Optimizing Projection in the Situation Calculus

Christopher James Ewin

orcid.org/0000-0002-4468-8519

Submitted in total fulfilment of the requirements

of the degree of Doctor of Philosophy

March 2018

School of Computing

and Information Systems

The University of Melbourne

Abstract

Among the most frequent reasoning tasks in the situation calculus are projection queries that query the truth of conditions in a future state of affairs. However, in long running action sequences involving thousands or millions of independent actions, solving the projection problem is complex. Existing approaches require either syntactically rewriting queries through each action that has occurred via a mechanism called *regression* or producing and maintaining an updated representation of the knowledge base via *progression*. This latter approach is often infeasible, as updating a knowledge base without loss of relevant information is not possible for many domains.

This thesis introduces a new technique which allows the length of the action sequences to be reduced by reordering independent actions and removing dominated actions; maintaining semantic equivalence with respect to the original action theory. This transformation allows for the removal of actions that are problematic with respect to progression, allowing for periodic update of the action theory to reflect the current state of affairs.

We provide the logical framework for the general case and give specific methods for important classes of action theories. We also show how more expressive cases may be handled, such as the reordering of sensing actions in order to delay progression. We investigate mechanisms for deciding which actions should be removed or reordered to improve the efficiency via a guided search and introduce appropriate heuristics.

The end result is a method that allows long-running situation calculus based agents to reason more efficiently about their current and future situations.

Declaration

This is to certify that:

1. the thesis comprises only my original work towards the PhD except where indicated in the Preface
2. due acknowledgement has been made in the text to all other materials used
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices

---

Christopher James Ewin

Preface

During the course of this research, the following publications and public presentations have been made based on the work in this thesis:

- Chris Ewin, Adrian R. Pearce and Stavros Vassos. Transforming Situation Calculus Action Theories for Optimised Reasoning In Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR) , pp. 448-457, 2014

- Chris Ewin, Adrian R. Pearce and Stavros Vassos. Optimizing Long-Running Action Histories in the Situation Calculus Through Search. In PRIMA 2015: Principles and Practice of Multi-Agent Systems 18th International Conference, pp. 85-100, Vol. 9387, LNCS, 2015

Acknowledgments

First, my heartfelt thanks to my supervisor, Adrian Pearce. He has always encouraged me to follow my own direction and be guided by my findings while also understanding where this research fits in the bigger picture. His open-mindedness to new ideas and his insights as to where they may be applied have been crucial. Thank you for all your support and guidance.

To Stavros Vassos, thank you for all of your help and your ideas for new research directions. Your focus and attention to detail has made all of this possible. Thank you for all of the late nights spent discussing theorems, proofs, and ideas.

To my advisory committee, Harald and Peter, thank you for all of your advice and support. You helped me to keep my ideas grounded, to pursue practical results and to aways keep my focus on the outcomes. Each progress meeting left me with a renewed sense of determination and clarity of direction.

My great thanks to the University of Melbourne, the Australian Government and National ICT Australia for providing my scholarships. Without their support this research would not have been possible.

To my fellow students and colleagues at the University of Melbourne, thank you for your support and encouragement. My particular thanks to Justin and Toby for always being willing to discuss new ideas and provide feedback.

Finally, my great thanks to my entire family. To my mother Cheryl and my sister Carrie, thank you for your unwavering support throughout the PhD years.

Contents

Chapter 1

Introduction

One of the most fundamental problems that software agents must solve is that of *projection*. Agents must be able to reason about the state of the world around them, the effects that their actions may have on the world, and how the world might look in the future if certain actions are performed. For expressive agents, this task is particularly significant as the effects of the agent's actions and knowledge may be expressed in first-order logic, making the projection problem challenging to solve.

Many formalisms have been developed to specify the behaviour of expressive autonomous agents and model the effects that their actions can have on the world around them. The situation calculus is one such formalism, which has been the focus of significant research and development since its inception in 1963. Complex programming languages such as Golog have been created using the situation calculus, allowing software agents capable of autonomous operation to be designed. Such agents can possess powerful cognitive abilities - they are capable of understanding what actions they may perform, storing information about the world around them and devising plans to achieve specified goals. Nonetheless, significant limitations currently exist in designing agents capable of operating autonomously for long periods of time over complex domains.

In such cases, we may expect thousands or even millions of individual ac-

tions to be performed, each affecting the state of the surrounding world. Such an agent must be capable of performing reasoning tasks such as querying the current state of objects in the world. The agent must also be able to perform projection queries to establish the truth of conditions in a future state of affairs.

Two well studied solutions exist for this problem. An agent may keep the whole action history and rely on a *regression* mechanism as introduced by Pirri and Reiter [76] for answering queries about the current and future states. This approach, taken in the traditional Golog programming language, involves maintaining a complete history of actions that the agent has performed and syntactically rewriting queries into a form that can easily be answered against the agent's initial knowledge base. Such an approach can be effective in automated planning, but becomes impractical for agent control as the length of the action history increases [95, 96]. Alternatively, an agent can use a *progression* mechanism to continually update the state of the knowledge base, answering projection queries against this progressed representation. While this can be an effective technique for many classes of problems, Vassos and Levesque [100] have shown that not all action theories are first-order progressable. Even in cases where first-order progressions exist, there is often no known mechanism for calculating them efficiently.

1.1 Motivation

Simple planning languages such as STRIPS [34] have existed for decades, and can be used to model the behaviour of simple software agents. More complex agents, however, require more sophisticated formalisms to efficiently describe and implement them. One such formalism is the situation calculus, which has a greater expressive power than most planning languages [30]. Selecting a formalism to model agent behaviour involves an inherent trade-off: more complex formalisms allow more complex behaviour to be implemented and produce a

more compact program, but the added complexity makes reasoning tasks more difficult.

Nonetheless complex agents have become an important tool, not just for designing mobile robots [60], but also for applications such as semantic web services [70], business processes [77] and many others. The computational limitations of complex agents become particularly significant as the lifetime of the agent increases [96, 19]. Many techniques have been developed over the years to help overcome these limitations, however the computational complexity of maintaining and reasoning about complex, long-lived agents remains a significant problem that we seek to address.

Here we present two motivating examples that we will refer to throughout. Consider first an agent designed to play the popular $Sokoban$ game. The agent controls a virtual robot which is capable of moving up, down, left or right through an $N \times M$ grid. The grid contains a number of blocks, each of which must be pushed by the agent to a target position on the grid. While moving through the grid, the agent must be able to answer queries such as:

"Is it possible to push a particular block forward right now?";
"Will this block become stuck after performing action α, preventing it being moved again?";
"Is it possible to win the game from the current position?"

Each of these queries requires the agent to reason about the current or future state of the game. As the the size of the game becomes larger, the number of actions required to solve the puzzle also increases. This poses great challenges for a regression based reasoning system, as it must rewrite queries over thousands or even millions of actions before being able to solve them.

Consider also an agent designed to operate in the so-called $wumpusworld$, a domain which has been extensively studied in AI research. The agent is positioned inside a similar $N \times M$ grid and must locate a pot of gold somewhere

inside the world. While doing so, the player must avoid falling into pits and also avoid the $wumpus$, a creature that will kill the player if the two collide. Since the grid is darkened, the player cannot see either the wumpus or the pits. He can, however, use his sense of smell to determine when the wumpus is nearby and also feel the breeze emanating from the pits if he is standing next to one. Here, the agent must perform similar queries such as:

"Can I safely move forward?";
"Have I already visited this grid square?"

This agent's reasoning system faces several additional challenges, however. After each move it must perform sensing actions to ascertain whether the wumpus or any pits are nearby then integrate that information into its knowledge base. Since feeling a breeze could mean that pits exist in any or all of the adjacent grid square, the agent must also be able to reason over uncertain information.

1.2 Contributions

We build upon the language of the situation calculus to provide an efficient method for handling automated reasoning problems such as *projection*. Motivated by approaches taken in classical partial order planning, we introduce mechanisms for determining which actions in a sequence are redundant to future reasoning, and which actions can be reordered to simplify the reasoning process. We consider restrictions on the *basic action theories* [80] that allow this process to be efficiently computed, and formally define the conditions under which actions can be removed from or reordered within a sequence. We then consider restrictions on the form of the initial knowledge base that allow us to estimate the cost of *progressing* actions through the use of a *database of possible closures*. Based on this database of possible closures, we introduce heuristics for determining which actions should be removed or reordered to enable automated

reasoning to be conducted efficiently and model the task of finding improved sequence manipulations as a classical planning problem. Our main contributions are as follows:

- The introduction of formal definitions that allow actions to be reordered or removed without affecting future reasoning tasks.
- A mechanism for determining which actions can be reordered or removed from a sequence when successor state axioms are restricted to a particular syntactic form, based on the concept of *effect sets*.
- Mechanisms for determining when more complex actions, such as *sensing actions* or *widening actions* can be removed from an action sequence in domains involving incomplete or uncertain information.
- A formal procedure for comparing actions offline, allowing actions to be eliminated and reordered efficiently during program execution.
- An examination of the cost of performing reasoning tasks such as progression for knowledge bases of a particular syntactic form.
- A modeling of the task of finding optimal action sequence manipulations as a classical planning problem and heuristics to aid the direction of the search.
- An empirical evaluation of the effects of action sequence manipulation over two popular domains.

1.3 Chapter Summary

The remainder of this thesis proceeds as follows:

- Chapter 2: We review relevant background information on the situation calculus and the automated reasoning tasks we are concerned with. We

examine existing approaches for handling these reasoning tasks, and introduce the Golog language commonly used to implement them.

- Chapter 3: We introduce mechanisms for determining when actions can be reordered or removed from an action sequence. We consider in detail syntactic restrictions that can be placed on the form of successor state axioms to enable that process to be conducted efficiently.

- Chapter 4: We examine the task of determining which action sequence manipulations will improve the efficiency of automated reasoning tasks. We model the task of finding an optimal sequence manipulation as a classical planning problem and consider restrictions on the initial knowledge base that allow useful heuristics to be developed.

- Chapter 5: We conclude with a summary of our findings and where they fit within the literature and suggest opportunities for further work.

Chapter 2

Background

This chapter provides a review of relevant background material. We begin by introducing the logical formalism of the situation calculus and the key reasoning tasks we seek to address. We further introduce the Golog programming language, which is commonly used to implement software agents using the situation calculus.

2.1 The Situation Calculus

The situation calculus, first introduced by [69] and formalised in [48], is a mathematical formalism used to describe the effects of actions and change within dynamic worlds. The variant most commonly used today is due to [80], but many extensions have since been proposed, such as [86] for handling epistemic domains and information gathering actions. This thesis follows Reiter's version as far as possible, in order to retain generality.

The language $\mathcal{L}$ of the situation calculus is a three-sorted first-order logic language with equality and some limited second-order features. The sorts are: *action*, *situation*, and a catch-all sort *object* for everything else depending on the domain of application. Similar to a normal one-sorted first-order language, $\mathcal{L}$ includes function and predicate symbols. In this case since there are three sorts,

each of the symbols has a type that specifies the sorts for the arguments it takes. The situation calculus includes symbols only of certain types each of which has a special role in the representation of the world and its dynamics.

An action term or simply an *action* represents an atomic action that may be performed in the world. For example consider an action $move(l_1, l_2)$ in the Sokoban domain. This action may be used to represent that a robot moves from location l_1 to location l_2 in the world. A situation term or simply a *situation* represents a world history as a sequence of actions. The constant S_0 is used to denote the *initial situation* where no actions have occurred. Sequences of actions are built using the function symbol *do*, such that $do(\alpha, \sigma)$ represents the successor situation resulting from performing action α in situation σ.

A *fluent* is a predicate whose last argument is a situation, and thus whose truth value can change from situation to situation. For example, $RobotAt(l, \sigma)$ may be used to represent that the robot lies at location l in situation σ. Actions need not be executable in all situations, and the predicate atom $Poss(\alpha, \sigma)$ states that action α is executable in situation σ. For example, $Poss(move(l_1, l_2), \sigma)$ is intended to represent that the action $move(l_1, l_2)$ can be executed in situation σ. Predicates that do not contain a situation term are referred to as *rigid* predicates.

The language $\mathcal{L}$ also includes the binary predicate symbol $\sqsubset$ which provides an ordering on situations. The atom $s \sqsubset s'$ means that the action sequence s' can be obtained from the sequence s by performing one or more actions in s. We will typically use the notation $\sigma \sqsubseteq \sigma'$ as a macro for $\sigma \sqsubset \sigma' \vee \sigma = \sigma'$. Of particular interest in this thesis are the set of so-called *uniform* formulas. Intuitively, these are formulas that refer to a common situation term. Formally, we follow the inductive definition of uniform formulas given by [80], presented below:

Definition 1. Let ϕ be a term of sort *situation*, g be a rigid function symbol, f be a fluent function symbol and t be a term not of sort situation. The set of

uniform terms is then the smallest set of terms τ such that:

$$\tau := t \mid g(\vec{\tau}) \mid f(\vec{\tau}, \phi)$$

Definition 2. Let ϕ be a term of sort *situation*, F be a fluent symbol, G be a rigid predicate, τ be a term uniform in ϕ and v be a term that is not of sort situation. The set of uniform formulas is then the smallest set of formulas such that:

$$\phi := F(\vec{\tau}, \phi) \mid R(\vec{\tau}) \mid \tau_1 = \tau_2 \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid (\exists v)\phi$$

2.2 Basic action theories

Within the language $\mathcal{L}$, one can formulate action theories that describe how the world changes as the result of the available actions. We consider *basic action theories (BATs)* as defined by [80] of the following form:

$$\mathcal{D} = \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_{una} \cup \mathcal{D}_0 \cup \mathcal{D}_{fnd},$$

where:

1. $\mathcal{D}_{ap}$ is the set of action precondition axioms (PAs), one per action symbol A, of the form $Poss(A(\vec{y}), s) \equiv \Pi_A(\vec{y}, s)$, where $\Pi_A(\vec{y}, s)$ is first-order and uniform in s. PAs characterize the conditions under which actions are physically possible.

2. $\mathcal{D}_{ss}$ is the set of successor state axioms (SSAs), one per fluent symbol F, of the form $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$, where $\Phi_F(\vec{x}, a, s)$ is first-order and uniform in s. SSAs describe how fluents change between situations as the result of actions.

3. $\mathcal{D}_{una}$ contains a set of unique names axioms for actions, which specify that actions with different names or arguments cannot be identical. For

example,

$$grab(l_1) \neq release(l_1)$$
$$move(l_1, l_2) = move(l_3, l_4) \supset l_1 = l_3 \wedge l_2 = l_4$$

4. $\mathcal{D}_0$, the *initial knowledge base (KB)*, is a set of first-order sentences uniform in S_0 describing the initial situation S_0.

5. $\mathcal{D}_{fnd}$ is the set of domain independent axioms of the situation calculus, formally defining the legal situations. In particular, $\mathcal{D}_{fnd}$ contains four key axioms. The first defines the fact that all situations produced by different sequences of actions are distinct. Formally,

   $$do(a_1, s_1) = do(a_2, s_2) \supset a_1 = a_2 \wedge s_1 = s_2$$

   The second is a second-order induction axiom that requires that all situations are constructed as a finite sequence of actions:

   $$\forall P.P(S_0) \wedge (\forall a, s)[P(s) \supset P(do(a, s))] \supset (\forall s)P(s)$$

   The third axiom requires that no situation can precede S_0:

   $$\neg(s \sqsubset S_0)$$

   The final axiom provides an ordering on situations after S_0:

   $$s \sqsubset do(a, s') \equiv s \sqsubseteq s'$$

Probably the most interesting component of a basic action theory is the set of successor state axioms, which together encode the dynamics of the domain being represented. Technically, successor state axioms are meant to capture the

effects and non-effects of actions. To achieve that in a parsimonious way, one typically follows the well-known solution to the frame problem[79] by means of successor state axioms of the following form [80]:

$$F(\vec{x}, do(a, s)) \equiv \gamma_F^+(\vec{x}, a, s) \vee F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a, s),$$

where both $\gamma_F^+(\vec{x}, a, s)$ and $\gamma_F^-(\vec{x}, a, s)$ are first-order formulas uniform in s encoding the positive and negative effects, respectively, of action a on fluent F at situation s.

