* $\mathbb{P}_{enforced}$ *has the following rule:*

$$achieve_e(\overline{Y}) \leftarrow execute_{p_\alpha}(\overline{X}). \tag{P4}$$

This rule says that one way to achieve a goal that occurs in the effects of an action is to execute that action.

– $\mathbb{P}_{achieves}$*: for each action* $\alpha = \langle p_\alpha(\overline{X}), Pre_\alpha, E_\alpha \rangle$ *in* $\mathbb{A}$*,* $\mathbb{P}_{achieves}$ *has the following rule:*

$$execute_{p_\alpha}(\overline{X}) \leftarrow (\|_{\ell \in Pre_\alpha} achieve_\ell) \otimes \odot p_\alpha(\overline{X}). \tag{P5}$$

This means that to execute an action, one must first achieve the precondition of the action and then perform the state changes prescribed by the action.

- $\mathbb{P}_G$*: Let* $G = \{g_1, ..., g_k\}$*. Then* $\mathbb{P}_G$ *has a rule of the form:*

$$achieve_G \leftarrow (\|_{i=1}^{k} achieve_{g_i}) \otimes (\wedge_{i=1}^{k} g_i). \tag{P6}$$

□

Given a set $\mathbb{R}$ of rules, a set $\mathbb{A}$ of *STRIPS* actions, an initial state **S**, and a goal G, Definition 10 gives a set of $\mathcal{TR}$ rules that specify a planning strategy for that problem. To find a solution for that planning problem, one simply needs to place the request

$$? - achieve_G\,. \tag{3.2}$$

at a desired initial state and use the $\mathcal{TR}$'s inference system of Section 2.3 to find a proof. The inference system in question is sound and complete for *concurrent Horn clauses*, and the rules in Definition 10 satisfy that requirement.

Example 3 (Planning rules for register exchange)**.** *Consider the classical problem of swapping two registers in a computer from [73]. The reason this problem is interesting is because it is the simplest problem where the original STRIPS is incomplete. Example 4 explains why and how our complete* $\mathcal{TR}$*-based planner handles the issue.*

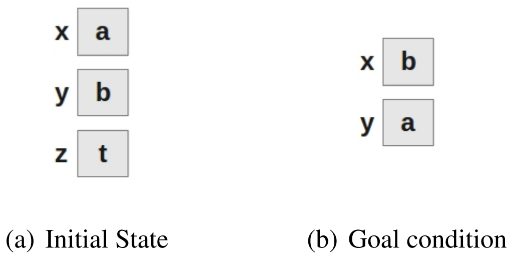


Figure 3.1: Initial state and goal condition of Example 3.

Consider two memory registers, x and y, with initial contents a and b, respectively. The goal is to find a plan to exchange the contents of these registers with the help of an auxiliary register, z. Let the extensional predicate $value(Reg, Val)$ represent the content of a register. Then the initial state of the system is $\{value(x, a), value(y, b), value(z, t)\}$, shown in Figure 3.1(a). Suppose the only available action is copy $=$ $\langle copy(Src, Dest, V), \{value(Src, V), value(Dest, V')\}, \{\neg value(Dest, V'), value(Dest, V)\}\rangle$, which copies the value V of the source register, Src, to the destination register $Dest$. The old value of $Dest$ is erased and the value of Src is written over. The planning goal is $G = \{value(x, b), value(y, a)\}$, shown in Figure 3.1(b). Per Definition 10, the planning rules for this problem are as follows.

Due to case (P2):

$$\begin{aligned} copy(Src, Dest, V) \;\leftarrow\;& value(Src, V) \otimes value(Dest, V') \otimes \\ & -value(Dest, V') \otimes +value(Dest, V). \end{aligned} \tag{3.3}$$

Due to (P3), (P4), and (P5):

$$\begin{aligned} &achieve_{value}(R, V) \leftarrow value(R, V). \\ &achieve_{\neg value}(R, V) \leftarrow \neg value(R, V). \end{aligned} \tag{3.4}$$

$$achieve_{value}(Dest, V) \leftarrow execute_{copy}(Src, Dest, V). \tag{3.5}$$

$$\begin{aligned} execute_{copy}(Src, Dest, V) \;\leftarrow\;& (\, achieve_{value}(Src, V) \mid \\ & \quad achieve_{value}(Dest, V')\,) \otimes \\ & copy(Src, Dest, V). \end{aligned} \tag{3.6}$$

Due to (P6):

$$\begin{aligned} achieve_G \;\leftarrow\;& (achieve_{value}(x, b) \mid achieve_{value}(y, a)) \\ & \otimes (value(x, b) \wedge value(y, a)). \end{aligned} \tag{3.7}$$

Case (P1) of Definition 10 does not contribute rules in this example because the planning problem does not involve intensional fluents. □

As mentioned before, a solution plan for a *STRIPS* planning problem is a sequence of actions leading to a state that satisfies the planning goal. Such a sequence can be extracted by picking out the atoms of the form p_α from a successful derivation branch generated by the $\mathcal{TR}$ inference system. Since each p_α uniquely corresponds to a *STRIPS* action, this provides us with the requisite sequence of actions that constitutes a plan.

Suppose $seq_0, \ldots, seq_m$ is a deduction by the $\mathcal{TR}$ inference system. Let $i_1, \ldots, i_n$ be exactly those indexes in that deduction where the inference rule (I1) was applied to some sequent using a rule of the form $tr(\alpha_{i_r})$ introduced in Definition 9. We will call $\alpha_{i_1}, \ldots, \alpha_{i_n}$ the ***pivoting sequence of actions***. The corresponding ***pivoting sequence of states*** $\mathbf{D}_{i_1}, \ldots, \mathbf{D}_{i_n}$ is a sequence where each $\mathbf{D}_{i_r}$, $1 \leq r \leq n$, is the state at which α_{i_r} is applied. We will prove that the pivoting sequence of actions is a solution to the planning problem.

Lemma 3 (Execution of *STRIPS* actions)**.** *Let $\alpha = \langle p_\alpha(\overline{X}), Pre_\alpha, E_\alpha \rangle \in \mathbb{A}$ be an action. Then:*

1. *If α is applied in state $\mathbf{D}$ in STRIPS and the system transitions to state $\mathbf{D}'$, then $\{tr(\alpha)\}, \mathbf{D} \ldots \mathbf{D}' \vdash p_\alpha(\overline{X})$.*

2. *If $\{tr(\alpha)\}, \mathbf{D} \ldots \mathbf{D}' \vdash p_\alpha(\overline{X})$ then applying the action α in $\mathbf{D}$ in STRIPS is possible and it transitions the system to state $\mathbf{D}'$.*

Proof. For the first claim, applying α in state $\mathbf{D}$ in *STRIPS* amounts to an execution of a sequence of inserts and deletes specified in E_α. By construction of $tr(\alpha)$, this is exactly what happens when one executes $p_\alpha(\overline{X})$, i.e., derives it via the inference rule (I1) with the help of the planning rule $tr(\alpha)$.

Conversely, if $\{tr(\alpha)\}, \mathbf{D} \ldots \mathbf{D}' \vdash p_\alpha(\overline{X})$ then, by construction of $tr(\alpha)$, the condition $\wedge_{\ell \in Pre_\alpha} \ell$ must be satisfied in $\mathbf{D}$. A derivation of $p_\alpha(\overline{X})$ using the planning rule $tr(\alpha)$ amounts to an execution of a the sequence of inserts and deletes specified in E_α, transitioning to state $\mathbf{D}'$. This means that α is applicable in *STRIPS* in state $\mathbf{D}$ and applying α in $\mathbf{D}$ leads to the same state $\mathbf{D}'$. □

All lemmas and theorems in this section assume that $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{D}_0 \rangle$ is a *STRIPS* planning problem and that $\mathbb{P}(\Pi)$ is the corresponding set of $\mathcal{TR}$ planning rules as in Definition 10.

Lemma 4 (Execution of transactions of the form p_α)**.** *Let $seq_0, \ldots, seq_m$ be a derivation of $\mathbb{P}(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_m \vdash achieve_G$, and $\alpha_1, \ldots, \alpha_n$ (resp., $\mathbf{D}_{i_1}, \ldots, \mathbf{D}_{i_n}$) be a pivoting sequence of actions (resp., states). Then $\mathbf{D}_0 = \mathbf{D}_{i_1}$, $\mathbb{P}(\Pi), \mathbf{D}_{i_r} \ldots \mathbf{D}_{i_{r+1}} \vdash p_{\alpha_r}$, for each r (p_{α_r} is the transaction associated with α_r—see Definition 9) such that $1 \leq r \leq n$, and $\mathbf{D}_{i_{n+1}} = \mathbf{D}_m$.*

Proof. We divide the proof in three parts, which together yield the desired result:

- $\mathbb{P}(\Pi), \mathbf{D}_0 \cdots \vdash p_{\alpha_1}$.
- $\mathbb{P}(\Pi), \mathbf{D}_{i_n} \ldots \mathbf{D}_m \vdash p_{\alpha_n}$.
- $\mathbb{P}(\Pi), \mathbf{D}_{i_r} \ldots \mathbf{D}_{i_{r+1}} \vdash p_{\alpha_r}$ for each r such that $1 \leq r \leq n$.

An easy examination of the inference rules and the $\mathcal{TR}$ planning rules with which these inference rules work shows that the only $\mathcal{TR}$ planning rules that directly cause the execution of an elementary $\mathcal{TR}$ transactions (and thus change the underlying states) are the rules of the form $tr(\alpha)$. Thus every elementary $\mathcal{TR}$ update in the derivation of $\mathbb{P}(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_m \vdash achieve_G$ is associated with some $tr(\alpha)$ and with a pivoting action α. By Definition 10, the only $\mathcal{TR}$ planning rules that call the heads of $tr(\alpha)$s are the rules of the form (P5). Because of the isolation operator $\odot$ in (P5), there can be no interleaving between the execution of p_αs (that is, once a p_α starts, the body of $tr(\alpha)$ will be executed without interleaving with any other update).

By the definition of pivoting sequence of actions and states, we have $\mathbb{P}(\Pi), \mathbf{D}_{i_1} \cdots \vdash p_{\alpha_1}$ where $\mathbf{D}_{i_1}$ is the pivoting state of α_1. We need to show that $\mathbf{D}_{i_1} = \mathbf{D}_0$. Assume, to the contrary, that the state $\mathbf{D}_{i_1}$ strictly follows state $\mathbf{D}_0$. Then there must be at least one elementary $\mathcal{TR}$ update $e \notin E_{\alpha_1}$ that was executed by the inference rule (I3) at state $\mathbf{D}_0$. Since, as noted above, every elementary $\mathcal{TR}$ update corresponds to a pivoting action, there must have been some other pivoting action α' that was executed before α_1. This contradicts the assumption that α_1 is the first pivoting action. Thus, $\mathbf{D}_{i_1} = \mathbf{D}_0$.

Suppose $\mathbb{P}(\Pi), \mathbf{D}_{i_n} \ldots \mathbf{D}_{i_{n+1}} \vdash p_{\alpha_{i_n}}$. We will show that $\mathbf{D}_{i_{n+1}} = \mathbf{D}_m$. If $\mathbf{D}_{i_{n+1}}$ strictly precedes $\mathbf{D}_m$, there must be some elementary $\mathcal{TR}$ update $e \notin E_{\alpha_n}$ that was executed by the inference rule (I3) at state $\mathbf{D}_{i_{n+1}}$. As every elementary $\mathcal{TR}$ update is associated with a pivoting action, there must have been another pivoting action α' that was executed after α_n. This contradicts the assumption that α_n is the last pivoting action. Thus, $\mathbf{D}_{i_{n+1}} = \mathbf{D}_m$.

By the definition of pivoting actions and states, we have $\mathbb{P}(\Pi), \mathbf{D}_{i_j} \cdots \vdash p_{\alpha_{i_j}}$ where $1 \leq j \leq n$. We need to show that $\mathbb{P}(\Pi), \mathbf{D}_{i_j} \ldots \mathbf{D}_{i_{j+1}} \vdash p_{\alpha_{i_j}}$. Assume, to the contrary, that $\mathbb{P}(\Pi), \mathbf{D}_{i_j} \ldots \mathbf{D}'_{i_j} \vdash p_{\alpha_{i_j}}$ and $\mathbf{D}'_{i_j} \neq \mathbf{D}_{i_{j+1}}$. Recall that $p_{\alpha_{i_{j+1}}}$ cannot start before $p_{\alpha_{i_j}}$ has finished due to the $\odot$ operator. Therefore the state $\mathbf{D}'_{i_j}$ precedes $\mathbf{D}_{i_{j+1}}$ and there must have been an elementary $\mathcal{TR}$ update $e \notin E_{\alpha_j}$ that was executed by the inference rule (I3) at state $\mathbf{D}'_{i_j}$. Since every elementary $\mathcal{TR}$ update in that derivation corresponds to a pivoting action, there must have been another pivoting action α' that was executed after α_j and before α_{j+1}. This contradicts the assumption that α_{j+1} is the pivoting action that immediately follows α_i. Thus $\mathbf{D}'_{i_j} = \mathbf{D}_{i_{j+1}}$, which concludes the proof. □

Lemmas 3 and 4 play key roles in proving the soundness of our *STRIPS*-inspired $\mathcal{TR}$ planning strategy.

Theorem 1 (Soundness of $\mathcal{TR}$ planning)**.** *Any pivoting sequence of actions in the derivation of* $\mathbb{P}(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_m \vdash achieve_G$ *is a solution plan.*[1]

Proof. Consider a derivation of $\mathbb{P}(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_m \vdash achieve_G$. Suppose $\alpha_{i_1}, \ldots, \alpha_{i_n}$ is the pivoting sequence of actions for this derivation and $\mathbf{D}_{i_1}, \ldots, \mathbf{D}_{i_n}$ is the corresponding pivoting sequence of states. Let $\mathbf{D}_{i_{n+1}}$ be $\mathbf{D}_m$. By the construction of $achieve_G$, we know that $\mathbf{D}_m \models G$. Due to the soundness of the $\mathcal{TR}$ inference rules, we also know that $\mathbb{P}(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_m \models achieve_G$. By Lemma 4 and by the soundness of the inference system, $\mathbb{P}(\Pi), \mathbf{D}_{i_r} \ldots \mathbf{D}_{i_{r+1}} \models p_{\alpha_r}$, for each $1 \leq r \leq n$. By Lemma 3, executing α_r in state $\mathbf{D}_{i_r}$ in *STRIPS* takes the system to state $\mathbf{D}_{i_{r+1}}$. This means that after the execution of the *STRIPS* sequence $\alpha_1, \ldots, \alpha_n$, one gets to state $\mathbf{D}_{i_{n+1}}$, which satisfies $achieve_G$ and thus G. The proof concludes by recalling that $\mathbf{D}_{i_1} = \mathbf{D}_0$ (by Lemma 4) and $\mathbf{D}_m = \mathbf{D}_{i_{n+1}}$ (by definition). □

Completeness of a planning strategy means that, for any *STRIPS* planning problem, if there is a solution, the planner will find *at least one* plan. A stronger statement about completeness is called ***comprehensive completeness***: if there is a *non-redundant* solution for a *STRIPS* planning problem, the planner will find *exactly that* plan.

Lemma 5 (Achieved literals for $\mathcal{TR}$ planning)**.** *Let* $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \_ \rangle$,[2] *be a planning problem and* $\mathbb{P}(\Pi)$ *be the set of* $\mathcal{TR}$ *planning rules in Definition 10. Then, for every literal* ℓ *and every state* $\mathbf{S}$*, the following holds:* $\mathbb{P}(\Pi), \mathbf{S} \models \ell$ *if and only if* $\mathbb{P}(\Pi), \mathbf{S} \vdash achieve_\ell$*.*

Proof. We have two cases:

[1]Sequents of the form $\mathbb{P}(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_m \vdash \ldots$ were defined at the very end of Section 2.3.

[2]In $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \_ \rangle$, $\_$ denotes that the initial state of Π is immaterial.

1. *ℓ is extensional.* Suppose $\mathbb{P}(\Pi), \mathbf{S} \models \ell$. Then due to the first planning rule in (P3), the sequent $\mathbb{P}(\Pi), \mathbf{S} \vdash achieve_\ell$ is derivable via the inference rules (I1) and (I2). To prove the *only if* part, suppose $\mathbb{P}(\Pi), \mathbf{S} \vdash achieve_\ell$. To derive $\mathbb{P}(\Pi), \mathbf{S} \vdash achieve_\ell$, two types of rules in $\mathbb{P}(\Pi)$ can be used: $achieve_\ell \leftarrow \ell$ and $achieve_\ell \leftarrow execute_{p_\alpha}$, where $\ell \in E(\alpha)$. In the first case, $\mathbb{P}(\Pi), \mathbf{S} \vdash \ell$ must be established as part of the proof of $\mathbb{P}(\Pi), \mathbf{S} \vdash achieve_\ell$, and thus $\mathbb{P}(\Pi), \mathbf{S} \models \ell$ must hold.

   In the second case, the derivation of $\mathbb{P}(\Pi), \mathbf{S} \vdash achieve_\ell$ implies that no state change has taken place, so it must be the case that $E(\alpha) \subseteq \mathbf{S}$. Therefore, $\ell \in \mathbf{S}$ and thus also $\mathbb{P}(\Pi), \mathbf{S} \models \ell$.

2. *ℓ is intensional.* Note that rules from $\mathbb{R}$ are the *only* rules in $\mathbb{P}(\Pi)$ that have intensional literals in their heads. Therefore, $\mathbb{P}(\Pi), \mathbf{S} \models \ell$ implies that there must be at least one derivation of ℓ formed by a set of ground instance rules from $\mathbb{R}$ that are true in $\mathbf{S}$, i.e., $\mathbb{R}, \mathbf{S} \vdash \ell$. By construction of $\mathcal{TR}$'s planning rules, each rule of the form $p(\overline{X}) \leftarrow \wedge_{i=1}^{n} p_i(\overline{X}_i)$ in $\mathbb{R}$ has a corresponding rule of the form $achieve_p(\overline{X}) \leftarrow \|_{i=1}^{n} achieve_{p_i}(\overline{X}_i)$ in $\mathbb{P}(\Pi)$. Therefore, any derivation of $\mathbb{R}, \mathbf{S} \vdash \ell$ has a corresponding derivation for $\mathbb{P}(\Pi), \mathbf{S} \vdash achieve_\ell$. To prove the *only if* part, suppose $\mathbb{P}(\Pi), \mathbf{S} \vdash achieve_\ell$. Since rules of the form $achieve_p(\overline{X}) \leftarrow \|_{i=1}^{n} achieve_{p_i}(\overline{X}_i)$ are the only ones in $\mathbb{P}(\Pi)$ that can be used to derive $\mathbb{P}(\Pi), \mathbf{S} \vdash achieve_\ell$, there must be at least one derivation of $achieve_\ell$ formed by a set of such rules. Again, using above mentioned correspondence, the derivation of $\mathbb{P}(\Pi), \mathbf{S} \vdash achieve_\ell$ can be used to construct a derivation for $\mathbb{P}(\Pi), \mathbf{S} \vdash \ell$. Thus $\mathbb{P}(\Pi), \mathbf{S} \models \ell$.

□

Lemma 6 (Plans of length zero)**.** *Let $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{D} \rangle$ be a planning problem that has a solution plan of length zero. Then the $\mathcal{TR}$ inference system can generate a derivation for $\mathbb{P}(\Pi), \mathbf{D} \vdash achieve_G$ whose pivoting sequence is empty.*

Proof. If Π has a plan of length zero then $G \subseteq \mathbf{D}$. By Lemma 5, $\mathbb{P}(\Pi), \mathbf{D} \vdash \|_{g \in G} achieve_g$, from which we readily get $\mathbb{P}(\Pi), \mathbf{D} \vdash achieve_G$. Since this derivation does not involve the planning rules $tr(\alpha)$, it has an empty pivoting sequence. □

Lemma 7 (Plans of length $> \mathbf{1}$: the inductive step)**.** *Suppose that, if a STRIPS planning problem $\Pi' = \langle \mathbb{R}, \mathbb{A}, G', \mathbf{D}_0 \rangle$ has a non-redundant plan σ' of length $k \leq n$, then it has a $\mathcal{TR}$-derivation for the sequent $\mathbb{P}(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \vdash achieve_{G'}$*

whose pivoting sequence is σ' and $\mathbf{D}_{f'}$ is the final state of the execution of σ' starting at $\mathbf{D}_0$.

Then, for any planning problem $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{D}_0 \rangle$ that has a non-redundant plan σ of length $n+1$, there is a $\mathcal{TR}$-derivation for $\mathbb{P}(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_f \vdash achieve_G$ whose pivoting sequence is σ and $\mathbf{D}_f$ is the final state of the execution of σ starting at $\mathbf{D}_0$.

Proof. Let $\sigma = \alpha_1, \ldots, \alpha_n, \alpha_{n+1}$ and $\sigma' = \alpha_1, \ldots, \alpha_n$ and $\mathbf{D}_f$, $\mathbf{D}_{f'}$ be as in the statement of the lemma. Consider the planning problem $\Pi' = \langle \mathbb{R}, \mathbb{A}, G', \mathbf{D}_0 \rangle$ where $G' = (\mathbf{D}_{f'} \cap G) \cup Pre_{\alpha_{n+1}}$ (to remind: $Pre_{\alpha_{n+1}}$ is the precondition of α_{n+1}). We will show that σ' is a non-redundant plan of length n for Π'. Since α_{n+1} is executable at $\mathbf{D}_{f'}$, we conclude that $Pre_{\alpha_{n+1}} \subseteq \mathbf{D}_{f'}$. Since, clearly, $\mathbf{D}_{f'} \cap G \subseteq \mathbf{D}_{f'}$, we obtain $G' \subseteq \mathbf{D}_{f'}$ and thus σ' is a plan for Π'. It is a *non-redundant* plan for Π' because if ς were a subsequence of σ' and also a plan for Π' then α_{n+1} would have been executable in the final state of ς (since $Pre_{\alpha_{n+1}} \subseteq G'$) and thus $\langle \varsigma, \alpha_{n+1} \rangle$ would have been a plan for Π. Since $\langle \varsigma, \alpha_{n+1} \rangle$ is a subsequence of σ, this would contradict the non-redundancy assumption about σ.

Since $\sigma' = \alpha_1 \ldots \alpha_n$ is a non-redundant n-plan for Π', the assumption of the lemma says that there is a $\mathcal{TR}$-derivation for a sequent of the form $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \vdash achieve_{G'}$ with σ' as the pivoting sequence. Therefore, with the help of the planning rule (P6), we also get a derivation for $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \vdash (\|_{g \in G'}\, achieve_g) \otimes (\wedge_{g \in G'}\, g)$. A prefix of that derivation (with the same pivoting sequence) gives us $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \vdash (\|_{g \in G'}\, achieve_g)$. Since $Pre_{\alpha_{n+1}} \subseteq \mathbf{D}_{f'}$, there must be a derivation for $\mathbb{P}(\Pi'), \mathbf{D}_{f'} \ldots \mathbf{D}_f \vdash p_{\alpha_{n+1}}$. Concatenating these two derivations yields a $\mathcal{TR}$-derivation for $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \ldots \mathbf{D}_f \vdash (\|_{g \in G'}\, achieve_g) \otimes p_{\alpha_{n+1}}$ and then $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \ldots \mathbf{D}_f \vdash (\|_{g \in G'}\, achieve_g) \otimes \odot p_{\alpha_{n+1}}$. This concatenation adds α_{n+1} to the pivoting sequence σ', turning it into σ. Recall that $\mathbb{P}(\Pi')$ has (P5), a rule of the form $execute_{p_{\alpha_{n+1}}} \leftarrow (\|_{g \in Pre_{\alpha_{n+1}}} achieve_g) \otimes \odot p_{\alpha_{n+1}}$, so the inference rule (I1) using (P5) yields a sequent of the form $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_f \vdash (\|_{g \in (G' \setminus Pre_{\alpha_{n+1}})}\, achieve_g) \,| execute_{p_{\alpha_{n+1}}}$, which can be rewritten as

$$\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_f \vdash (\|_{g \in G''}\, achieve_g) \,|\, execute_{p_{\alpha_{n+1}}} \tag{3.8}$$

where $G'' = (G \cap \mathbf{D}_{f'}) \setminus Pre_{\alpha_{n+1}}$.

Since σ is a non-redundant plan for Π, it follows that $G \setminus \mathbf{D}_{f'} \neq \emptyset$: if not, $\alpha_1, \ldots, \alpha_n$ would have been a plan for Π (starting at $\mathbf{D}_0$ and ending at $\mathbf{D}_{f'}$), contrary to non-redundancy of σ.

Let $\varrho \in G \setminus \mathbf{D}_{f'}$. Since $\mathbb{P}(\Pi'), \mathbf{D}_{f'} \ldots \mathbf{D}_f \vdash p_{\alpha_{n+1}}$, by Lemma 3, $\mathbf{D}_f$ is the final state of the execution of α_{n+1} at $\mathbf{D}_{f'}$. Therefore, $\mathbf{D}_f$ is the result of execution of σ

starting at $\mathbf{D}_0$. Clearly, $G \subseteq \mathbf{D}_f$, as σ is a plan for the planning problem Π. Since $\varrho \in G \subseteq \mathbf{D}_f$, we conclude that $\mathbb{P}(\Pi), \mathbf{D}_f \models \varrho$ and, by Lemma 5, $\mathbb{P}(\Pi), \mathbf{D}_f \vdash achieve_\varrho$. On the other hand, since $\varrho \notin \mathbf{D}_{f'}$, we have $\mathbb{P}(\Pi), \mathbf{D}_{f'} \nvdash achieve_\varrho$.