Example 1. Take the Sokoban domain mentioned earlier. Allow:

- Object variables $block_1 \cdots block_n$ representing each block in the domain and $loc_1 \cdots loc_n$ representing each location in the grid.
- Two actions, *move*(l), which moves the player to l and *push*(b, l_1, l_2) which pushes block b from l_1 to l_2.
- Two fluents *PlAt*(l, s), which represents that the player is at location l in situation s and *BlAt*(b, l, s), which represents that block b is at location l in situation s.
- Two helper functions $Adjacent(l_1, l_2)$ which returns $true$ if two locations are horizontally or vertically adjacent to another, and $LineUp(l_1, l_2, l_3)$, which returns true if l_1, l_2 and l_3 all line up along a row or column.

If formalised as a situation calculus basic action theory, it might contain the following:

Unique Name Axioms:

$$\textit{move}(\vec{x}) \neq \textit{push}(\vec{y})$$

$$\textit{move}(x_1) = \textit{move}(x_2) \supset x_1 = x_2$$

$$\textit{push}(b_1, x_1, y_1) = \textit{push}(b_2, x_2, y_2) \supset b_1 = b_2 \wedge x_1 = x_2 \wedge y_1 = y_2$$

Action Precondition Axioms:

$$Poss(\textit{move}(l_1), s) \equiv \exists l_2(\textit{PlAt}(l_2, s) \wedge Adjacent(l_1, l_2)) \wedge \neg\exists b\, \textit{BlAt}(b, l_1, s)$$

$$Poss(\textit{push}(b_1, l_1, l_2)) \equiv \textit{BlAt}(b_1, l_1, s) \wedge \neg\exists b_2(\textit{BlAt}(b_2, l_2, s)) \wedge$$
$$\exists l_3(\textit{PlAt}(l_3) \wedge LineUp(l_3, l_1, l_2))$$

Successor State Axioms:

$$\textit{PlAt}(l, \textit{do}(a, s)) \equiv a = move(l) \vee \exists b, y(a = push(b, l, y))$$
$$\vee\, \textit{PlAt}(l, s) \wedge \neg\exists b, x, y[a = move(x) \vee a = push(b, x, y)]$$
$$\textit{BlAt}(b, l, \textit{do}(a, s)) \equiv \exists x(a = push(b, x, l)$$
$$\vee\, \textit{BlAt}(b, l, s) \wedge \neg\exists y(a = push(b, l, y))$$

Initial Situation:

One sample initial situation could be the following:

$$\textit{PlAt}(loc_1, S_0)$$
$$\textit{BlAt}(block_1, loc_3, S_0)$$
$$\textit{BlAt}(block_2, loc_6, S_0)$$

2.3 Reasoning via Regression

In the most general case, query answering in $\mathcal{D}$ corresponds to theorem-proving in second-order logic (SOL). This is clearly problematic for automated reasoning. Observe, however, that second-order axioms exist only in $\mathcal{D}_{\textit{fnd}}$ which relate situations together, therefore $\mathcal{D} - \mathcal{D}_{\textit{fnd}}$ is a purely first-order theory. Based on their earlier work on state constraints [64] Lin and Reiter [65] use this observation to demonstrate that for any ground situation term σ and any first-order formula ϕ uniform in σ, $\mathcal{D} \models \phi$ iff $(\mathcal{D} - \mathcal{D}_{\textit{fnd}}) \models \phi$.

Pirri and Reiter [76] obtain an even stronger result in their work on $regression$. They define a set of regressable formulas as follows for which query answering in future situations can be reduced to queries posed solely in S_0:

Definition 3. A first-order formula ϕ in $\mathcal{L}$ is *regressable* iff the following conditions hold:

- Every situation term mentioned in ϕ is rooted at S_0;
- For every atom of the form $Poss(\alpha, \sigma)$ that appears in ϕ, α has the form $A(\vec{t})$, where A is some n-ary action function symbol;
- ϕ does not quantify over situations;
- ϕ does not mention the predicate symbol $\sqsubset$ nor any equality atom built on situation terms

Regressable formula are more general than uniform formula, as they can contain predicates that refer to multiple situations so long as they all remain rooted in the same initial situation. They cannot, however, quantify over situation terms.

Pirri and Reiter go on to define a regression operator $\mathcal{R}$ that syntactically rewrites a query formula in $\mathcal{L}$ into a new formula uniform in S_0, enabling it to be answered in first-order logic. Essentially, the regression operator 'unwinds' a situation. Fluents mentioning $do(a, \sigma)$ are replaced with fluents mentioning σ using the successor state axioms in $\mathcal{D}_{ss}$. This process is repeated until no situation terms constructed via do remain. The full definition of the operator is complex, and omitted here for conciseness, but can be found in [80]. Crucially, Pirri and Reiter show that $\mathcal{R}$ is both sound and complete for the set of regressable formula. Formally, they prove the following theorem:

Theorem 1. *Suppose W is a regressable sentence of $\mathcal{L}$ and $\mathcal{D}$ is a basic action theory. Then,*

- $\mathcal{R}[W]$ *is a sentence uniform in* S_0
- $\mathcal{D} \models W$ *iff* $\mathcal{D}_0 \cup \mathcal{D}_{una} \models \mathcal{R}[W]$

While regression is not an efficient reasoning technique over all domains, it has nonetheless been shown to be effective for many practical cases. It has been widely implemented for many automated reasoning modules, such as [61] and [42].

2.4 Reasoning via Progression

Despite the flexibility of reasoning via regression, it has significant limitations in dealing with long action histories. The regression operator must be applied to each ground fluent for each action in the sequence, resulting in a potentially exponential increase in the cost of reasoning. While techniques such as caching can improve the efficiency of reasoning [21], it is frequently impractical to use regression to reason about long-lived agents. Thielscher [95] effectively demonstrates the significance of this problem, and argues strongly for progression-based reasoning as a more efficient approach.

Progression refers to the notion of updating the knowledge base with new information as it becomes available, however due to the second-order nature of $\mathcal{D}_{fnd}$ this is not straightforward to accomplish. In their seminal paper, Lin and Reiter [65] defined a sound and complete progression mechanism in the following way:

Definition 4. Let S_α be the situation that results after performing action α in situation S_0. Then $\mathcal{D}_{S\alpha}$ is a progression of $\mathcal{D}_{S0}$ if the following conditions hold:

1. The sentences of $\mathcal{D}_{S\alpha}$ are uniform in S_α
2. $\mathcal{D} \models (\mathcal{D} - \mathcal{D}_{S0}) \cup \mathcal{D}_{S\alpha}$

3. For every model $\mathcal{M}_\alpha$ of $(\mathcal{D} - \mathcal{D}_{S0}) \cup \mathcal{D}_{S\alpha}$ there is a model $\mathcal{M}$ of $\mathcal{D}$ such that:

   - $\mathcal{M}$ and $\mathcal{M}_\alpha$ have the same domains
   - $\mathcal{M}$ and $\mathcal{M}_\alpha$ interpret all non-fluent symbols that do not take arguments of sort situation identically
   - $\mathcal{M}$ and $\mathcal{M}_\alpha$ interpret all fluents about the future of S_α identically
   - $\mathcal{M}$ and $\mathcal{M}_\alpha$ interpret Poss identically for all future situations of S_α

This definition is known as $LR - Progression$. Lin and Reiter conjectured that if the set of sentences $\mathcal{D}_{S\alpha}$ were held to be first-order then the progression would not be correct according to Definition 5 for all basic action theories. As a result, some queries posed directly in $\mathcal{D}_{S\alpha}$ could return a different result than if regressed and posed in $\mathcal{D}_{S0}$. Vassos and Levesque [100] later proved this conjecture to be true by way of a counter-example. They introduce a specially crafted domain in which the axioms that describe the initial situation are only consistent in the presence of *unnamed objects*. These are objects for which no ground term is mentioned in the model. After (first-order) progressing along a certain action sequence, they show that this property no longer holds and demonstrate an example of a projection query that returns a different result for the progressed representation.

Generating an LR-Progression in the general case is difficult, requiring the use of second-order axioms [65, 100, 102] for the updated representation. The identified problematic cases that require second-order logic are mathematically involved and are rarely encountered, occurring infrequently in practice. Consequently, syntactically restricted forms of action theories have been investigated in order to ensure that a first-order progression exists and can be effectively computed.

For example, by building on the work of Vassos [98, 99] , Liu and Lakemeyer [66] use the notion of *forgetting* [63] to show that for *local-effect* and

normal actions, a first-order progression can always be defined and computed. By requiring that successor-state axioms be defined in a specified form, they define the *argument set* of a fluent F wrt an action α as follows:

$$\Delta_F = \{\vec{t} \mid \vec{x} = \vec{t} \text{ appears in } \gamma_F(\vec{x}, \alpha, s)\}$$

They then define the *characteristic set* of α as follows:

$$\Omega(s) = \{F(\vec{t}, s) \mid \text{F is a fluent and } \vec{t} \in \Delta_F\}$$

Based on these notions, they are able to demonstrate that a progression of the initial knowledge base, $\mathcal{D}_{S_0}$ can be defined as:

$$\bigwedge \mathcal{D}_{una} \wedge \bigvee_{\theta \in \mathcal{M}(\Omega(S_0))} (\mathcal{D}_{S_0} \cup \mathcal{D}_{ss}[\Omega])[\theta](S_0/S_\alpha)$$

Where $\mathcal{D}_{ss}[\Omega]$ is the instantiation of D_{ss} wrt Ω and $\mathcal{M}(\Omega(S_0))$ is a model of all sentences in $\Omega(S_0)$.

They go on to show that if the initial knowledge base is specified in a certain form, known as *proper+* [57], then the progression can be computed in $O(n)$ time, where n is the size of the knowledge base. Fan et. al. [31] later developed a more efficient implementation of this technique via *grounding*, and used the technique to develop a situation calculus based programming language capable of performing progression. It should be noted, however, that whilst such a technique allows for efficient first-order reasoning via progression in restricted domains, it requires that the programmer specify the entire initial knowledge base in proper+ form, placing a significant limitation on expressivity. It is also incapable of handling more general action theories, such as those including *global actions*.

Since then, significant research has been conducted into identifying restric-

tions that can be placed on basic action theories to enable a progression to be efficiently computed. Solutions have been found that encompass a number of complex domains, including those involving a form of incomplete information [101]. A map of progressable action theory classes can be found in [103]. Despite this research, action theories for which first-order progressions are known to exist are quite limited and a wide range of cases do not fit in the identified classes—for which we have no means of performing a (logically correct) first-order update. In practice this means that if the desired action theory includes even a single action that does not fit into the identified progressable cases, then updating of the action theory cannot be guaranteed: from the time that a problematic action α_1 is executed no more updating can be performed. Of course, projection queries can still be decided by the original action theory by forming appropriate entailment queries of the form: 'Will ϕ be true after actions $\langle \alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_m \rangle$ are performed in the initial state of the environment?', where $\alpha_1, \ldots, \alpha_n$ are the actions that have been already executed (for which the action theory unfortunately cannot be updated due to α_1) and $\beta_1, \ldots, \beta_m$ are the ones with respect to which we want to project in the future. Even when operating within the identified progressable classes, the progression operation is not guaranteed to be computationally efficient. [66] For example, progressing up to the final state may occasionally blow up the size of the KB due to incomplete information and the way it is encoded in the final state.

2.5 Reasoning in the Fluent Calculus

While most situation calculus based reasoning tools use regression as their primary reasoning tool, we are motivated to consider how these issues are handled in other first-order formalisms. The fluent calculus [93] uses the principle of *reification* to denote statements by terms instead of atoms and is designed heavily around the principle of progression. Rather than retaining a situation as a

sequence of consecutive actions, the fluent calculus maintains the current state of the world in terms of current fluent values. The fluent calculus shares many concepts with the situation calculus, to the extent that formal translations have been developed between the two calculi [88, 87, 97], leading us to examine it closely. Thielscher uses a binary predicate $Holds(f, s)$ to denote the fact that a particular fluent, f, holds in situation s. This allows a complete description of an initial situation to be specified using only a single situation argument. For example, an initial situation describing the fact that two lights are plugged in could be defined as follows:

$$Holds(f, S_0) \equiv f = PluggedIn(Light1) \vee f = PluggedIn(Light2)$$

Using the same representation, a successor state axiom for switching on a light could be defined as follows:

$$\begin{aligned} &Holds(f, Do(SwitchOn(x), s)) \equiv \\ &\qquad f = Lit(x) \wedge Holds(PluggedIn(x), S_0) \vee Holds(f, S_0) \end{aligned}$$

Now the state reached by turning on Light1 in the initial situation can be reached by substituting $x = Light1$ into the successor state axiom and replacing the subformula $Holds(f, S_0)$ with an equivalent result from the initial situation:

$$\begin{aligned} &Holds(f, Do(SwitchOn(Light1), s)) \equiv \\ &\qquad f = Lit(Light1) \wedge Holds(PluggedIn(Light1), S_0) \vee Holds(f, S_0) \\ &Holds(f, Do(SwitchOn(Light1), s)) \equiv f = Lit(Light1) \vee Holds(f, S_0) \end{aligned}$$

This representation cannot capture incomplete information, however, which is a significant limitation. For this reason, Thielscher introduces a function symbol $\circ$, representing the reified conjunction of fluents. The union of all fluents

that hold in a particular situation is defined as the $State$ of that situation. An initial situation describing the fact that two lights are plugged in, along with the possibility of other, unknown information could therefore be defined as follows:

$$\exists z[State(S_0) = PluggedIn(Light1) \circ PluggedIn(Light2) \circ z]$$

With this new representation, the $Holds$ predicate can be defined as follows:

$$Holds(f, s) \equiv \exists z.State(s) = f \circ z$$

Thielscher introduces a set of state update axioms that function similarly to the successor state axioms in the situation calculus. Instead of producing a new situation term, however, the state update axioms progress the current state of the world. The State Update axiom for switching on a light can be defined as follows:

$$Holds(PluggedIn(x), s) \supset State(Do(SwitchOn(x), s)) = State(s) \circ Lit(x)$$

The result of any action in the world can be defined by the application of one state update axiom. In the case of complete information, each state update axiom basically corresponds to the addition or removal of a finite number of literals from the database representing the current state of the world. This allows a projection to be efficiently computed. In the case of incomplete information, however, the state update axioms only allow for the representation of the old state, plus some sentences representing the update, which is not a uniform set of facts in any situation. Thus, these fluent calculus techniques do not constitute an LR-progression as defined above.

2.6 GOLOG

Based off the situation calculus, Levesque, et. al. implemented a non-deterministic agent programming language, GOLOG (alGOl in LOGic) [61]. Unlike traditional programming languages, programs written in GOLOG may have some arbitrary level of non-determinism, allowing the interpreter to find a legal execution at runtime. This approach has the potential to make programs simpler to write, as the programmer need not explicitly plan the next execution in each and every state, and more powerful, as optimal decisions can be taken at runtime. GOLOG has the following basic constructs:

α	Execute a primitive action, α
ϕ?	Test whether condition ϕ is true
$\delta_1; \delta_2$	Execute action δ_1 followed by δ_2
$\delta_1 \vert \delta_2$	Non-deterministically select either action δ_1 or δ_2 to execute
if ϕ then δ_1 else δ_2	Execute δ_1 if condition ϕ is true, otherwise execute δ_2
while ϕ do δ	Execute δ as long as ϕ is true
proc $P(\vec{x})\delta$ end	Define a high level procedure, P
$\pi(\vec{x}, \delta(\vec{x}))$	Non-deterministically select arguments for delta
δ *	Execute δ zero or more times

Since the development of the original GOLOG language, there have been a number of versions written to enhance its functionality. The most well known are:

- ConGolog [42], which provides support for concurrent actions
- IndiGolog [24], which provides support for online execution.

Many other extensions have been proposed, to incorporate multiple agents [51, 50, 33], decision theoretic [14] and game-theoretic aspects [35, 36], and additional control operators [10, 40, 92]. The original GOLOG language, as well as the ConGolog and IndiGolog extensions use regression for reasoning

about action theories, although recent research by Fan et. al. [31] has shown the the viability of using progression in conjunction with a classical planner in place of regression based reasoning. The use of progression allowed Fan to achieve significant performance gains when dealing with a restricted domain.

2.7 Compilations of Basic Action Theories

Given the difficulty of computing and reasoning about plans directly in expressive formalisms like the situation calculus, significant work has been undertaken to encode the action planning problem as a classical planning problem. This has the potential to allow the expressivity of agent programming languages, such as Golog, to be combined with the efficiency of PDDL Planning [37]. The most common approach is to use the 'ADL' fragment of PPDL first developed by Pednault [75], also referred to as PDDL level 1.

This approach has proven quite promising. Claßen et. al. [19, 18, 20] demonstrate a method for encoding a fragment of the situation calculus called $\mathcal{ES}$ [58] as an ADL planning problem. They show that the semantics of ADL can be understood as progression in the situation calculus, and develop a system for integrating ADL planning with a Golog interpreter. The interpreter makes periodic calls to the ADL planner, using it to generate action sequences that fulfil a particular sub-goal. They demonstrate that their system can be effective in situations where classical planning sub-problems naturally occur during the execution of a high-level Golog program.

Their system, however, requires that the domain be expressed using the less expressive ADL formalism. Previous work has been conducted to identify subsets of situation calculus basic action theories that can be mapped to ADL. Eyerich [30] first uses Nebel's compilation framework [74] to describe a set of restrictions that can be placed upon an action theory to enable it to be described in ADL. Röger et al. [82, 81] further examine these conditions to identify a max-

imal fragment of basic action theories that is equivalent to ADL. Nonetheless, converting general situation calculus action theories to ADL is not possible, as the situation calculus is a more expressive formalism. Even when a domain can be represented by ADL, such representations are not guaranteed to be compact, and may be exponentially larger than the situation calculus equivalent.

In parallel work, Baier, Fritz and McIlraith [8, 7] develop a Golog style language that translates into ADL planning. They use the Golog semantics to specify rules that 'guide' the planner, excluding action sequences that don't comply with the procedural rules. Using a modified version of TLPLAN [5], they are able to show that the Golog rules substantially reduces the search space required to find a valid execution. They build upon this work in [38] to develop a procedure for compiling ConGolog programs into ADL. Their approach has a number of advantages of that of Claßen et. al. By incorporating the procedural rules contained within the Golog program directly into the planner, they are able to guide the planning process in a way not otherwise possible. It also enables a more holistic approach to finding an execution, whereas Claßen's method requires that each case of non-determinism be handled as a separate planning problem, and is unable to reuse information from previous plans.

Despite this, Baier's work shares similar limitations to that of Claßen et. al. The domain must still be expressed using ADL specifications, and the Golog rules can only be used for restrict the transition semantics, not for defining them. While powerful, owing to the reduced expressivity of the ADL language, these compilation based approaches are only useful for restricted subsets of the situation calculus.

Other compilation based approaches have also been considered in the literature. Lee and Palla [59] develop a method for formulating situation calculus domains as Answer Set Programming (ASP) problems. They go on to develop and evaluate a system called F2LP [59] that converts first-order formula into a form that can be solved by ASP solvers in a manner similar to previous work in

the event calculus [52]. This idea is not entirely new. Kautz and Selman [49] adopt an intuitively similar approach to model situation calculus domains as SAT problems, as does Lin [62] when developing successor state axioms from causal theories. But all of these implementations require propositionalizing the domain, which results in a significant loss of expressivity.

Chapter 3

Optimizing Action Histories by Reordering and Dominating Actions

This chapter explores ways of improving the efficiency of reasoning by projection and progressing a knowledge base by manipulating a situation calculus action history. First, we introduce theorems for determining when actions can be safely reordered or eliminated. We examine the extent to which these operations can be determined off-line in advance of program execution, and propose mechanisms for reordering selected types of sensing actions. Finally, we introduce a GOLOG interpreter that demonstrates the efficacy of these techniques.

3.1 Background

In many practical domains, neither progression nor regression alone provide an optimal reasoning mechanism. Our approach allows these two mechanisms to be combined. An agent may progress only up to a point but leave a final sequence of actions for which queries about the current state will be answered by means of regression. Before progressing over an action sequence, reordering

and eliminating actions can significantly reduce the effort needed to reach the final state. An important observation is that similar to action α_1, that makes further updates impossible, there can be another action, α_k, that brings the environment to a state of affairs that is easier to represent and reason about. For example, α_k could be a *resetting* action that brings the environment to the initial state of affairs, or one that essentially *subsumes* the effects of α_1 in a way that the new representation is first-order progressable—via setting all problematic atoms to a fixed or sensed truth value. Current approaches do not consider this type of reasoning with respect to the action history, leaving out a wide range of practical cases that can be handled by such a hybrid approach to progression.

Our approach can be also useful in cases where all actions fall in the first-order progressable classes—for the purpose of obtaining computational benefits. Since the updating of the action theory is typically a much more computationally demanding procedure than that of projection, it can be beneficial to keep a history of executed actions up to some fixed (past horizon) length anticipating that a future action may greatly simplify the history. Depending on the characteristics of the domain and the projection queries that are needed, such an approach has the potential to save a lot of computational resources. For instance, in cases where there is a large degree of uncertainty in the form of incomplete or disjunctive information which would, under normal circumstances, lead to an exponential growth of the progressed theory with respect to the number of actions executed, a later sensing action that settles uncertainty would lead to larger computational savings, as compared to those consumed in order to perform projection queries using the longer history of executed actions.

We provide a framework for specifying transformations in a logically correct way in the general case. A promising new class of global-effect actions is introduced, termed *resetting* actions, which facilitate transformation based on dominance principles. These actions are general enough to encode complex operator-based planning domains, such as PDDL, as well as concepts such as

sensing and non-determinism. We specify practical procedures for realizing the proposed transformations, allowing for swapping and elimination of actions for handling non-progressable instances; or the interleaving of regression and progression as required.

Our framework allows the problem of finding more efficient action sequence representations to be expressed as a local-search problem. We demonstrate how the notion of *dominatable pairs* in an action history can be used to develop an efficient heuristic that can be used to guide the search. We further show that when a knowledge base expresses disjunctive information in a particular form it is possible to analyse the intermediate effort for updating the representation based on the notions of *progression speedup* and *progression slowdown* factors. This makes it possible to identify sequences of equivalent length that are less costly to progress, and therefore should be favoured for reasoning. We demonstrate that much of the computational effort involved in determining permissible alternative sequences can be obtained offline, through careful examination of the successor-state axioms for the domain. Consequently, we introduce an *offline* precomputation strategy allowing efficient online detection as to which actions can be swapped or eliminated. Finally, we evaluate the benefits of our technique over several illustrative domains.

3.2 Transforming situation calculus histories

In this section we provide the logical foundations for two basic operations that will allow to transform an action history in a way that is logically correct with respect to a basic action theory $\mathcal{D}$. We consider here a reordering operator, that will swap two consecutive actions in an action history as well as an elimination operator, which will remove an action from an action history. As we will demonstrate, a combination of these two operators acting on different actions in a history can be used to generate alternative action sequences that are nonethe-

less equivalent for reasoning over a wide variety of queries.

We first introduce some notation to specify sequences of actions and situation terms that are *rooted* at some other situation term, similarly to [39, 80].

Definition 5. Let σ be a situation term and δ be a (possibly empty) vector of action terms $\langle \alpha_1, \ldots, \alpha_n \rangle$. We use $do(\delta, \sigma)$ to denote the following situation: $do(\alpha_n, do(\alpha_{n-1}, \ldots do(\alpha_1, \sigma) \ldots))$. We say that a situation term κ is *rooted at* σ iff κ is syntactically the same term as $do(\delta, \sigma)$, for some vector of action terms δ. Finally, we use $\delta_1 \cdot \delta_2$ to denote the concatenation of vectors δ_1 and δ_2.

The intuition is that a situation term κ is rooted at some other situation term σ iff κ is of the form $do(\delta, \sigma)$ for some vector of action terms δ, i.e., iff it can be obtained from σ by "adding" a sequence of actions using the function *do*.

Example 2. The situation terms $do(\alpha, S_0)$, $do(a, S_0)$, and $do(\alpha_2, do(\alpha_1, S_0))$ are all rooted at S_0. The latter is also rooted at $do(\alpha_1, S_0)$. Note that every situation term σ is always rooted at σ. For instance, the situation term $do(a, s)$ is rooted at s but also at $do(a, s)$.

3.2.1 Swapping consecutive actions

We start with the logical specification of the notion of consecutive actions being *swappable* in an action history of the form $do(\delta, S_0)$, where δ is a sequence of ground actions. We will introduce two versions of the notion, *just-in-time swappable* and *always swappable*. For the following definitions, let $\mathcal{D}$ be a situation calculus basic action theory, δ be a sequence of ground actions and $\langle \alpha, \alpha' \rangle$ be a pair of ground actions that occur immediately after δ.

Definition 6. We say that actions α and α' are *just-in-time swappable* after δ wrt $\mathcal{D}$ iff for all fluents $F(\vec{x}, s)$ in $\mathcal{L}$ the following holds:

$$\mathcal{D} \models \forall \vec{x}.\ F(\vec{x}, do(\delta \cdot \langle \alpha, \alpha' \rangle, S_0)) \equiv F(\vec{x}, do(\delta \cdot \langle \alpha', \alpha \rangle, S_0)).$$

Essentially, this definition says that we can swap actions α and α' after δ as long as in all models of $\mathcal{D}$ it happens that the interpretation of all fluent atoms is preserved in the two versions of the action history. The *just-in-time* terminology is used to stress that it may be due to the axioms in this particular instance of $\mathcal{D}_0$ and the action sequence δ that ensure that this condition holds, while in fact actions α and α' would not necessarily be swappable if another $\mathcal{D}_0$ and δ were assumed.

Example 3. Consider the case of fluent $F(s)$ that is only affected by actions set_F and unset_F such that set_F has a conditional effect that makes $F(s)$ true when $G(s)$ holds, and unset_F has the conditional effect that makes $F(s)$ false when $G(s)$ holds. The SSA for $F(s)$ is then as follows:

$$F(\mathit{do}(a, s)) \equiv \ (a = \mathit{set}_F \wedge G(s)) \ \vee F(s) \wedge \neg(a = \mathit{unset}_F \wedge G(s)).$$

Consider now an action sequence $\delta \cdot \langle \alpha, \alpha' \rangle$ as above in which action α is set_F and α' is unset_F, assuming there are also other fluents and actions in the language. By looking at the SSA for $F(s)$ we can conclude that these actions may have "conflicting" effects and therefore they should not be considered swappable in general, as the order of applying these two actions may result to a different truth value for $F(s)$.

Nonetheless, if we also take into account $\mathcal{D}_0$ and the particular course of action that takes us to the situation where α and α' are performed, we may conclude that we can actually swap the actions. For instance, consider a $\mathcal{D}_0$ that includes the fact $\neg G(S_0)$, and the following SSA for $G(s)$:

$$G(\mathit{do}(a, s)) \equiv a = \mathit{set}_G \ \vee G(s) \wedge \neg(a = \mathit{unset}_G).$$

In the case that none of the actions in δ is set_G then condition $G(s)$ is in fact not satisfied in the situation when α is performed, and also in the resulting

situation in which α' is performed. As a result, actions α and α' have no effect in the truth value of $F(s)$, and actually no effects whatsoever in δ in every model of $\mathcal{D}$. For the same reason, reversing the order of these two actions also results in no effects in the truth value of any fluent atom, and we can conclude that they can be safely swapped, or more precisely that α and α' are just-in-time swappable after δ wrt $\mathcal{D}$. Note then that if none of the actions in δ is unset_G and exactly one is set_G then condition $G(s)$ is satisfied in the situation when α is performed, and also in the resulting situation in which α' is performed. As a result, actions α and α' have conflicting effects and are not just-in-time swappable after δ wrt $\mathcal{D}$.

Definition 6 is a precise characterization of when two consecutive actions may be swapped in a particular action sequence without affecting the interpretations of fluents. In order to check this in general one needs to look into the properties of the situation where the actions are performed and verify that no conflict arises. As we saw in Example 3 the same actions for the same $\mathcal{D}$ may be just-in-time swappable after one sequence of actions and not after another. There can also be cases of actions that the actual situation in which they are performed does not play any role, such as for actions that may only affect different fluents. We can capture this notion which we call *always swappable* with a similar definition that removes the dependency on the particular $\mathcal{D}_0$ and the action sequence δ as follows.

Definition 7. We say that actions α and α' are *always swappable* wrt $\mathcal{D}$ iff for all fluents $F(\vec{x}, s)$ in $\mathcal{L}$ the following holds:

$$\mathcal{D} - \mathcal{D}_0 \models \forall \vec{x}.\ F(\vec{x}, \mathit{do}(\langle \alpha, \alpha' \rangle, S_0)) \equiv F(\vec{x}, \mathit{do}(\langle \alpha', \alpha \rangle, S_0)).$$

Essentially, Definition 7 requires that starting from any possible extension for all fluents, applying the two actions in question will have the same effect regardless of the order the actions are applied.

Example 4. The actions set_F and $unset_F$ defined earlier are not always swappable wrt $\mathcal{D}$, and also the same holds for all α, α' such that $\alpha \in \{set_F, unset_F\}$ and $\alpha' \in \{set_G, unset_G\}$. On the other hand, if we assume that $\mathcal{D}$ also includes the following successor state axiom for H:

$$H(do(a, s)) \equiv\ a = set_H\ \vee H(s) \wedge \neg(a = unset_H)$$

Then for all α, α' such that $\alpha \in \{set_F, unset_F, set_G, unset_G\}$ and $\alpha' \in \{set_H, unset_H\}$, α, α' are always swappable wrt $\mathcal{D}$.

As we will see later, this distinction will become important when we specify algorithmic ways to decide whether consecutive actions are swappable. Next, we show some straightforward results about projection that follow from these definitions. The *simple projection* problem is deciding whether a condition about a particular situation in the future holds [80], and the next theorem shows that swapping actions according to our definitions preserves the result for any extension of the action sequence in the question.

Theorem 2. *Let δ be a sequence of ground action terms. Let $\phi(s)$ be a (first-order or second-order) formula in $\mathcal{L}$ that is uniform in s. If actions α and α are always swappable wrt $\mathcal{D}$ or just-in-time swappable after δ wrt $\mathcal{D}$ then the following holds:*

$$\mathcal{D} \models \phi(do(\delta \cdot \langle \alpha, \alpha' \rangle \cdot \zeta, S_0)) \equiv \phi(do(\delta \cdot \langle \alpha', \alpha \rangle \cdot \zeta, S_0)),$$

where ζ is any sequence of ground action terms.

Proof. For the case of just-in-time swappable we work as follows. Let M be an arbitrary model of $\mathcal{D}$. We show by induction on the situation terms σ such that $do(\delta, S_0) \sqsubseteq \sigma$, that for all fluents F and all sequences of action terms ζ, $M \models F(\vec{x}, do(\delta \cdot \langle \alpha, \alpha' \rangle \cdot \zeta, S_0)$ iff $M \models F(\vec{x}, do(\delta \cdot \langle \alpha', \alpha \rangle \cdot \zeta, S_0)$. For the base case we apply Definition 6 directly. For the induction step, consider the

successor state axioms $F(\vec{x}, do(a, s)) = \Phi_F(\vec{x}, a, s)$. The induction step follows by applying Definition 6 to $\Phi_F(\vec{x}, a, s)$. The theorem then follows by induction on the construction of the formulas ϕ that are uniform in s.

For the case of always swappable let M be an arbitrary model of $\mathcal{D}$. Since $\mathcal{D} - \mathcal{D}_0$ admits all possible extensions for the fluents in S_0 it follows that there is a model M' of $\mathcal{D} - \mathcal{D}_0$ with the same domain for all sorts as M such that for all μ and for all fluents F, $M, \mu \models F(\vec{x}, \mathit{do}(\delta, S_0))$ iff $M', \mu \models F(\vec{x}, S_0)$. By Definition 7 then it follows that $M \models F(\vec{x}, \mathit{do}(\delta \cdot \langle \alpha, \alpha' \rangle, S_0)$ iff $M \models F(\vec{x}, \mathit{do}(\delta \cdot \langle \alpha', \alpha \rangle, S_0)$. Then proceed by induction. for the base case, apply Definition 7 directly. For the induction step, apply Definition 7 to the successor state axioms $F(\vec{x}, do(a, s)) = \Phi_F(\vec{x}, a, s)$, as in the case of just-in-time swappable actions. □

Theorem 2 shows that swapping actions that are just-in-time or always swappable in δ preserves the entailment of any projection query that refers to some particular situation that comes after $\mathit{do}(\delta, S_0)$. This generalizes to more expressive sentences that refer to situations that come after $\mathit{do}(\delta, S_0)$, not necessary being uniform in one situation.