Next we will show that $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_f \vdash (\|_{g \in G''}\, achieve_g) \,|\, achieve_\varrho$. There are two cases:

1. ϱ *is extensional*: Then $\varrho \in E_{\alpha_{n+1}}$. If not, recall that $\varrho \notin \mathbf{D}_{f'}$ and since $\mathbf{D}_f$ is the state obtained by executing α_{n+1} at $\mathbf{D}_{f'}$, it would follow that $\varrho \notin \mathbf{D}_f$. Therefore, assuming $\varrho \notin E_{\alpha_{n+1}}$ contradicts the earlier conclusion that $\mathbb{P}(\Pi), \mathbf{D}_f \models \varrho$. Thus, $\varrho \in E_{\alpha_{n+1}}$ and, by Definition 10, $\mathbb{P}(\Pi)$ has a rule $achieve_\varrho \leftarrow execute_{p_{\alpha_{n+1}}}$. Applying the inference rule (I1) to the sequent (3.8) using $achieve_\varrho \leftarrow execute_{p_{\alpha_{n+1}}}$ yields the sequent $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_f \vdash (\|_{g \in G''}\, achieve_g) \,|\, achieve_\varrho$.

2. ϱ *is intensional*: Consider a derivation of $\mathbb{P}(\Pi), \mathbf{D}_f \vdash achieve_\varrho$. Note that the planning rules (P1) reproduced below

$$achieve_p(\overline{X}) \leftarrow \|_{i=1}^{n} achieve_{p_i}(\overline{X}_i) \tag{3.9}$$

are the only ones in $\mathbb{P}(\Pi)$ that can be used in the derivation of $\mathbb{P}(\Pi), \mathbf{D}_f \vdash achieve_\varrho$ via the inference rule (I1), and there is no other way to derive that sequent except the rules (I1) and (I2). Consider a *non-redundant* such derivation seq_0, ..., seq_m, where seq_0 is an axiom and seq_m is $\mathbb{P}(\Pi), \mathbf{D}_f \vdash achieve_\varrho$. By Lemma 1, each seq_i (except seq_0) was derived from seq_{i-1} via the inference rules (I1) and (I2), where the latter can use only the planning rules (3.9). This derivation must have a sequent of the form $\mathbb{P}(\Pi), \mathbf{D}_f \vdash achieve_h$, obtained via the $\mathcal{TR}$ inference rule (I1) using a planning rule of the form $achieve_h(\overline{X}) \leftarrow \ldots |achieve_{\varrho'}(\overline{X_{\varrho'}})$ in (3.9), where ϱ' is an extensional literal such that $\varrho' \in E_{\alpha_{n+1}}$. If there were no such ϱ', we could have used the same derivation to show that $\mathbb{P}(\Pi), \mathbf{D}_{f'} \models \varrho$, contrary to the earlier conclusion that $\mathbb{P}(\Pi), \mathbf{D}_{f'} \nvdash achieve_\varrho$. Since $\varrho' \in E_{\alpha_{n+1}}$, Definition 10 says that $\mathbb{P}(\Pi)$ has a rule of the form $achieve_{\varrho'}(\overline{X}) \leftarrow execute_{p_{\alpha_{n+1}}}(\overline{X})$. An application of the $\mathcal{TR}$ inference rule (I1) to the sequent (3.8) using $achieve_{\varrho'}(\overline{X}) \leftarrow execute_{p_{\alpha_{n+1}}}(\overline{X})$ then yields

$$\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_f \vdash (\|_{g \in G''}\, achieve_g) \,|\, achieve_{\varrho'}. \tag{3.10}$$

Now, rerun the inference steps for the aforesaid derivation seq_0, ..., seq_m starting with the sequent (3.10) instead of seq_0 and omitting the step that introduces the literal $achieve_{\varrho'}$ to the goal on the right-hand side of the sequents. This step can be omitted because $achieve_{\varrho'}$ already exists in (3.10). The result of such a derivation is the sequent $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_f \vdash (\|_{g \in G''}\, achieve_g) \,|\, achieve_{\varrho}$.

Let $L = G \setminus (G'' \cup \{\varrho\})$. Clearly, $L \subseteq \mathbf{D}_f$ since the entire G is already in $\mathbf{D}_f$. Therefore, by Lemmas 5 and 2, starting with $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_f \vdash (\|_{g \in G''}\, achieve_g) \,|\, achieve_{\varrho}$, one can derive $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_f \vdash (\|_{g \in G''}\, achieve_g) \,|\, achieve_{\varrho} \,|\, (\|_{g \in L}\, achieve_g)$, which is simply $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_f \vdash (\|_{g \in G}\, achieve_g)$. Since $G \subseteq \mathbf{D}_f$, the inference rule (I2) also gives us $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_f \vdash (\|_{g \in G}\, achieve_g) \otimes (\wedge_{g \in G}\, g)$. Finally, $\mathbb{P}(\Pi)$ differs from $\mathbb{P}(\Pi')$ only in the form of the rule (P6). In case of $\mathbb{P}(\Pi)$, that rule has exactly the form that lets one apply the inference rule (I1) to get $\mathbb{P}(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_f \vdash achieve_G$. □

Theorem 2 (Completeness of $\mathcal{TR}$ planning)**.** *If there is a plan that achieves the goal G from the initial state $\mathbf{D}_0$ then the $\mathcal{TR}$-based STRIPS planner will find a plan.*

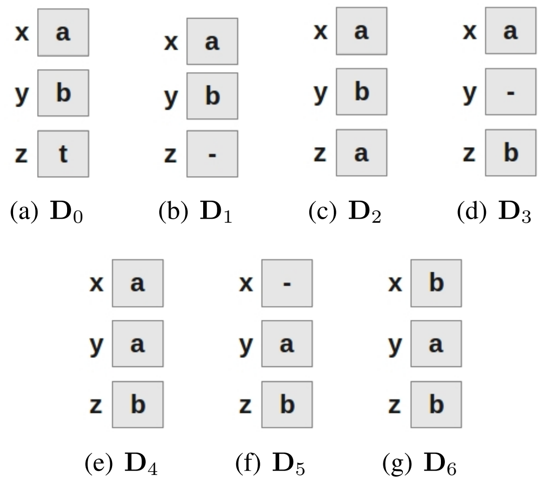
Proof. By induction on the length of the plan. Lemma 6 proves the base case of the induction and Lemma 7 establishes the inductive step. □

Theorem 2 establishes the completeness of the planner that is comprised of the $\mathcal{TR}$ proof theory and the rules that express the original *STRIPS* strategy.

Recall that the *classical STRIPS* planner described in [32, 73] was incomplete. The next example illustrates the reason for this incompleteness and contrasts the situation to the $\mathcal{TR}$-based planner.

Example 4 (Register exchange, continued)**.** *Consider the register exchange problem of Example 3. The original STRIPS planner fails to find a plan if, in the initial state, the auxiliary register z has the value t distinct from a and b [73]. We will now illustrate how the $\mathcal{TR}$ based planner deals with this case. Let $\mathbb{P}$ be the set of $\mathcal{TR}$ rules (3.3-3.6) that constitute the planner for the $\mathcal{TR}$-based planner for this problem. Given the planning goal $G = \{value(x, b), value(y, a)\}$ and the initial state $\mathbf{D}_0$, where $\{value(x, a), value(y, b)\} \subseteq \mathbf{D}_0$, we will show how the $\mathcal{TR}$ inference system constructs a derivation (and thus a plan) for the sequent $\mathbb{P}, \mathbf{D}_0 \cdots \mathbf{D}_n \vdash achieve_G$ for some $\mathbf{D}_n$ such that $\{value(x, b), value(y, a)\} \subseteq \mathbf{D}_n$.*

Consider the sequent $\mathbb{P}, \mathbf{D}_0 \cdots \vdash achieve_G$ that corresponds to the query (3.2). Applying the inference rule (I1) to that sequent using the rule (3.7), we get:



(a) $\mathbf{D}_0$ (b) $\mathbf{D}_1$ (c) $\mathbf{D}_2$ (d) $\mathbf{D}_3$

(e) $\mathbf{D}_4$ (f) $\mathbf{D}_5$ (g) $\mathbf{D}_6$

Figure 3.2: State changes by $\mathcal{TR}$ planning rules in Example 4.

$$\begin{aligned}\mathbb{P}, \mathbf{D}_0 \cdots \vdash \quad & (achieve_{value}(x,b) | achieve_{value}(y,a)) \\ & \otimes (value(x,b) \wedge value(y,a))\end{aligned}$$

Applying the inference rule (I1) twice to the resulting sequent using the rules (3.5–3.6) with appropriate substitutions result in:

$$\begin{aligned}\mathbb{P}, \mathbf{D}_0 \cdots \vdash \quad & (((achieve_{value}(z,b) | achieve_{value}(x,a)) \otimes copy(z,x,b)) \\ & \quad | achieve_{value}(y,a)) \\ & \otimes (value(x,b) \wedge value(y,a))\end{aligned}$$

Applying the inference rule (I1) once more and again using the rules (3.5–3.6) we get:

$$\begin{aligned}\mathbb{P}, \mathbf{D}_0 \cdots \vdash \quad & (\,(\,(\,(\,(achieve_{value}(y,b) \mid achieve_{value}(z,t)) \otimes copy(y,z,b)\,) \\ & \mid achieve_{value}(x,a)\,) \otimes copy(z,x,b)\,) \\ & \mid achieve_{value}(y,a)\,) \otimes (value(x,b) \wedge value(y,a))\end{aligned}$$

One more application of the inference rule (I1) but this time in conjunction with (3.4) yields:

$$\begin{aligned}\mathbb{P}, \mathbf{D}_0 \cdots \vdash \quad & (\,(\,(\,(\,(value(y,b) \mid achieve_{value}(z,t)) \otimes copy(y,z,b)\,) \\ & | achieve_{value}(x,a)\,) \otimes copy(z,x,b)\,) \\ & \mid achieve_{value}(y,a)\,) \otimes (value(x,b) \wedge value(y,a))\end{aligned}$$

Since $value(y,b) \in \mathbf{D}_0$, we can eliminate it by an application of the inference rule (I2). Then, another application of the inference rule (I1) using (3.4), followed by an application of the inference rule (I2), yields:

$$\begin{array}{rl}\mathbb{P}, \mathbf{D}_0 \cdots \vdash & (\,(\,(\,copy(y,z,b) \mid achieve_{value}(x,a)\,) \otimes copy(z,x,b)\,) \\ & \mid achieve_{value}(y,a)\,) \otimes (value(x,b) \wedge value(y,a))\end{array}$$

Then we can replace the first $copy$ using its definition (3.3) due to the inference rule (I1). Then, a couple of applications of the inference rule (I2) yields

$$\begin{array}{rl}\mathbb{P}, \mathbf{D}_0 \cdots \vdash & (\,(\,(\,(\,-value(z,t) \otimes +value(z,b)\,) \mid achieve_{value}(x,a)) \\ & \quad \otimes copy(z,x,b)\,) \mid achieve_{value}(y,a)\,) \\ & \otimes (value(x,b) \wedge value(y,a))\end{array}$$

Applying the inference rule (I3) twice to the primitive updates at the front first yields

$$\begin{array}{rl}\mathbb{P}, \mathbf{D}_1 \cdots \vdash & (\,(\,(\,+value(z,b) \mid achieve_{value}(x,a)) \otimes copy(z,x,b)\,) \\ & \mid achieve_{value}(y,a)\,) \otimes (value(x,b) \wedge value(y,a))\end{array}$$

and then

$$\begin{array}{rl}\mathbb{P}, \mathbf{D}_2 \cdots \vdash & (\,(\,achieve_{value}(x,a) \otimes copy(z,x,b)\,) \mid achieve_{value}(y,a)) \otimes \\ & (value(x,b) \wedge value(y,a))\end{array}$$

where $\mathbf{D}_1$ is $\mathbf{D}_0$ with $value(z,t)$ (where t denotes the old value of z) deleted and $\mathbf{D}_2$ is $\mathbf{D}_1$ with $value(z,b)$ added. Then, an application of inference rule (I1) using 3.4 followed by an application of inference rule (I2) yields

$$\begin{array}{rl}\mathbb{P}, \mathbf{D}_2 \cdots \vdash & (copy(z,x,b) | achieve_{value}(y,a)) \otimes \\ & (value(x,b) \wedge value(y,a))\end{array}$$

Now we can use the inference rule (I4) to explore the subgoal $achieve_{value}(y,a)$. Namely, we can expand this subgoal with the inference rule (I1) three times, first using (3.5–3.6) and then using (3.4), obtaining

$$\begin{array}{rl}\mathbb{P}, \mathbf{D}_2 \cdots \vdash & (\,copy(z,x,b) \\ & |(\,(\,value(x,a) \mid achieve_{value}(y,b)\,) \otimes copy(x,y,a)\,)\,) \\ & \otimes (value(x,b) \wedge value(y,a))\end{array}$$

Since $value(x,a)$ is true in $\mathbf{D}_2$, it can be removed. Then, an application of the inference rule (I1) using (3.4) can replace $achieve_{value}(y,b)$ with $value(y,b)$, which *can be eliminated by the inference rule (I2) because* $value(y, b)$ *is also true in* $\mathbf{D}_2$*. Then we get*

$$\begin{array}{rl} \mathbb{P}, \mathbf{D}_2 \cdots \vdash & (\ copy(z, x, b) \mid copy(x, y, a)\ ) \\ & \otimes (value(x, b) \wedge value(y, a)) \end{array}$$

Finally, the two $copy$*'s can be replaced by their definition (3.3) and then the remaining* $+value(...)$ *and* $-value(...)$ *can be executed using the inference rule (I3). This will advance the database (via three intermediate states) to state* $\mathbf{D}_6$ *containing* $\{value(x, b), value(y, a), value(z, b)\}$ *in which both* $value(x, b)$ *and* $value(y, a)$ *are true. Therefore, the inference rule (I2) can be used to derive the* $\mathcal{TR}$ *axiom* $\mathbb{P}, \mathbf{D}_6 \cdots \vdash ()$*, thus concluding the proof. The pivoting sequence of actions in this proof is* $\langle copy(y, z, b), copy(x, y, a), copy(z, x, b)\rangle$*, which constitutes the desired plan.* $\square$

3.2 The *fSTRIPS* Planner

In this section, we introduce *fSTRIPS* — a modification of the previously introduced *STRIPS* transform, which represents to a new planning strategy, which we call *fast STRIPS*. We show that although the new strategy explores a smaller search space, it is still sound and complete. Chapter 7 shows that *fSTRIPS* can be orders of magnitude faster than *STRIPS*.

Definition 11 ($\mathcal{TR}$ planning rules for *fSTRIPS*)**.** *Let* $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{S}\rangle$ *be a STRIPS planning problem as in Definition 4 and* $\mathbb{P}(\Pi)$ *is as in Definition 10. We define* $\mathbb{P}^f(\Pi)$ *to be exactly as* $\mathbb{P}(\Pi)$ *except for the* $\mathbb{P}_{enforced}$ *part. For* $\mathbb{P}^f(\Pi)$*, we redefine* $\mathbb{P}^f_{enforced}$ *(the replacement of* $\mathbb{P}_{enforced}$*) as follows:*

For each action $\alpha = \langle p_\alpha(\overline{X}), Pre_\alpha, E_\alpha\rangle$ *in* $\mathbb{A}$ *and each* $e(\overline{Y}) \in E_\alpha$*,* $\mathbb{P}^f_{enforced}$ *has the following rule:*

$$achieve_e(\overline{Y}) \leftarrow \neg e(\overline{Y}) \otimes execute_{p_\alpha}(\overline{X}). \qquad \text{(P7)}$$

This rule says that an action, α*, should be attempted only if it helps to achieve the currently pursued, unsatisfied goal.* $\square$

The other key aspect of *fSTRIPS* is that it uses a modified (general, unrelated to planning) proof theory for $\mathcal{TR}$, which relies on *tabling*, a technique analogous to [86]. This theory was introduced in [33] and was shown to be sound and complete. Here we use it for two reasons. First, it terminates if the number of base fluents is finite. Second, it has the property that it will not attempt to construct plans that have extraneous loops and thus will not attempt to large and unnecessary parts of the search space.

To construct a plan, as before, we can extract a pivoting sequence of actions with respect to *fSTRIPS* and show that the new pivoting sequence of actions is still a solution plan.

Similarly to Section 3.1, we assume till the end of this section that $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{D}_0 \rangle$ is a *STRIPS* planning problem, that $\mathbb{P}(\Pi)$ is the set of planning rules in Definition 10, and that $\mathbb{P}_f(\Pi)$ is the set of planning rules as specified in Definition 11.

Theorem 3 (Soundness of *fSTRIPS*)**.** *Any pivoting sequence of actions in the derivation of* $\mathbb{P}^f(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \vdash achieve_G$ *is a solution plan.*

Proof. Since Lemmas 3 and 4 hold for *fSTRIPS*, the proof of this theorem is almost identical to that of Theorem 1. □

Theorem 4 (Completeness of *fSTRIPS*)**.** *If there is a non-redundant plan that achieves the goal G from the initial state* $\mathbf{D}_0$ *then the* $\mathcal{TR}$*-based fSTRIPS planner will find that plan.*

Proof. In order to prove the completeness of *fSTRIPS*, we need to show that Lemmas 6 and 7 are held if $\mathbb{P}(\Pi)$ is replaced with $\mathbb{P}^f(\Pi)$. Since both of those lemmas are using Lemma 5 in their proofs, we also need to show that we can replace $\mathbb{P}(\Pi)$ with $\mathbb{P}^f(\Pi)$ in Lemma 5 as well. The proof of Lemma 5 carries over almost without change for the new situation, since the only difference is that the rules of the forms $achieve_e(\overline{Y}) \leftarrow execute_{p_\alpha}(\overline{X})$ and $execute_{p_\alpha}(\overline{X}) \leftarrow (\|_{\ell \in Pre_\alpha} achieve_\ell) \otimes \odot p_\alpha(\overline{X})$ are replaced with the rules of the form $achieve_e(\overline{Y}) \leftarrow execute_{p_\alpha}(\overline{X}, e(\overline{Y}))$ and $execute_{p_\alpha}(\overline{X}, e(\overline{Y})) \leftarrow (\|_{\ell \in Pre_\alpha} achieve_\ell) \otimes \neg e(\overline{Y}) \otimes \odot p_\alpha(\overline{X})$, respectively. Replacing $\mathbb{P}(\Pi)$ with $\mathbb{P}^f(\Pi)$ does not require any changes in the proof of Lemma 6.

To reestablish Lemma 7 with $\mathbb{P}^f(\Pi)$, in addition to the substitutions of the rules of the forms $achieve_e(\overline{Y}) \leftarrow execute_{p_\alpha}(\overline{X})$ and $execute_{p_\alpha}(\overline{X}) \leftarrow (\|_{\ell \in Pre_\alpha} achieve_\ell) \otimes \odot p_\alpha(\overline{X})$ with their correspondent rules in $\mathbb{P}^f(\Pi)$, we need another simple change. Consider the part in the proof where concatenating the derivations for $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \vdash (\|_{g \in G'} achieve_g)$ and $\mathbb{P}(\Pi'), \mathbf{D}_{f'} \ldots \mathbf{D}_f \vdash p_{\alpha_{n+1}}$ results in a $\mathcal{TR}$-derivation for $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \ldots \mathbf{D}_f \vdash (\|_{g \in G'} achieve_g) \otimes \odot p_{\alpha_{n+1}}$. Further in that proof, we show that there must be an extensional literal ϱ (or ϱ' in case ϱ is intensional) such that $\varrho \in G \setminus \mathbf{D}_{f'}$. Since $\varrho \notin \mathbf{D}_{f'}$, applying the inference rule (I2) using $\neg\varrho$ derives $\mathbb{P}(\Pi'), \mathbf{D}_{f'} \vdash \neg\varrho$. Therefore, concatenating the derivations for $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \vdash (\|_{g \in G'} achieve_g)$, $\mathbb{P}(\Pi'), \mathbf{D}_{f'} \vdash \neg\varrho$, and

$\mathbb{P}(\Pi'), \mathbf{D}_{f'} \ldots \mathbf{D}_f \vdash p_{\alpha_{n+1}}$ yields a $\mathcal{TR}$-derivation for $\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \ldots \mathbf{D}_f \vdash (\|_{g \in G'}\, achieve_g) \otimes \neg \varrho \otimes \odot p_{\alpha_{n+1}}$. The rest of the proof is almost identical to that of Section 3.1.

Since the proof of theorem 2 is just an inductive argument based on Lemmas 6 and 7, it carries through *fSTRIPS* as well. $\square$

Theorem 5 (*fSTRIPS* finds no more plans than *STRIPS*)**.** *Any plan found by the fSTRIPS planner will also be found by the STRIPS planner.*

Proof. Suppose $seq'_0, \ldots, seq'_m$ is a deduction of $achieve_G$ by the $\mathcal{TR}$ inference system using $\mathbb{P}^f(\Pi)$. By Definition 11, the sequent seq'_i produced by an application of the inference rule (I1) using a rule of the form $execute_{p_\alpha}(\overline{X}, e(\overline{Y})) \leftarrow \neg e(\overline{Y}) \otimes \odot p_\alpha(\overline{X})$ will have the form $\mathbb{P}^f(\Pi), \mathbf{D} \cdots \vdash (\exists)(left \mid (\neg e(\overline{Y}) \otimes \odot p_\alpha(\overline{X})) \mid right)$. One can show that $\mathbf{D} \models (\exists)(\neg e(\overline{Y}))$, so we can derive the sequent seq'_{i+1} of the form $\mathbb{P}^f(\Pi), \mathbf{D} \cdots \vdash (\exists)(left \mid \odot p_\alpha(\overline{X}) \mid right)$. Therefore, by eliminating the sequent seq'_i and those like it from the proof $seq'_0, \ldots, seq'_m$ we will get a proof of $achieve_G$ using $\mathbb{P}(\Pi)$. $\square$

In other words, the *STRIPS* strategy may generate more plans than *fSTRIPS*. The plans that are not generated by *fSTRIPS* are those that contain actions whose effects were not immediately required at the time of the action selection. This has the effect of ignoring longer plans when shorter plans are already found. The upshot of all this is that *STRIPS* has a larger search space to explore, and this explains the inferior performance of *STRIPS* compared to *fSTRIPS*, as the experiments in the next section show.

Chapter 4

Planning With Regression Analysis

In this chapter, first we give a brief introduction of the concept of regression analysis in a *STRIPS* planning problem. Second, we extend the $\mathcal{TR}$-based *STRIPS* planner in Section 3.1 with regression analysis.

4.1 Regression Analysis

In this section, first we give a formal definition of the regression of literals through *STRIPS* actions. Then, we show how one can compute the regression of a literal through an action.

Definition 12 (Regression of a *STRIPS* action)**.** *Consider a STRIPS action $\alpha = \langle p(\overline{X}), Pre, E\rangle$ and a consistent set of fluents L. The* ***regression of** L **through*** *α, denoted $\mathfrak{R}(\alpha, L)$, is a set of actions such that, for every $\beta \in \mathfrak{R}(\alpha, L)$ of the form $\beta = \langle p(\overline{X}), Pre_\beta, E\rangle$, where $Pre_\beta \supseteq Pre$ is a minimal (with respect to $\subsetneqq$) set of fluents satisfying the following condition: For every state $\mathbf{S}$ and substitution θ such that $\theta(\alpha)(\mathbf{S})$ exists, if $\mathbf{S} \models \theta(Pre_\beta) \wedge \theta(L)$, then $\theta(\alpha)(\mathbf{S}) \models \theta(L)$ (So $\theta(\alpha)(\mathbf{S}) = \theta(\beta)(\mathbf{S})$). In other words, β has the same effects as α, its precondition may be more restrictive, and performing β preserves the set of literals L.*
Each action in $\mathfrak{R}(\alpha, L)$ will also be called a regression of L via α. □

Next, we will show how regression can be computed. In the following we use an *identity* operator, $=$, which will be treated as an immutable extensional predicate, i.e., a predicate defined by a non-changeable set of facts: For any state S and a pair of constants or ground fluents ℓ and ℓ', $(\ell = \ell') \in$ S if and only if ℓ

and ℓ' are identical. Similarly, *nonidentity* is defined as follows: $\ell \neq \ell' \in \mathbf{S}$ if and only if ℓ, ℓ' are distinct.

The next example illustrates Definition 12. Consider a *STRIPS* action $copy = \langle copy(Src, Dest, V), \{value(Src, V), value(Dest, V')\}, \{\neg value(Dest, V'), value(Dest, V)\}\rangle$ from Example 3 in Section 3.1. Clearly, for every state $\mathbf{S}$ and substitution θ such that $\theta(copy)(\mathbf{S})$ exists, if $\mathbf{S} \models \theta(value(Src, V))$, then $\theta(\alpha)(\mathbf{S}) \models \theta(value(Src, V))$. Therefore, $copy \in \mathfrak{R}(copy, value(Src, V))$. This example is a special case of the following property of regression, which directly follows from the definitions: if ℓ is an extensional literal and $\alpha = \langle p(\overline{X}), Pre, E\rangle$ is a *STRIPS* action,

- $\mathfrak{R}(\alpha, \ell) = \emptyset$ if and only if, for every ground substitution, $\neg\theta(\ell) \in \theta(E)$.
- $\mathfrak{R}(\alpha, \ell) = \{\alpha\}$ if and only if, for every ground substitution $\neg\theta(\ell) \notin \theta(E)$.

The following proposition and lemmas present a method to compute regression. The method is complete for extensional literals; for intentional literals, it may find some, but possibly not all, regressions.

Proposition 1 (Regression of sets of literals)**.** *Given a set of literals $L = L_1 \cup L_2$ and a STRIPS action $\alpha = \langle p(\overline{X}), Pre_\alpha, E_\alpha\rangle$, let $\beta_1 \in \mathfrak{R}(\alpha, L_1)$ and $\beta_2 \in \mathfrak{R}(\alpha, L_2)$, where $\beta_1 = \langle p(\overline{X}), Pre_{\beta_1}, E_\alpha\rangle$ and $\beta_2 = \langle p(\overline{X}), Pre_{\beta_2}, E_\alpha\rangle$. Then there is some $\beta = \langle p(\overline{X}), Pre_\beta, E_\alpha\rangle \in \mathfrak{R}(\alpha, L)$ such that $Pre_\beta \subseteq Pre_{\beta_1} \cup Pre_{\beta_2}$.*

Proof. From the assumptions, it follows that for every state $\mathbf{S}$ and substitution θ such that $\theta(\alpha)(\mathbf{S})$ exists, if $\mathbf{S} \models \theta(Pre_{\beta_1} \cup Pre_{\beta_2}) \wedge \theta(L)$, then $\theta(\alpha)(\mathbf{S}) \models \theta(L)$.