For example, we can show that swapping actions preserves sentences that may also quantify over future situations such as in the set $\mathcal{L}^F_\sigma$ defined by Vassos and Levesque in [102], which is a form of the so-called *generalized projection task*. Vassos and Levesque define this set as follows:

Definition 8. Let σ, κ be situation terms and ϕ a rectified formula in $\mathcal{L}$. κ *is in the future of* σ *in* ϕ iff one of the following holds:

- κ is σ, or
- κ is rooted at some situation term κ' that is in the future of σ in ϕ, or
- κ is a variable and $\forall\kappa(\kappa' \sqsubseteq \kappa \supset \beta)$ or $\exists\kappa(\kappa' \sqsubseteq \kappa \wedge \beta)$ is a sub-formula of ϕ, where κ' is a situation term that is in the future of σ in ϕ.

The formula ϕ *is about the future of* σ iff the situation terms in ϕ that appear as arguments of $Poss$ or some fluent or the equality predicate are all in the future of σ in ϕ. $\mathcal{L}_\sigma^F$ is the set of all rectified sentences ϕ in $\mathcal{L}$ such that ϕ is about the future of σ.

Essentially, the truth of sentences in $\mathcal{L}_\sigma^F$ depend only on situation σ and situations after σ. For σ being $do(\alpha, S_0)$, an example of a sentence in $\mathcal{L}_\sigma^F$ is the following: $\forall s(do(\alpha, S_0) \sqsubseteq s \supset \psi(s))$, which states that after executing action α in S_0 then $\psi(s)$ remains true always for all the future situations s from that point on.

Theorem 3. *Let $\mathcal{L}_\sigma^F$ be the set of first-order formulas that refer to the future of σ as defined in Definition 8 that allows limited quantification over situations, and $\phi(\sigma)$ be a sentence in $\mathcal{L}_\sigma^F$ for σ being $do(\delta, S_0)$. If actions α and α' are always or just-in-time swappable in δ wrt $\mathcal{D}$ then the following holds:*

$$\mathcal{D} \models \phi(do(\delta \cdot \langle \alpha, \alpha' \rangle \cdot \zeta, S_0)) \equiv \phi(do(\delta \cdot \langle \alpha', \alpha \rangle \cdot \zeta, S_0)),$$

where ζ is any sequence of ground action terms.

Proof. Follow the proof of Theorem 2 but also perform induction on the construction of the formulas ϕ in $\mathcal{L}_\sigma^F$. □

Also it is easy to show that always swappable implies just-in-time swappable but not the opposite.

Corollary 1. *If actions α and α' are always swappable wrt $\mathcal{D}$ then for any sequence of actions δ after which they appear they are just-in-time swappable in δ wrt $\mathcal{D}$, but the opposite is not true.*

We now turn our attention to how swappable actions may be detected in practice. Definitions 6 and 7 provide a semantical account of the requirement

that characterizes when two actions can be safely swapped in an action sequence. In particular, as we noted earlier, for just-in-time swappable actions we need to be able to project on the situation where the actions are applied and check the effects of actions conditioned on the action history. On the other hand, the notion of actions being always swappable is decoupled from the action history, allowing us to check for a simpler condition. The following theorem shows an equivalent characterization of always swappable actions in terms of the effects of the actions and their successor state axioms.

Theorem 4. *Actions α and α' are always swappable wrt $\mathcal{D}$ iff for all fluents $F(\vec{x}, s)$ in $\mathcal{L}$ the following holds:*

$$\begin{aligned}\mathcal{D} - \mathcal{D}_0 \models \forall \vec{x}\{&(\gamma_F^+(\vec{x}, \alpha, S_0) \vee \gamma_F^+(\vec{x}, \alpha', \mathit{do}(\alpha, S_0)) \\ &\equiv \gamma_F^+(\vec{x}, \alpha', S_0) \vee \gamma_F^+(\vec{x}, \alpha, \mathit{do}(\alpha', S_0))) \\ &\wedge (\gamma_F^-(\vec{x}, \alpha, S_0) \vee \gamma_F^-(\vec{x}, \alpha', \mathit{do}(\alpha, S_0)) \\ &\equiv \gamma_F^-(\vec{x}, \alpha', S_0) \vee \gamma_F^-(\vec{x}, \alpha, \mathit{do}(\alpha', S_0)))\} \\ \wedge \neg\exists \vec{x}\{&\gamma_F^+(\vec{x}, \alpha, S_0) \wedge \gamma_F^-(\vec{x}, \alpha', S_0) \\ &\vee \gamma_F^+(\vec{x}, \alpha', S_0) \wedge \gamma_F^-(\vec{x}, \alpha, S_0)\}.\end{aligned}$$

Proof. Consider the sequence $\langle \alpha, \alpha' \rangle$. If $\mathcal{D} - \mathcal{D}_0 \models F(\vec{x}, \mathit{do}(\langle \alpha, \alpha' \rangle, S_0))$ then either $\mathcal{D} - \mathcal{D}_0 \models \gamma_F^+(\vec{x}, \alpha, S_0)$ or $\mathcal{D} - \mathcal{D}_0 \models \gamma_F^+(\vec{x}, \alpha, \mathit{do}(\alpha', S_0))$, and similarly if $\mathcal{D} - \mathcal{D}_0 \models \neg F(\vec{x}, \mathit{do}(\langle \alpha, \alpha' \rangle, S_0))$ then either $\mathcal{D} - \mathcal{D}_0 \models \gamma_F^-(\vec{x}, \alpha, S_0)$ or $\mathcal{D} - \mathcal{D}_0 \models \gamma_F^-(\vec{x}, \alpha, \mathit{do}(\alpha', S_0))$. For Definition 7 to hold, we therefore require $\mathcal{D} - \mathcal{D}_0 \models \forall \vec{x}\{\gamma_F^+(\vec{x}, \alpha, S_0) \vee \gamma_F^+(\vec{x}, \alpha', \mathit{do}(\alpha, S_0)) \equiv \gamma_F^+(\vec{x}, \alpha', S_0) \vee \gamma_F^+(\vec{x}, \alpha, \mathit{do}(\alpha', S_0))) \wedge (\gamma_F^-(\vec{x}, \alpha, S_0) \vee \gamma_F^-(\vec{x}, \alpha', \mathit{do}(\alpha, S_0)) \equiv \gamma_F^-(\vec{x}, \alpha', S_0) \vee \gamma_F^-(\vec{x}, \alpha, \mathit{do}(\alpha', S_0)))\}$. Definition 7 also requires that α and α' don't affect a fluent in different ways, so also require $\mathcal{D} - \mathcal{D}_0 \models \neg\exists \vec{x}\{\gamma_F^+(\vec{x}, \alpha, S_0) \wedge \gamma_F^-(\vec{x}, \alpha', S_0) \vee \gamma_F^+(\vec{x}, \alpha', S_0) \wedge \gamma_F^-(\vec{x}, \alpha, S_0)\}$. □

Essentially, this theorem requires that two swappable actions have the same

effects on all fluent literals irrespective of the order they're performed in, and that they do not have different effects on any fluent literal. For certain classes of action theories, stronger results may be obtained. Consider the case of *context-free* action theories defined below:

Definition 9. Let F be a fluent symbol, a be an action and s be a situation term. Let γ_F^+ and γ_F^- refer to the positive and negative successor-state axioms for F respectively. A successor-state axiom is *context-free* if it has the following form: $F(\vec{x}, do(a, s)) \equiv \gamma_F^+(\vec{x}, a) \vee F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a)$. A basic action theory $\mathcal{D}$ is *context-free* iff all successor state axioms in $\mathcal{D}$ are context-free.

Intuitively, context-free action theories are those for which the effects of actions do not depend on the current situation. While restricted, they are nonetheless sufficient to encompass many interesting domains. For context-free action theories, Theorem 4 simplifies as follows:

Corollary 2. *Let $\mathcal{D}$ be a context-free basic action theory according to Definition 9 Actions α and α' are always swappable wrt $\mathcal{D}$ iff for all fluents $F(\vec{x}, s)$ in $\mathcal{L}$ the following holds:*

$$\mathcal{D} - \mathcal{D}_0 \models \neg\exists\vec{x}\{\gamma_F^+(\vec{x}, \alpha, S_0) \wedge \gamma_F^-(\vec{x}, \alpha', S_0) \vee \gamma_F^+(\vec{x}, \alpha', S_0) \wedge \gamma_F^-(\vec{x}, \alpha, S_0)\}.$$

These results allow us to reduce the semantic notion expressed in Definition 7 into a more concrete one about the effects of actions as expressed in the positive and negative effect formulas γ^+ and γ^- in the successor state axioms of the fluents in $\mathcal{L}$. Even though this condition is still an entailment question about the effects of actions, as we will see later for special cases of successor state axioms we will be able to further reduce it to simpler tests that we can easily evaluate.

Next we proceed to a similar analysis for actions that can be considered redundant and can be safely omitted.

3.2.2 Eliminating actions

Now we turn our attention to simplifying the action history by *eliminating* an action α that is *dominated* by a subsequent action α' after a sequence of ground action terms δ. As in the case of swappable actions, we identify two versions for this notion.

Definition 10. We say that action α is *just-in-time dominated* by α' after δ wrt $\mathcal{D}$ iff for all fluents $F(\vec{x}, s)$ in $\mathcal{L}$ the following holds:

$$\mathcal{D} \models \forall \vec{x}.\, F(\vec{x}, do(\delta \cdot \langle \alpha, \alpha' \rangle, S_0)) \equiv F(\vec{x}, do(\delta \cdot \langle \alpha' \rangle, S_0)).$$

Example 5. Consider the fluent $F(x, s)$ with the following successor state axiom:

$$\begin{aligned} F(x, do(a, s)) \equiv\ & a = set_F(x) \vee a = resetTrue_F \wedge G(s) \vee \\ & F(s) \wedge \neg(a = unset_F(x) \vee a = resetFalse_F). \end{aligned}$$

Action $set_F(c)$ sets the truth value of atom $F(c, s)$ to true, $unset_F(x)$ sets it to false, and there are two resetting actions that set all atoms to true and false. Action $resetTrue_F$ is conditional and only has this global effect when $G(s)$ holds, while $resetFalse_F$ is unconditional. Assume that $G(s)$ has the same successor state axiom as in Example 3. Consider the action sequence δ: $\langle unset_G, unset_F(c), set_F(c), resetTrue_F \rangle$. Then, $unset_F(c)$ is just-in-time dominated by $set_F(c)$ in δ wrt to $\mathcal{D}$, but $set_F(c)$ is not just-in-time dominated by $resetTrue_F$ in δ wrt to $\mathcal{D}$.

Similarly to always swappable actions we introduce the notion of *always dominated* actions that removes the dependency on the particular $\mathcal{D}_0$ and the action sequence δ as follows.

Definition 11. We say that action α is *always dominated* by α' wrt $\mathcal{D}$ iff for all

fluents $F(\vec{x}, s)$ in $\mathcal{L}$ the following holds:

$$\mathcal{D} - \mathcal{D}_0 \models \forall \vec{x}.\, F(\vec{x}, do(\langle \alpha, \alpha' \rangle, S_0) \equiv F(\vec{x}, do(\alpha', S_0)).$$

Example 6. Consider the action sequence δ': $\langle set_G, unset_F(c), set_F(c), resetFalse_F \rangle$. Then $set_F(c)$ is always dominated by $resetFalse_F$ in δ wrt to $\mathcal{D}$.

Similar to the analysis of swappable actions, the notions of just-in-time and always dominated actions preserve the entailment of first-order (and second-order) sentences that refer to situations after δ. The next theorem shows this for sentences of the simple projection problem like Theorem 2, and a similar theorem can be obtained for more expressive sentences for the generalized progression problem such as the one addressed in Theorem 3.

Theorem 5. *Let δ be a sequence of ground action terms. Let $\phi(s)$ be a first-order or second-order formula in $\mathcal{L}$ that is uniform in s. If action α is always dominated by α' wrt $\mathcal{D}$ or just-in-time dominated by α' in δ wrt $\mathcal{D}$ then the following holds:*

$$\mathcal{D} \models \phi(do(\delta \cdot \langle \alpha, \alpha' \rangle \cdot \zeta, S_0)) \equiv \phi(do(\delta \cdot \langle \alpha' \rangle \cdot \zeta, S_0)),$$

where ζ is any vector of ground action terms.

Proof. The proof here is similar to that of Theorem 2. For the case of just-in-time dominated, let M be an arbitrary model of $\mathcal{D}$. By induction on the situation terms σ such that $do(\delta, S_0) \sqsubseteq \sigma$, for all fluents F and all sequences of action terms ζ, $M \models F(\vec{x}, do(\delta \cdot \langle \alpha, \alpha' \rangle \cdot \zeta, S_0)$ iff $M \models F(\vec{x}, do(\delta \cdot \langle \alpha' \rangle \cdot \zeta, S_0)$. For the base case we use Definition 11.

The theorem then follows by induction on the construction of the formulas ϕ that are uniform in s.

For the case of always dominated let M be an arbitrary model of $\mathcal{D}$. Since $\mathcal{D} - \mathcal{D}_0$ admits all possible extensions for the fluents in S_0 it follows that there is a model M' of $\mathcal{D} - \mathcal{D}_0$ with the same domain for all sorts as M such that for all μ and for all fluents F, $M, \mu \models F(\vec{x}, do(\delta, S_0))$ iff $M', \mu \models F(\vec{x}, S_0)$. By Definition 11 then it follows that $M \models F(\vec{x}, do(\delta \cdot \langle \alpha, \alpha' \rangle, S_0)$ iff $M \models F(\vec{x}, do(\delta \cdot \langle \alpha' \rangle, S_0)$, from which point we proceed as in the case of just-in-time dominated actions. □

Also it is easy to show that always domination implies just-in-time domination but not the opposite.

Corollary 3. *If action α is always dominated by α' wrt $\mathcal{D}$ then for any sequence of actions δ, α is just-in-time dominated by α' after δ wrt $\mathcal{D}$, but the opposite is not true.*

The following theorem shows an equivalent characterization of always domination of actions in terms of the effects of the actions and their successor state axioms.

Theorem 6. *Action α is always dominated by α' wrt $\mathcal{D}$ iff for all fluents $F(\vec{x}, s)$ in $\mathcal{L}$ the following holds:*

$$\begin{aligned}\mathcal{D} - \mathcal{D}_0 \models \forall \vec{x}\{&\gamma_F^+(\vec{x}, \alpha', S_0) \equiv \gamma_F^+(\vec{x}, \alpha', do(\alpha, S_0)) \\ &\wedge \gamma_F^-(\vec{x}, \alpha', S_0) \equiv \gamma_F^-(\vec{x}, \alpha', do(\alpha, S_0))\} \\ \wedge \forall \vec{x}\{&(\gamma_F^+(\vec{x}, \alpha, S_0) \vee \ \gamma_F^-(\vec{x}, \alpha, S_0)) \\ &\supset (\gamma_F^+(\vec{x}, \alpha', S_0) \vee \gamma_F^-(\vec{x}, \alpha', S_0))\}.\end{aligned}$$

Proof. From Definition 11 and the construction of the successor state axioms. □

Essentially, this theorem requires that removing an action does not change the behaviour of the subsequent action, and that the effects of the dominating

action subsume those of the action being removed. In the case of *context-free* actions such that γ_F^+ and γ_F^- do not depend on s, this is simplified as follows.

Corollary 4. *Let $\mathcal{D}$ be a basic action theory such that all successor state axioms are context-free in the sense that they have the following form: $F(\vec{x}, do(a, s)) \equiv \gamma_F^+(\vec{x}, a) \vee F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a)$. Action α is always dominated by α' wrt $\mathcal{D}$ iff for all fluents $F(\vec{x}, s)$ in $\mathcal{L}$ the following holds:*

$$\mathcal{D} - \mathcal{D}_0 \models \forall\vec{x}\{(\gamma_F^+(\vec{x}, \alpha, S_0) \vee \ \gamma_F^-(\vec{x}, \alpha, S_0)) \\ \supset(\gamma_F^+(\vec{x}, \alpha', S_0) \vee \gamma_F^-(\vec{x}, \alpha', S_0))\}.$$

These notions provide the logical specification for transforming and simplifying action histories in a way that preserves the entailment of first-order projection queries. What becomes interesting then is identifying efficient procedures that can identify when sets of actions can be swapped and when an action can be eliminated for certain classes of action theories $\mathcal{D}$.