To find a minimal subset of $Pre_{\beta_1} \cup Pre_{\beta_2}$ satisfying the regression property, one can repeatedly remove elements from $Pre_{\beta_1} \cup Pre_{\beta_2}$ and check if the regression property still holds. When no removable elements remain, we get a desired set Pre_β. □

Lemma 8 (Regression of extensional literals)**.** *Consider an extensional literal ℓ and a STRIPS action $\alpha = \langle p(\overline{X}), Pre, E\rangle$ where α and ℓ do not share variables. Let $Pre_\beta = Pre \cup \{\, \ell \neq e \mid \neg e \in E \;\wedge\; \exists\theta \text{ s.t. } \theta(e) = \theta(\ell) \,\}$.
Then $\beta \in \mathfrak{R}(\alpha, \ell)$, where $\beta = \langle p(\overline{X}), Pre_\beta, E\rangle$.*

Proof. Let $\mathbf{S}$ be a state and a substitution θ is such that $\theta(\alpha)(\mathbf{S})$ exists. Clearly, if $\mathbf{S} \models \theta(Pre_\beta)$, there is no $\neg e \in E$ such that $\theta(e) = \theta(\ell)$. Therefore, if $\mathbf{S} \models \theta(Pre_\beta) \wedge \theta(\ell)$, then $\theta(\alpha)(\mathbf{S}) \models \ell$.

We need to show that Pre_β is a minimal set of literals satisfying the above property. Assume, to the contrary, that there is some $Pre_{\beta'}$, $Pre \subseteq Pre_{\beta'} \subsetneq$

Pre_β, such that for every state $\mathbf{S}$ and substitution θ, if $\theta(\alpha)(\mathbf{S})$ exists and $\mathbf{S} \models \theta(Pre_{\beta'}) \wedge \theta(\ell)$, then $\theta(\alpha)(\mathbf{S}) \models \theta(\ell)$. Since $Pre_{\beta'} \subset Pre_\beta$, there must be $(\ell \neq e) \in Pre_\beta \setminus Pre_{\beta'}$. Let θ_1 be a substitution such that $\theta_1(e) = \theta_1(\ell)$. In that case, for every $\mathbf{S}$ such that $\theta_1(\alpha)(\mathbf{S})$ exists, we have that $\mathbf{S} \models \theta_1(Pre_{\beta'}) \wedge \theta_1(\ell)$ but $\theta_1(\alpha)(\mathbf{S}) \not\models \theta_1(\ell)$, since $\theta_1(\neg\ell) = \theta_1(\neg e) \in \theta_1(E)$. This contradicts the assumption about $Pre_{\beta'}$. Thus, $Pre_{\beta'}$ does not exist and Pre_β is a minimal set of fluents satisfying the regression condition for ℓ, so $\beta \in \mathfrak{R}(\alpha, \ell)$. $\square$

To illustrate the lemma, consider a ground extensional literal $value(r, c)$ and the ***STRIPS*** action $copy = \langle copy(S, D, V), Pre_{copy}, E_{cppy}\rangle$, where $Pre_{copy} = \{value(S, V), value(D, V')\}$ and $E_{copy} = \{\neg value(D, V'), value(D, V)\}$. Then $\beta \in \mathfrak{R}(copy, value(r, c))$, where $\beta = \langle copy(S, D, V), Pre_\beta, E_{copy}\rangle$ and $Pre_\beta = Pre_{copy} \cup \{value(r, c) \neq value(D, V')\}$.

Lemma 9 (Regression of intensional literals)**.** *Consider a set of rules $\mathbb{R}$, an intensional literal ℓ, and a STRIPS action $\alpha = \langle p(\overline{X}), Pre, E\rangle$, where α and ℓ do not share variables. Let L be a minimal set of extensional literals such that $\mathbb{R} \cup L \cup \{\leftarrow \ell\}$ has an SLD-refutation [67]. Then for every $\beta \in \mathfrak{R}(\alpha, L)$ of the form $\beta = \langle p(\overline{X}), Pre_\beta, E\rangle$, there is $L_\beta \subseteq Pre_\beta \cup L$ such that $\langle p(\overline{X}), L_\beta, E\rangle \in \mathfrak{R}(\alpha, \ell)$.*

Proof. By Definition 12, for every state $\mathbf{S}$ and substitution θ such that $\theta(\alpha)(\mathbf{S})$ exists, if $\mathbf{S} \models \theta(Pre_\beta) \wedge \theta(L)$, then $\theta(\alpha)(\mathbf{S}) \models \theta(L)$. Due to the soundness of SLD-refutation [67], if $\mathbf{S} \models \theta(Pre_\beta) \wedge \theta(L)$ then $\mathbf{S} \models \theta(\ell)$; and if $\theta(\alpha)(\mathbf{S}) \models \theta(L)$ then $\theta(\alpha)(\mathbf{S}) \models \theta(\ell)$. Therefore, for every state $\mathbf{S}$ and substitution θ such that $\theta(\alpha)(\mathbf{S})$ exists, if $\mathbf{S} \models \theta(Pre_\beta) \wedge \theta(L) \wedge \theta(\ell)$, then $\theta(\alpha)(\mathbf{S}) \models \theta(\ell)$. Therefore, $Pre_\beta \cup L$ satisfies the conditions for regressing ℓ through α except, possibly, minimality. To get the minimality, we can start removing elements from this set, as in Proposition 8, until a minimal set is reached. $\square$

Definition 13 (Perfect regression)**.** *Given a set of literals L, a STRIPS action α, and a substitution θ, $\beta \in \mathfrak{R}(\alpha, L)$ is called a* **perfect regression of L through α with respect to θ** *if, for every state $\mathbf{S}$ such that $\theta(\alpha)(\mathbf{S})$ exists, $\theta(\beta)$ is also executable at $\mathbf{S}$.* $\square$

In Definition 13, β is simply called a ***perfect regression of*** L ***through*** α if α and L are ground. Next lemma shows that, given a ground action α and a set of literal L, Proposition 1 and Lemmas 8 and 9 are always able to find the perfect regression of L through α.

Lemma 10 (Perfect regression of ground literals)**.** *Given a ground set of literals L and a ground STRIPS action $\alpha = \langle p_\alpha, Pre, E\rangle$, let $\mathbf{S}$ be a state such that $Pre \subseteq \mathbf{S}$ and $\alpha(\mathbf{S}) \models L$. Then, there exists $\beta = \langle p_\beta, Pre_\beta, E\rangle$, a perfect regression of L through α such that $Pre_\beta \setminus Pre$ is a set of literals of the form $\ell = e$ or $\ell \neq e$.*

Proof. By Proposition 1 and Lemmas 8 and 9, one can find $\beta = \langle p_\beta, Pre_\beta, E\rangle \in \mathfrak{R}(\alpha, L)$ such that $Pre_\beta \setminus Pre$ is a set of literals of the form $\ell = e$ or $\ell \neq e$. We need to show that at least one of those regression actions exists and it is executable at every state.

To show that such β exists, we will show how Lemmas 8 and 9 construct such action. Since $Pre \in \mathbf{S}$, to show that β is also executable at every state, we need to show that identity and nonidentity literals in $Pre_\beta \setminus Pre$ are always true. Due to Lemmas 8 and 9, $Pre_\beta \setminus Pre$ is formed based on L and α as follows:

For every $\ell \in L$, we have two cases:

1. *ℓ is extensional.* By Lemma 8, $Pre_\beta \setminus Pre$ contains a set of literals of the form $\ell \neq e$, where $\neg e \in E$. By construction of such ground nonidentity literal, it is independent of $\mathbf{S}$ and it only depends on ground literals ℓ and e. Then, one can show that such literal is true in every state. If not, $\ell \notin \alpha(\mathbf{S})$, a contrary to the original assumption that $\alpha(\mathbf{S}) \models \ell$.

2. *ℓ is intensional.* By completeness of SLD-refutation [67], since $\alpha(\mathbf{S}) \models \ell$, there must be a minimal set of extensional literals $L_M \subseteq \alpha(\mathbf{S})$ such that $\mathbb{R} \cup L_M \cup \{\leftarrow \ell\}$ has an SLD-refutation [67]. Then, by Lemmas 8 and 9, one can show that $Pre_\beta \setminus Pre$ contains a set of literals of the form $\ell' \neq e$, where $\ell' \in L_M$ and $\neg e \in E$. By construction of such ground nonidentity literals, they are independent of $\mathbf{S}$, i.e. they depend on L_M and E. Clearly, such literals are true. If not, $L_M \nsubseteq \alpha(\mathbf{S})$, a contrary to the original assumption that there was a SLD-refutation for $\mathbb{R} \cup L_M \cup \{\leftarrow \ell\}$ in $\mathbf{S}$.

Since all of the literals in $Pre_\beta \setminus Pre$ are independent of $\mathbf{S}$ and they are always true, for every state $\mathbf{S}'$, β is executable at $\mathbf{S}'$ if $Pre \subseteq \mathbf{S}'$. Thus β is a perfect regression of L through α. □

From now on, we assume that the set of actions $\mathbb{A}$ is regression deterministic and closed under regression.

4.2 The *STRIPS*^{R} Planner

Regression analysis of literals has been shown to be a potential enhancement for planning strategies [2]. A planning strategy can use such analysis to protect goal literals that are already achieved during its execution such that further actions taken by the planning strategy cannot remove those literals [73][45]. This section shows how we use $\mathcal{TR}$ to lay the ground work of regression analysis for *STRIPS* planning strategy.

In this section, to track achieved goal literals, we introduce a built-in binary predicate of the form $lock(\ell, n)$, where ℓ is a literal, such that for any planning problem $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{D}_0 \rangle$ and every ground literal ℓ, we have $lock(\ell, 0) \in \mathbf{D}_0$ but $lock(\ell, k) \notin \mathbf{D}_0$ if $k > 0$. The counter in $lock(\ell, n)$ indicates the number of concurrent transactions that consider ℓ as an already achieved goal. In other words, for every state $\mathbf{S}$ and literal ℓ, if $lock(\ell, n) \in \mathbf{S}$ and $n > 0$, then ℓ is assumed to be an already achieved goal in $\mathbf{S}$.

In what follows, we will be using the unary $protect$ and $unprotect$ transactions, defined by (4.1) and (4.2), to protect and unprotect planning goals. The first transaction gets a literal and if it is achieved, increases its associated counter to be protected from "unachieving"; the second transaction decreases the counter when it no longer needs the protection by the current transaction.

$$\begin{aligned} protect(p(\overline{X})) \leftarrow\ & \odot(\ p(\overline{X}) \otimes lock(p(\overline{X}), N) \otimes \\ & - lock(p(\overline{X}), N) \otimes + lock(p(\overline{X}), N+1)\ ). \end{aligned} \tag{4.1}$$

$$\begin{aligned} unprotect(p(\overline{X})) \leftarrow\ & \odot(\ lock(p(\overline{X}), N) \otimes - lock(p(\overline{X}), N) \otimes \\ & + lock(p(\overline{X}), N-1)\ ). \end{aligned} \tag{4.2}$$

Clearly, both of these transactions will always succeed. We also assume $regress$ to be a built-in binary predicate such that, for every state $\mathbf{S}$ and a set of literals $L = \{\ell \mid lock(\ell, k) \in \mathbf{S},\ k > 0\}$, if there exists a perfect regression of L through α, then $regress(p_\alpha(\overline{X}), p_{\alpha'}(\overline{X})) \in \mathbf{S}$, for some perfect regression $\alpha' \in \mathfrak{R}(\alpha, L)$.

Definition 14 ($\mathcal{TR}$ planning rules for *STRIPS*^{R})**.** *Let* $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{S} \rangle$ *be a STRIPS planning problem (see Definition 4). Similar to* $\mathbb{P}(\Pi)$ *(see Definition 10), we define* $\mathbb{P}^r(\Pi)$ *to be a set of* $\mathcal{TR}$ *rules, which will be shown to provide a sound and complete solution to the STRIPS planning problem.* $\mathbb{P}^r(\Pi)$ *has three disjoint parts,* $\mathbb{P}^r_{\mathbb{R}}$*,* $\mathbb{P}^r_{\mathbb{A}}$*, and* $\mathbb{P}^r_G$*, as follows:*

- *The $\mathbb{P}^r_{\mathbb{R}}$ part: for each rule $p(\overline{X}) \leftarrow p_1(\overline{X}_1) \wedge \cdots \wedge p_n(\overline{X}_n)$ in $\mathbb{R}$, $\mathbb{P}^r_{\mathbb{R}}$ has a rule of the form*

$$\begin{array}{rl} achieve^r_p(\overline{X}) \leftarrow & (\|^n_{i=1} achieve^r_{p_i}(\overline{X}_i)) \otimes \\ & \odot(\ \otimes^n_{i=1}\ unprotect(p_i(\overline{X}_i)) \otimes protect(p(\overline{X}))\ ). \end{array} \tag{P8}$$

  This rule is virtually identical to Rule (P1) in Definition 10.

- *The part $\mathbb{P}^r_{\mathbb{A}} = \mathbb{P}^r_{actions} \cup \mathbb{P}^r_{atoms} \cup \mathbb{P}^r_{achieves}$ has three subparts, as follows:*

  - $\mathbb{P}^r_{actions}$*: for each $\alpha \in \mathbb{A}$, $\mathbb{P}^r_{actions}$ has a rule of the form*

$$p_\alpha(\overline{X}) \leftarrow (\wedge_{\ell \in Pre_\alpha} \ell) \otimes (\otimes_{u \in Enf(E_\alpha)} u). \tag{P9}$$

    This rule is identical to Rule (P2) in Definition 10.

  - $\mathbb{P}^r_{atoms} = \mathbb{P}^r_{achieved} \cup \mathbb{P}^r_{constraints} \cup \mathbb{P}^r_{enforced}$ *has three disjoint subparts as follows:*

    - $\mathbb{P}^r_{achieved}$*: for each extensional predicate $p \in \mathcal{P}_{ext}$, $\mathbb{P}^r_{achieved}$ has the rules*

$$\begin{array}{rcl} achieve^r_p(\overline{X}) & \leftarrow & protect(p(\overline{X})). \\ achieve^r_{\neg p}(\overline{X}) & \leftarrow & protect(\neg p(\overline{X})). \end{array} \tag{P10}$$

      These rules are similar to the rules in (P3), but we add protection to $p(\overline{X})$ (resp., $\neg p(\overline{X})$), which are the goals considered to be achieved. Note that $protect(p(\overline{X}))$ also checks if $p(\overline{X})$ is true in the current state.

    - $\mathbb{P}^r_{constraints}$*: for each identity or nonidentity literal ℓ_c, i.e. a literal of the form $\ell = \ell'$ or $\ell \neq \ell'$, $\mathbb{P}^r_{constraints}$ has a rule*

$$achieve^r_{\ell_c} \leftarrow \ell_c. \tag{P11}$$

      This rule say that if an identity or nonidentity literal is true in a state then that literal has already been achieved as a goal.

    - $\mathbb{P}^r_{enforced}$*: for each action $\alpha = \langle p_\alpha(\overline{X}), Pre_\alpha, E_\alpha \rangle$ in $\mathbb{A}$ and each $e(\overline{Y}) \in E_\alpha$, $\mathbb{P}^r_{enforced}$ has the following rule:*

$$achieve^r_e(\overline{Y}) \leftarrow execute^r_{p_\alpha}(\overline{X}, e(\overline{Y})). \tag{P12}$$

      This rule is similar to (P4).

– $\mathbb{P}^{r}_{achieves}$: *for each action* $\alpha = \langle p_\alpha(\overline{X}), Pre_\alpha, E_\alpha\rangle$ *in* $\mathbb{A}$ *and each* $e(\overline{Y}) \in E_\alpha$, $\mathbb{P}^{r}_{achieves}$ *has the following rule:*

$$\begin{array}{rl} execute^{r}_{p_\alpha}(\overline{X}, e(\overline{Y})) \leftarrow & (\|_{\ell \in Pre_\alpha} achieve^{r}_{\ell}) \otimes \\ & \odot(\ unprotect(Pre_\alpha)\ \otimes \\ & \quad regress(p_\alpha(\overline{X}), p_{\alpha'}(\overline{X}))\ \otimes \\ & \quad p_{\alpha'}(\overline{X})\ \otimes \\ & \quad protect(e(\overline{Y}))\ ). \end{array} \tag{P13}$$

The above rule means that to execute an action, one must first achieve the precondition of the action, then remove the precondition of the action from the list of already achieved protected goals. Moreover, before taking the action α *to achieve* $e(\overline{Y})$*, it chooses* α' *that is a regression of already achieved goals through* α*. Then, it protects the newly achieved goal in the set of effects of* α*, and finally perform the state changes prescribed by the action.*

- $\mathbb{P}^{r}_{G}$: *Let* $G = \{g_1, ..., g_k\}$*. Then* $\mathbb{P}^{r}_{G}$ *has a rule of the form:*

$$achieve^{r}_{G} \leftarrow \|_{i=1}^{k} achieve^{r}_{g_i} \otimes (\wedge_{i=1}^{k} g_i). \tag{P14}$$

□

Regression analysis enhances the $\mathcal{TR}$ planning rules of Definition 10 with a search mechanism that intuitively makes more sense: it avoids seemingly useless work whereby planning goals might be achieved in the interim only to be unachieved later. We will again use the register exchange example of Section 3.1 to illustrate how $\mathcal{TR}$-based planning with regression works.

Example 5 (Planning rules of *STRIPS*^{R} for register exchange)**.** *Recall the register exchange problem of Example 3. Per Definition 14, the planning rules with regression analysis for this problem are as follows.*

As explained in Lemma 8, given a non-ground extensional literal $value(Reg, V'')$*,* $\beta \in \mathfrak{R}(\alpha, value(Reg, V''))$*, a regression of* $value(Reg, V'')$ *through the STRIPS action* $\alpha = copy(Src, Dest, V)$*, can be computed as follows:*

$$\begin{array}{rl} \beta = \langle & copy(Src, Dest, V), \\ & \{value(Src, V), value(Dest, V'), value(Reg, V''), \\ & (value(Reg, V'') \neq value(Dest, V'))\}, \\ & \{\neg value(Dest, V'), value(Dest, V)\}\rangle \end{array} \tag{4.3}$$

Due to case (P9):

$$\begin{array}{rcl} copy(Src, Dest, V) & \leftarrow & value(Src, V) \otimes value(Dest, V') \otimes \\ & & -value(Dest, V') \otimes +value(Dest, V). \\ copy_\beta(Src, Dest, V) & \leftarrow & value(Src, V) \otimes \\ & & value(Dest, V') \otimes value(Reg, V'') \otimes \\ & & (value(r, c) \neq value(Dest, V')) \otimes \\ & & -value(Dest, V') \otimes +value(Dest, V). \end{array} \tag{4.4}$$

Due to (P10), (P12), and (P13):

$$\begin{array}{l} achieve^r_{value}(R, V) \leftarrow protect(value(R, V)). \\ achieve^r_{\neg value}(R, V) \leftarrow protect(\neg value(R, V)). \end{array} \tag{4.5}$$

$$\begin{array}{ll} achieve^r_{value} & (Dest, V) \leftarrow \\ & execute^r_{copy}(Src, Dest, V). \end{array} \tag{4.6}$$

Due to (P11):

$$\begin{array}{l} achieve^r_{\ell_{c_1}} \leftarrow value(R_1, V_1) = value(R_2, V_2). \\ achieve^r_{\ell_{c_2}} \leftarrow value(R_1, V_1) \neq value(R_2, V_2). \end{array} \tag{4.7}$$

where $\ell_{c_1} = value(r, c) = value(R, V)$ *and* $\ell_{c_2} = value(r, c) \neq value(R, V)$.
Due to (P13):

$$\begin{array}{ll} execute^r_{copy}(& Src, Dest, V) \leftarrow \\ & (\ achieve^r_{value}(Src, V) \mid achieve^r_{value}(Dect, V')\ ) \otimes \\ & \odot\ (\ unprotect(\{value(Src, V), value(Dest, V')\}) \otimes \\ & \quad regress(copy(Src, Dest, V), copy'(Src, Dest, V)) \otimes \\ & \quad copy'(Src, Dest, V) \otimes \\ & \quad protect(value(Dest, V))\ ). \end{array} \tag{4.8}$$

where $copy'(Src, Dest, V))$ *is the regression of* $copy(Src, Dest, V))$ *through the set of already achieved goals in the current state.*
Due to (P14):

$$\begin{array}{rcl} achieve^r_G & \leftarrow & (achieve^r_{value}(x, b) \mid achieve^r_{value}(y, a)) \\ & & \otimes\, (value(x, b)\ \wedge value(y, a)). \end{array} \tag{4.9}$$

Case (P8) of Definition 14 does not add any rule in this example because the planning problem does not involve intensional fluents. □

To construct a plan, as before, we can extract a pivoting sequence of actions with respect to *STRIPS*^{R} and show that the new pivoting sequence of actions is still a solution plan. It is easy to show that Lemma 3 is still valid under *STRIPS*^{R}. We will also show that Lemma 4 can be extended to accommodate *STRIPS*^{R}.

We assume till the end of this section $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{D}_0 \rangle$ will denote a *STRIPS* planning problem and $\mathbb{P}^r(\Pi)$ will be the set of planning rules defined in Definition 14. We also introduce notation to conveniently separate the *lock(...)*-facts in every state $\mathbf{D}$. Thus $\mathbf{D}^{lock}$ will stand for the set of all the *lock(...)*-facts in $\mathbf{D}$, while $\mathbf{D}^{\cancel{lock}}$ will denote the rest of the literals in $\mathbf{D}$.

Proposition 2 (Reduction of states in a $\mathcal{TR}$ derivation)**.** *If there is a derivation for $\mathbb{P}^r(\Pi), \mathbf{D}_i \ldots \mathbf{D}_j \vdash p_\alpha(\overline{X})$, then the following holds:*

1. $\mathbf{D}_i^{lock} = \mathbf{D}_j^{lock}$*; and*

2. *For every $\mathbf{D}_k$ and $\mathbf{D}_l$ such that $\mathbf{D}_k^{\cancel{lock}} = \mathbf{D}_i^{\cancel{lock}}$, $\mathbf{D}_l^{\cancel{lock}} = \mathbf{D}_j^{\cancel{lock}}$, and $\mathbf{D}_k^{lock} = \mathbf{D}_l^{lock}$, one can derive $\mathbb{P}^r(\Pi), \mathbf{D}_k \ldots \mathbf{D}_l \vdash p_\alpha(\overline{X})$.*

Proof. By Lemma 3, the derivation of $\mathbb{P}^r(\Pi), \mathbf{D}_i \ldots \mathbf{D}_j \vdash p_\alpha(\overline{X})$ shows that the execution of $p_\alpha(\overline{X})$ changes the state from $\mathbf{D}_i$ to $\mathbf{D}_j$. Since, for every $\alpha = \langle p_\alpha(\overline{X}), Pre_\alpha, E_\alpha \rangle \in \mathbb{A}$, Pre_α and E_α do not have $lock(\_,\_)$-literals, the execution of $p_\alpha(\overline{X})$ does not affect these literals. Therefore, $\mathbf{D}_i^{lock} = \mathbf{D}_j^{lock}$ and one can simply derive $\mathbb{P}^r(\Pi), \mathbf{D}_k^{\cancel{lock}} \ldots \mathbf{D}_l^{\cancel{lock}} \vdash p_\alpha(\overline{X})$ using the same derivation steps of $\mathbb{P}^r(\Pi), \mathbf{D}_i \ldots \mathbf{D}_j \vdash p_\alpha(\overline{X})$. Similarly, one can show that the same $\mathcal{TR}$ derivation holds even if a set of $lock(\_,\_)$-literals, is added to all of the states in the derivation of $\mathbb{P}^r(\Pi), \mathbf{D}_k^{\cancel{lock}} \ldots \mathbf{D}_l^{\cancel{lock}} \vdash p_\alpha(\overline{X})$. Hence $\mathbb{P}^r(\Pi), \mathbf{D}_k \ldots \mathbf{D}_l \vdash p_\alpha(\overline{X})$. □

Lemma 11 (Execution of transactions of the form p_α in *STRIPS*^{R})**.** *Let $seq_0, \ldots, seq_m$ be a derivation of $\mathbb{P}^r(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_m \vdash achieve_G$, and let $\alpha_1, \ldots, \alpha_n$ (resp., $\mathbf{D}_{i_1}, \ldots, \mathbf{D}_{i_n}$) be a pivoting sequence of actions (resp., states). Then for each r ($1 \le r \le n$) there are i_r, i_{r+1} such that $\mathbf{D}_0^{\cancel{lock}} = \mathbf{D}_{i_1}^{\cancel{lock}}$, $\mathbf{D}_{i_{n+1}}^{\cancel{lock}} = \mathbf{D}_m^{\cancel{lock}}$, and $\mathbb{P}(\Pi), \mathbf{D}_{i_r}^{\cancel{lock}} \ldots \mathbf{D}_{i_{r+1}}^{\cancel{lock}} \vdash p_{\alpha_r}$. (Recall that p_{α_r} is the transaction associated with α_r—see Definition 9.)*

Proof. Similar to Lemma 4, we divide the proof in three parts, namely, proving:

- $\mathbb{P}^r(\Pi), \mathbf{D}_0^{\cancel{lock}} \cdots \vdash p_{\alpha_1}$,
- $\mathbb{P}^r(\Pi), \mathbf{D}_{i_n}^{\cancel{lock}} \ldots \mathbf{D}_m^{\cancel{lock}} \vdash p_{\alpha_n}$, and

- $\mathbb{P}^r(\Pi), \mathbf{D}_{i_r}^{\cancel{lock}} \ldots \mathbf{D}_{i_{r+1}}^{\cancel{lock}} \vdash p_{\alpha_r}$, for each r $(1 \leq r < n)$.