3.3 Action theories with single-value fluents and resetting actions

We now turn our attention to basic action theories with relational fluents that have a function-like behaviour in the following sense. For each fluent $F(\vec{y}, s)$, we distinguish the last argument of sort object as the *output* of the fluent and the rest of the arguments of sort object as the *input*. In order to make this explicit we will write $F(\vec{x}, v, s)$ where v is the the output and $\vec{x}$ is input. We will also require that in every model of the action theory for every input $\vec{x}$ there is exactly one object for v (up to equality) such that $F(\vec{x}, v, s)$ holds. We call fluents of this type *single-value fluents*.

Definition 12. *Single-value* fluents are fluents for which the following holds:

$$\mathcal{D} \models \forall \vec{x}, v\{F(\vec{x}, v, s) \supset \neg\exists v'(F(\vec{x}, v', s) \wedge v' \neq v)\}.$$

This is a very common case in many practical domains, such as those involving mobile agents (i.e. a mobile agent can be in only one position in any given situation). We note that a "regular" relational fluent could still be expressed by means of special constants *true* and *false*. We introduce a class of actions which we term *resetting actions* that preserve the property of single-value fluents. The intuition here is that the application of a resetting action "resets" the value of fluent $F(\vec{x}, v, s)$ to a known value v by setting $F(\vec{x}, v, s)$ and unsetting $F(\vec{x}, v', s)$ for all $v' \neq v$. We note that resetting actions are expressive enough to implement many practical domains, such as PDDL planning. As we will see later, they can also be extended to express more complex concepts such as sensing and non-determinism.

Definition 13. The successor state axiom for $F(\vec{x}, v, s)$ has *resetting-actions* iff $\gamma_F^+(\vec{x}, v, a, s)$ is a disjunction of formulas of the form:

$$\exists \vec{z}(a = A(\vec{y}) \wedge v = y_i \wedge \phi(\vec{y}, s)),$$

and $\gamma_F^-(\vec{x}, v, a, s)$ is a disjunction of formulas of the form:

$$\exists \vec{z}(a = A(\vec{y}) \wedge \neg v = y_i \wedge \phi(\vec{y}, s)),$$

where A is an action symbol, $\vec{y}$ contains $\vec{x}$, $\vec{z}$ corresponds to the remaining variables of $\vec{y}$, y_i is a variable in $\vec{y}$, and $\phi(\vec{y}, s)$ (called a *context formula*) is a first-order formula uniform in s, and γ_F^+,γ_F^- come in pairs in the sense that for each disjunct in γ_F^+ there is one in γ_F^- that is identical except for the atom $v = y_i$ and vice versa. As in the case of functional fluents in [80], we also require the

following sentence be entailed by $\mathcal{D}$:

$$\mathcal{D} \models \gamma_F^+(\vec{x}, v, \alpha, s) \supset \neg\exists v'(v' \neq v \wedge \gamma_F^-(\vec{x}, v', \alpha, s)).$$

A basic action theory $\mathcal{D}$ is one of resetting-actions if every SSAs in $\mathcal{D}_{ss}$ is a resetting SSA. Actions in these basic action theories are called *resetting actions*.

We now demonstrate that resetting actions preserve single-value fluents.

Theorem 7. *Let s be a situation, F be a single-value fluent and a be a resetting action. Then F will remain a single-value fluent after $do(a, s)$ is performed.*

Proof. Assume F is a single-value fluent in S_0, therefore $\forall\vec{x}, v\{F(\vec{x}, v, S_0) \supset \neg\exists v'(F(\vec{x}, v', S_0) \wedge v' \neq v)\}$. Consider the induction axiom from $\mathcal{D}_{fnd}$: $\forall P.P(S_0) \wedge (\forall a, s)[P(s) \supset P(do(a, s))] \supset (\forall s)P(s)$. Take P(s) to be $\forall\vec{x}, v\{F(\vec{x}, v, s) \supset \neg\exists v'(F(\vec{x}, v', s) \wedge v' \neq v)\}$ and apply the successor state axioms in Definition 13 □

Note that the positive effects of a resetting SSA are in fact local-effect as they have exactly the same form, requiring that any affected atom is built using arguments of action $A(\vec{y})$, while the negative effects are a special form of global effects. Similarly to [67], the instantiation of a resetting SSA on a ground action term can be simplified using the unique names axioms for actions as follows.

Lemma 1. *Let α be the ground action term $A(\vec{e})$, where $\vec{e}$ is a vector of constants and suppose that the SSA for F is resetting. Then $\gamma_F^+(\vec{x}, v, \alpha, s)$ is logically equivalent to a formula of the following form:*

$$(\vec{x} = \vec{c}_1 \wedge v = d_1 \wedge \phi_1(s)) \vee \cdots \vee (\vec{x} = \vec{c}_m \wedge v = d_m \wedge \phi_m(s)),$$

where each of the $\vec{c}_1, \ldots, \vec{c}_m$ is a vector of constants contained in $\vec{e}$, and $\phi_1(s), \ldots, \phi_m(s)$ are first-order and uniform in s. Each combination of $\vec{c}_i$, $\vec{d}_i$ must be distinct, and for any $\vec{x} = \vec{c}_i$, at most one $\phi_i(s)$ can hold for any model

of s. Similarly, $\gamma_F^-(\vec{x}, \alpha, s)$ is logically equivalent to a formula of the following form:

$$(\vec{x} = \vec{c}_1 \wedge \neg v = d_1 \wedge \phi_1(s)) \vee \cdots \vee (\vec{x} = \vec{c}_m \wedge \neg v = d_m \wedge \phi_m(s)).$$

Proof. Since the SSAs for F are resetting, $\gamma_F^+(\vec{x}, v, a, s)$ is represented by a disjunction of formulas of the form given by Definition 13. Let one of those disjunctions be $\exists \vec{z}(a = A(\vec{y}) \wedge v = y_i \wedge \phi(\vec{y}, s))$, where $\vec{y}$ contains $\vec{x}$, $\vec{z}$ corresponds to the remaining variables of $\vec{y}$, and y_i is a variable in $\vec{y}$. Since action names are unique, $\mathcal{D}_{una} \models \exists \vec{z}(A(\vec{e}) = A(\vec{y}) \wedge v = y_i \wedge \phi(\vec{y}, s))$ iff $\mathcal{D}_{una} \models (\vec{x} = \vec{c} \wedge v = d \wedge \phi(s))$ where $\vec{x} = \vec{c}, v = d$ are contained in $\vec{y} = \vec{e}$. The lemma follows using the same argument for each disjunct. □

We now demonstrate the form of resetting actions using an extended form of Sokoban[105, 46], a popular IPC planning domain.

Example 7. Consider a modified instance of the Sokoban problem such that the surface of any cell can be either clean or oily. Consider three actions: $move(x)$, which moves the player to position x, $push(b, x, y)$, which pushes block b from position x to position y and $clean(x, t)$, which sets the truth value of x being oily to t. We use three fluents: *PlAt*(x, s), which says the player is in position x in situation s, *BlAt*(b, x, s), which says that block b is in position x in situation s and *Oily*(x, t, s), which says the surface of x contains oil where t is a variable that will take an object representing True or False. There are also appropriate $Poss$ axioms for walls, connectivity, and other necessary restrictions.

The successor state axioms for this problem are as follows:

$$\begin{aligned}
&\mathit{PlAt}(v, \mathit{do}(a, s)) \equiv \exists x(a = move(x) \wedge v = x) \\
&\quad \vee \exists b, x, y(a = push(b, x, y) \wedge v = x) \\
&\quad \vee \mathit{PlAt}(v, s) \wedge \neg(\exists x(a = move(x) \wedge \neg v = x) \\
&\quad \vee \exists b, x, y(a = push(b, x, y) \wedge \neg v = x)) \\
&\mathit{BlAt}(b, v, \mathit{do}(a, s)) \equiv \exists x, y(a = push(b, x, y) \wedge v = y) \\
&\quad \vee \mathit{BlAt}(b, v, s) \wedge \neg \exists x, y(a = push(b, x, y) \wedge \neg v = y) \\
&\mathit{Oily}(x, v, \mathit{do}(a, s)) \equiv \exists t(a = clean(x, t) \wedge v = t)) \\
&\quad \vee \mathit{Oily}(x, v, s) \wedge \neg \exists t(a = clean(x, t) \wedge \neg v = t))
\end{aligned}$$

We begin by introducing the notion of the *effect set* of an action. The idea is to represent a set of situation-suppressed ground fluent atoms that will be affected by an action, as well as the conditions under which the fluent atom will be affected. This is analogous to a set containing effect axioms as contained in Reiter [80], but defined specifically for resetting actions. In particular, we need only include the positive effects of an action as the negative effects can be implicitly determined. The effect set is formalised as follows:

Definition 14. Let F be a fluent symbol and α_k be a ground resetting action. Let $\gamma_F^+(\vec{x}, v, \alpha_k, s)$ be the instantiation of $\gamma_F^+(\vec{x}, v, \alpha, s)$ on α_k, simplified according to Lemma 1. The *effect set* of α_k is the following set:

$$\Phi_k = \{\phi \supset F(\vec{c}, d) \mid \vec{x} = \vec{c} \wedge \phi \wedge v = d \text{ appears in } \gamma_F^+(\vec{x}, v, \alpha_k, s)\}.$$

Without loss of generality, we assume that the ground versions of the γ^+ and γ^- formulas are simplified according to Lemma 1. The effect sets can then be obtained directly. We observe the following fundamental property of the effect set:

Lemma 2. *Let $F(\vec{x}, v, s)$ be a single-value fluent with positive and negative successor state axioms $\gamma_F^+(\vec{x}, v, \alpha, s)$ and $\gamma_F^-(\vec{x}, v, \alpha, s)$ respectively. Let M be any model of $\mathcal{D}$. Let α_k be a ground resetting action with effect set Φ_k. Then for all $\vec{x} = \vec{c}, v = d$: $M \models \gamma_F^+(\vec{x}, v, \alpha_k, s)$ iff there exists a $\phi \supset F(\vec{c}, d)$ in Φ_k such that $M \models \phi$.*
$M \models \gamma_F^-(\vec{x}, v, \alpha_k, s)$ iff there exists a $\phi \supset F(\vec{c}, e)$ in Φ_k such that $d \neq e$ and $M \models \phi$.

Proof. From the successor state axioms implied by Lemma 1 it follows that $M \models \gamma_F^+(\vec{x}, v, \alpha_k, s)$ iff $M \models \vec{x} = \vec{c} \wedge v = d \wedge \phi$ for some c, d, ϕ mentioned in $\gamma_F^+(\vec{x}, v, \alpha_k, s)$. From Definition 14, if $\vec{x} = \vec{c_1} \wedge v = d \wedge \phi$ appears in $\gamma_F^+(\vec{x}, v, \alpha_k, s)$ then $\phi \supset F(\vec{c}, d)$ will appear in Φ, and similarly for the negative effects. □

3.3.1 Resetting actions without contexts

We begin by considering a subset of resetting actions where each ϕ_i is $true$. This allows us to reorder or eliminate actions without needing to consider the effects that one action can have on the contexts of the other. We introduce a procedure for determining whether two context-free actions are swappable by directly inspecting their effect sets:

Theorem 8. *Let actions α_1 and α_2 be consecutive resetting actions without contexts with effect sets Φ_1 and Φ_2 respectively. Then α_1 and α_2 are always swappable iff for all elements of Φ_1 of the form $true \supset F(\vec{c}, d_1)$ there is no element of Φ_2 of the form $true \supset F(\vec{c}, d_2)$ such that $d_1 \neq d_2$.*

Proof. From Corollary 2 and Lemma 2. □

We note that in the case of context-free actions, the processes for determining whether two actions are always swappable and just-in-time-swappable are

the same. This is due to the fact that all fluents that are specified in the effect set of an action are guaranteed to have their value set by that action. As a result, it is never necessary to examine the knowledge base to determine the value of a fluent mentioned in the effect set of action α after α has been performed. Formally,

Theorem 9. *Let actions α_1 and α_2 be consecutive resetting actions without contexts with effect sets Φ_1 and Φ_2 respectively. Then α_1 and α_2 are just-in-time swappable iff they are always swappable*

Proof. The only if direction follows directly from Corollary 1. For the if direction, assume there is a domain $\mathcal{D}$ and an action sequence δ such that α_k and α_{k+1} are not always swappable. Then there must exist an element of Φ_1 of the form $true \supset F(\vec{c}, d_1)$ and an element of Φ_2 of the form $true \supset F(\vec{c}, d_2)$ such that $d_1 \neq d_2$. But if such an element exists then Definition 10 no longer holds and actions α_k and α_{k+1} are not just-in-time swappable. □

While this is a restricted class of actions, it is nonetheless sufficient to cover a number of interesting cases, such as the Sokoban example.

Example 8. Consider the SSAs used in Example 7 and a sample action sequence $\langle move(pos_1), clean(pos_1, F)\rangle$. We first calculate the effect sets for each action:

$$\Phi_1 = \{true \supset \textit{PlAt}(pos_1)\}$$
$$\Phi_2 = \{true \supset \textit{Oily}(pos_1, F)\}$$

From Theorem 8 we can observe that $move(pos_1)$ and $clean(pos_1, F)$ are always swappable.

We now introduce a procedure for determining whether one resetting action without a context always dominates another by directly inspecting their effect sets.

Theorem 10. *Let actions α_1 and α_2 be consecutive resetting actions without contexts with effect sets Φ_1 and Φ_2 respectively. Then α_2 always dominates α_1 iff for all elements of Φ_1 of the form $true \supset F(\vec{c}, d_1)$ there exists an element of Φ_2 of the form $true \supset F(\vec{c}, d_2)$.*

Proof. From Corollary 4 and Lemma 2 □

Note that determining whether actions can be just-in-time eliminated purely by inspection of the successor state axioms is not possible. This is because of the possibility that some or all of those fluent atoms already held the same value before the action was taken. In the extreme case, for example, that an action occurred twice and identically set all fluent atoms, the second occurrence of that action could be said to be dominated by *any* subsequent action. To encapsulate this possibility, we introduce the following theorem for determining when an action just-in-time dominates another:

Theorem 11. *Let α_1 and α_2 be consecutive context-free resetting actions with effect sets Φ_1 and Φ_2 respectively and S_k be the situation before α_1 was applied. Then α_2 just-in-time dominates α_1 iff for all elements of Φ_1 of the form $true \supset F(\vec{c}, d_1)$ either there exists an element of Φ_2 of the form $true \supset F(\vec{c}, d_2)$ or $\mathcal{D} \models F(\vec{c}, d_1, S_k)$*

Proof. Consider the action sequence $\langle \alpha_1, \ldots, \alpha_{k-1}, \alpha_k \rangle$. By Lemma 2, for any F, $\vec{x}$ such that $F(\vec{x}, do(\langle \alpha_1, \ldots, \alpha_{k-1}, \alpha_k \rangle, S_0)) \not\equiv F(\vec{x}, do(\langle \alpha_1, \ldots, \alpha_{k-1} \rangle, S_0))$ there will exist en element of Φ_k of the form $true \supset F(\vec{c}, d_1)$ such that $\vec{c}$ is the first elements of $\vec{x}$. Similarly, for any F, $\vec{x}$ such that $F(\vec{x}, do(\langle \alpha_1, \ldots, \alpha_{k-1}, \alpha_k, \alpha_{k+1} \rangle, S_0)) \not\equiv F(\vec{x}, do(\langle \alpha_1, \ldots, \alpha_{k-1}, \alpha_k \rangle, S_0))$ there will exist en element of Φ_{k+1} of the form $true \supset F(\vec{c}, d_2)$ such that $\vec{c}$ is the first elements of $\vec{x}$. After removing α_k, Definition 10 therefore holds for all F, $\vec{x}$ for which $true \supset F(\vec{c}, d_1)$ does not exist in Φ_k such that $\vec{c}$ is the first elements of $\vec{x}$. For each $true \supset F(\vec{c}, d_1)$

in Φ_k, there are two circumstances under which Definition 10 holds: either $F(\vec{x}, do(\langle \alpha_1, \ldots, \alpha_{k-1} \rangle, S_0))$ held where $\vec{c}$ is the first elements of $\vec{x}$ and d_1 is the final element of x, or $true \supset F(\vec{c}, d_2)$ exists in Φ_2. □

Note that this definition is more computationally expensive than that of always-dominating, since establishing entailment of $F(\vec{c}, d_1, S_k)$ is not trivial.

Below is an example demonstrating the idea of always-dominating for resetting actions without contexts.

Example 9. Consider the SSAs used in Example 7 and a sample action sequence $\langle move(pos_1), push(b_1, pos_2, pos_3) \rangle$. We first calculate the effect sets for each action:

$$\Phi_1 = \{true \supset \mathit{PlAt}(pos_1)\}$$
$$\Phi_2 = \{true \supset \mathit{PlAt}(pos_2), true \supset \mathit{BlAt}(b_1, pos_2)\}$$

From Theorem 10 we can observe that $push(b_1, pos_2, pos_3)$ dominates $move(pos_1)$ and thus the action sequence can be simplified to $\langle push(b_1, pos_2, pos_3) \rangle$.

3.3.2 Resetting actions with conjunctive contexts

We now consider a larger class of resetting actions where the context formulas ϕ_i consist of a quantifier-free conjunction of ground fluent atoms. We require that each context formula ϕ_i is satisfiable in the sense that if an element $F(\vec{c}, d)$ is mentioned then neither $\neg F(\vec{c}, d)$ nor $F(\vec{c}, e)$ is mentioned for some $e \neq d$. Reasoning about the properties of these actions is not so straightforward, as determining the value of a context formula for an action requires detailed inspection of the knowledge base. Moreover, the application of one action can affect the contexts of another. We introduce the following sufficient but not necessary method for determining whether a pair of conjunctive resetting actions

are always swappable:

Theorem 12. *Let actions α_1 and α_2 be consecutive conjunctive resetting actions with effect sets Φ_1 and Φ_2 respectively. Then α_1 and α_2 are always swappable if:*

- *For all elements of Φ_1 of the form $\phi_1 \supset F_1(\vec{c_1}, d_1)$, there does not exist an element of Φ_2 of the form $\phi_2 \supset F_2(\vec{c_2}, d_2)$ such that $F_1 = F_2, \vec{c_1} = \vec{c_2}$, $d_1 \neq d_2$, $\phi_1 \wedge \phi_2$ is satisfiable.*

- *For all elements of Φ_1 of the form $\phi_1 \supset F_1(\vec{c_1}, d_1)$, there does not exist an element of Φ_2 of the form $\phi_2 \supset F_2(\vec{c_2}, d_2)$ such that ϕ_2 mentions $F_1(\vec{c_1}, e)$ for some e.*

- *For all elements of Φ_2 of the form $\phi_2 \supset F_2(\vec{c_2}, d_2)$, there does not exist an element of Φ_1 of the form $\phi_1 \supset F_1(\vec{c_1}, d_1)$ such that ϕ_1 mentions $F_2(\vec{c_2}, e)$ for some e.*

Proof. The first point above corresponds with the second condition of Theorem 4 and follows from Lemma 2. For the second condition of Theorem 4, we consider a slightly stronger requirement such that $\gamma_F^+(\vec{x}, v, \alpha_1, S_0) \equiv \gamma_F^+(\vec{x}, \alpha_1, do(\alpha_2, S_0))$ and $\gamma_F^+(\vec{x}, v, \alpha_2, S_0) \equiv \gamma_F^+(\vec{x}, \alpha_2, do(\alpha_1, S_0))$ for all $\vec{c}, v$ and similarly for negative SSAs. From Lemma 2, any $F(\vec{c}, d)$ for which there is no $\phi \supset F(\vec{c}, e)$ in Φ_1 for some e cannot affect $\gamma_F^+(\vec{x}, v, \alpha_2, S_0) \equiv \gamma_F^+(\vec{x}, \alpha_2, do(\alpha_1, S_0))$, and similarly for the other conditions. □

This result is incomplete due to the possibility that for all models, a pair of context formulas in Φ_1 and Φ_2 referring to the same fluent may be unsatisfiable, or that one may imply the other. We expect such cases to be uncommon in practice, however, as in the majority of cases we expect to reorder actions that do not affect the same fluents. We also note that the satisfiability check can be omitted while still retaining a sound result. We now consider the complexity of always-swapping for conjunctive contexts:

Theorem 13. *The complexity of determining whether two conjunctive resetting actions are always swappable is $O(mn)$ where n is the number of elements in the effect set and m is the number of elements in the context formula ϕ. For resetting actions without contexts, this reduces to $O(n)$.*

Proof. Let $\phi_1 \supset F_1(\vec{c_1}, d_1)$ be an element of Φ_1 and $\phi_2 \supset F_2(\vec{c_2}, d_2)$ be an element of Φ_2. The first item in Theorem 12 requires that for each element of the effect sets such that $F_1 = F_2$ and $\vec{c_1} = \vec{c_2}$, the satisfiability of $\phi_1 \wedge \phi_2$ is determined. Since $\phi_1 \wedge \phi_2$ is a conjunction of ground literals, the satisfiability check reduces to table lookup and so can be done in $O(m)$ time. As there are n elements of the effect set, the total time is $O(mn)$. The second and third points require the n instances of $F_1(\vec{c_1}, d_1)$ be checked for membership in the set of $n \times m$ conjunctive literals mentioned in all instances of ϕ_2, with a complexity of $O(mn)$. For resetting actions without contexts, the second and third points are not required, and the first part does not require a satisfiability check. This reduces to $O(n)$, as the problem is essentially checking set membership. □

To demonstrate, we consider a version of the Sokoban example used previously, modified to include conjunctive contexts. Imagine now that pushing a box over an oily surface will cause it to move one additional unit further than expected. We modify the formalism as follows:

Example 10. Let the push action now take five arguments, $push(b, x, y, z, t)$ where t is a variable that will take an object representing a truth value. This action will push block b from position x to position y. If the surface of y is oily, the block will be moved to z instead. The successor state axiom for *BlAt* is now

as follows:

$$
\begin{aligned}
&\mathit{BlAt}(b, v, \mathit{do}(a, s)) \equiv \\
&\exists x, y, z(a = push(b, x, y, z, t) \wedge v = y \wedge \neg\, \mathit{Oily}(y, t, s)) \\
\vee &\exists x, y, z(a = push(b, x, y, z, t) \wedge v = z \wedge \mathit{Oily}(y, t, s)) \\
\vee\, &\mathit{BlAt}(b, v, s) \wedge \neg\{ \\
&\exists x, y, z(a = push(b, x, y, z, t) \wedge \neg v = y) \wedge \neg\, \mathit{Oily}(y, t, s) \\
\vee &\exists x, y, z(a = push(b, x, y, z, t) \wedge \neg v = z \wedge \mathit{Oily}(y, t, s))\}
\end{aligned}
$$

Consider now an action sequence $\langle clean(pos_1, F), push(b_1, pos_2, pos_3, pos_4, T)\rangle$. We firstly calculate the effect sets of the two actions:

$$
\begin{aligned}
\Phi_1 &= \{true \supset \mathit{Oily}(pos_1, F)\} \\
\Phi_2 &= \{\neg\, \mathit{Oily}(pos_3, T) \supset \mathit{BlAt}(b_1, pos_3), \\
&\qquad \mathit{Oily}(pos_3, T) \supset \mathit{BlAt}(b_1, pos_4), \\
&\qquad true \supset \mathit{PlAt}(pos_2)\}.
\end{aligned}
$$

Now using Theorem 12, we can observe that $clean(pos_1, F)$ and $push(b_1, pos_2, pos_3, pos_4, T)$ are always swappable.

We now propose a method for identifying when one resetting action is always dominated by a subsequent action, and can thus be removed.

Theorem 14. *Let α_1 and α_2 be consecutive conjunctive resetting actions with effect sets Φ_1 and Φ_2 respectively. Let $\phi_1[F(\vec{c})]$ refer to the disjunction of all ϕ_1 for which $\phi_1 \supset F(\vec{c}, d)$ is in Φ_1 for any d and $\phi_2[F(\vec{c})]$ refer to the disjunction of all ϕ_2 for which $\phi_2 \supset F(\vec{c}, d)$ is in Φ_2 for any d. Then α_2 always dominates α_1 iff for all $\vec{c}$, $\phi_1[F(\vec{c})] \supset \phi_2[F(\vec{c})]$ is valid and for all elements of Φ_1 of the form $\phi_1 \supset F_1(\vec{c_1}, d_1)$, there is no element of Φ_2 of the form $\phi_2 \supset F_2(\vec{c_2}, d_2)$ such that ϕ_2 mentions $F_1(\vec{c_1}, e)$ for some e and $\phi_1 \wedge \phi_2$ is satisfiable.*

Proof. Focus firstly on the second part of Theorem 6. From Lemma 2,

$\gamma_F^-(\vec{x}, v, \alpha_1, S_0) \vee \gamma_F^+(\vec{x}, v, \alpha_1, S_0)$ iff there exists a $\phi_1 \supset F(\vec{x}, v')$ in Φ_1 such that ϕ_1 holds. Taking together elements of Φ_1 referring to the same F, $\vec{x}$, such a $\phi_1 \supset F(\vec{x}, v')$ exists iff $\phi_1[F(\vec{x})]$ holds. Similarly, $\gamma_F^-(\vec{x}, v, \alpha_2, S_0) \vee \gamma_F^+(\vec{x}, v, \alpha_2, S_0)$ iff there exists a $\phi_2 \supset F(\vec{x}, v')$ in Φ_2 such that ϕ_2 holds. Taking together elements of Φ_2 referring to the same F, $\vec{x}$, such a $\phi_2 \supset F(\vec{x}, v')$ exists iff $\phi_2[F(\vec{x})]$ holds.

Therefore $(\gamma_F^-(\vec{x}, v, \alpha_1, S_0) \vee \gamma_F^+(\vec{x}, v, \alpha_1, S_0)) \supset (\gamma_F^-(\vec{x}, v, \alpha_2, S_0) \vee \gamma_F^+(\vec{x}, v, \alpha_2, S_0))$ is valid iff $\phi_1[F(\vec{c})] \supset \phi_2[F(\vec{c})]$ is valid for all models.

For the first part of Theorem 6, for each $\phi_2 \supset F_2(\vec{c_2}, d_2)$ in Φ_2, consider a model, M such that every $F(\vec{x}, y)$ mentioned in ϕ_2 holds with the exception of some $F(\vec{c}, d)$ for which $\phi_1 \supset F(\vec{c}, d)$ appears in Φ_1. Then $\gamma_F^+(\vec{x}, v, \alpha_2, S_0) \not\equiv \gamma_F^+(\vec{x}, v, \alpha_2, \mathit{do}(\alpha_1, S_0))$ for $x = c_2, v = d_2$ iff ϕ_1 is also satisfiable M. The approach is similar for the case of $F(\vec{c}, e)$ in ϕ_1 as well as for the negative effects. □

We now consider the complexity of always-dominating for conjunctive contexts:

Theorem 15. *The complexity of determining whether one conjunctive-resetting action dominates another is $O(n.2^{mk})$ where n is the number of elements in the effect set, k is the number of times a $\phi \supset F(\vec{c}, d)$ appears in the effect set for fixed $\vec{c}$ but varying d and m is the number of elements in the context formula ϕ. For resetting actions without contexts, this reduces to $O(n)$.*

Proof. The first part of Theorem 14 requires that for each of the n elements of the effect set, the validity of $\phi_1[F(\vec{c})] \supset \phi_2[F(\vec{c})]$ is determined. As $\phi[F(\vec{c})]$ contains $m \times k$ terms, the complexity of determining the validity for each formula of this type is thus $O(2^{mk})$. For the second part, each of the $n \times m$ elements of the contexts in Φ_2 must be checked for membership in Φ_1. If found, the satisfiability of $\phi_1 \wedge \phi_2$ must be determined. Since $\phi_1 \wedge \phi_2$ is a conjunction of literals, the satisfiability check can be done in $O(m)$ time. The total complexity is thus

$O(n.2^{mk}) + O(nm^2)$. For resetting actions without contexts, the second part is not required, and the first part does not require a validity check. This reduces to $O(n)$, as the problem is essentially checking set membership. □

We expect such a method to be feasible in practice, as it is exponential only in the size of the context formulas and the number of times a particular $F(\vec{c})$ appears in the effect set.

We now show an example of dominating conjunctive resetting actions.

Example 11. Using the successor state axioms developed in Example 10, consider now an action sequence $\langle \textit{push}(b_1, pos_1, pos_2, pos_3, T), \textit{push}(b_1, pos_2, pos_3, pos_4, T)\rangle$. We firstly calculate the effect sets of the two actions:

$$\begin{aligned}
\Phi_1 = \{\neg\, \textit{Oily}(pos_2, T) &\supset \textit{BlAt}(b_1, pos_2),\\
\textit{Oily}(pos_2, T) &\supset \textit{BlAt}(b_1, pos_3),\\
true &\supset \textit{PlAt}(pos_1)\}\\
\Phi_2 = \{\neg\, \textit{Oily}(pos_3, T) &\supset \textit{BlAt}(b_1, pos_3),\\
\textit{Oily}(pos_3, T) &\supset \textit{BlAt}(b_1, pos_4),\\
true &\supset \textit{PlAt}(pos_2)\}.
\end{aligned}$$

Now using Theorem 14, $\textit{push}(b_1, pos_2, pos_3, pos_4, T)$ always dominates $\textit{push}(b_1, pos_1, pos_2, pos_3, T)$.

The case of just-in-time swapping is more complex as it is necessary to determine whether contexts are entailed by the knowledge base. It is also no longer sufficient to assume that the presence of a ground fluent atom in a context formula means that atom will affect the result of the context formula. To account for this, we need to determine whether changing the values of a particular set of fluents will affect the results of a context formula. We introduce a function, *aff*, for this purpose:

Definition 15. Let Φ be an effect set containing elements of the form $\phi \supset F(\vec{c}, d)$ and ψ be a conjunction of literals. Let W be the set of elements $F(\vec{c}, d)$ such that $\phi \supset F(\vec{c}, d)$ is in Φ and $\mathcal{D} \models \phi(s)$. For each $F(\vec{c}, d) \in W$, let ψ' be ψ with the following modifications:

- Replace all occurrences of $F(\vec{c}, d)$ in ψ' with $\top$ in ψ'.
- Replace all occurrences of $F(\vec{c}, e)$ where $e \neq d$ with $\bot$ in ψ'.