Examination of the $\mathcal{TR}$ planning rules in Definition 14 shows that there are two groups of such rules that directly cause the execution of an elementary transitions (and thus change the underlying states): the rules of the form $tr(\alpha)$ and the rules defining $protect$ and $unprotect$ transactions in (4.1). Thus every elementary $\mathcal{TR}$ update in the derivation of $\mathbb{P}^r(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_m \vdash achieve_G$ is caused by either a $tr(\alpha)$, or one of the rules of the form $protect(\ell) \leftarrow \cdots$ or $unprotect(\ell) \leftarrow \cdots$ in (4.1) and (4.2). By Definition 14, the only $\mathcal{TR}$ planning rules that call the heads of those transactions are the rules of the form (P10) and (P13). Because of the isolation operators $\odot$ in (4.1), (4.2), and (P13), there can be no interleaving between the execution of p_αs, $protect$s, and $unprotect$s. Clearly, the rules of the form $protect(\ell) \leftarrow \cdots$ and $unprotect(\ell) \leftarrow \cdots$ can only cause $\mathcal{TR}$ elementary updates of the form $+\ lock(\ell, n)$ or $-\ lock(\ell, n)$. Therefore, for any $0 \leq j < m$, if $\mathbf{D}_j^{\cancel{lock}} \neq \mathbf{D}_{j+1}^{\cancel{lock}}$, the state change must have been caused by a rule $tr(\alpha)$, i.e., by execution of a pivoting action α.

By the definition of pivoting sequence of actions and states, we have $\mathbb{P}^r(\Pi), \mathbf{D}_{i_1} \cdots \vdash p_{\alpha_1}$ where $\mathbf{D}_{i_1}$ is the pivoting state of α_1. We need to show that $\mathbf{D}_{i_1}^{\cancel{lock}} = \mathbf{D}_0^{\cancel{lock}}$. Assume, to the contrary, that for some $0 \leq j < i_1$, $\mathbf{D}_0^{\cancel{lock}} = \mathbf{D}_j^{\cancel{lock}}$ and $\mathbf{D}_j^{\cancel{lock}} \neq \mathbf{D}_{i_1}^{\cancel{lock}}$. Then there must be at least one elementary $\mathcal{TR}$ update $e \notin E_{\alpha_1}$ that was executed by the inference rule (I3) at state $\mathbf{D}_j$. Since, as noted above, every elementary $\mathcal{TR}$ update corresponds to a pivoting action, there must have been some other pivoting action α' that was executed before α_1. This contradicts the assumption that α_1 is the first pivoting action. Thus, $\mathbf{D}_{i_1}^{\cancel{lock}} = \mathbf{D}_0^{\cancel{lock}}$ and, by Proposition 2, $\mathbb{P}^r(\Pi), \mathbf{D}_0^{\cancel{lock}} \cdots \vdash p_{\alpha_1}$.

Suppose $\mathbb{P}^r(\Pi), \mathbf{D}_{i_n} \ldots \mathbf{D}_{i_{n+1}} \vdash p_{\alpha_n}$. We need to show that $\mathbf{D}_{i_{n+1}}^{\cancel{lock}} = \mathbf{D}_m^{\cancel{lock}}$. Assume, to the contrary, that $\mathbf{D}_{i_{n+1}}^{\cancel{lock}} \neq \mathbf{D}_m^{\cancel{lock}}$. Let $i_{n+1} \leq j < m$ such that $\mathbf{D}_j^{\cancel{lock}} \neq \mathbf{D}_{i_{n+1}}^{\cancel{lock}}$ and $\mathbf{D}_j^{\cancel{lock}} = \mathbf{D}_m^{\cancel{lock}}$. Then, there must be some elementary $\mathcal{TR}$ update $e \notin E_{\alpha_n}$ that was executed by the inference rule (I3) at state $\mathbf{D}_{i_{n+1}}$. As every elementary $\mathcal{TR}$ update is associated with a pivoting action, there must have been a pivoting action α' that was executed after α_n—contrary to the assumption that α_n is the last pivoting action. Thus $\mathbf{D}_{i_{n+1}}^{\cancel{lock}} = \mathbf{D}_m^{\cancel{lock}}$ and, by Proposition 2, $\mathbb{P}^r(\Pi), \mathbf{D}_{i_n}^{\cancel{lock}} \ldots \mathbf{D}_m^{\cancel{lock}} \vdash p_{\alpha_n}$.

By the definition of pivoting actions and states, we have $\mathbb{P}^r(\Pi), \mathbf{D}_{i_j} \cdots \vdash p_{\alpha_j}$ where $1 \leq j \leq n$, and by Proposition 2, we have $\mathbb{P}^r(\Pi), \mathbf{D}_{i_j}^{\cancel{lock}} \cdots \vdash p_{\alpha_j}$. We need to show that $\mathbb{P}(\Pi), \mathbf{D}_{i_j}^{\cancel{lock}} \ldots \mathbf{D}_{i_{j+1}}^{\cancel{lock}} \vdash p_{\alpha_j}$. Assume, to the contrary, that $\mathbb{P}(\Pi), \mathbf{D}_{i_j}^{\cancel{lock}} \ldots \mathbf{D}_k^{\cancel{lock}} \vdash p_{\alpha_j}$ and $\mathbf{D}_k^{\cancel{lock}} \neq \mathbf{D}_{i_{j+1}}^{\cancel{lock}}$. Due to the isolation operator in (P13), $p_{\alpha_{j+1}}$ cannot start before p_{α_j} has finished. Therefore, the state $\mathbf{D}_k^{\cancel{lock}}$ precedes $\mathbf{D}_{i_{j+1}}^{\cancel{lock}}$. Thus, there must have been an elementary $\mathcal{TR}$ update $e \notin E_{\alpha_j}$ that was executed by the inference rule (I3) at state $\mathbf{D}_k$, i.e., after α_j finished and before α_{j+1} started. Since every elementary $\mathcal{TR}$ update in that derivation corresponds to a pivoting action, this means that there must have been another pivoting action α' that was executed after α_j and before α_{j+1} — contrary to the assumption that α_{j+1} follows immediately after α_j. Thus $\mathbf{D}_k^{\cancel{lock}} = \mathbf{D}_{i_{j+1}}^{\cancel{lock}}$, which concludes the proof. □

Theorem 6 (Soundness of *STRIPS*^{R})**.** *Any pivoting sequence of actions in the derivation of* $\mathbb{P}^r(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_m \vdash achieve_G^r$ *is a solution plan.*

Proof. Given a derivation of $\mathbb{P}^r(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_m \vdash achieve_G^r$, let $\alpha_{i_1}, \ldots, \alpha_{i_n}$ be the pivoting sequence of actions for this derivation and $\mathbf{D}_{i_1}, \ldots, \mathbf{D}_{i_n}$ be its corresponding pivoting sequence of states. Let $\mathbf{D}_{i_{n+1}}$ be $\mathbf{D}_m$. By the construction of $achieve_G^r$, we know that $\mathbf{D}_m \models G$. Since G does not contain any literal of the form $lock(\_, \_)$, $\mathbf{D}_m^{\cancel{lock}} \models G$. Due to the soundness of the $\mathcal{TR}$ inference system, we also know that $\mathbb{P}^r(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_m \models achieve_G^r$. By Lemma 11 and by the soundness of the inference system, $\mathbb{P}^r(\Pi), \mathbf{D}_{i_r}^{\cancel{lock}} \ldots \mathbf{D}_{i_{r+1}}^{\cancel{lock}} \models p_{\alpha_r}$, where $1 \leq r \leq n$. By Lemma 3, executing α_r in state $\mathbf{D}_{i_r}^{\cancel{lock}}$ in *STRIPS* takes the system to state $\mathbf{D}_{i_{r+1}}^{\cancel{lock}}$. This means that after the execution of the *STRIPS* sequence $\alpha_1, \ldots, \alpha_n$, one gets to state $\mathbf{D}_{i_{n+1}}^{\cancel{lock}}$, which satisfies $achieve_G$ and thus G. The proof concludes by recalling that $\mathbf{D}_{i_1}^{\cancel{lock}} = \mathbf{D}_0^{\cancel{lock}}$ (by Lemma 11) and $\mathbf{D}_m = \mathbf{D}_{i_{n+1}}$ (by definition). □

Lemma 12 (Extending derivations with a $protect$ transaction)**.** *Given a planning problem* $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{D} \rangle$*, let* $seq_1 \ldots seq_m$ *be a non-redundant derivation where* $seq_m = \mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_f \vdash \phi \mid \psi$*. If* $\ell \in \mathbf{D}_f$*, one can extend such derivation to derive* $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_{f'} \vdash (\ \phi\ \otimes\ protect(\ell)\ ) \mid \psi$*, where* $\mathbf{D}_{f'}^{\cancel{lock}} = \mathbf{D}_f^{\cancel{lock}}$ *and if* $lock(\ell, n) \in \mathbf{D}_f$ *then* $lock(\ell, n+1) \in \mathbf{D}_{f'}$*.*

Proof. Consider the sequent $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_f \vdash \phi \mid \psi$. Since $\ell \in \mathbf{D}_f$, applications of the inference rule (I2) using ℓ and $lock(\ell, n)$ results in $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_f \vdash (\ \phi \otimes\ \odot(\ell \otimes lock(\ell, n)))\ ) \mid \psi$. Then, applying the inference rule (I3) twice to the elementary updates $-lock(\ell, n)$ and $+lock(\ell, n+1)$, one can get:

$$\begin{aligned} \mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_{f'} \vdash (\ \phi\ \otimes\ \odot(\quad & \ell\ \otimes \\ & lock(\ell, n)\ \otimes \\ & -lock(\ell, n)\ \otimes \\ & +lock(\ell, n+1)\ )\ )\ ) \mid \psi \end{aligned} \tag{4.10}$$

Note that the last two derivation steps increase the counter in $lock(\ell, n)$ in $\mathbf{D}_{f'}$ to $n + 1$, but $\mathbf{D}_{f'}^{\cancel{lock}} = \mathbf{D}_f^{\cancel{lock}}$. Finally, $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_{f'} \vdash (\ \phi\ \otimes\ protect(\ell)\ ) \mid \psi$ is derivable via the inference rule (I1) using a rule of the form (4.1). □

Lemma 13 (Extending derivations with $unprotect$ transactions)**.** *Given a planning problem $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{D} \rangle$, let $L = \{\ell_1, \ldots, \ell_n\}$ be a set of literals and $seq_1 \ldots seq_m$ be a non-redundant derivation where $seq_m = \mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_f \vdash \phi \mid \psi$. Such derivation can be extended to derive $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_{f'} \vdash (\ \phi \otimes \odot\ unprotect(L)\ ) \mid \psi$ where $\mathbf{D}_{f'}^{\cancel{lock}} = \mathbf{D}_f^{\cancel{lock}}$ and, for each $\ell \in L$, if $lock(\ell, n) \in \mathbf{D}_f$ then $lock(\ell, n-1) \in \mathbf{D}_{f'}$.*

Proof. Let $\ell_i \in L$. Applying the inference rule (I2) using $lock(\ell_i, n)$ in $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_f \vdash \phi \mid \psi$, one can get $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_f \vdash (\ \phi \otimes \odot(lock(\ell_i, n)))\ ) \mid \psi$. Then, a couple of applications of the inference rule (I3) using the elementary updates $-lock(\ell_i, n)$ and $+lock(\ell_i, n-1)$ results in:

$$\begin{aligned} \mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_{f_i'} \vdash (\ \phi \otimes \odot (\quad & lock(\ell_i, n) \otimes \\ & - lock(\ell_i, n) \otimes \\ & + lock(\ell_i, n-1)\ )\ ) \mid \psi \end{aligned} \tag{4.11}$$

Clearly, the counter in $lock(\ell, n)$ is decreased from $\mathbf{D}_f$ to $\mathbf{D}_{f'}$ while $\mathbf{D}_{f_i'}^{\cancel{lock}} = \mathbf{D}_f^{\cancel{lock}}$. Then, an application of the inference rule (I1) using a rule of the form (4.2) results in $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_{f_i'} \vdash (\ \phi \otimes unprotect(\ell_i)\ ) \mid \psi$. Repeating the above derivation steps for every $1 \leq i \leq n$ results in $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_{f_n'} \vdash (\ \phi \otimes \odot\ unprotect(L)\ ) \mid \psi$ where $\mathbf{D}_{f_n'} = \mathbf{D}_{f'}$ and $\mathbf{D}_{f'}^{\cancel{lock}} = \mathbf{D}_f^{\cancel{lock}}$. □

The following lemmas are analogous to Lemmas 5, 6, and 7; it will be used to prove the completeness of the *STRIPS*^{R} planner of Definition 14.

Lemma 14 (Achieved literals for $\mathcal{TR}$ planning with regression)**.** *Let $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \_ \rangle$ be a planning problem and $\mathbb{P}^r(\Pi)$ be the set of $\mathcal{TR}$ planning rules of Definition 14. For every literal ℓ and every state $\mathbf{D}$, if $\mathbb{P}^r(\Pi), \mathbf{D} \models \ell$, then there must be $\mathbf{D}_f$ such that $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_f \vdash achieve^r_\ell$, where $\mathbf{D}^{\cancel{lock}} = \mathbf{D}_f^{\cancel{lock}}$ and $\mathbf{D}_f^{lock}$ is exactly like $\mathbf{D}^{lock}$ except if $lock(\ell, n) \in \mathbf{D}^{lock}$ then $lock(\ell, n+1) \in \mathbf{D}_f^{lock}$.*

Proof. We have two cases:

1. *ℓ is extensional.* By Lemma 12, $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_f \vdash protect(\ell)$, where $\mathbf{D}^{\cancel{lock}} = \mathbf{D}_f^{\cancel{lock}}$ and $\mathbf{D}_f^{lock}$ is exactly same as $\mathbf{D}^{lock}$ except the counter in $lock(\ell, n)$ is increased by one from $\mathbf{D}^{lock}$ to $\mathbf{D}_f^{lock}$. Then, $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_f \vdash achieve_\ell$ is derivable via an application of inference rule (I1) using planning rule of the form (P10).

2. *ℓ is intensional.* First, we need to show how planning rules of the form (P8) can be used in a derivation. Then, we will show how to derive $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_f \vdash achieve^r_\ell$ for some $\mathbf{D}_f$. Let $h \leftarrow b_1 \wedge \cdots \wedge b_n$ be a ground instance of a rule from $\mathbb{R}$ and $B_h = \{b_1, \ldots, b_n\}$. By definition of ground instances of rules, if for some $\mathbf{D}_m$ we have $\mathbb{R}, \mathbf{D}_m \vdash B_h$ then $\mathbb{R}, \mathbf{D}_m \vdash h$ is derivable. By construction of $\mathbb{P}^r(\Pi)$, it has a planning rule of the form $achieve_h \leftarrow \|_{b_i \in B_h} achieve_{b_i} \otimes \odot(unprotect(B_h) \otimes protect(h))$. Suppose $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_m \vdash \|_{b_i \in B_h} achieve_{b_i} \mid rest$ is a derivable sequent via $\mathcal{TR}$'s inference rules. Then, by Lemma 13, we can derive

$$\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_m \ldots \mathbf{D}_{f_{B_h}} \vdash \begin{array}{l} \|_{b_i \in B_h} achieve_{b_i} \otimes \\ \odot unprotect(B_h) \mid rest \end{array}$$

where $\mathbf{D}_m^{\cancel{lock}} = \mathbf{D}_{f_{B_h}}^{\cancel{lock}}$ and, for every $b_i \in B_h$, the counter in $lock(b_i, n)$ is changed to $lock(b_i, n-1)$ from $\mathbf{D}_m^{lock}$ to $\mathbf{D}_{f_{B_h}}^{lock}$. Then, by Lemma 12, we get:

$$\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_m \ldots \mathbf{D}_{f_{B_h}} \ldots \mathbf{D}_{f_h} \vdash \begin{array}{l} \|_{b_i \in B_h} achieve_{b_i} \otimes \\ \odot(unprotect(B_h) \otimes protect(h)) \mid \\ rest \end{array}$$

where $\mathbf{D}_{f_{B_h}}^{\cancel{lock}} = \mathbf{D}_{f_h}^{\cancel{lock}}$ and $lock(h, n)$ is changed to $lock(h, n+1)$ from $\mathbf{D}_{f_{B_h}}^{lock}$ to $\mathbf{D}_{f_h}^{\cancel{lock}}$. Therefore, $\mathbf{D}_m^{\cancel{lock}} = \mathbf{D}_{f_h}^{\cancel{lock}}$ and $\mathbf{D}_{f_h}^{lock}$ is like $\mathbf{D}_m^{lock}$ except the counter in $lock(h, n)$ is increased by one from $\mathbf{D}_m^{lock}$ to $\mathbf{D}_{f_h}^{lock}$ and, for every $b_i \in B_h$, the counter in $lock(b_i, n)$ is decreased by one going from $\mathbf{D}_m^{lock}$ to $\mathbf{D}_{f_h}^{lock}$. These derivation steps show how an application of a rule of the form $p(\overline{X}) \leftarrow \wedge_{i=1}^n p_i(\overline{X}_i)$ can correspond to an application of a $\mathcal{TR}$ planning rule of the form $achieve_p(\overline{X}) \leftarrow \|_{i=1}^n achieve_{p_i}(\overline{X}_i) \otimes \odot (\otimes_{i=1}^n unprotect(p_i(\overline{X_i})) \otimes protect(p(\overline{X})))$. Recall that rules from $\mathbb{R}$ are the *only* rules in $\mathbb{P}^r(\Pi)$ that have intensional literals in their heads. Therefore, $\mathbb{P}^r(\Pi), \mathbf{D} \models \ell$ implies that there must be at least one derivation of ℓ formed by a set of ground instances of rules from $\mathbb{R}$ that are true in $\mathbf{D}$, i.e., $\mathbb{R}, \mathbf{D} \vdash \ell$. Due to the correspondence between rules in $\mathbb{R}$ and planning rules of the form (P8), any derivation of $\mathbb{R}, \mathbf{D} \vdash \ell$ has a corresponding derivation for $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_f \vdash achieve_\ell$ where $\mathbf{D}^{\cancel{lock}} = \mathbf{D}_f^{\cancel{lock}}$.

   Next, we will prove that the counter in $lock(\ell, n)$ is increased by one going from $\mathbf{D}^{lock}$ to $\mathbf{D}_f^{lock}$ and the rest of the literals of $\mathbf{D}^{lock}$ are not changed going from $\mathbf{D}^{lock}$ to $\mathbf{D}_f^{lock}$. Recall that $protect(\ell)$ is the only transaction that can increment the lock count n in $lock(\ell, n)$, while $unprotect(\ell)$ is the only transaction that can decrement n in $lock(\ell, n)$.

Let $seq_0, \ldots, seq_n$ be a non-redundant derivation of $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_f \vdash achieve^r_\ell$. For each literal ℓ_i, let $A(\ell_i)$, $P(\ell_i)$, and $U(\ell_i)$ denote the number of sequents (among $seq_0, \ldots, seq_n$) that are derived by applying the $\mathcal{TR}$ inference rule (I1) using a planning rule whose head is $achieve(\ell_i)$, $protect(\ell_i)$, and $unprotect(\ell_i)$, respectively.

Literals of the form $protect(\ell_i)$ appear in the body of planning rules (P8) and (P10), or in the body of a planning rule of the form (P13), which in turn is called by a rule of the form (P12). Therefore, for every seq_i derived by an application of inference rule (I1) using a rule of the form $protect(\ell_i) \leftarrow \ldots$, there is a sequent seq_j such that $i < j$ and seq_j is derived by an application of inference rule (I1) using a rule of the form $achieve_{\ell_i} \leftarrow \ldots$. Thus, $A(\ell_i) \geq P(\ell_i)$. On the other hand, all of the rules of the form $achieve_{\ell_i} \leftarrow \ldots$ in $\mathbb{P}^r(\Pi)$ call $protect(\ell_i)$ in their bodies. Then, one can show that for every seq_j derived by an application of inference rule (I1) using a rule of the form $achieve_{\ell_i} \leftarrow \ldots$, there is a sequent seq_i such that $i < j$ and seq_i is derived by an application of inference rule (I1) using a rule of the form $protect(\ell_i) \leftarrow \ldots$. Then, $A(\ell_i) \leq P(\ell_i)$, which together with the above implies $A(\ell_i) = P(\ell_i)$.

Likewise, by the construction of $\mathcal{TR}$ planning rules, $achieve_{\ell_i}$ appears either in the body of a planning rule of the form (P8) or (P13), or is called in the body of a rule of the form (P14). Clearly, the rule of the form (P14) is not used in this derivation. In the body of planning rules of the form (P8) and (P13), calling $achieve_{\ell_i}$ is always followed by a call to $unprotect(\ell_i)$. There are two cases to consider:

(a) $\ell_i \neq \ell$: Every $achieve_{\ell_i}$ is called in the body of planning rules of the form (P8) or (P13), that is always followed by a call to $unprotect(\ell_i)$. Thus $A(\ell_i) = U(\ell_i)$. Since $A(\ell_i) = P(\ell_i)$, $protect(\ell_i)$ and $unprotect(\ell_i)$ are called the same number of times and thus, the counter in $lock(\ell_i, n)$ is not increased from $\mathbf{D}$ to $\mathbf{D}_f$.

(b) $\ell_i = \ell$: Clearly, $achieve_\ell$ appears only once in seq_m. If not, $seq_0, \ldots, seq_m$ would be a redundant derivation. Then, $protect(\ell)$ is never followed by an $unprotect(\ell)$. Therefore, the counter in $lock(\ell_i, n)$ is increased by one from $\mathbf{D}$ to $\mathbf{D}_f$.

□

Lemma 15 (Plans of length zero with regression analysis)**.** *Let* $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{D} \rangle$ *be a planning problem that has a solution plan of length zero. Then the* $\mathcal{TR}$ *inference system can generate a derivation for* $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_f \vdash achieve^r_G$ *whose pivoting sequence is empty,* $\mathbf{D}^{\cancel{lock}} = \mathbf{D}_f^{\cancel{lock}}$*, and, for every* $g \in G$*,* $lock(g, 1) \in \mathbf{D}_f^{lock}$ *while, for every* $g \notin G$*,* $lock(g, 0) \in \mathbf{D}_f^{lock}$*.*

Proof. If Π has a plan of length zero then $G \subseteq \mathbf{D}$. By Lemma 14, $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_f \vdash \|_{g \in G} achieve^r_g$, where $\mathbf{D}^{\cancel{lock}} = \mathbf{D}_f^{\cancel{lock}}$ and, for every $g \in G$, $lock(g, 1) \in \mathbf{D}_f^{lock}$ while for every $g \notin G$, $lock(g, 0) \in \mathbf{D}_f^{lock}$. This gives us $\mathbb{P}^r(\Pi), \mathbf{D} \ldots \mathbf{D}_f \vdash achieve^r_G$. Since this derivation does not involve the planning rules $tr(\alpha)$, it has an empty pivoting sequence. □

Lemma 16 (Plans of length $> \mathbf{1}$ with regression analysis: the inductive step)**.** *Suppose that, if a STRIPS planning problem* $\Pi' = \langle \mathbb{R}, \mathbb{A}, G', \mathbf{D}_0 \rangle$ *has a non-redundant plan* σ' *of length* $k \leq n$*, then it has a* $\mathcal{TR}$*-derivation for the sequent* $\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \vdash achieve^r_{G'}$ *such that*

- *the pivoting sequence of this derivation is* σ'*;*
- $\mathbf{D}_{f'}^{\cancel{lock}} = \mathbf{S}_{f'}^{\cancel{lock}}$*, where* $\mathbf{S}_{f'}$ *is the final state of the execution of* σ' *starting at* $\mathbf{D}_0$*;*
- *for every* $g \in G'$*,* $lock(g, 1) \in \mathbf{D}_{f'}^{lock}$*; and*
- *for every* $g \notin G'$*,* $lock(g, 0) \in \mathbf{D}_{f'}^{lock}$*.*

Then, for any planning problem $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{D}_0 \rangle$ *that has a non-redundant plan* σ *of length* $n+1$*, there must be a* $\mathcal{TR}$*-derivation for* $\mathbb{P}^r(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_f \vdash achieve^r_G$ *such that*

- *the pivoting sequence of that derivation is* σ*;*
- $\mathbf{D}_f^{\cancel{lock}} = \mathbf{S}_f^{\cancel{lock}}$*, where* $\mathbf{S}_f$ *is the final state of the execution of* σ *starting at* $\mathbf{D}_0$*;*
- *for every* $g \in G$*,* $lock(g, 1) \in \mathbf{D}_f^{lock}$*; and*
- *for every* $g \notin G$*,* $lock(g, 0) \in \mathbf{D}_f^{lock}$*.*

Proof. We will show that any non-redundant *STRIPS* plan is also an *STRIPS*^{R} plan. The proof mimics Lemma 7, but is slightly more complex because we need to keep track of the $lock$-facts.

Let $\sigma = \alpha_1, \ldots, \alpha_n, \alpha_{n+1}$ and $\sigma' = \alpha_1, \ldots, \alpha_n$. Suppose $\mathbf{S}_f$ and $\mathbf{S}_{f'}$ are the final state of the executions of σ and σ' starting at $\mathbf{D}_0$, as in the statement of the lemma. Consider the set of literals $G' = (\mathbf{S}_{f'} \cap G) \cup Pre_{\alpha_{n+1}}$ where $Pre_{\alpha_{n+1}}$ is the precondition of α_{n+1}. Consider the planning problem $\Pi' = \langle \mathbb{R}, \mathbb{A}, G', \mathbf{D}_0 \rangle$. We will show that σ' is a non-redundant plan of length n for Π'. Since α_{n+1} is executable at $\mathbf{S}_{f'}$, it follows that $Pre_{\alpha_{n+1}} \subseteq \mathbf{S}_{f'}$. As $\mathbf{S}_{f'} \cap G \subseteq \mathbf{S}_{f'}$, we obtain $G' \subseteq \mathbf{S}_{f'}$ and thus σ' is a plan for Π'. It is a *non-redundant* plan for Π' because if ς were a subsequence of σ' and also a plan for Π' then α_{n+1} would have been executable in the final state of ς (since $Pre_{\alpha_{n+1}} \subseteq G'$) and thus $\langle \varsigma, \alpha_{n+1} \rangle$ would have been a plan for Π. Since $\langle \varsigma, \alpha_{n+1} \rangle$ is a subsequence of σ, this would contradict the non-redundancy assumption about σ.