Then *aff*(Φ, ψ) iff $\mathcal{D} \models \psi \not\equiv \psi'$

Using the procedure specified here for *aff* in conjunction with Theorem 16 below produces a reordering algorithm that is both sound and complete, in the sense that any two resetting actions that are just-in-time swappable according to Definition 6. It is, however, very expensive to compute for conjunctive contexts due to the need to establish entailment of context formulas ϕ. One might also consider a sound but incomplete specification of *aff* that omits these entailment checks. Such a specification would enable just-in-time swappability to be determined in a manner that is sound but incomplete. We introduce the following procedure for determining whether a pair of resetting actions are just-in-time swappable:

Theorem 16. *Let actions α_1 and α_2 be consecutive conjunctive resetting actions with effect sets Φ_1 and Φ_2 respectively. Then α_1 and α_2 are just-in-time swappable iff:*

- *For all elements of Φ_1 of the form $\phi_1 \supset F_1(\vec{c_1}, d_1)$, there does not exist an element of Φ_2 of the form $\phi_2 \supset F_2(\vec{c_2}, d_2)$ such that $F_1 = F_2, \vec{c_1} = \vec{c_2}$, $d_1 \neq d_2$, $\mathcal{D} \models \phi_1 \wedge \phi_2$.*
- *For all elements of Φ_1 of the form $\phi_1 \supset F_1(\vec{c_1}, d_1)$, $\neg$ aff(Φ_2, ϕ_1) or $\mathcal{D} \models F_1(\vec{c_1}, d_1)$*

- *For all elements of* Φ_2 *of the form* $\phi_2 \supset F_2(\vec{c_2}, d_2)$, $\neg$ *aff*(Φ_1, ϕ_2) *or* $\mathcal{D} \models F_2(\vec{c_1}, d_1)$

Proof. Follow the same proof as for Theorem 12, making use of Definition 15 □

We now propose a method for identifying whether one strictly local-effect action is just-in-time dominated by a subsequent action:

Theorem 17. *Let* α_1 *and* α_2 *be two consecutive conjunctive resetting actions with effect sets* Φ_1 *and* Φ_2 *respectively. Let* s_k *be the situation before* α_1 *is applied. Let* Φ_t *be the set of all* $\phi \supset F(\vec{c}, d)$ *in* Φ_1 *such that* $\mathcal{D} \models \phi(s_k)$ *and* $\mathcal{D} \not\models F(\vec{c}, d, s_k)$*. An action can then safely be eliminated if the following conditions hold:*

- *For each element* $\phi_t \supset F(\vec{c}, d_1)$ *of* Φ_t*, there exists a* $\phi_2 \supset F(\vec{c}, d_2)$ *in* Φ_2 *such that* $\mathcal{D} \models \phi_2(s_k)$
- *There is no element in* Φ_2 *of the form* $\phi_2 \supset F(\vec{c}, d)$ *for which aff*(Φ_t, ϕ_2) *is true*

Proof. Follow the proof of Theorem 14 making use of Definition 15 □

For the remainder of this thesis, we restrict ourselves to considering always-swapping and always-dominating actions. We expect these procedures to be far more applicable in practice, as examining the knowledge base to determine whether actions are just-in-time swappable or can be just-in-time eliminated is not feasible in general. As we will see in Chapter 4, the concepts of always-swapping and always-dominating also lend themselves to offline processing.

3.4 Transforming action sequences with sensing

In this section we consider reordering and dominating actions in the presence of sensing actions and incomplete information. Sensing actions are particu-

larly helpful for transforming action sequences, as they naturally dominate other types of actions by introducing new information to an agent's knowledge base. We adopt the semantics of sensing result axioms due to [86], but make an important distinction. Rather than consider an explicit account of knowledge in the epistemic situation calculus, we follow the approach of [25] and [101] and incorporate the sensing result directly into the action theory in the manner of Indigolog [24]. This approach avoids the complications of reordering and eliminating actions in an epistemic formalism while allowing a more straightforward application of our results to Indigolog programs.

We consider here a particular class of sensing actions which can be modelled as resetting actions and introduce a definition of the effect set for sensing actions.

Definition 16. A is a *resetting sensing action* iff it has a sensing-result axiom (SRA) of the form $sr(A(\vec{y}), s){=}r \equiv \bigwedge_i [\phi_i(\vec{y}, r, s) \supset \bigvee_j F_{ij}(\vec{x}_{ij}, z_{ij}, s)]$, where $\{\vec{x}_{ij}, z_{ij}\}{\subseteq}\{\vec{y}, r\}$.

Recall that sensing actions differ from physical actions in that they can only "filter" models of the basic action theory by means of adding an atom of the form $sr(\alpha, \sigma)$ to the theory, hence requiring that the right-hand side of the corresponding SRA holds. We model a sensing action as a set of implications (the conjuncts i), each of which provides under the context condition ϕ_i, disjunctive information about fluent atoms that are built using the arguments of the action and the sensing result. We define the effect set for such actions similarly to physical actions.

Definition 17. Let α be a ground sensing resetting action along with a sensing result. The *effect set* Φ of α is the set of all those formulas $\phi_i \supset \bigvee_{ij} F_{ij}(\vec{c}_{ij}, d_{ij})$ that appear in the instantiated (and simplified similarly as Definition 14) SRA. The *expanded* effect set Φ^* is the set where each element of Φ is broken down to j elements of the form $\phi_i \supset F_{ij}(\vec{c}_{ij}, d_{ij})$.

The resetting sensing actions require special treatment within the existing framework, and except for the normal effect set we will use also expanded version that characterizes all fluent terms that *may* be affected by a sensing action.

Sensing actions can be reordered in a manner similar to physical actions. The key difference is that sensing actions may be reordered even if they affect the contexts of other actions, as a fluent unaffected by action α, that was sensed after performing action α must have held the same value prior to performing α. The following theorems identify sound methods for determining whether two actions can be swapped.

Theorem 18. *Any two consecutive resetting sensing actions are always swappable.*

Proof. From Theorem 4 and Definition 16 □

Theorem 19. *Let action α_1 be a resetting sensing action with expanded effect set Φ_1^*, and α_2 be a (physical) resetting action with effect set Φ_2, that occurs immediately before or after α_1. Then α_1 and α_2 are always swappable if for all elements of Φ_2 of the form $\phi_1 \supset F_1(\vec{c_1}, d_1)$, there does not exist an element of Φ_1^* of the form $\phi_2 \supset F_2(\vec{c_2}, d_2)$ such that $F_1 = F_2, \vec{c_1} = \vec{c_2}$ or s.t. ϕ_1 mentions $F_2(\vec{c_2}, e)$ for some e.*

Proof. From Theorem 4 and Definition 16 □

Theorem 20. *Theorem 14 that specifies the condition according to which α_2 always dominates α_1, holds also with resetting sensing actions when the expanded effect set is used instead, for action α_1.*

Proof. Follow the same proof as for Theorem 14, making use of Theorem 17 □

We now turn our attention to actions with nondeterministic effects. A nondeterministic action can be modeled as corresponding to a set of actions with

deterministic effects, and assuming that the agent does not know which of them was performed [4]. This is typically handled in an epistemic version of situation calculus, but one can also model the notion of the agent *knowing some sentence to hold* in this setting, by means of entailment as follows.

Definition 18. We assume that the domain includes for each fluent $F(\vec{x}, v, s)$ a particular resetting action $A_F(\vec{x}, v)$ with only effect to set the output of F to be v for the input $\vec{x}$. A *widening* ground action α^* is defined as a finite set of resetting actions $\{A_F(\vec{c}, d_1), \ldots, A_F(\vec{c}, d_n)\}$ for some fluent F. For a ground sequence δ^* with widening actions and a FO sentence $\phi(do(\delta^*, S_0))$, we define the *unfolding* of ϕ as the resulting sentence after recursively replacing all subformulas of the form $\phi(do(\alpha^*, \sigma)$ by $\forall a.(a = \alpha_1 \vee \cdots \vee a = \alpha_n) \supset \phi(do(a, \sigma))$, where $\alpha^* = \{\alpha_1, \ldots, \alpha_n\}$ is a widening action.

The notion of the effect set then generalizes naturally to widening actions, and our previous results about always swappable and always dominatable also apply.

Definition 19. The *effect set* Φ of a widening action $\alpha^* = \{\alpha_1, \ldots, \alpha_n\}$ is the union of the effect sets of actions α_i.

Theorem 21. *Widening actions follow the same conditions as physical actions for always swappable and always domination, using the corresponding definition for effect sets.*

Proof. Follow the proof for the relevant condition, making use of Definition 19 □

Our framework then allows for a practical alternative for reasoning with widening actions in the context of regular BATs, as by reordering and eventually eliminating such actions we can in some cases arrive back to having regular action sequences and reason using the standard methods.

Chapter 4

A mechanism for determining an optimal action history

This chapter will examine approaches and metrics that can be used to determine what the optimal action history is for progressing a knowledge base or reasoning via projection, given the possible presence of incomplete information and non-deterministic actions. We formalise the reasoning tasks, and analyse their costs by introducing appropriate heuristics. We also examine restrictions on both the structure of the initial knowledge base and the form of fluents mentioned in projection queries to develop a more efficient action history optimization. This new action history will maintain semantic equivalence with the original action history only for knowledge bases and queries that fall into these restricted cases.

4.1 A rewriting system and search framework

The previous section along with results in [29] show sound methods for reordering and eliminating physical and sensing actions in the case of single-valued fluents. We now look into finding preferred sequence transformations. First, we introduce some notation.

Definition 20. Let δ be an action sequence of the form $\langle \alpha_1, \ldots, \alpha_n \rangle$, and $1 \leq i \leq n-1$. We use r_i to denote the operation of swapping the position of actions α_i and α_{i+1} in δ, and $\mathsf{r}_i(\delta)$ to denote the resulting sequence δ'. Similarly, we use e_i to denote the operation of eliminating action α_i in δ, and $\mathsf{e}_i(\delta)$ for the resulting δ'.

Next, we define an abstract rewrite system for transforming action sequences in the context of a basic action theory.

Definition 21. Let $\mathcal{D}$ be a basic action theory. $\mathcal{A}^{\mathcal{D}} = (A, \rightarrow)$ is an abstract rewrite system, where A is the set of all finite sequences of ground actions of the language of $\mathcal{D}$, and $\rightarrow$ is a binary relation between sequences such that $\delta \rightarrow \delta'$ iff there exists an i, $1 \leq i \leq n-1$, where n is the size of δ, such that $\delta' = \mathsf{r}_i(\delta)$ and actions α_i, α_{i+1} are always swappable in δ wrt $\mathcal{D}$ or $\delta' = \mathsf{e}_i(\delta)$ and action α_{i+1} dominates α_i δ wrt $\mathcal{D}$. Also, $\stackrel{*}{\rightarrow}$ is the reflexive transitive closure of $\rightarrow$.

Essentially $\delta \rightarrow \delta'$ is true when we can perform exactly one step of reordering or eliminating wrt $\mathcal{D}$, while $\delta \stackrel{*}{\rightarrow} \delta'$ is true when δ' can be derived by a finite number of steps. This characterizes a class of sequences that are to be considered equivalent as far as the final state is concerned.

Definition 22. For a sequence δ we define the set *FSE*(δ) of *final-state equivalent* sequences as the set $\{\delta' \mid \delta \stackrel{*}{\rightarrow} \delta'\}$.

Observe that the set $\{\delta' \mid \delta \stackrel{*}{\rightarrow} \delta'\}$ is finite. This is because the permutations of a finite sequence δ are finite and sequences can only shrink to a smaller size, not grow.

Depending on which type of reasoning task we intend to apply over the action sequence, the conditions that identify a preferred sequence may vary greatly. Without yet committing to a particular reasoning task, one natural way to evaluate action sequences is by their size, that is, shorter sequences are to be preferred over longer ones. The best action sequences then are the ones that have minimal size among all those (finitely many) that are final-state equivalent.

Finding a sequence in $FSE(\delta)$ with a global minimal size can be posed as an *optimization problem* where the *objective function* is the size of the sequence. For example, a hill-climbing local search algorithm for this problem that uses the size of sequences as the objective function would work as follows: start from the original sequence δ; at every iteration look into the current sequence δ and all successors δ' such that $\delta \to \delta'$; keep a successor with minimal size smaller than δ and continue; if one does not exist then return δ. Other local search methods could be employed in a similar manner, e.g., simulated annealing.

Observe though that the size of sequences is not a good choice for evaluating the successors toward the path to a global minimal. This is because often it will be needed to perform several reordering steps before an elimination step can take place, which leads to large plateaus. In order to better illustrate the space of heuristic cost functions that can be investigated we introduce the following heuristic.

A better approach is to use a heuristic cost function that evaluates how far we are at making the current sequence shorter. A simple way to quantify this is to look over all pairs of actions in δ such that one can dominate the other and measure how far they are to each other.

Definition 23. We define the set of *dominatable pairs* $Dom(\delta)$ as $\{(\alpha, \alpha') \mid \alpha$ always dominates α' in δ wrt $\mathcal{D}\}$. The *dominatable pairs distance heuristic* h^{e_i} is the function $h^{e_i}(\delta) = \sum_{(\alpha,\alpha') \in Dom(\delta)} dist(\alpha, \alpha')$, where $dist(\alpha, \alpha')$ is the distance of the actions in terms of their index in δ.

Example 12. Consider an action sequence $\delta = \langle \alpha_1, \alpha_2, \alpha_3 \rangle$. Let F and G be fluent symbols and Φ_1, Φ_2, Φ_3 the corresponding effect sets of the actions as

follows:

$$\Phi_1 = \{true \supset F(\vec{x}, v)\},$$
$$\Phi_2 = \{true \supset G(\vec{x}, v)\},$$
$$\Phi_3 = \{true \supset F(\vec{x}, v)\}.$$

Then applying hill-climbing with heuristic h^{e_i} will produce the following transition sequence: $\langle \alpha_1, \alpha_2, \alpha_3 \rangle \rightarrow r_2 \langle \alpha_2, \alpha_1, \alpha_3 \rangle \rightarrow e_1 \langle \alpha_2, \alpha_3 \rangle$, resulting in a sequence of length 2.

While this heuristic can be helpful, hill-climbing with h^{e_i} is nonetheless not optimal. As with any greedy local search algorithm that does not occasionally backtrack to sequences with a lower heuristic cost value, hill-climbing with h^{e_i} may decide to eliminate an action too early. This is illustrated with the following example.

Example 13. Consider a sequence $\delta = \langle \alpha_1, \alpha_2, \alpha_3, \alpha_4 \rangle$. Let F, G, H be fluent symbols and $\Phi_1, \Phi_2, \Phi_3, \Phi_4$ the effect sets of actions as follows:

$$\Phi_1 = \{G(\vec{x}, v) \supset F(\vec{x}, v)\},$$
$$\Phi_2 = \{true \supset G(\vec{x}, v)\}$$
$$\Phi_3 = \{true \supset F(\vec{x}, v)\}$$
$$\Phi_4 = \{true \supset F(\vec{x}, v),$$
$$G(\vec{x}, v) \supset H(\vec{x}, v)\}.$$

Then applying hill-climbing with heuristic h^{e_i} will perform only one transition $\delta \rightarrow e_3 \ \delta'$, resulting in a sequence of length 3. The optimal approach follows the transitions: $\delta \rightarrow r_2 \langle \alpha_1, \alpha_3, \alpha_2, \alpha_4 \rangle \rightarrow e_1 \langle \alpha_3, \alpha_2, \alpha_4 \rangle \rightarrow r_1 \langle \alpha_2, \alpha_3, \alpha_4 \rangle \rightarrow e_2 \langle \alpha_2, \alpha_4 \rangle$, resulting in a length 2. Note that this cannot be obtained by any transformation from δ'.

More sophisticated heuristics can be specified based on the dominatable

pairs, also taking into account the interactions between the pairs through their effect sets. In this way, for instance, an appropriate causal ordering can be defined that requires certain eliminating steps to occur before others, avoiding the problem illustrated in Example 13. For example, the number of fluent atoms that would prevent an action from being reordered if present in the effect set of a second action can be measured. This can be used as a measure of the difficulty of reordering this action, with the heuristic prioritising operations that eliminate difficult actions.

We now turn our attention the effort required to reason and optimize with respect to particular actions.

4.2 A case of DBs with disjunctive information

In the previous section we looked into finding optimal sequences of actions in the sense that their size is minimal. Depending on the reasoning task we want to perform with an action sequence, also the particular actions in the sequence and their order may play an important role on the effort needed for the reasoning task. In order to investigate this further, we look into a particular case inspired by databases.

We look into a particular type of KB that captures disjunctive information, and investigate the impact of manipulating action sequences in the case that progression is the reasoning task we are interested in performing, i.e., updating the KB to the final state after all actions have been performed.

The KB is a *database of possible closures* that explicitly specify a set of possible worlds for single-valued fluents.

Definition 24. Let d be a constant and τ a fluent atom of the form $F(\vec{c}, w, S_0)$, where $\vec{c}$ is a vector of constants and w a variable. We say that τ has the *ground input* $\vec{c}$ and the *output* w. The *atomic closure* χ of τ on d is the sentence: $\forall w.F(\vec{c}, w, S_0) \equiv (w = d)$. The *closure* of vector $\vec{\tau} = \langle \tau_1, \ldots, \tau_n \rangle$ of dis-

tinct fluent atoms on a ground input vector $\vec{d} = \langle d_1, \ldots, d_n \rangle$ of constants is the conjunction $(\chi_1 \wedge \ldots \wedge \chi_n)$, where each χ_i is the atomic closure of τ_i on d_i.