Since $\sigma' = \alpha_1 \ldots \alpha_n$ is a non-redundant plan of length n for Π', the assumption of the lemma guarantees a $\mathcal{TR}$-derivation for a sequent of the form $\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \vdash achieve^r_{G'}$ where $\mathbf{D}^{\cancel{lock}}_{f'} = \mathbf{S}^{\cancel{lock}}_{f'}$ and the pivoting sequence of that derivation is σ'. Therefore, with the help of the planning rule (P14), we also derive $\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \vdash (\|_{g \in G'}\, achieve^r_g) \otimes (\wedge_{g \in G'}\, g)$. A prefix of that derivation (with the same pivoting sequence) derives $\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \vdash (\|_{g \in G'}\, achieve^r_g)$. Due to the assumption about $\mathbf{D}^{lock}_{f'}$, for every $\ell \in Pre_{\alpha_{n+1}}$, $lock(\ell, 1) \in \mathbf{D}_{f'}$. By Lemma 13, we get:

$$\begin{aligned} \mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \ldots \mathbf{D}_{f_u} \vdash\ & (\|_{g \in G''}\, achieve^r_g) \mid \\ & (\|_{\ell \in Pre_{\alpha_{n+1}}}\, achieve^r_\ell) \otimes \\ & \odot unprotect(Pre_{\alpha_{n+1}}) \end{aligned} \tag{4.12}$$

where $\mathbf{D}^{\cancel{lock}}_{f'} = \mathbf{D}^{\cancel{lock}}_{f_u}$, $G'' = G' \setminus Pre_{\alpha_{n+1}}$, and, for every $\ell \in Pre_{\alpha_{n+1}}$, the counter in $lock(\ell, n)$ is decreased by one going from $\mathbf{D}_{f'}$ to $\mathbf{D}_{f_u}$. Thus, for every $g \in G''$ we have $lock(g, 1) \in \mathbf{D}_{f_u}$ and for every $g \notin G''$, $lock(g, 0) \in \mathbf{D}_{f_u}$. By definition, G'' is the set of already achieved goals in $\mathbf{D}_{f_u}$. Since $G' = (\mathbf{S}_{f'} \cap G) \cup Pre_{\alpha_{n+1}}$ and $G'' = G' \setminus Pre_{\alpha_{n+1}}$, then $G'' \subseteq \mathbf{S}_{f'} \cap G$. Clearly, G'' and α_{n+1} are ground. Then, since α_{n+1} is executable at $\mathbf{S}_{f'}$ and $G'' \subseteq G \subseteq \mathbf{S}_f$, by Lemma 10, there exists $\alpha'_{n+1} \in \mathfrak{R}(\alpha_{n+1}, G'')$, which is a perfect regression of G'' through α_{n+1}. Then, the goal $regress(p_{\alpha_{n+1}}(\overline{X}), p_{\alpha'_{n+1}}(\overline{X}))$ succeeds at state $\mathbf{D}_{f_u}$. Now, applying the inference rule (I2) using $regress(p_{\alpha_{n+1}}(\overline{X}), p_{\alpha'_{n+1}}(\overline{X}))$ to (4.12), we get:

$$\begin{aligned}\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \ldots \mathbf{D}_{f_u} \vdash \; & (\|_{g \in G''}\, achieve^r_g) \mid \\ & (\|_{\ell \in Pre_{\alpha_{n+1}}}\, achieve^r_\ell) \otimes \\ & \odot(\, unprotect(Pre_{\alpha_{n+1}}) \otimes \\ & \quad regress(p_{\alpha_{n+1}}(\overline{X}), p_{\alpha'_{n+1}}(\overline{X}))\,)\end{aligned} \tag{4.13}$$

Since $\mathbf{S}_f$ is the final state of the execution of α_{n+1} at $\mathbf{S}_{f'}$ and α'_{n+1} is a perfect regression of G'' through α_{n+1}, it follows that α'_{n+1} is also executable at $\mathbf{S}_{f'}$ and $\alpha'_{n+1}(\mathbf{S}_{f'}) = \mathbf{S}_f$. Therefore, by Lemma 3, $\mathbb{P}^r(\Pi'), \mathbf{S}_{f'} \ldots \mathbf{S}_f \vdash p_{\alpha'_{n+1}}$. Moreover, by Proposition 2, $\mathbf{S}^{lock}_{f'} = \mathbf{S}^{lock}_f$. Clearly, $\mathbf{S}^{\cancel{lock}}_{f'} = \mathbf{D}^{\cancel{lock}}_{f_u}$ because $\mathbf{D}^{\cancel{lock}}_{f'} = \mathbf{S}^{\cancel{lock}}_{f'}$ and $\mathbf{D}^{\cancel{lock}}_{f'} = \mathbf{D}^{\cancel{lock}}_{f_u}$. Let $\mathbf{D}_{f_{ue}} = \mathbf{S}^{\cancel{lock}}_f \cup \mathbf{D}^{lock}_{f_u}$. Then, by Proposition 2, one can derive $\mathbb{P}^r(\Pi'), \mathbf{D}_{f_u} \ldots \mathbf{D}_{f_{ue}} \vdash p_{\alpha_{n+1}}$. Appending this latter derivation to the derivation of (4.13), we get

$$\begin{aligned}\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \ldots \mathbf{D}_{f_u} \ldots \; & \mathbf{D}_{f_{ue}} \vdash \\ & (\|_{g \in G''}\, achieve^r_g) \mid \\ & (\|_{\ell \in Pre_{\alpha_{n+1}}}\, achieve^r_\ell) \otimes \\ & \odot(\, unprotect(Pre_{\alpha_{n+1}}) \otimes \\ & \quad regress(p_{\alpha_{n+1}}(\overline{X}), p_{\alpha'_{n+1}}(\overline{X})) \otimes \\ & \quad p_{\alpha'_{n+1}}(\overline{X})\,)\end{aligned} \tag{4.14}$$

where $\mathbf{D}^{lock}_{f_u} = \mathbf{D}^{lock}_{f_{ue}}$ and $\mathbf{D}^{\cancel{lock}}_{f_{ue}}$ is the result of the execution of α'_{n+1} at $\mathbf{D}^{\cancel{lock}}_{f_u}$. Consider the set $G \setminus \mathbf{S}_{f'}$. Clearly, $G \setminus \mathbf{S}_{f'} \neq \emptyset$ for otherwise $\alpha_1, \ldots, \alpha_n$ would have been a plan for Π (starting at $\mathbf{D}_0$ and ending at $\mathbf{S}_{f'}$), contrary to non-redundancy of σ. Let $\varrho \in G \setminus \mathbf{S}_{f'}$. Recall that $G \subseteq \mathbf{S}_f$ and thus $\varrho \in G \subseteq \mathbf{S}_f$. Therefore, for any state $\mathbf{S}$ (including $\mathbf{D}_{f_{ue}}$) such that $\mathbf{S}^{\cancel{lock}} = \mathbf{S}^{\cancel{lock}}_f$, we have $\mathbb{P}^r(\Pi), \mathbf{S} \models \varrho$. Moreover, since $\varrho \notin \mathbf{S}_{f'}$, for every state $\mathbf{S}$ such that $\mathbf{S}^{\cancel{lock}} = \mathbf{S}^{\cancel{lock}}_{f'}$ (including $\mathbf{D}_{f'}$ and $\mathbf{D}_{f_u}$), $\mathbb{P}^r(\Pi), \mathbf{S} \not\models \varrho$.

Next we will show that $\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f_{ue}} \ldots \mathbf{D}_{f_{uep}} \vdash (\|_{g \in G''}\, achieve_g) \mid achieve_\varrho$ for some $\mathbf{D}_{f_{uep}}$ such that $\mathbf{D}^{\cancel{lock}}_{f_{uep}} = \mathbf{S}^{\cancel{lock}}_f$ and the counter in $lock(\varrho, n)$ is increased by one going from $\mathbf{D}_{f_{ue}}$ to $\mathbf{D}_{f_{uep}}$. There are two cases:

1. *ϱ is extensional*: By Lemma 12, one can derive

$$\begin{aligned}\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \ldots \; & \mathbf{D}_{f_u} \ldots \mathbf{D}_{f_{ue}} \ldots \mathbf{D}_{f_{uep}} \vdash \\ & (\|_{g \in G''}\, achieve^r_g) \mid \\ & (\|_{\ell \in Pre_{\alpha_{n+1}}}\, achieve^r_\ell) \otimes \\ & \odot(\, unprotect(Pre_{\alpha_{n+1}}) \otimes \\ & \quad regress(p_{\alpha_{n+1}}(\overline{X}), p_{\alpha'_{n+1}}(\overline{X})) \otimes \\ & \quad p_{\alpha'_{n+1}}(\overline{X}) \otimes protect(\varrho)\,)\end{aligned} \tag{4.15}$$

where $\mathbf{D}_{fue}^{\cancel{lock}} = \mathbf{D}_{fuep}^{\cancel{lock}}$ and the counter in $lock(\varrho, n)$ is increased by one from $\mathbf{D}_{fue}$ to $\mathbf{D}_{fuep}$. Since ϱ is an extensional literal, it must be that $\varrho \in E_{\alpha_{n+1}}$. If not, since $\mathbf{D}_{fue}$ is the state obtained by executing α'_{n+1} at $\mathbf{D}_{f_u}$ and $\varrho \notin \mathbf{D}_{f_u}$, it would follow that $\varrho \notin \mathbf{D}_{f_{ue}}$, which means $\varrho \notin \mathbf{S}_f$. Therefore, assuming $\varrho \notin E_{\alpha_{n+1}}$ contradicts the earlier conclusion that $\mathbb{P}^r(\Pi'), \mathbf{S}_f \models \varrho$. Thus, $\varrho \in E_{\alpha_{n+1}}$ and, by construction of $\mathcal{TR}$ planning rules, $\mathbb{P}^r(\Pi')$ has a rule of the form (P13), which can be used to derive

$$\begin{aligned}\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \ldots \mathbf{D}_{f_u} \ldots \quad & \mathbf{D}_{f_{ue}} \ldots \mathbf{D}_{f_{uep}} \vdash \\ & (\|_{g \in G''}\, achieve^r_g) \mid \\ & execute^r_{p_{\alpha'_{n+1}}}(\varrho)\end{aligned} \tag{4.16}$$

Moreover, by Definition 14, $\mathbb{P}^r(\Pi')$ has a rule $achieve^r_\varrho \leftarrow execute^r_{p_{\alpha_{n+1}}}(\varrho)$. Applying the inference rule (I1) to the sequent (4.16) using $achieve^r_\varrho \leftarrow execute^r_{p_{\alpha_{n+1}}}(\varrho)$ yields the sequent $\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f_{uep}} \vdash (\|_{g \in G''}\, achieve^r_g) \mid achieve^r_\varrho$.

2. *ϱ is intensional*: Since $\varrho \in \mathbf{S}_f$, by Lemma 14, there must be a derivation of $\mathbb{P}^r(\Pi'), \mathbf{S}_f \ldots \mathbf{S}'_f \vdash achieve_\varrho$. Consider a *non-redundant* such derivation seq_0, ..., seq_m, where seq_0 is an axiom and seq_m is $\mathbb{P}^r(\Pi'), \mathbf{S}_f \ldots \mathbf{S}'_f \vdash achieve^r_\varrho$. By Lemma 1, each seq_i (except seq_0) was derived from seq_{i-1}. As shown in Lemma 14, such derivation is produced from a derivation of $\mathbb{R}, \mathbf{S}_f \models \varrho$ using planning rules of the form

$$\begin{aligned}achieve^r_p(\overline{X}) \leftarrow \quad & (\|^n_{i=1} achieve^r_{p_i}(\overline{X}_i)) \otimes \\ & \odot(\ \otimes^n_{i=1}\ unprotect(p_i(\overline{X}_i)) \otimes \\ & \quad protect(p(\overline{X}))\ ).\end{aligned} \tag{4.17}$$

This derivation must have a sequent of the form $\mathbb{P}^r(\Pi), \mathbf{S}_f \ldots \mathbf{S}'_f \vdash achieve^r_h$, obtained via the $\mathcal{TR}$ inference rule (I1) and a planning rule of the form $achieve^r_h(\overline{X}) \leftarrow \ldots | achieve^r_\ell(\overline{X_\ell}) \otimes \ldots$, where ℓ is an extensional literal such that $\ell \in E_{\alpha_{n+1}}$ and $\ell \notin \mathbf{S}_{f'}$. If there were no such ℓ, we could have used the above derivation to show that $\mathbb{P}^r(\Pi), \mathbf{S}_{f'} \models \varrho$, contrary to the earlier conclusion that $\mathbb{P}^r(\Pi), \mathbf{S}_{f'} \not\models \varrho$. Thus, from (4.14) and Lemma 12, one can derive

$$
\begin{aligned}
\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \ldots \quad & \mathbf{D}_{f_u} \ldots \mathbf{D}_{f_{ue}} \ldots \mathbf{D}_{f_{uep_\ell}} \vdash \\
& (\|_{g \in G''}\, achieve_g^r) \mid \\
& (\|_{\ell \in Pre_{\alpha_{n+1}}}\, achieve_\ell^r) \otimes \\
& \odot(\ unprotect(Pre_{\alpha_{n+1}}) \otimes \\
& \quad regress(p_{\alpha_{n+1}}(\overline{X}), p_{\alpha'_{n+1}}(\overline{X})) \otimes \\
& \quad p_{\alpha'_{n+1}}(\overline{X}) \otimes protect(\ell)\ )
\end{aligned}
\tag{4.18}
$$

where $\mathbf{D}_{fue}^{\cancel{lock}} = \mathbf{D}_{fuep_\ell}^{\cancel{lock}}$ and the counter in $lock(\ell, n)$ is increased by one from $\mathbf{D}_{f_{ue}}$ to $\mathbf{D}_{f_{uep_\ell}}$. By Definition 14, $\mathbb{P}^r(\Pi')$ has a rule of the form (P13) whose head is $execute^r_{p_{\alpha_{n+1}}}(\overline{X}, \ell)$. This rule can be used by the $\mathcal{TR}$ inference rule (I1) to derive:

$$
\begin{aligned}
\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \ldots \mathbf{D}_{f_u} \ldots \quad & \mathbf{D}_{f_{ue}} \ldots \mathbf{D}_{f_{uep_\ell}} \vdash \\
& (\|_{g \in G''}\, achieve_g^r) \mid \\
& execute^r_{p_{\alpha_{n+1}}}(\overline{X}, \ell).
\end{aligned}
\tag{4.19}
$$

Since $\ell \in E_{\alpha_{n+1}}$, Definition 14 says that $\mathbb{P}^r(\Pi')$ has a rule of the form $achieve_\ell^r \leftarrow execute^r_{p_{\alpha_{n+1}}}(\overline{X}, \ell)$. An application of the $\mathcal{TR}$ inference rule (I1) to the sequent (4.19) using $achieve_\ell^r \leftarrow execute^r_{p_{\alpha_{n+1}}}(\overline{X}, \ell)$ then yields

$$
\mathbb{P}(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f_{uep_\ell}} \vdash (\|_{g \in G''}\, achieve_g^r) \mid achieve_\ell^r. \tag{4.20}
$$

Now, rerun the inference steps for the aforesaid derivation seq_0, ..., seq_m starting with the sequent (4.20) instead of seq_0 and omit the step that introduces the literal $achieve_\ell^r$ to the goal on the right-hand side of the sequents. This step can be omitted because $achieve_\ell^r$ already exists in (4.20). By Lemma 14, the result of such a derivation is the sequent $\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f_{uep}} \vdash (\|_{g \in G''}\, achieve_g^r) \mid achieve_\varrho^r$ where $\mathbf{D}_{f_{uep_\ell}}^{\cancel{lock}} = \mathbf{D}_{f_{uep}}^{\cancel{lock}}$ and $lock(\ell, n_\ell)$, $lock(\varrho, n_\varrho)$ in $\mathbf{D}_{f_{uep_\ell}}^{\cancel{lock}}$ become $lock(\ell, n_\ell - 1)$ and $lock(\varrho, n_\varrho + 1)$ in $\mathbf{D}_{f_{uep}}$, respectively. Therefore, $\mathbf{D}_{f_{ue}}^{\cancel{lock}} = \mathbf{D}_{f_{uep}}^{\cancel{lock}}$ and the counter in $lock(\varrho, n)$ is increased by one going from $\mathbf{D}_{f_{ue}}$ to $\mathbf{D}_{f_{uep}}$, while the counter in $lock(\ell, n)$ is not changed.

Let $L = G \setminus (G'' \cup \{\varrho\})$. Clearly, $L \subseteq \mathbf{D}_{f_{uep}}^{\cancel{lock}}$ since the entire G is assumed to be in $\mathbf{S}_f^{\cancel{lock}}$ and $\mathbf{S}_f^{\cancel{lock}} = \mathbf{D}_{f_{uep}}^{\cancel{lock}}$. Therefore, by Lemma 14, starting with $\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f_{uep}} \vdash (\|_{g \in G''}\, achieve_g^r) \mid achieve_\varrho^r$, one can derive the sequent

$$
\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_{f_{uep}} \ldots \mathbf{D}_f \vdash (\|_{g \in G''}\, achieve_g^r) \mid achieve_\varrho^r \mid (\|_{g \in L}\, achieve_g^r)
$$

which is simply $\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_f \vdash (\|_{g\in G}\, achieve^r_g)$. Here $\mathbf{D}_f$ is chosen so that $\mathbf{S}^{\cancel{lock}} = \mathbf{D}^{\cancel{lock}}_{f_{uep}} = \mathbf{D}^{\cancel{lock}}_f$ and, by Lemma 14, for every $\ell \in L$, the counter in $lock(\ell, n)$ is increased by one going from $\mathbf{D}_{f_{uep}}$ to $\mathbf{D}_f$. Therefore, for every $g \in G$, $lock(g,1) \in \mathbf{D}^{lock}_f$ and for every $g \notin G$, $lock(g,0) \in \mathbf{D}^{lock}_f$. Since $G \subseteq \mathbf{D}_f$, the inference rule (I2) also gives us $\mathbb{P}^r(\Pi'), \mathbf{D}_0 \ldots \mathbf{D}_f \vdash (\|_{g\in G}\, achieve^r_g) \otimes (\wedge_{g\in G}\, g)$. Finally, $\mathbb{P}^r(\Pi)$ differs from $\mathbb{P}^r(\Pi')$ only in the form of the rule (P14). In case of $\mathbb{P}^r(\Pi)$, that rule has exactly the form that lets one apply the inference rule (I1) to get $\mathbb{P}^r(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_f \vdash achieve^r_G$. □

Theorem 7 (Completeness of *STRIPS*^{R})**.** *If there is a non-redundant plan that achieves the goal G from the initial state* $\mathbf{D}_0$ *then the* $\mathcal{TR}$*-based STRIPS*^{R} *planner will find a plan.*

Proof. By induction on the length of the plan. Lemma 15 proves the base case of the induction and Lemma 16 establishes the inductive step.

□

The following example shows how the inference rules of $\mathcal{TR}$ inference system in Definition 7 are applied to the planning rules in Example 5 to construct a plan. This example also illustrates why planning rules in Example 5 are more efficient than those in Example 3.

Example 6 (Register exchange with regression analysis, continued)**.** *Consider the register exchange problem in Example 3. Let* $\mathbb{P}^r$ *be the set of* $\mathcal{TR}$ *rules (4.4-4.8) that constitute the planner for the* $\mathcal{TR}$*-based planner with regression analysis for this problem. Given the planning goal* $G = \{value(x,b), value(y,a)\}$ *and the initial state* $\mathbf{D}_0$*, where* $\{value(x,a), value(y,b)\} \subseteq \mathbf{D}_0$*, we will show how the inference system in Definition 7 constructs a derivation (and thus a plan) for the sequent* $\mathbb{P}^r, \mathbf{D}_0 \cdots \mathbf{D}_n \vdash achieve^r_G$ *for some* $\mathbf{D}_n$ *such that* $\{value(x,b), value(y,a)\} \subseteq \mathbf{D}_n$.

Consider the sequent $\mathbb{P}^r, \mathbf{D}_0 \cdots \vdash achieve^r_G$ *that corresponds to the query (3.2). Applying the inference rule (I1) to that sequent using the rule (4.9), we get:*

$$\mathbb{P}, \mathbf{D}_0 \cdots \vdash \begin{array}{l} (\ achieve^r_{value}(x,b) \mid achieve^r_{value}(y,a)\ ) \\ \otimes (value(x,b)\ \wedge value(y,a)). \end{array}$$

Applying the inference rule (I1) to the resulting sequent using the rule (4.6) with appropriate substitutions, we get:

$$\mathbb{P}, \mathbf{D}_0 \cdots \vdash \begin{array}{l} (\ execute_{copy}(z,x,b) \mid achieve^r_{value}(y,a)\ ) \\ \otimes (value(x,b)\ \wedge value(y,a)). \end{array}$$

from which, via an application of the inference rule (I1) using the rule (4.8), we get:

$$\begin{array}{rl}\mathbb{P}, \mathbf{D}_0 \cdots \vdash & (\,(\,(\,achieve^r_{value}(z,b) \mid achieve^r_{value}(x,a)\,) \otimes \\ & \odot\,(\,unprotect(\{value(z,b), value(x,a)\}) \otimes \\ & \quad regress(copy(z,x,b), copy'(z,x,b)) \otimes \\ & \quad copy'(z,x,b) \ \otimes\ protect(value(x,b))\,)\,) \mid \\ & \quad achieve^r_{value}(y,a)\,) \\ & \otimes\,(value(x,b)\ \wedge value(y,a)).\end{array}$$

Then, three applications of the inference rule (I1) using the rules (4.6), (4.8), and (4.5) with appropriate substitutions results in:

$$\begin{array}{rl}\mathbb{P}, \mathbf{D}_0 \cdots \vdash & (\,(\,(\,(\,(\,protect(value(y,b)) \mid achieve^r_{value}(z,t)\,) \otimes \\ & \odot\,(\,unprotect(\{value(y,b), value(z,t)\}) \otimes \\ & \quad regress(copy(y,z,b), copy'(y,z,b)) \otimes \\ & \quad copy'(y,z,b) \ \otimes\ protect(value(z,b))\,)\,) \mid \\ & \quad achieve^r_{value}(x,a)\,) \otimes \\ & \odot\,(\,unprotect(\{value(z,b), value(x,a)\}) \otimes \\ & \quad regress(copy(z,x,b), copy'(z,x,b)) \otimes \\ & \quad copy'(z,x,b) \ \otimes\ protect(value(x,b))\,)\,) \mid \\ & \quad achieve^r_{value}(y,a)\,) \\ & \otimes\,(value(x,b)\ \wedge value(y,a)).\end{array}$$

from which, by Lemma 12, we get

$$\begin{array}{rl}\mathbb{P}, \mathbf{D}_1 \cdots \vdash & (\,(\,(\,(\,achieve^r_{value}(z,t) \otimes \\ & \odot\,(\,unprotect(\{value(y,b), value(z,t)\}) \otimes \\ & \quad regress(copy(y,z,b), copy'(y,z,b)) \otimes \\ & \quad copy'(y,z,b) \ \otimes\ protect(value(z,b))\,)\,) \mid \\ & \quad achieve^r_{value}(x,a)\,) \otimes \\ & \odot\,(\,unprotect(\{value(z,b), value(x,a)\}) \otimes \\ & \quad regress(copy(z,x,b), copy'(z,x,b)) \otimes \\ & \quad copy'(z,x,b) \ \otimes\ protect(value(x,b))\,)\,) \mid \\ & \quad achieve^r_{value}(y,a)\,) \\ & \otimes\,(value(x,b)\ \wedge value(y,a)).\end{array}$$

where $\mathbf{D}_0^{\cancel{lock}} = \mathbf{D}_1^{\cancel{lock}}$ *and* $lock(value(y,b),0)$ *is changed to* $lock(value(y,b),1)$ *from* $\mathbf{D}_0^{lock}$ *to* $\mathbf{D}_1^{lock}$*. In order to highlight the role of regression and clarify the difference between executions of rules in Examples 3 and 5, we use inference rule*

(I4) at this point. Then an application of the inference rule (I1) to the resulting sequent using the rule (4.6) with appropriate substitutions results in:

$$\begin{array}{ll} \mathbb{P}, \mathbf{D}_1 \cdots \vdash & (\,(\,(\,(\,achieve^r_{value}(z,t) \otimes \\ & \odot\,(\,unprotect(\{value(y,b), value(z,t)\}) \otimes \\ & \quad regress(copy(y,z,b), copy'(y,z,b)) \otimes \\ & \quad copy'(y,z,b) \;\otimes\; protect(value(z,b))\,)\,)\,| \\ & \quad achieve^r_{value}(x,a)\,) \otimes \\ & \odot\,(\,unprotect(\{value(z,b), value(x,a)\}) \otimes \\ & \quad regress(copy(z,x,b), copy'(z,x,b)) \otimes \\ & \quad copy'(z,x,b) \;\otimes\; protect(value(x,b))\,)\,)\,| \\ & \quad execute^r_{copy}(x,y,a)\,) \\ & \otimes\,(value(x,b) \;\wedge value(y,a)). \end{array}$$

Then, applying the inference rule (I1) twice using the rules (P13) and (4.5), we get:

$$\begin{array}{ll} \mathbb{P}, \mathbf{D}_1 \cdots \vdash & (\,(\,(\,(\,achieve^r_{value}(z,t) \otimes \\ & \odot\,(\,unprotect(\{value(y,b), value(z,t)\}) \otimes \\ & \quad regress(copy(y,z,b), copy'(y,z,b)) \otimes \\ & \quad copy'(y,z,b) \;\otimes\; protect(value(z,b))\,)\,)\,| \\ & \quad achieve^r_{value}(x,a)\,) \otimes \\ & \odot\,(\,unprotect(\{value(z,b), value(x,a)\}) \otimes \\ & \quad regress(copy(z,x,b), copy'(z,x,b)) \otimes \\ & \quad copy'(z,x,b) \;\otimes\; protect(value(x,b))\,)\,)\,| \\ & \;(\,(\,protect(value(x,a)) \mid achieve^r_{value}(y,b)\,) \otimes \\ & \;\;\odot\;\,(\,unprotect(\{value(x,a), value(y,b)\}) \otimes \\ & \quad\;\; regress(copy(x,y,a), copy'(x,y,a)) \otimes \\ & \quad\;\; copy'(x,y,a) \;\otimes\; protect(value(y,a))\,)\,)\,)\,) \\ & \otimes\,(value(x,b) \;\wedge value(y,a)). \end{array}$$

from which, by Lemma 12, we get:

$$\begin{array}{ll}
\mathbb{P}, \mathbf{D}_2 \cdots \vdash & (\;(\;(\;(\;(\; achieve^r_{value}(z,t) \otimes \\
& \odot\;(\; unprotect(\{value(y,b), value(z,t)\}) \otimes \\
& \qquad regress(copy(y,z,b), copy'(y,z,b)) \otimes \\
& \qquad copy'(y,z,b) \;\otimes\; protect(value(z,b))\;)\;)\;| \\
& \qquad achieve^r_{value}(x,a)\;) \otimes \\
& \odot\;(\; unprotect(\{value(z,b), value(x,a)\}) \otimes \\
& \qquad regress(copy(z,x,b), copy'(z,x,b)) \otimes \\
& \qquad copy'(z,x,b) \;\otimes\; protect(value(x,b))\;)\;)\;| \\
& \quad (\; achieve^r_{value}(y,b) \otimes \\
& \quad\; \odot\;\;(\; unprotect(\{value(x,a), value(y,b)\}) \otimes \\
& \qquad\;\; regress(copy(x,y,a), copy'(x,y,a)) \otimes \\
& \qquad\;\; copy'(x,y,a) \;\otimes\; protect(value(y,a))\;)\;)\;)\;) \\
& \otimes\,(value(x,b) \;\wedge value(y,a)).
\end{array}$$

where $\mathbf{D}_1^{\cancel{lock}} = \mathbf{D}_2^{\cancel{lock}}$ *and* $lock(value(x,a),0)$ *is changed to* $lock(value(x,a),1)$ *from* $\mathbf{D}_1^{lock}$ *to* $\mathbf{D}_2^{lock}$*. Applying the inference rule (I1) once more and again using the rules (4.5), we get:*

$$\begin{array}{ll}
\mathbb{P}, \mathbf{D}_2 \cdots \vdash & (\;(\;(\;(\;(\; achieve^r_{value}(z,t) \otimes \\
& \odot\;(\; unprotect(\{value(y,b), value(z,t)\}) \otimes \\
& \qquad regress(copy(y,z,b), copy'(y,z,b)) \otimes \\
& \qquad copy'(y,z,b) \;\otimes\; protect(value(z,b))\;)\;)\;| \\
& \qquad achieve^r_{value}(x,a)\;) \otimes \\
& \odot\;(\; unprotect(\{value(z,b), value(x,a)\}) \otimes \\
& \qquad regress(copy(z,x,b), copy'(z,x,b)) \otimes \\
& \qquad copy'(z,x,b) \;\otimes\; protect(value(x,b))\;)\;)\;| \\
& \quad (\; protect(value(y,b)) \otimes \\
& \quad\; \odot\;\;(\; unprotect(\{value(x,a), value(y,b)\}) \otimes \\
& \qquad\;\; regress(copy(x,y,a), copy'(x,y,a)) \otimes \\
& \qquad\;\; copy'(x,y,a) \;\otimes\; protect(value(y,a))\;)\;)\;)\;) \\
& \otimes\,(value(x,b) \;\wedge value(y,a)).
\end{array}$$

Then, by Lemma 12, we get:

$$\begin{array}{rl}
\mathbb{P}, \mathbf{D}_3 \cdots \vdash & (\,(\,(\,(\,achieve^r_{value}(z,t) \otimes \\
& \odot\,(\,unprotect(\{value(y,b), value(z,t)\}) \otimes \\
& \quad regress(copy(y,z,b), copy'(y,z,b)) \otimes \\
& \quad copy'(y,z,b) \,\otimes\, protect(value(z,b))\,)\,)\,\mid \\
& \quad achieve^r_{value}(x,a)\,) \otimes \\
& \odot\,(\,unprotect(\{value(z,b), value(x,a)\}) \otimes \\
& \quad regress(copy(z,x,b), copy'(z,x,b)) \otimes \\
& \quad copy'(z,x,b) \,\otimes\, protect(value(x,b))\,)\,)\,\mid \\
& \;\; \odot\,\,(\,unprotect(\{value(x,a), value(y,b)\}) \otimes \\
& \quad\; regress(copy(x,y,a), copy'(x,y,a)) \otimes \\
& \quad\; copy'(x,y,a) \,\otimes\, protect(value(y,a))\,)\,) \\
& \otimes\,(value(x,b) \,\wedge value(y,a)).
\end{array}$$

where $\mathbf{D}_2^{\cancel{lock}} = \mathbf{D}_3^{\cancel{lock}}$ *and* $lock(value(y,b),1)$ *is changed to* $lock(value(y,b),2)$ *from* $\mathbf{D}_2^{lock}$ *to* $\mathbf{D}_3^{lock}$*. Then, by Lemma 13, we get:*

$$\begin{array}{rl}
\mathbb{P}, \mathbf{D}_5 \cdots \vdash & (\,(\,(\,(\,achieve^r_{value}(z,t) \otimes \\
& \odot\,(\,unprotect(\{value(y,b), value(z,t)\}) \otimes \\
& \quad regress(copy(y,z,b), copy'(y,z,b)) \otimes \\
& \quad copy'(y,z,b) \,\otimes\, protect(value(z,b))\,)\,)\,\mid \\
& \quad achieve^r_{value}(x,a)\,) \otimes \\
& \odot\,(\,unprotect(\{value(z,b), value(x,a)\}) \otimes \\
& \quad regress(copy(z,x,b), copy'(z,x,b)) \otimes \\
& \quad copy'(z,x,b) \,\otimes\, protect(value(x,b))\,)\,)\,\mid \\
& \;\; \odot\,\,(\,regress(copy(x,y,a), copy'(x,y,a)) \otimes \\
& \quad\; copy'(x,y,a) \,\otimes\, protect(value(y,a))\,)\,) \\
& \otimes\,(value(x,b) \,\wedge value(y,a)).
\end{array}$$

where $\mathbf{D}_3^{\cancel{lock}} = \mathbf{D}_5^{\cancel{lock}}$*. Moreover,* $lock(value(x,a),1)$ *and* $lock(value(y,b),2)$ *is changed to* $lock(value(x,a),0)$ *and* $lock(value(y,b),1)$ *from* $\mathbf{D}_3^{lock}$ *to* $\mathbf{D}_5^{lock}$*. Note that*
$regress(copy(x,y,a), copy'(x,y,a))$ *cannot bind* $copy'(x,y,a)$ *to any action because currently* $value(y,b)$ *is an achieved goal and* $\mathfrak{R}(copy(x,y,a), value(y,b)) = \emptyset$*. Then* $regress(copy(x,y,a), copy'(x,y,a))$ *fails at this point. One can continue the rest of inference as shown in Example 4. Clearly, with similar application of inference rule in Example 4, the inference can proceed without failure and backtracking because of the lack of regression analysis and* $regress(copy(x,y,a),$

$copy'(x, y, a))$. *Therefore, the failure and backtracking could be postponed which increases the execution time of planning algorithm. This simple example illustrates how regression analysis can provide a more efficient planning algorithm.* $\square$

In this chapter, we showed how a modification of the $\mathcal{TR}$-based *STRIPS* planner in Section 3.1 results in a more efficient planner. Chapter 7 shows that *STRIPS*^{R} can be two orders of magnitude faster than *STRIPS*.

Chapter 5

Planning Problems With Negative Intensional Literals

Recall from Chapter 2 that a literal is either an atom or a negated extensional atom, which excludes negated intensional atoms. In this chapter, we first extend the planning context with the negated intensional atoms. Then, we introduce *STRIPS*^{neg}, which is an extension of *STRIPS* planner in Section 3.1 with respect to negative intensional literals. We also prove the soundness and completeness of the new strategy with respect to the semantics of negative intensional literals.

5.1 Negative Intensional Literals

Consider an intensional atom of the form $p(t_1, ..., t_n)$, where $p \in \mathcal{P}_{int}$. An intensional literal of the form $\mathsf{naf}\ p(t_1, ..., t_n)$ is called a ***default negation of intensional literal***. Such negation is understood *not* in the first-order sense, but as *negation-as-failure*. These negated literals can appear in rule bodies only, not in rule heads.

Another form of negation is *explicit negation*, where ***explicit negation of an intensional literal*** is an atom of the form $\neg p(t_1, ..., t_n)$. These literals are understood as new positive predicates and correspond to the notion of classical negation introduced in [39]. For the semantics of the planning states that allow both default and explicit negation, we use well-founded semantics with explicit negation as defined in [3]. This semantics implies that, for every intensional atom of the form $p(t_1, ..., t_n)$, the following holds:

$$\neg p(t_1, ..., t_n) \models \mathsf{naf}\ p(t_1, ..., t_n) \tag{5.1}$$

Moreover, we assume that the background theory contains a rule of the form

$$\neg p(t_1, ..., t_n) \leftarrow \mathsf{naf}\ p(t_1, ..., t_n) \tag{5.2}$$

This makes $\neg p(t_1, ..., t_n)$ equivalent to $\mathsf{naf}\ p(t_1, ..., t_n)$ in every well-founded model. Therefore, we can replace every occurrence of $\mathsf{naf}\ p$ with $\neg p$ and in the sequel we will only deal with negative literals of the latter form. We now modify the definition of planning states accordingly.

Definition 15 (State – with negative intensional literals)**.** *Given a set $\mathbb{R}$ of rules (possibly with negated literals in the bodies) and a set of extensional literals $\mathbf{S}_{ext}$, a **state** is a set of literals $\mathbf{S} = \mathbf{S}_{ext} \cup \mathbf{S}_{int}$ such that*

1. *$\mathbf{S}$ is the well-founded model of $\mathbb{R} \cup \mathbf{S}_{ext}$.*
2. *For every literal ℓ, either $\ell \in \mathbf{S}$ or $\neg\ell \in \mathbf{S}$.* □

Since a state is a model of $\mathbb{R}$, every rule of $\mathbb{R}$ is true in $\mathbf{S}$. In addition to the above, we assume that negative literals can appear in facts and in rule bodies of $\mathbb{R}$, but not in the rule heads in $\mathbb{R}$.

5.2 The *STRIPS*^{neg} Planner

In this section, first we extend $\mathcal{TR}$-based *STRIPS* planner introduced in Section 3.1 to achieve negative intensional literals. Then, we prove the soundness and completeness of the extended planner with respect to the semantics of the set of rules that defines intensional atoms.

Definition 16 ($\mathcal{TR}$ planning rules for *STRIPS*^{neg})**.** *Let $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{S} \rangle$ be a STRIPS planning problem and $\mathbb{P}(\Pi)$ be a set of $\mathcal{TR}$ planning rules, as in Definition 10. We define $\mathbb{P}^{neg}(\Pi)$ to be exactly as $\mathbb{P}(\Pi)$ except for the $\mathbb{P}_{\mathbb{R}}$ part. Namely, $\mathbb{P}^{neg}_{\mathbb{R}}$—the replacement of $\mathbb{P}_{\mathbb{R}}$— has two disjoint parts, $\mathbb{P}^{neg}_{positive}$ and $\mathbb{P}^{neg}_{negative}$, and is constructed as follows:*

- *The $\mathbb{P}^{neg}_{positive}$ part: for each rule $p(\overline{X}) \leftarrow p_1(\overline{X}_1) \wedge \cdots \wedge p_n(\overline{X}_n)$ in $\mathbb{R}$, $\mathbb{P}^{neg}_{positive}$ has a rule of the form*

$$achieve_p(\overline{X}) \leftarrow \|_{i=1}^{n} achieve_{p_i}(\overline{X}_i). \tag{P15}$$

This rule is identical to Rule (P1) in Definition 10.

- $\mathbb{P}^{neg}_{negative} = \mathbb{P}^{neg}_{rules} \cup \mathbb{P}^{neg}_{literals}$ *has two disjoint parts as follows:*
  - $\mathbb{P}^{neg}_{rules}$*: For each positive intensional literal* ℓ*, let* $r_1, \ldots, r_k \in \mathbb{R}$ *be all the rules that have* ℓ *in their heads. For each such rule* r_i *of the form* $p(\overline{X}) \leftarrow p_1(\overline{X}_1) \wedge \cdots \wedge p_n(\overline{X}_n)$*,* $\mathbb{P}^{neg}_{rules}$ *has* n *rules of the form*

$$achieve_{disable\_r_i}(\overline{X}) \leftarrow achieve_{\neg p_j}(\overline{X_j}). \qquad \text{(P16)}$$

    where r_i *is a new predicate. These rules say that, to disable rule* r_i *we need to achieve the negation of one of* r_i*'s body literals.*
  - $\mathbb{P}^{neg}_{literals}$*: for each intensional predicate* $p \in \mathcal{P}_{int}$*,* $\mathbb{P}^{neg}_{literals}$ *has a rule of the form:*

$$achieve_{\neg p}(\overline{X}) \leftarrow \|_{i=1}^{n} achieve_{disable\_r_i}(\overline{X}). \qquad \text{(P17)}$$

    where $\{r_1, \ldots, r_n\} \subseteq \mathbb{R}$ *are all the rules that have* $p(\overline{X})$ *in their heads. This rule says that, to achieve a negative intensional literal* $\neg p(\overline{X})$*, all of the rules that can possibly make* $p(\overline{X})$ *true must be disabled.* □

To construct a plan, we extract a pivoting sequence of actions with respect to *STRIPS*^{neg}, as before, and show that this sequence of actions is a solution plan. The following lemma shows that Lemma 4 is still valid under *STRIPS*^{neg}. Similarly to Section 3.1, we assume till the end of this section that $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{D}_0 \rangle$ is a *STRIPS* planning problem, $\mathbb{P}(\Pi)$ is the set of planning rules in Definition 10, and $\mathbb{P}^{neg}(\Pi)$ is the set of planning rules from Definition 16.

Lemma 17 (Execution of transactions of the form p_α in *STRIPS*^{neg})**.** *Let* $seq_0, \ldots, seq_m$ *be a derivation of* $\mathbb{P}^{neg}(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_m \vdash achieve_G$*, and* $\alpha_1, \ldots, \alpha_n$ *(resp.,* $\mathbf{D}_{i_1}, \ldots, \mathbf{D}_{i_n}$*) be a pivoting sequence of actions (resp., states). Then:*

- $\mathbf{D}_0 = \mathbf{D}_{i_1}$
- $\mathbb{P}^{neg}(\Pi), \mathbf{D}_{i_r} \ldots \mathbf{D}_{i_{r+1}} \vdash p_{\alpha_r}$*, where* $r = 1, ..., n$
  (recall that p_{α_r} *is the* $\mathcal{TR}$ *transaction associated with action* α_r*—see Definition 9)*
- $\mathbf{D}_{i_{n+1}} = \mathbf{D}_m$.

Proof. Since none of the rules added to *STRIPS* by *STRIPS*^{neg} (rules (P17) and (P16)) have an elementary update in their bodies, the proof is identical to that of Lemma 4, where $\mathbb{P}(\Pi)$ is replaced with $\mathbb{P}^{neg}(\Pi)$. □

Theorem 8 (Soundness of *STRIPS*^{neg})**.** *Any pivoting sequence of actions in the derivation of* $\mathbb{P}^{neg}(\Pi), \mathbf{D}_0 \ldots \mathbf{D}_{f'} \vdash achieve_G$ *is a solution plan.*

Proof. Since Lemmas 3 and 4 hold for *STRIPS*^{neg}, the proof of this theorem is almost identical to that of Theorem 1. □

Definition 17 (Ground-non-recursive set of rules)**.** *A set of rules* $\mathbb{R}$ *is said to be* **ground-non-recursive** *if and only if the set* $ground(\mathbb{R})$ *of ground instances of* $\mathbb{R}$ *is non-recursive (viewed as propositional rules). In other words, there is a partial order on the ground literals of* $ground(\mathbb{R})$ *such that*

1. $\ell < \neg\ell$ *for any positive literal* ℓ *in* $ground(\mathbb{R})$*; and*

2. *For every rule* $\ell \leftarrow \cdots \wedge \ell' \wedge \cdots \in ground(\mathbb{R})$*, we have* $\ell' < \ell$*.* □

It is easy to see that any ground-non-recursive set of rules is locally stratified [76]. From now on, we deal only with planning problems $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{S} \rangle$ where $\mathbb{R}$ is ground-non-recursive. With this assumption, we will be able to show that our planner for *STRIPS*^{neg} is not only sound but also complete.

Definition 18 (Negation-eliminated form of rules)**.** *The* **negation-eliminated form** $\mathcal{N}(\mathbb{R})$ *of a set of rules* $\mathbb{R}$ *is defined as follows:*

– *For each positive intensional literal* ℓ*, let* $r_1, \ldots, r_k \in \mathbb{R}$ *be the set of all rules that have* ℓ *in the rule heads. Then, for each such* ℓ*,* $\mathcal{N}(\mathbb{R})$ *has the rule*

$$\neg\ell \leftarrow \wedge_{i=1}^{n} disable\_r_i. \tag{5.3}$$

where $\neg\ell$ *and* $disable\_r_i$ *are new positive literals and propositions, respectively.*

– *For every rule* $r_i \in \mathbb{R}$ *of the form* $\ell \leftarrow \ell_1 \wedge \cdots \wedge \ell_n$*,* $r_i \in \mathcal{N}(\mathbb{R})$ *and* $\mathcal{N}(\mathbb{R})$ *has* n *rules of the form:*

$$disable\_r_i \leftarrow \neg\ell_j. \tag{5.4}$$

where $1 \leq j \leq n$*.* □

Recall that, by assumption, $\mathbb{R}$ does not have negative literals in the rule heads. However, $\mathcal{N}(\mathbb{R})$ reintroduces such rules in (5.3).

Lemma 18 (Soundness of negation elimination)**.** *Consider a set of rules $\mathbb{R}$ and a set of ground extensional literals $\mathbf{S}_{ext}$. Let ℓ be a ground positive intensional literal that appears in the head of some rules in $\mathbb{R}$. Then, the following holds:*

- $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models \ell$ *if and only if* $\mathbb{R}, \mathbf{S}_{ext} \models \ell$.
- $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models \neg\ell$ *if and only if* $\mathbb{R}, \mathbf{S}_{ext} \not\models \ell$.

Proof. Consider a total order of ground literals of $ground(\mathbb{R})$ obtained a topological sort of the partial order of Definition 17. The induction is on the number of elements in that total order.

For any literal ℓ_i, let $s(\ell_i)$ be the index of ℓ_i in the aforesaid total order. Suppose ℓ appears in the head of the ground rule instances R_1,...,R_m from $\mathbb{R}$ and let $Body(R_i)$ denote the set of ground literals in the body of R_i. In each part of the inductive proof, first we show that $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models \ell$ if and only if $\mathbb{R}, \mathbf{S}_{ext} \models \ell$. Then, we will prove that $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models \neg\ell$ if and only if $\mathbb{R}, \mathbf{S}_{ext} \not\models \ell$.

Base case: $s(\ell) = 1$. By definition of our order, each $Body(R_i)$ contains only extensional literals. Since, by construction of $\mathcal{N}(\mathbb{R})$, $\{R_1, \ldots, R_m\} \subseteq ground(\mathcal{N}(\mathbb{R}))$ and R_1,...,R_m are the only ground rule instances in both $\mathbb{R}$ and $\mathcal{N}(\mathbb{R})$ that have ℓ in the rule heads, then $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models \ell$ if and only if $\mathbb{R}, \mathbf{S}_{ext} \models \ell$.

By definition of negative intensional literals, $\mathbb{R}, \mathbf{S}_{ext} \not\models \ell$ entails that, for $1 \leq i \leq m$, there must be at least a ground extensional literal $b \in Body(R_i)$ such that $\neg b \in \mathbf{S}_{ext}$. If not, $\mathbb{R}, \mathbf{S}_{ext} \models \ell$ as R_i is assumed to be true. Since $\neg b \in \mathbf{S}_{ext}$, by a rule of the form $disable\_R_i \leftarrow \neg b$, we have $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models disable\_R_i$. But then, $\neg\ell \leftarrow \wedge_{i=1}^{n} disable\_R_i$ implies $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models \neg\ell$.

For the other direction, let $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models \neg\ell$. Note that $\neg\ell$ occurs only in the head of the rule $\neg\ell \leftarrow \wedge_{i=1}^{n} disable\_R_i$ in $\mathcal{N}(\mathbb{R})$, so then we must have $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models disable\_R_i$, where $i = 1, ..., m$. Since $s(\ell) = 1$, $Body(R_i)$ contains only extensional literals. By construction of $\mathcal{N}(\mathbb{R})$, the rules of the form $disable\_R_i \leftarrow \neg b_i$ are the only rules that have $disable\_R_i$ in their heads, where $b_i \in Body(R_i)$. Thus, there must be an extensional literal $b_i \in Body(R_i)$ such that $\neg b_i \in \mathbf{S}_{ext}$. Otherwise, $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \not\models disable\_R_i$. Therefore, the bodies of all $\mathbb{R}_i$'s are false and thus $\mathbb{R}, \mathbf{S}_{ext} \not\models \ell$.

Inductive case: $s(\ell) = n + 1$ and we assume that, for every ground positive intensional literal ℓ_j such that $s(\ell_j) \leq n$, $\mathbb{R}, \mathbf{S}_{ext} \models \ell_j$ if and only if $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models \ell_j$, and $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models \neg\ell_j$ if and only if $\mathbb{R}, \mathbf{S}_{ext} \not\models \ell_j$.

For every $b \in Body(R_i)$, where $1 \leq i \leq m$, by Definition 17, we have $s(b) < s(\ell) = n + 1$. By the inductive assumption, $\mathbb{R}, \mathbf{S}_{ext} \models b$ if and only if

$\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models b$. Since, by construction of $\mathcal{N}(\mathbb{R})$, $\{R_1, \ldots, R_m\} \subseteq ground(\mathcal{N}(\mathbb{R}))$ and R_1,...,R_m are the only ground rule instances in both $\mathbb{R}$ and $\mathcal{N}(\mathbb{R})$ that have ℓ in the rule heads, it follows that $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models \ell$ if and only if $\mathbb{R}, \mathbf{S}_{ext} \models \ell$.

If $\mathbb{R}, \mathbf{S}_{ext} \not\models \ell$, then, for $1 \leq i \leq m$, there must be a literal $b \in Body(R_i)$ such that $\mathbb{R}, \mathbf{S}_{ext} \not\models b$. If not, $\mathbb{R}, \mathbf{S}_{ext} \models \ell$ as R_i is assumed to be true. By the definition of the order, $s(b) < s(\ell) = n+1$. So, by the inductive assumption, $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models \neg b$. Since, by construction of $\mathcal{N}(\mathbb{R})$, there is a rule of the form $disable\_R_i \leftarrow \neg b$, we have $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models disable\_R_i$. Then, $\neg \ell \leftarrow \wedge_{i=1}^{n} disable\_R_i$ entails $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models \neg \ell$.

For the other direction, consider the rule of the form $\neg \ell \leftarrow \wedge_{i=1}^{n} disable\_R_i$. Again, this is the only rule that has $\neg \ell$ in its head. Thus, to have $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models \neg \ell$, we must have $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models disable\_R_i$ for each $i = 1, ...m$. Since $disable\_R_i$ appears only in the heads of the rules of the form $disable\_R_i \leftarrow \neg b_i$, where $b_i \in Body(R_i)$, to have $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models disable\_R_i$, there must be a literal b_i such that $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models \neg b_i$. Since $b_i \in Body(R_i)$, as before, we must have $s(b) < s(\ell) = n+1$ and, by the inductive assumption, $\mathcal{N}(\mathbb{R}), \mathbf{S}_{ext} \models \neg b_i$ entails $\mathbb{R}, \mathbf{S}_{ext} \not\models b_i$. Since, none of the rules R_1,...,R_m entails $\mathbb{R}, \mathbf{S}_{ext} \models \ell$, and thus $\mathbb{R}, \mathbf{S}_{ext} \not\models \ell$. $\square$

Lemma 19 (Transforming negation eliminated form into planning rules)**.** *Given a planning problem* $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{D}_0 \rangle$*, let* $\Pi^{neg} = \langle \mathcal{N}(\mathbb{R}), \mathbb{A}, G, \mathbf{D}_0 \rangle$ *. Let* $\mathbb{P}(\Pi^{neg})$ *and* $\mathbb{P}^{neg}(\Pi)$ *be the sets of* $\mathcal{TR}$ *planning rules in Definition 10 and Definition 16, respectively. Then,* $\mathbb{P}(\Pi^{neg}) = \mathbb{P}^{neg}(\Pi)$.

Proof. An examination of the $\mathcal{TR}$ planning rules in Definitions 10 and 16 shows that $\mathbb{P}(\Pi)$ and $\mathbb{P}^{neg}(\Pi)$ are identical, except the planning rules obtained from $\mathbb{R}$, i.e., the rules by (P15), (P16), and (P17). Likewise, besides the rules added to $\mathbb{R}$ by Definition 18, Π and Π^{neg} are identical. Consequently, the planning rules obtained from $\mathbb{R}$ are the only difference between $\mathbb{P}(\Pi)$ and $\mathbb{P}(\Pi^{neg})$. To prove that $\mathbb{P}(\Pi^{neg}) = \mathbb{P}^{neg}(\Pi)$, we just need to show that the $\mathcal{TR}$ planning rules obtained from $\mathbb{R}$ in $\mathbb{P}(\Pi^{neg})$ and $\mathbb{P}^{neg}(\Pi)$ are identical.

Let h be a positive intensional literal that appears in the head of rules $r_1, \ldots, r_m \in \mathbb{R}$, where each r_i has the form $h \leftarrow \ell_{i_1} \wedge \cdots \wedge \ell_{i_{n_i}}$. Corresponding to each such rule, Definition 18 introduces a rule of the form $\neg \ell \leftarrow \wedge_{i=1}^{n} disable\_r_i$ to $\mathcal{N}(\mathbb{R})$, where $disable\_r_1, \ldots, disable\_r_m$ are new positive intensional literals. Therefore, $\mathbb{P}(\Pi^{neg})$ has a planning rule of the form

$$achieve_{\neg h} \leftarrow \|_{i=1}^{m} achieve_{disable\_r_i} \tag{5.5}$$

Also, by (5.3), for each rule r_i, $\mathcal{N}(\mathbb{R})$ has the following rules:

$$disable\_r_i \leftarrow \neg \ell_{i_j}. \tag{5.6}$$

where $1 \leq j \leq i_{n_j}$. By construction of $\mathbb{P}(\Pi^{neg})$, each of these gives rise to to a planning rule of the form

$$achieve_{disable\_r_i} \leftarrow achieve_{\neg\ell_{i_j}}. \tag{5.7}$$

By (P17), $\mathbb{P}^{neg}(\Pi)$ has the rule (5.5), and by (P16), it has also the rules (5.7). Thus, $\mathbb{P}(\Pi^{neg}) = \mathbb{P}^{neg}(\Pi)$. □

Theorem 9 (Completeness of *STRIPS*^{neg})**.** *Consider a planning problem* $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{D}_0 \rangle$*, where* $\mathbb{R}$ *is a set of ground-non-recursive rules and let* $\mathbb{P}^{neg}(\Pi)$ *be the set of* $\mathcal{TR}$ *planning rules in Definition (16). If there is a plan to achieve the goal* G *from the initial state* $\mathbf{D}_0$*, then, using* $\mathbb{P}^{neg}(\Pi)$*, the* $\mathcal{TR}$ *inference system will find that plan.*

Proof. Let σ be a non-redundant plan of Π. We need to show that there is a $\mathcal{TR}$ derivation of $\mathbb{P}^{neg}(\Pi), \mathbf{D}_0 \cdots \vdash achieve_G$ whose pivoting sequence of actions is σ. Consider a planning problem $\Pi^{neg} = \langle \mathcal{N}(\mathbb{R}), \mathbb{A}, G, \mathbf{D}_0 \rangle$ and let $\mathbb{P}(\Pi^{neg})$ be the set of $\mathcal{TR}$ planning rules in Definition 10. Clearly, σ is also a non-redundant plan of Π^{neg}. Then, by Theorem 2, there is a $\mathcal{TR}$ derivation for $\mathbb{P}(\Pi^{neg}), \mathbf{D}_0 \cdots \vdash achieve_G$ whose pivoting sequence of actions is σ. By Lemma 18, in every state appearing in the $\mathcal{TR}$ derivation of $\mathbb{P}(\Pi^{neg}), \mathbf{D}_0 \cdots \vdash achieve_G$, we can replace every literal of the form $\neg\ell$ with $\neg\ell$, where ℓ is a positive intensional literal. By Lemma 19, we also have $\mathbb{P}(\Pi^{neg}) = \mathbb{P}^{neg}(\Pi)$. Thus, if we replace $\mathbb{P}(\Pi^{neg})$ in the derivation of $\mathbb{P}(\Pi^{neg}), \mathbf{D}_0 \cdots \vdash achieve_G$, we will get a derivation for $\mathbb{P}^{neg}(\Pi), \mathbf{D}_0 \cdots \vdash achieve_G$ with the same pivoting sequence of actions. □

Chapter 6

The $\mathcal{TR}^*\mathcal{G}$*raphPlan* Planner

In this chapter, we encode the $\mathcal{G}$*raphPlan* planning algorithm [16]—one of the leading neoclassical planners—as a set of $\mathcal{TR}$ planning rules. Since $\mathcal{G}$*raphPlan* algorithm uses a planning graph to guide its search mechanism, we will first explain the planning graph. Then, we explain $\mathcal{TR}^*\mathcal{G}$*raphPlan* planning algorithm, which is the representation of the well-known $\mathcal{G}$*raphPlan* planning algorithm [16] in $\mathcal{TR}$. Since, in the original $\mathcal{G}$*raphPlan*, all predicates were extensional, we also disallow the intensional atoms and rules in the specifications of planning problems. Extending $\mathcal{TR}^*\mathcal{G}$*raphPlan* with rules is a subject for future work. Thus, from now on, in every planning problem $\Pi = \langle \mathbb{R}, \mathbb{A}, G, \mathbf{S}\rangle$, the set of planning rules $\mathbb{R}$ will be replaced with $\emptyset$.

Similar to $\mathcal{G}$*raphPlan*, $\mathcal{TR}^*\mathcal{G}$*raphPlan* is a two-step planning algorithm whose main contribution is using compact reachability graph to reduce the search space. In the first step, $\mathcal{G}$*raphPlan* builds a graph, called the ***planning graph*** which reflects the relationship between possible actions in different steps of a solution plan. The graph is of polynomial size and can be built in polynomial time in the size of the input. Since building such graphs is fairly straightforward in logic programming, we omit this step in order to avoid distraction.

To simplify the development of planning graph, for every literal ℓ, $\mathcal{G}$*raphPlan* introduces a neutral action $no\_op_\ell = \langle no\_op_\ell, \{\ell\}, \{\ell\}\rangle$, called ***no-op of*** ℓ and assumes that $no\_op_\ell \in \mathbb{A}$ for every literal ℓ. The construction of planning graph relies on the existence of these no-op actions.

Definition 19 (Planning graph)**.** *Given a planning problem* $\Pi = \langle \emptyset, \mathbb{A}, G, \mathbf{S}\rangle$ *(see Definition 4), a planning graph for* Π *expanded to level* i*, denoted* $\mathcal{G}(\Pi, i)$*, is an undirected graph whose nodes are literals, labeled with* $0, 1, 2, ..., i$*, and actions,* *labeled with $1, 2, ..., i$. Let L_j denote the set of all of the literals labeled with j, where $0 \leq j \leq i$, and A_k be the set of all of the actions labeled with k, where $1 \leq k \leq i$. Then, $\mathcal{G}(\Pi, i)$ forms a sequence of levels of nodes $\langle L_0, A_1, L_1 \ldots, A_i, L_i \rangle$, where $L_0 = \mathbf{S}$ and, for every $1 \leq j \leq i$, L_j and A_j are constructed as follows:*

- *For every $\alpha \in \mathbb{A}$, if $Pre_\alpha \subseteq L_{j-1}$, then $\alpha \in A_j$, represented by a fact of the form $g_{act}(\alpha, j)$. Moreover, there is a* **precondition** *edge between $\ell \in L_{j-1}$ and $\alpha \in A_j$, represented by a fact of the form $g_{pre}(\ell, \alpha, j)$, if and only if $\ell \in Pre_\alpha$.*

- *$L_j = \bigcup\limits_{\alpha \in A_j} E_\alpha$, where E_α denotes the effects of α. Every node $\ell \in L_j$ is represented by a fact of the form $g_{lit}(\alpha, j)$. In addition, there is an* **effect** *edge between $\alpha \in A_j$ and $\ell \in L_j$, represented by a fact of the form $g_{eff}(\alpha, \ell, j)$, if and only if $\ell \in E_\alpha$. Observe that $L_{j-1} \subseteq L_j$ because A_j also contains no-op actions.*

- *Every level also contains* **mutex** *edges that are constructed as follows:*

  – *For every $\alpha, \beta \in A_j$, A_j contains a* **mutex** *edge between α and β, represented by the fact of the form $g_{mut}(\alpha, \beta, j)$, if one of the following conditions holds:*

    ∗ *$\ell \in Pre_\alpha$ and $\neg\ell \in E_\beta$, for some ℓ;*

    ∗ *$\ell \in Pre_\beta$ and $\neg\ell \in E_\alpha$, for some ℓ;*

    ∗ *$\ell \in E_\alpha$ and $\neg\ell \in E_\beta$, for some ℓ;*

    ∗ *$\ell_1 \in Pre_\alpha$, $\ell_2 \in Pre_\beta$ and $g_{mut}(\ell_1, \ell_2, j-1)$, for some ℓ_1, ℓ_2.*

  – *For every $\ell_1, \ell_2 \in L_j$, L_j also has a* **mutex** *edge between ℓ_1 and ℓ_2, represented by the fact of the form $g_{mut}(\ell_1, \ell_2, j)$, if any of the following holds:*

    ∗ *$\ell_1 = \neg\ell_2$;*

    ∗ *For every pair of actions $\alpha, \beta \in A_j$ it is the case that: $\ell_1 \in E_\alpha$ and $\ell_2 \in E_\beta$, the fact $g_{mut}(\alpha, \beta, j)$ holds, and there is no $\gamma \in A_j$ such that $\{\ell_1, \ell_2\} \subseteq E_\gamma$. Both, α and β can be no-op actions. Without no-op actions, the above condition would have produced false mutexes, pairs of literals that should not be considered mutexes.*

$\square$

Given a planning problem, $\mathcal{G}$*raphPlan* finds a sequence of sets of actions as a solution plan. For example, it might output $\langle \sigma_1, \sigma_2, \sigma_3 \rangle$, where $\sigma_1 = \{\alpha_1, \alpha_2\}$, $\sigma_2 = \{\alpha_3, \alpha_4\}$, and $\sigma_3 = \{\alpha_5, \alpha_6, \alpha_7\}$. Such sequence represents sequences starting with α_1 and α_2 in any order, followed by α_3 and α_4 in any order, followed by α_5, α_6, and α_7 in any order. To extract such a sequence of sets out of the planning graph in Definition 19, $\mathcal{G}$*raphPlan* proposes the concept of independent sets of actions. For instance, α_3 and α_4 are independent actions as they can appear in any order after α_1 and α_2 in the above solution plan. Mutex edges in a planning graph reflect the independence between actions (resp. literals) to guide $\mathcal{G}$*raphPlan*'s search procedure and reduce the search space. As shown in [17], planning graph is finite. The following example illustrates Definition 19.

Example 7 (Planning graph for register exchange)**.** *Recall the register exchange problem of Example 3 in Chapter 3. Consider a subset of that problem where there are two memory registers, x and y, with initial contents a and b, respectively. Let the extensional predicate $v(Reg, Val)$ represent the content of a register and suppose the only available action is $cp = \langle cp(S, D, V), \{v(S, V)\}, \{\neg v(D, V'), v(D, V)\}\rangle$, which copies the value V of the source register, S, to the destination register D. The planning graph for this problem has seven levels. The first three levels of the graph have been shown in Figure 6.1 while the subsequent levels are too big to be included in this dissertation.* □

The $\mathcal{G}$*raphPlan*'s search procedure looks for a solution plan in a planning graph by proceeding back from some level L_i that includes all goal-literals and the goal is mutex-free, i.e.,

$$G \subseteq L_i \quad and\ for\ every\ pair\ \ g_1, g_2 \in G,\ \ g_{mut}(g_1, g_2, i) \notin L_i \tag{6.1}$$

The search procedure then looks for a set $\sigma_i \subseteq A_i$ of nonmutex actions that achieve literals of G. The preconditions of the actions in σ_i thus become the new goal for level $i-1$ and so on. In fact, at each level j, the search procedure looks for a set of independent actions from A_j that ***achieve*** a set of nonmutex ***to-be-achieved*** literals in level j. Failure to achieve the currently active goal at some level j causes backtracking and selection of other subsets of A_{j+1}. If level 0 is successfully reached, then the corresponding sequence $\langle \sigma_1, \ldots, \sigma_i \rangle$ is a solution plan [72].

Next, we introduce a set of $\mathcal{TR}$ rules, $\mathcal{TR}^*\mathcal{G}$*raphPlan*, that simulate the search mechanism of $\mathcal{G}$*raphPlan*. At each level, to track the literals that are expected to be achieved, we introduce a built-in extensional predicate of the form $achieve_\ell(j)$, which indicates that $\ell \in L_j$ is expected to be achieved by $\mathcal{TR}^*\mathcal{G}$*raphPlan*.

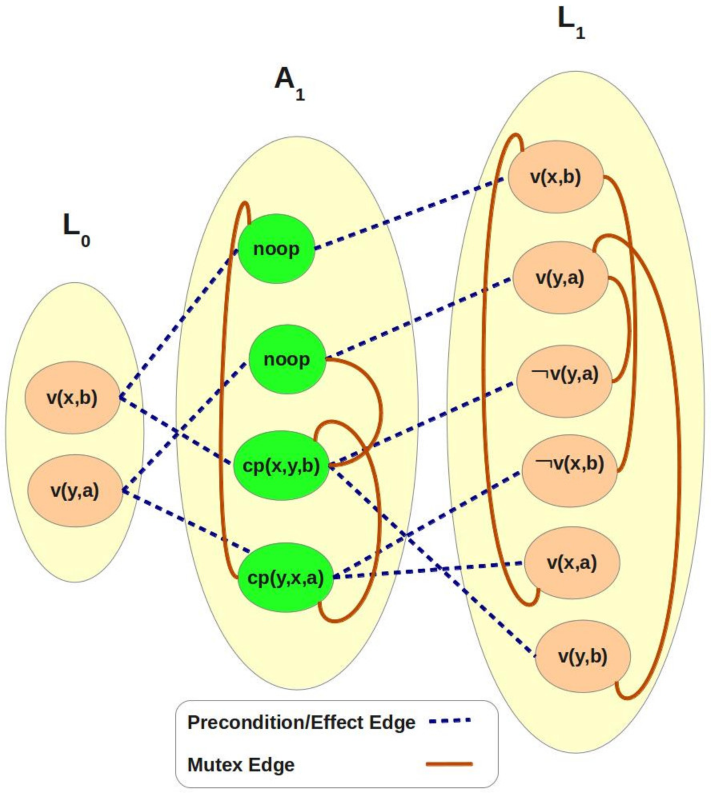


Figure 6.1: The first three levels of planning graph of Example 7.

Definition 20 ($\mathcal{TR}^*\mathcal{G}raphPlan$ rules)**.** *Given a planning problem* $\Pi = \langle \emptyset, \mathbb{A}, G, \mathbf{S} \rangle$ *(see Definition 4), let* $\mathcal{G}(\Pi, i)$ *be a planning graph of* Π *(see Definition 19). We define a set of* $\mathcal{TR}$ *rules,* $\mathbb{P}^g(\Pi)$*, which provides a solution to* Π*.* $\mathbb{P}^g(\Pi)$ *has three parts,* $\mathbb{P}^g_M$*,* $\mathbb{P}^g_{\mathbb{A}}$*, and* $\mathbb{P}^g_G$ *described below.*

- *The* $\mathbb{P}^g_M$ *part contains the following rules:*

$$\begin{aligned} plan(0) \leftarrow&\ \ . \\ plan(I) \leftarrow&\ \ choose\_act(I) \otimes \textsf{naf}\ incomplete(I) \otimes plan(I-1). \\ plan(I) \leftarrow&\ \ choose\_act(I) \otimes incomplete(I) \otimes plan(I). \end{aligned} \tag{P18}$$

  These rules construct a sequence of sets of independent actions based on the provided planning graph.

- *The* $\mathbb{P}^g_{\mathbb{A}} = \mathbb{P}^g_{incomplete} \cup \mathbb{P}^g_{choose}$ *has two disjoint parts as follows:*

  – $\mathbb{P}^g_{incomplete}$*: for every literal* ℓ*,* $\mathbb{P}^g_{incomplete}$ *has a rule of the form*

$$incomplete(I) \leftarrow achieve_\ell(I). \tag{P19}$$

  This rule says that $\mathcal{TR}^*\mathcal{G}raphPlan$ *has not completed planning at level* I *because there is an* ℓ*, that is yet-to-be-achieved at level* I*.*

  – $\mathbb{P}^g_{choose}$ *has a rule of the form*

$$\begin{aligned} choose\_act(I) \leftarrow&\ \ achieve_\ell(I) \otimes g_{eff}(\alpha, \ell, I) \otimes \\ &\ \ \textsf{naf}\ (\, chosen(\beta, I) \wedge g_{mut}(\alpha, \beta, I)\,) \otimes \\ &\ \ (\, \otimes_{e \in E_\alpha} - achieve_e(I)\,) \otimes \\ &\ \ (\, \otimes_{p \in Pre_\alpha} + achieve_p(I-1)\,) \otimes \\ &\ \ +chosen(\alpha, I). \end{aligned} \tag{P20}$$

  This rule is a natural verbalization of the mechanism of choosing actions in $\mathcal{G}raphPlan$ *algorithm [17]. This rule says that if there is a yet-to-be-achieved literal* ℓ *in level* I*,* $\mathcal{TR}^*\mathcal{G}raphPlan$ *chooses the action* α *from the* I*-th level of the planning graph such that* $\ell \in E_\alpha$ *and* α *is not conflicting with any other previously chosen action. Then, it removes all of the to-be-achieved literals in level* I *that appear in* E_α *and adds* Pre_α *to the to-be-achieved literals in level* $I-1$*. Finally, it marks* α *as a chosen action in the* I*-th level.*

- $\mathbb{P}^g_G = \mathbb{P}^g_{start} \cup \mathbb{P}^g_{search}$ *has two disjoint parts as follows:*

– $\mathbb{P}^g_{start}$ *is a rule of the form*

$$start\_level(I) \leftarrow \begin{array}{l} (\wedge_{g \in G}\ g_{lit}(g, I)) \ \wedge \\ (\wedge_{g_1, g_2 \in G}\ \neg g_{mut}(g_1, g_2, I)). \end{array} \tag{P21}$$

This rule checks if the I-th level of a planning graph can be considered as the starting point for planning (cf. the earlier condition (6.1) of $\mathcal{G}$raphPlan).

– $\mathbb{P}^g_{search}$ *is a rule of the form*

$$graphPlan_G \leftarrow \begin{array}{l} start\_level(I) \otimes \\ (\otimes_{g \in G}\ +achieve_g(I)) \otimes plan(I). \end{array} \tag{P22}$$

This rule starts the backward search for a plan from the I-th level of the planning graph.

□

Given a set $\mathbb{A}$ of *STRIPS* actions, an initial state $\mathbf{S}$, and a goal G, Definition 19 gives a set of facts that specify a planning graph for that problem. Using that planning graph and the goal G, Definition 20 produces a set of $\mathcal{TR}$ rules that specify *$\mathcal{G}$raphPlan* planning strategy for that planning problem. To find a solution for that problem, one simply needs to place the request

$$?- graphPlan_G\,. \tag{6.2}$$

at the state formed by the set of facts of the planning graph and use the $\mathcal{TR}$'s inference system of Section 2.3.

As mentioned before, *$\mathcal{G}$raphPlan* outputs a sequence of sets of independent actions that can be used to produce a solution plan for the given planning problem. Such a sequence can be extracted by picking out the atoms of the form $chosen(\alpha, j)$, where $1 \leq j \leq i$, from a successful derivation branch generated by the $\mathcal{TR}$ inference system.

Suppose $seq_0, \ldots, seq_m$ is a deduction by the $\mathcal{TR}$ inference system. Let $\mathbf{D}_0$ and $\mathbf{D}_n$ be the first and last states of that derivation, respectively. Then, $\mathbf{D}_n$ contains literals of the form $chosen(\alpha, j)$, where $1 \leq j \leq i$ and $start\_level(i) \in \mathbf{D}_0$. For $1 \leq j \leq i$, let $\sigma_j = \{\alpha \mid chosen(\alpha, j) \in \mathbf{D}_n\}$. We will call $\langle \sigma_1, \ldots, \sigma_i \rangle$ the ***pivoting sequence of sets of independent actions***. It can be shown using the techniques of previous sections that the pivoting sequence of sets of independent actions is a solution produced *$\mathcal{G}$raphPlan*. Since *$\mathcal{G}$raphPlan* is sound [17], this means that $\mathcal{TR}^*$*$\mathcal{G}$raphPlan* is also sound. We believe that $\mathcal{TR}^*$*$\mathcal{G}$raphPlan* is also complete but proving this is a future work.

Chapter 7

Experiments

In this chapter we report on our experiments that compare *STRIPS*, *fSTRIPS*, *STRIPS*^{R}, and *STRIPS*^{neg}. The test environment was a $\mathcal{TR}$ interpreter [33] implemented in XSB and running on Intel®Xeon(R) CPU E5-1650 0 @ 3.20GHz 12 CPU and 64GB memory running under 64-bit Mint Linux 14.

All test cases are taken from [12] and represent so called *state modifying access control policies*. A typical use of such a policy is to determine if a particular access request (say, to play digital contents) should be granted. The first test case, a *Movie Store*, is shown in Example 8. The second test case, a *Health Care Authorization* example, is too large to be included here and can be found at *http://ewl.cewit.stonybrook.edu/planning/* along with the first test case and all the necessary items needed to reproduce the results.

Example 8 (State Modifying Policy for a Movie Store)**.** *The following represents a policy that regulates how users buy movies online, try them, and sell them, if not satisfied.*

$$
\begin{array}{rcl}
buy(X,M) & \leftarrow & \neg bought(\_,M) \otimes +bought(X,M) \\
play1(X,M) & \leftarrow & bought(X,M) \otimes \neg played1(X,M) \otimes +played1(X,M) \\
keep(X,M) & \leftarrow & bought(X,M) \otimes \neg played1(X,M) \otimes +played1(X,M) \\
 & & \otimes + happy(X,M) \\
play2(X,M) & \leftarrow & played1(X,M) \otimes \neg played2(X,M) \otimes +played2(X,M) \\
play3(X,M) & \leftarrow & played2(X,M) \otimes \neg played3(X,M) \otimes +played3(X,M) \\
sell(X,M) & \leftarrow & played1(X,M) \otimes \neg played3(X,M) \otimes \neg happy(X,M) \\
 & & \otimes + sold(X,M) \otimes -bought(X,M)
\end{array}
\tag{7.1}
$$

Size of goal	*STRIPS*		*fSTRIPS*	
	CPU	Mem.	CPU	Mem.
6	0.0160	1095	0.0080	503
9	0.2760	14936	0.1360	6713
12	9.4120	409293	5.8840	184726

Table 7.1: Results of different goal sizes (number of literals in the goals) for the movie store case. The initial state is fixed and has 6 extensional atoms.

Size of goal	*STRIPS*		*fSTRIPS*	
	CPU	Mem.	CPU	Mem.
3	10.0240	246520	0.0400	2011
4	32.9540	774824	0.2040	8647
5	46.1380	1060321	0.3080	13622

Table 7.2: Results of different goal sizes (number of literals in the goals) for the health-care case. The initial state is fixed and has 6 extensional atoms.

The first rule describes an action of a user, X, buying a movie, M. The action is possible only if the movie has not already been purchased by somebody. The second rule says that, to play a movie for the first time, the user must buy it first and not have played it before. The third rule deals with the case when the user is happy and decides to keep the movie. The remaining rules are self-explanatory. □

A reachability query in a state modifying policy is a specification of a *target state* (usually an undesirable state), and the administrator typically wants to check if such a state is reachable by a sequence of actions. The target state specification consists of a set of literals, and the reachability query is naturally expressed as a planning problem. For instance, in Example 8, the second rule can be seen as a *STRIPS* action whose precondition is $\{bought(X, M) \otimes \neg played1(X, M)\}$ and the effect is $\{+played1(X, M)\}$. The initial and the target states in this example are sets of facts that describe the movies that have been bought, sold, and played by various customers.

The main difference between the two test cases is that the Health Care example has many actions and intensional rules, while the movie store case has only six actions and no intensional predicates. As seen from Tables 7.1, 7.2, 7.3, and 7.4, for the relatively simple Movie Store example, *fSTRIPS* is about twice more ef-

initial state	*STRIPS*		*fSTRIPS*	
	CPU	Mem.	CPU	Mem.
20	9.2560	409293	5.8800	184726
30	9.2600	409293	5.7440	184726
40	9.2520	409293	5.8000	184726
50	9.4120	409293	5.8840	184726
60	9.3720	409293	5.8240	184726

Table 7.3: Results of different sizes (number of facts) in initial states for the movie store case. The planning goal is fixed: 6 extensional literals in the "movie store" case and 3 extensional literals in the "health care" case.

initial state	*STRIPS*		*fSTRIPS*	
	CPU	Mem.	CPU	Mem.
3	0.148	5875	0.012	718
6	10.076	246519	0.04	2011
9	689.3750	9791808	0.124	5443
12	>1000	N/A	0.348	14832
18	>1000	N/A	0.94	38810

Table 7.4: Results of different sizes (number of facts) in initial states for the health care case. The planning goal is fixed: 6 extensional literals in the "movie store" case and 3 extensional literals in the "health care" case.

ficient both in time and space.[1] However, in the more complex Health Care example, *fSTRIPS* beats *STRIPS* by more than two orders of magnitude—both time-wise and space-wise. While in the Movie Store example the statistics for the two strategies seem to grow at the same rate, in the Health Care case, the *fSTRIPS* time seems to grow at slower rate.

The main goal of the second set of experiments is to demonstrate the benefits of regression analysis in *STRIPS* planning. In describing the results, we use tables that compare two different implementations of *fSTRIPS*: with and without regression analysis. The improvement in the Movie Store example is not that pronounced, so we do not consider that example here. As shown in Tables 7.5 and 7.6, in the more complex Health Care example, *fSTRIPS* with regression analysis can be two orders of magnitude better both time-and space-wise. Table 7.5 indicates

[1]Time is measured in seconds and memory in kilobytes.

	w/o regression		with regression	
Size of goal	CPU	Mem.	CPU	Mem.
3	1.036	42250	0.16	5098
6	22.037	797019	3.324	116513
9	329.72	10755859	54.899	1862270
12	>5000	N/A	809.986	25003988

Table 7.5: Results of different goal sizes (number of literals in the goals) to show the effect of regression analysis. The initial state is fixed and has 15 extensional atoms.

	w/o regression		with regression	
Size of init. state	CPU	Mem.	CPU	Mem.
6	0.83	38000	0.984	35438
9	2.504	108664	1.668	56492
12	7.556	301403	2.376	80442
15	21.673	797009	3.36	116514
18	61.451	2056640	4.384	150446
21	172.97	5313468	5.576	188573
24	489.582	14047510	6.964	239397
27	>1000	N/A	8.676	314697
30	>1000	N/A	10.228	367817

Table 7.6: Results of different sizes (number of facts) in initial states. The planning goal is fixed: 6 extensional literals.

that both implementations seem to show growth at the same rate, but regression analysis still speeds up *fSTRIPS* by an order of magnitude. Table 7.6 shows an even stronger effect of regression analysis—again both in time and space. Example 6 in Section 4.2 explains how regression analysis accrues such an improvement. First, regression avoids un-achieving already achieved goals during planning. Second, it causes fewer backtracking and selects more promising actions at each step. Note that, for planning problems where the chance of un-achieving already achieved goals is small (e.g., when the initial state is small), regression analysis may cause some overhead and slow down the planning process. The overall conclusion from these experiments is that regression analysis incur relatively small overheads for small problems, but brings substantial savings for larger problems and makes them scale better.

Number of Blocks	*fSTRIPS*		*STRIPS*^{neg}	
	CPU	Mem	CPU	Mem
3	0.02	2442	0.004	462
4	0.512	35800	0.028	2324
5	65.264	1396968	0.164	15463
6	> 1000	N/A	1.488	173038
7	> 1000	N/A	27.405	3525475

Table 7.7: Results for the Blocks World test case.

The other difference between the two test cases is that the Health Care example has intensional rules, while the Movie Store case does not. Although the intensional rules in the Health Care example makes it a potential benchmark for our *STRIPS*^{neg} planning algorithm, we use Blocks World test case [4], a simpler example, to compare our *STRIPS*^{neg} and *fSTRIPS* planning algorithms.

The next test case compares *STRIPS*^{neg} and *fSTRIPS* using the well-known Blocks World example [4]. We choose Blocks World example for comparison of *STRIPS*^{neg} and *fSTRIPS* because the simplicity of Blocks World test case allows us to illustrate the impact of adding negative derived literals on the performance of planning algorithms. We modified this example so that some extensional literals become intensional, which allows us to reduce the size of specification. Namely, we replaced the extensional literal $handempty$ with the rule $handempty \leftarrow \neg holding(V)$ and also changed the extensional literal $clear(X)$ to a literal defined with the following rule:

$$clear(X) \leftarrow \neg holding(X) \otimes \neg on(Y, X).$$

As shown in Table 7.7, in the Blocks World example, using two intensional literals and *STRIPS*^{neg} can be two orders of magnitude faster than *fSTRIPS* without any intensional literals. The reason is that, replacing extensional atoms with intensional ones reduces not only the size of the problem but also the need to perform elementary update operations on predicates that used to be extensional and now are intensional. As a consequence, the state search space is also reduced.

Chapter 8

Discussion

In this chapter, first we briefly compare our technique with some of the recent logic based planning frameworks. Then, we draw a brief conclusion for the dissertation.

8.1 Related Works

Although several deductive planners have been developed over the last four decades, only few of them are still being developed and attracting attention of the planning community. This section briefly overviews those developing deductive planners to help the reader position our contribution in a proper spot in this area.

Answer set programming is one of the leading logic-based planning techniques [66][37]. The encoding of planning domains in answer set programs resembles the representation of planning domains in situation calculus [35, 36]. Although answer set programming has been shown to be able to compete with leading non-deductive planners, its encoding is not flexible enough for representing domain-independent heuristics provided by the AI planning community. In contrast, as shown in Chapters 3, 4, and 5, $\mathcal{TR}$-based planning is not only flexible enough to represent complex planning algorithms, but also expressive enough to facilitate developing novel planning heuristics. A number of search heuristics are embedded in answer set programming systems, which is largely responsible for the competitiveness of this approach. However, answer set programming is not flexible enough to be able to encode the different planning strategies developed by the mainstream in the planning field. This makes this technique less scalable and is partly responsible for the skepticism with which it is met by that mainstream. Also, unlike our $\mathcal{TR}$-based planning, it is unclear how to use answer set programming as an experimentation and prototyping framework.

Planning with satisfiability, called *SATPlan*, is another dominating approach among existing deductive planning techniques [59][60][61][58][44][30]. The main idea in SATPlan is to formulate a planning problem as a propositional satisfiability problem, i.e. as the problem of determining whether a propositional formula is satisfiable. Recent improvements in the performance of general purpose algorithms for propositional satisfiability enables SATPlan to solve relatively large planning problems. In fact, the ability to exploit the efforts and the results in a very active field of research in computer science is one of the main advantages of SATPlan. Despite such advantage, SATPlan has the same limitation as answer set programming. Namely, although there have been works to encode domain-*dependent* knowledge and heuristics [62][63], SATPlan still cannot represent domain-*independent* heuristics, which is the key advantage of our approach. Likewise, unlike our $\mathcal{TR}$-based planning formalism, SATPlan cannot be used as a tool for analyzing, testing, and prototyping of planning algorithms and domain-independent heuristics. The same limitations apply to planning techniques based on Constraint Satisfaction Problem (CSP) [84][83][31][54].

Picat is another logic programming framework that relies on forward state space search planning technique. Due to its novel tabling technique, Picat has been shown to be an efficient logic-based system for solving planning problems [7][88][6][87]. Picat's planner tables every state encountered during search to avoid repeating the exploration of the same state. It applies various optimizations, like hash-consing and non-tabled arguments, to avoid certain overheads associated with tabling [89]. The techniques employed by Picat are orthogonal to the results presented here and, we believe, this work provides a natural direction for incorporation of complex planning strategies, like *STRIPS* and *STRIPS*^{R}, into Picat. In addition, Picat utilizes tabled states to perform efficient resource-bounded search, which is also used to generate optimal plans. In this dissertation, we did not focus on resource-bounded and optimal planning. However, it is not too difficult to enhance our $\mathcal{TR}$-based planning techniques to enable it to solve resource-bounded and optimal planning problems.

8.2 Conclusion

This dissertation has demonstrated that the use of Transaction Logic accrues significant benefits in the area of planning, especially in showing the way for various generalizations, analyses, and optimization of planning strategies. It also provides a bridge between traditional mainstream planning and logic based planning.

We have shown that sophisticated planning strategies, such as *STRIPS* and $\mathcal{G}$*raphPlan*, can not only be naturally represented in $\mathcal{TR}$, but also such representations can yield several benefits. For instance, a $\mathcal{TR}$ representation of the *STRIPS* planning algorithm can be used to design the non-linear version of *STRIPS*, which turns out to be complete. We have also shown that the use of this powerful logic opens up new possibilities for generalizations and devising new, more efficient algorithms. As an example, we have shown that once the *STRIPS* algorithm is cast as a set of rules in $\mathcal{TR}$, the framework can be extended, almost for free, to support such advanced aspects as action ramification and negation.

By tweaking these rules just slightly, we obtained a new, much more efficient planner, which we dubbed *fSTRIPS* (fast *STRIPS*), which is also proven to be complete. In addition, we described a non-linear version of regression-based *STRIPS*, which we called *STRIPS*^{R}. This planner generalizes and improves upon *RSTRIPS*, the original regression-based off-shot of *STRIPS*. Again, unlike the original, our version is provably complete. Moreover, we extended the *STRIPS* planning strategy with negative intentional literals, which allows us to talk about negative ramifications of actions. Finally, we have also demonstrated the power of our technique by applying it to $\mathcal{G}$*raphPlan*, which is significantly different from *STRIPS*. We also conjecture that $\mathcal{TR}$ representation of $\mathcal{G}$*raphPlan* can also be extended with ramification and negation.

These non-trivial insights were acquired merely due to the use of $\mathcal{TR}$ and not much else. The same technique can be used to cast even more advanced strategies such as *ABSTRIPS* [73], and HTN [72] as $\mathcal{TR}$ rules, and the aforesaid optimizations and enhancements would still be applicable.

There are several promising directions to continue this work. One is to investigate other planning strategies and, hopefully, accrue similar benefits. Other possible directions include heuristics, plans with loops [56, 55, 82], planning with preferences [25, 5], plan failure analysis [53, 57], and assumption based planning [27, 1].

Bibliography

[1] Albore, A., Bertoli, P.: Generating safe assumption-based plans for partially observable, nondeterministic domains. In: McGuinness, D.L., Ferguson, G. (eds.) Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA. pp. 495–500. AAAI Press / The MIT Press (2004)

[2] Alcázar, V., Borrajo, D., Fernández, S., Fuentetaja, R.: Revisiting regression in planning. In: Rossi, F. (ed.) IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013. IJCAI/AAAI (2013)

[3] Alferes, J.J., Pereira, L.M., Przymusinski, T.C.: Strong and explicit negation in non-monotonic reasoning and logic programming. In: Proceedings of the European Workshop on Logics in Artificial Intelligence. pp. 143–163. JELIA '96, Springer-Verlag, London, UK, UK (1996)

[4] Bacchus, F.: AIPS-00 planning competition (May 2000)

[5] Baier, J.A., McIlraith, S.A.: Planning with preferences. AI Magazine 29(4), 25–36 (2008)

[6] Barták, R., Zhou, N.: On modeling planning problems: Experience from the petrobras challenge. In: Espinoza, F.C., Gelbukh, A.F., González-Mendoza, M. (eds.) Advances in Soft Computing and Its Applications - 12th Mexican International Conference on Artificial Intelligence, MICAI 2013, Mexico City, Mexico, November 24-30, 2013, Proceedings, Part II. Lecture Notes in Computer Science, vol. 8266, pp. 466–477. Springer (2013)

[7] Barták, R., Zhou, N.: Using tabled logic programming to solve the petrobras planning problem. TPLP 14(4-5), 697–710 (2014)

[8] Basseda, R.: Doctoral consortium extended abstract: Planning with concurrent transaction logic. In: Calimeri, F., Ianni, G., Truszczynski, M. (eds.) Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9345, pp. 545–551. Springer (2015)

[9] Basseda, R., Kifer, M.: Planning with regression analysis in transaction logic. In: ten Cate, B., Mileo, A. (eds.) Web Reasoning and Rule Systems - 9th International Conference, RR 2015, Berlin, Germany, August 4-5, 2015. Proceedings. Lecture Notes in Computer Science, vol. 8741, pp. 29–44. Springer (2015)

[10] Basseda, R., Kifer, M.: State space planning using transaction logic. In: Pontelli, E., Son, T.C. (eds.) Practical Aspects of Declarative Languages - 17th International Symposium, PADL 2015, Portland, OR, USA, June 18-19, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9131, pp. 17–33. Springer (2015)

[11] Basseda, R., Kifer, M., Bonner, A.J.: Planning with transaction logic. In: Kontchakov, R., Mugnier, M. (eds.) Web Reasoning and Rule Systems - 8th International Conference, RR 2014, Athens, Greece, September 15-17, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8741, pp. 29–44. Springer (2014)

[12] Becker, M.Y., Nanz, S.: A logic for state-modifying authorization policies. ACM Trans. Inf. Syst. Secur. 13(3), 20:1–20:28 (Jul 2010)

[13] Bibel, W.: A deductive solution for plan generation. New Generation Computing 4(2), 115–132 (1986)

[14] Bibel, W.: A deductive solution for plan generation. In: Schmidt, J.W., Thanos, C. (eds.) Foundations of Knowledge Base Management, pp. 453–473. Topics in Information Systems, Springer Berlin Heidelberg (1989)

[15] Bibel, W., del Cerro, L.F., Fronhfer, B., Herzig, A.: Plan generation by linear proofs: On semantics. In: Metzing, D. (ed.) GWAI-89 13th German Workshop on Artificial Intelligence, Informatik-Fachberichte, vol. 216, pp. 49–62. Springer Berlin Heidelberg (1989)

[16] Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artificial Intelligence 90(12), 281 – 300 (1997)

[17] Blum, A.L., Furst, M.L.: Fast planning through planning graph analysis. Artif. Intell. 90(1-2), 281–300 (Feb 1997)

[18] Bonet, B., Geffner, H.: Planning as heuristic search: New results. In: Biundo, S., Fox, M. (eds.) Recent Advances in AI Planning, Lecture Notes in Computer Science, vol. 1809, pp. 360–372. Springer Berlin Heidelberg (2000)

[19] Bonner, A., Kifer, M.: Transaction logic programming. In: Int'l Conference on Logic Programming. pp. 257–282. MIT Press, Budapest, Hungary (June 1993)

[20] Bonner, A., Kifer, M.: Applications of transaction logic to knowledge representation. In: Proceedings of the International Conference on Temporal Logic. pp. 67–81. No. 827 in Lecture Notes in Artificial Inteligence, Springer-Verlag, Bonn, Germany (July 1994)

[21] Bonner, A., Kifer, M.: Transaction logic programming (or a logic of declarative and procedural knowledge). Tech. Rep. CSRI-323, University of Toronto (November 1995), `http://www.cs.toronto.edu/~bonner/transaction-logic.html`

[22] Bonner, A., Kifer, M.: Concurrency and communication in transaction logic. In: Joint Int'l Conference and Symposium on Logic Programming. pp. 142–156. MIT Press, Bonn, Germany (September 1996)

[23] Bonner, A., Kifer, M.: A logic for programming database transactions. In: Chomicki, J., Saake, G. (eds.) Logics for Databases and Information Systems, chap. 5, pp. 117–166. Kluwer Academic Publishers (March 1998)

[24] Bonner, A.J., Kifer, M.: An overview of transaction logic. Theoretical Computer Science 133 (1994)

[25] Brafman, R.I., Chernyavsky, Y.: Planning with goal preferences and constraints. In: Biundo, S., Myers, K.L., Rajan, K. (eds.) Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA. pp. 182–191. AAAI (2005)

[26] Cresswell, S., Smaill, A., Richardson, J.: Deductive synthesis of recursive plans in linear logic. In: Biundo, S., Fox, M. (eds.) Recent Advances in AI Planning, Lecture Notes in Computer Science, vol. 1809, pp. 252–264. Springer Berlin Heidelberg (2000)

[27] Davis-Mendelow, S., Baier, J.A., McIlraith, S.A.: Assumption-based planning: Generating plans and explanations under incomplete knowledge. In: desJardins, M., Littman, M.L. (eds.) Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA. AAAI Press (2013)

[28] Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: A logic programming approach to knowledge-state planning, ii: The {DLVK} system. Artificial Intelligence 144(12), 157 – 211 (2003)

[29] Eiter, T., Faber, W., Leone, N., Pfeifer, G., Polleres, A.: A logic programming approach to knowledge-state planning: Semantics and complexity. ACM Trans. Comput. Log. 5(2), 206–263 (2004)

[30] Ernst, M.D., Millstein, T.D., Weld, D.S.: Automatic sat-compilation of planning problems. In: Proceedings of the Fifteenth International Joint Conference on Artifical Intelligence - Volume 2. pp. 1169–1176. IJCAI'97, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1997)

[31] Erol, K., Hendler, J.A., Nau, D.S.: UMCP: A sound and complete procedure for hierarchical task-network planning. In: Hammond, K.J. (ed.) Proceedings of the Second International Conference on Artificial Intelligence Planning Systems, University of Chicago, Chicago, Illinois, USA, June 13-15, 1994. pp. 249–254. AAAI (1994)

[32] Fikes, R.E., Nilsson, N.J.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2(34), 189 – 208 (1971)

[33] Fodor, P., Kifer, M.: Tabling for transaction logic. In: Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming. pp. 199–208. PPDP '10, ACM, New York, NY, USA (2010)

[34] Fox, M., Long, D.: Pddl2.1: An extension to pddl for expressing temporal planning domains. J. Artif. Int. Res. 20(1), 61–124 (Dec 2003)

[35] Gebser, M., Kaminski, R., Knecht, M., Schaub, T.: plasp: A prototype for PDDL-based planning in ASP. In: Delgrande, J., Faber, W. (eds.) Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11). Lecture Notes in Artificial Intelligence, vol. 6645, pp. 358–363. Springer-Verlag (2011)

[36] Gebser, M., Kaminski, R., Knecht, M., Schaub, T.: plasp: A prototype for pddl-based planning in ASP. In: Delgrande, J.P., Faber, W. (eds.) Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6645, pp. 358–363. Springer (2011)

[37] Gebser, M., Kaufmann, R., Schaub, T.: Correct reasoning. chap. Gearing Up for Effective ASP Planning, pp. 296–310. Springer-Verlag, Berlin, Heidelberg (2012)

[38] Geffner, H.: Pddl 2.1: Representation vs. computation. J. Artif. Int. Res. 20(1), 139–144 (Dec 2003)

[39] Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9(3-4), 365–385 (1991)

[40] Gelfond, M., Lifschitz, V.: Action languages. Electron. Trans. Artif. Intell. 2, 193–210 (1998)

[41] Gerevini, A., Haslum, P., Long, D., Saetti, A., Dimopoulos, Y.: Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. Artif. Intell. 173(5-6), 619–668 (2009)

[42] Ghallab, M., Isi, C.K., Penberthy, S., Smith, D.E., Sun, Y., Weld, D.: PDDL - The Planning Domain Definition Language. Tech. rep., CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control (1998)

[43] Giunchiglia, E., Lifschitz, V.: Dependent fluents. In: Proceedings of International Joint Conference on Artificial Intelligence (IJCAI). pp. 1964–1969 (1995)

[44] Giunchiglia, E., Massarotto, A., Sebastiani, R.: Act, and the rest will follow: Exploiting determinism in planning as satisfiability. In: Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative

Applications of Artificial Intelligence. pp. 948–953. AAAI ’98/IAAI ’98, American Association for Artificial Intelligence, Menlo Park, CA, USA (1998)

[45] Givan, R., Dean, T.L.: Model minimization, regression, and propositional STRIPS planning. In: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes. pp. 1163–1168. Morgan Kaufmann (1997)

[46] Green, C.: Application of theorem proving to problem solving. In: Proceedings of the 1st International Joint Conference on Artificial Intelligence. pp. 219–239. IJCAI’69, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1969)

[47] Gregory, P., Long, D., Fox, M.: Constraint based planning with composable substate graphs. In: Coelho, H., Studer, R., Wooldridge, M. (eds.) ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings. Frontiers in Artificial Intelligence and Applications, vol. 215, pp. 453–458. IOS Press (2010)

[48] Gupta, P., Stoller, S.D., Xu, Z.: Abductive analysis of administrative policies in rule-based access control. IEEE Trans. Dependable Sec. Comput. 11(5), 412–424 (2014)

[49] Haslum, P.: Improving heuristics through relaxed search: An analysis of tp4 and hspa* in the 2004 planning competition. J. Artif. Int. Res. 25(1), 233–267 (Feb 2006)

[50] Haslum, P., Bonet, B., Geffner, H.: New admissible heuristics for domain-independent planning. In: Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3. pp. 1163–1168. AAAI’05, AAAI Press (2005)

[51] Hoffmann, J., Porteous, J., Sebastia, L.: Ordered landmarks in planning. J. Artif. Int. Res. 22(1), 215–278 (Nov 2004)

[52] Hölldobler, S., Schneeberger, J.: A new deductive approach to planning. New Generation Computing 8(3), 225–244 (1990)

[53] Howe, A.E.: Improving the reliability of artificial intelligence planning systems by analyzing their failure recovery. IEEE Trans. on Knowl. and Data Eng. 7(1), 14–25 (Feb 1995)

[54] Joslin, D., Pollack, M.E.: Is ”early commitment” in plan generation ever a good idea? In: Clancey, W.J., Weld, D.S. (eds.) Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, August 4-8, 1996, Volume 2. pp. 1188–1193. AAAI Press / The MIT Press (1996)

[55] Kahramanogullari, O.: Towards planning as concurrency. In: Hamza, M.H. (ed.) Artificial Intelligence and Applications. pp. 387–393. IASTED/ACTA Press (2005)

[56] Kahramanogullari, O.: On linear logic planning and concurrency. Information and Computation 207(11), 1229 – 1258 (2009), special Issue: 2nd International Conference on Language and Automata Theory and Applications (LATA 2008)

[57] Kambhampati, S., Hendler, J.A.: A validation-structure-based theory of plan modification and reuse. Artif. Intell. 55(2-3), 193–258 (Jun 1992)

[58] Kautz, H., Selman, B.: Planning as satisfiability. In: Proceedings of the 10th European Conference on Artificial Intelligence. pp. 359–363. ECAI ’92, John Wiley & Sons, Inc., New York, NY, USA (1992)

[59] Kautz, H.A., McAllester, D.A., Selman, B.: Encoding plans in propositional logic. In: Aiello, L.C., Doyle, J., Shapiro, S.C. (eds.) Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning (KR’96), Cambridge, Massachusetts, USA, November 5-8, 1996. pp. 374–384. Morgan Kaufmann (1996)

[60] Kautz, H.A., Selman, B.: Planning as satisfiability. In: ECAI. pp. 359–363 (1992)

[61] Kautz, H.A., Selman, B.: Pushing the envelope: Planning, propositional logic and stochastic search. In: Clancey, W.J., Weld, D.S. (eds.) Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI

96, IAAI 96, Portland, Oregon, August 4-8, 1996, Volume 2. pp. 1194–1201. AAAI Press / The MIT Press (1996)

[62] Kautz, H.A., Selman, B.: The role of domain-specific knowledge in the planning as satisfiability framework. In: Simmons, R.G., Veloso, M.M., Smith, S.F. (eds.) Proceedings of the Fourth International Conference on Artificial Intelligence Planning Systems, Pittsburgh, Pennsylvania, USA, 1998. pp. 181–189. AAAI (1998)

[63] Kautz, H.A., Selman, B.: Unifying sat-based and graph-based planning. In: Dean, T. (ed.) Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages. pp. 318–325. Morgan Kaufmann (1999)

[64] Koehler, J., Hoffmann, J.: On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. J. Artif. Int. Res. 12(1), 339–386 (Jun 2000)

[65] Korf, R.E.: Planning as search: A quantitative approach. Artif. Intell. 33(1), 65–88 (1987)

[66] Lifschitz, V.: Answer set programming and plan generation. Artificial Intelligence 138(12), 39 – 54 (2002), knowledge Representation and Logic Programming

[67] Lloyd, J.W.: Foundations of Logic Programming. Springer-Verlag New York, Inc., New York, NY, USA (1984)

[68] McAllester, D., Rosenblitt, D.: Systematic nonlinear planning. In: Proceedings of the Ninth National Conference on Artificial Intelligence - Volume 2. pp. 634–639. AAAI'91, AAAI Press (1991)

[69] McDermott, D.V.: A heuristic estimator for means-ends analysis in planning. In: Drabble, B. (ed.) Proceedings of the Third International Conference on Artificial Intelligence Planning Systems, Edinburgh, Scotland, May 29-31, 1996. pp. 142–149. AAAI (1996)

[70] McDermott, D.V.: The 1998 AI planning systems competition. AI Magazine 21(2), 35–55 (2000)

[71] McDermott, D.V.: PDDL2.1 - the art of the possible? commentary on fox and long. J. Artif. Intell. Res. (JAIR) 20, 145–148 (2003)

[72] Nau, D., Ghallab, M., Traverso, P.: Automated Planning: Theory & Practice. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2004)

[73] Nilsson, N.: Principles of Artificial Intelligence. Tioga Publ. Co., Paolo Alto, CA (1980)

[74] Pednault, E.P.D.: Adl: Exploring the middle ground between strips and the situation calculus. In: Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning. pp. 324–332. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1989)

[75] Penberthy, S.J., Weld, D.S.: Ucpop: A sound, complete, partial order planner for ADL. In: Nebel, B., Rich, C., Swartout, W. (eds.) KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference, pp. 103–114. Morgan Kaufmann (1992)

[76] Przymusinski, T.C.: Foundations of deductive databases and logic programming. chap. On the Declarative Semantics of Deductive Databases and Logic Programs, pp. 193–216. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1988)

[77] Rezk, M., Kifer, M.: Transaction logic with partially defined actions. J. Data Semantics 1(2), 99–131 (2012)

[78] Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Pearson Education, 2 edn. (2003)

[79] Sacerdoti, E.D.: Planning in a hierarchy of abstraction spaces. Artif. Intell. 5(2), 115–135 (1974)

[80] Sacerdoti, E.D.: The nonlinear nature of plans. In: Proceedings of the 4th International Joint Conference on Artificial Intelligence - Volume 1. pp. 206–214. IJCAI'75, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1975)

[81] Smith, D.E.: The case for durative actions: A commentary on PDDL2.1. J. Artif. Intell. Res. (JAIR) 20, 149–154 (2003)

[82] Srivastava, S., Immerman, N., Zilberstein, S., Zhang, T.: Directed search for generalized plans using classical planners. In: Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS-2011). AAAI (June 2011)

[83] Stefik, M.: Planning with constraints (MOLGEN: part 1). Artif. Intell. 16(2), 111–140 (1981)

[84] Stefik, M.J.: Planning with Constraints. Ph.D. thesis, Stanford, CA, USA (1980), aAI8016868

[85] Sussman, G.J.: A Computer Model of Skill Acquisition. Elsevier Science Inc., New York, NY, USA (1975)

[86] Swift, T., Warren, D.: Xsb: Extending the power of prolog using tabling. Theory and Practice of Logic Programming (2011)

[87] Zhou, N., Dovier, A.: A tabled prolog program for solving sokoban. In: IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011, Boca Raton, FL, USA, November 7-9, 2011. pp. 896–897. IEEE Computer Society (2011)

[88] Zhou, N., Dovier, A.: A tabled prolog program for solving sokoban. Fundam. Inform. 124(4), 561–575 (2013)

[89] Zhou, N.f., Have, C.t.: Efficient tabling of structured data with enhanced hash-consing. Theory Pract. Log. Program. 12(4-5), 547–563 (Sep 2012)