A closure of $\vec{\tau}$ expresses complete information about the output of all input-grounded single-valued fluents in $\vec{\tau}$. For example, consider fluents *RobotAt*(x, s) and *RobotDir*(x, s) that represent information about the location of the robot and the direction it is facing. Let χ_1 be the the atomic closure of *RobotAt*(w, S_0) on loc_1 and χ_2 the atomic closure of *RobotDir*(w, S_0) on *north*, that is:

$$\forall w.\, \textit{RobotAt}(w, S_0) \equiv (w = loc_1), \tag{χ_1}$$

$$\forall w.\, \textit{RobotDir}(w, S_0) \equiv (w = \textit{north}). \tag{χ_2}$$

Then χ_1 and χ_2 express complete information about the location of the robot and the direction it is facing. We can combine closure statements to express incomplete information.

Definition 25. A *possible closures axiom (PCA)* for a vector of input-grounded fluents $\vec{\tau}$ is a disjunction of the form $(\chi_1 \vee \cdots \vee \chi_n)$, where each χ_i is a closure of $\vec{\tau}$ on a vector of constants $\vec{d_n}$ of the same size.

A PCA for $\vec{\tau}$ expresses *disjunctive information* about the output of all fluents in $\vec{\tau}$, by stating how such outputs can be combined together (in n possible ways).

Example 14. Let χ_1 and χ_2 represent as before the atomic closures of *RobotAt*(w, S_0) on loc_1 and *RobotDir*(w, S_0) on *north* respectively. Let χ_3 be the closure of *RobotAt*(w, S_0) on loc_2 and χ_4 the closure of *RobotDir*(w, S_0) on *east*, which both express complete information. Then $((\chi_1 \wedge \chi_2) \vee (\chi_3 \wedge \chi_4))$ is a PCA for $\langle$*RobotAt*(w, S_0), *RobotDir*$(w, S_0)\rangle$ which states two possible combinations for the location and direction.

Using a set of PCAs, each one referring to different input-grounded fluent atoms, we are now able to define the form of the KB we want to consider.

Definition 26. A *database of possible closures (DBPC)* is a set $\mathcal{D}_0 = \{\Xi^{\vec{\tau_1}}, \ldots, \Xi^{\vec{\tau_\ell}}\}$, where each $\Xi^{\vec{\tau_i}}$ is a PCA for $\vec{\tau_i}$ s.t. $\vec{\tau_i} \cap \vec{\tau_j} = \emptyset$, for all distinct $i, j \in \{1, \ldots, \ell\}$. For every i, each disjunct of $\Xi^{\vec{\tau_i}}$ is called a *possible closure* wrt $\mathcal{D}_0$.

So, for every fluent atom τ with a ground input either the output of τ is completely unknown in S_0 (i.e., τ is not mentioned in $\mathcal{D}_0$) or there is just one PCA $\Xi_{\vec{\tau_i}}$ (with $\tau \in \vec{\tau_i}$) that specifies its output value in several possible "partial worlds" (one per disjunct in the PCA). This can be viewed also as a set of regular database tables, each of which holds the possible values for one of the $\vec{\tau_i}$ by means of listing the partial possible worlds as rows. We will refer to the number of ground-input atoms in a PCA as the *width*, and the number of disjuncts in PCA as the *depth* of the PCA.

Note that as the DBPC evolves under the effects of actions, more input-ground fluents may be needed to be put together in a PCA in order to express the resulting state, affecting the size of a PCA through the combinations of values that need to be explicitly represented by means of a cross product. This increase or decrease of the width and depth of the DBPC provides then a means for evaluating action sequences wrt to the effort needed to update to the final state.

Consider an action with contexts that do not mention fluents consisting of disjunctive information. This action can impose complete information about a fluent, removing that fluent from the PCA, which reduces the width of the PCA by 1, as well as potentially the depth, reducing the difficulty of progression. Likewise, consider an action with contexts that depend on fluents with disjunctive information not already mentioned in the PCA. This action will result in the merger of two PCAs, resulting in a single PCA which is larger, increasing the complexity of progression.

Definition 27. Let α be a ground action and Φ the effect set, consisting of ele-

ments of the form $\phi \supset F(\vec{c}, d)$. We define the *progression speedup factor* $h_+(\alpha)$ of α as the total number of elements in Φ such that *(i)* $F(\vec{c}, d)$ appears in a PCA in $\mathcal{D}_0$ with depth greater than 1 and *(ii)* this is not true for any fluent atom mentioned in ϕ. We define the *progression slowdown factor* $h_-(\alpha)$ of α as the total number of fluent atoms mentioned in ϕ in the elements of Φ such that the atom appears in a PCA in $\mathcal{D}_0$ with depth greater than 1.

We note that sensing actions with an equivalent speedup factor are inherently more desirable than physical actions. As well as imposing complete information about a fluent, sensing actions allow all partial worlds inconsistent with the sensed fluent to be removed, reducing the depth of the PCA. This is demonstrated in the following example:

Example 15. Consider the robot described above with the initial situation $\mathcal{D}_0 = \{((\chi_1 \wedge \chi_2) \vee (\chi_3 \wedge \chi_4))\}$. The PCAs for the initial situation can be represented by the following table:

RobotAt	*RobotDir*
loc_1	*north*
loc_2	*east*

After performing the action *moveTo*(loc_3) in the initial situation, the PCAs can be represented by the following table:

RobotAt	*RobotDir*
loc_3	*north*
loc_3	*east*

Now allow an action *senseLocation*, which returns the current location of the robot. If the robot is initially at loc_1, then the PCAs after performing this action in the initial situation can be represented by the following table:

RobotAt	*RobotDir*
loc_1	*north*

Example 16. Consider the robot described above, along with another fluent *Holding*(x, s) which represents information about an object the robot is holding. Let χ_5 be the atomic closure of *Holding*(x, S_0) on *drill* and χ_6 be the atomic closure of *Holding*(x, s_0) on *spade*. Let $(\chi_5 \vee \chi_6)$ be a PCA for *Holding*(x, S_0). Then Actions *moveTo*(loc_3) and *turnTo*$(south)$, which move and turn the robot to a specified position and direction respectively, both have a speedup factor of 1. Action *moveIfHolding*(x, y, S_0), which moves the robot to location x if it is holding a required object y, has a progression slowdown factor of 1.

Note that the notions of speedup and slowdown we defined can help distinguish what should be preferred between action sequences of the same size. In particular we can specify heuristic cost functions for identifying the best permutation of a sequence that cannot be simplified further by elimination steps. As this relates to reordering actions, we use the r_i notation to denote the heuristic functions.

Definition 28. The following are heuristic cost functions based on the effort imposed by intermediate updates when progressing to the final state:

- *speedup heuristic* $h_+^{r_i}(\delta) = \sum_{\alpha \in \delta} -h_+(\alpha)$,
- *slowdown heuristic* $h_-^{r_i}(\delta) = \sum_{\alpha \in \delta} h_-(\alpha)$,
- *balanced heuristic* $h_b^{r_i}(\delta) = \sum_{\alpha \in \delta} h_-(\alpha) - h_+(\alpha)$.

Hill-climbing using these heuristics is not optimal, however. A key observation is that they take into account only of disjunctive information in the initial state, rather than considering the evolution of disjunctive information as actions occur in the sequence. A more sophisticated approach would keep track of the fluents mentioned in the PCAs and compare these to the fluents mentioned in the effect set.

Example 17. Consider a simplified version of the Wumpus World domain [83, Chapter 7] with a *senseBreeze*(y_1, y_2) action, which senses whether one

of two surrounding locations contains a pit, and a *sensePit*(y) action, which senses whether a particular location contains a pit. Allow fluent *IsPit*(x, s) to capture whether (from the point of view of the agent) that x contains a pit, and let the initial situation be represented by the PCAs $(\chi_1), (\chi_2), (\chi_3), (\chi_4)$, with each χ_i being the closure of *IsPit* on loc_i. Now consider the sequence $\langle$*senseBreeze*(loc_1, loc_2), *sensePit*$(loc_1)\rangle$ and assume that both sensing actions return $true$. Progressing wrt the first action leads to the following DBPC, which contains a single PCA shown as a table.

IsPit(loc_1)	*IsPit*(loc_2)
$true$	$true$
$true$	$false$
$false$	$true$

According to Definition 28, *senseBreeze*(loc_1, loc_2) has a speedup heuristic value of 0 and a slowdown heuristic value of 1, while *sensePit*(loc_1) has a speedup of -1 and a slowdown 0. This motivates us to consider reordering the *sensePit*(loc_1) actions towards the start of the action sequence using Theorem 18. This leads to the sequence $\langle$*sensePit*(loc_1), *senseBreeze*$(loc_1, loc_2)\rangle$, which is final-state equivalent wrt Definition 7. The DBPC after the first action in this sequence can be represented as follows (which happens to be equivalent to the DBPC in the final situation):

IsPit(loc_1)
$true$

IsPit(loc_2)
$true$
$false$

As expected, the DBPC for the final situation is the same for both sequences. Note though that for the reordered sequence, table sizes are maintained between 1×1 and 1×2, as compared with the 2×3 table produced for the original sequence, with a corresponding reduction in the cost progression.

4.3 An offline precomputing step

We now examine the approach of precomputing actions off-line, in order to determine whether any pair of actions can be reordered or eliminated. This approach leads to a considerable increase in the efficiency of the online reasoning task, at the cost of producing and maintaining a version of the effect sets for each pair of action symbols in the domain. As an added benefit, the expensive offline reasoning need only be executed once for every pair of action symbols in the domain, rather than for every pair of ground actions in an action sequence. We take each pair of nonground action symbols in the domain, and reformulate the γ^+, γ^- formulas of SSAs according to the following lemma:

Lemma 3. *Let α be the nonground resetting action $A(\vec{y})$. Then, for each action symbol A in the domain, $\gamma_F^+(\vec{x}, v, \alpha, s)$ is logically equivalent to a formula of the form $(\vec{x} = \vec{w}_1 \wedge v = z_1 \wedge \phi_1(s)) \vee \cdots \vee (\vec{x} = \vec{w}_m \wedge v = z_m \wedge \phi_m(s))$, where each of the $\vec{w}_1, \ldots, \vec{w}_m$ is a vector of variables contained in $\vec{y}$, each z_i is a variable in $\vec{y}$ and $\phi_1(s), \ldots, \phi_m(s)$ are first-order and uniform in s. Each combination of $\vec{w}_i$, $\vec{z}_i$ must be distinct. Similarly, $\gamma_F^-(\vec{x}, \alpha, s)$ is logically equivalent to a formula of the form $(\vec{x} = \vec{w}_1 \wedge \neg v = z_1 \wedge \phi_1(s)) \vee \cdots \vee (\vec{x} = \vec{w}_m \wedge \neg v = z_m \wedge \phi_m(s))$.*

Proof. Since the SSAs for F are resetting, $\gamma_F^+(\vec{x}, v, a, s)$ is represented by a disjunction of formulas of the form given by Definition 13. Let one of those disjunctions be $\exists \vec{z}(a = A(\vec{y}) \wedge v = y_i \wedge \phi(\vec{y}, s))$, where $\vec{y}$ contains $\vec{x}$, $\vec{z}$ corresponds to the remaining variables of $\vec{y}$, and y_i is a variable in $\vec{y}$. Since action names are unique, there exists a variable assignment μ such that $\mathcal{D} \models \exists \vec{z}(a = A(\vec{y}) \wedge v = y_i \wedge \phi(\vec{y}, s))$ iff $\mathcal{D} \models (\vec{x} = \vec{w} \wedge v = z \wedge \phi(\vec{w}, v, s))$ where $\vec{w}, z$ are contained in $\vec{y}$. The lemma follows using the same argument for each disjunct. □

The form here is similar to that of Lemma 1 for ground action terms, but here represents a syntactic translation of the successor state axioms into a form

that allows an effect set to be determined. We now define a variation of the effect sets that encapsulates the same property for nonground actions:

Definition 29. Let F be a fluent symbol and α be an nonground resetting action of the form $A(\vec{y})$. Let γ_F^+ and γ_F^- be the positive and negative successor state axioms for F respectively, simplified according to Lemma 3. Let ϕ be the situation-suppressed version of formula $\phi(s)$. The *effect set* Φ of α is the set: $\{\phi \supset F(\vec{x}, v) \mid \vec{x} = \vec{w} \wedge \phi \wedge v = z \text{ appears in } \gamma_F^+(\vec{x}, v, \alpha, s)\}$.

We are now able to split the process of determining whether two actions can be reordered into two parts: First, a more complex off-line precomputing step that produces a set of equality relations between variables mentioned in the action terms, which we call the *online determination set (ODS)*. Second, a comparatively simple online procedure that verifies whether the equality relations hold. For resetting actions without contexts, determining whether two actions can be reordered proceeds as follows:

Theorem 22. *Let actions α_1 and α_2 be consecutive nonground resetting actions without contexts with effect sets Φ_1 and Φ_2 respectively. For each element of Φ_1 of the form $true \supset F(\vec{w_1}, z_1)$, if there exists an element of Φ_2 of the form $true \supset F(\vec{w_2}, z_2)$, add $\vec{w_1} = \vec{w_2} \supset z_1 \neq z_2$ to the ODS. Then α_1 and α_2 are always swappable if all formulas in the ODS hold after grounding.*

Proof. Follow the proof of Theorem 8 making use of Definition 29 □

For *conjunctive resetting actions*, which have contexts that are comprised of quantifier free conjunctions of fluent atoms, the process for determining whether two actions can be reordered or eliminated is more complex. This is because the outcome of one action can affect the context of another. As a result, the potential benefits from offline precomputing are far more significant. We firstly define a set of inequalities between vectors of arguments, $arg(F, \vec{x}, \phi)$, which ensures that fluent $F(\vec{x}, v, s)$ does not appear in ϕ: We note that these results are similar for resetting sensing actions.

Definition 30. Let F be a fluent symbol, $\vec{x}$ be a vector of variables and ϕ a situation-suppressed conjunctive context formula. Then *arg*$(F, \vec{x}, \phi)$ is the smallest set such that for all $\vec{y}, v$ if $F(\vec{y}, v)$ appears in ϕ then $\vec{y} \neq \vec{x}$ is in *arg*$(F, \vec{x}, \phi)$.

Theorem 23. *Let actions α_1 and α_2 be consecutive conjunctive resetting actions with effect sets Φ_1, Φ_2. Let Φ_1 contain elements of the form $\phi_1 \supset F_1(\vec{w_1}, z_1)$ and Φ_2 contain elements of the form $\phi_2 \supset F_2(\vec{w_2}, z_2)$. Then the offline pre-computing step is as follows: For all combinations of elements in Φ_1 and Φ_2, add arg$(F_1, \vec{w_1}, \phi_2)$ and arg$(F_2, \vec{w_2}, \phi_1)$ to the ODS. If $F_1 = F_2$, also add $\vec{w_1} = \vec{w_2} \supset z_1 \neq z_2$ to the ODS. Then α_1 and α_2 are always swappable if all formulas in the ODS hold after grounding.*

Proof. Follow the proof of Theorem 12 making use of Definition 29. □

Example 18. Consider the modified Sokoban example presented in Example 10. Consider the action symbols $a_1 = clean(w_{11}, w_{12})$ and $a_2 = push(w_{21}, w_{22}, w_{23}, w_{24}, w_{25})$. According to Definition 29, the effect sets for these actions are as follows:

$$\begin{aligned}\Phi_1 &= \{true \supset Oily(w_{11}, w_{12})\} \\ \Phi_2 = \{\neg Oily(w_{23}, w_{25}) &\supset BlAt(w_{21}, w_{23}), \\ Oily(w_{23}, w_{25}) &\supset BlAt(w_{21}, w_{24}), \\ true &\supset PlAt(w_{22})\}.\end{aligned}$$

Applying Theorem 23 yields the following ODS:

$$\{arg(Oily, w_{11}, \neg Oily(w_{23}, w_{25})), \quad arg(Oily, w_{11}, Oily(w_{23}, w_{25}))\}$$

Expanding using Definition 30 yields the following

$$\{w_{11} \neq w_{23}, w_{11} \neq w_{23}\}$$

Which simplifies to just

$$\{w_{11} \neq w_{23}\}$$

For the online component, let the ground actions be $a_1 = clean(pos_1, F)$ and $a_2 = \textit{push}(b_1, pos_2, pos_3, pos_4, T)$. After grounding, the ODS is:

$$\{pos_1 \neq pos_3\}$$

Which holds in the presence of the unique name axioms. These two actions are therefore swappable by Theorem 23.

Now, to eliminate α_1 it is necessary to establish that the eliminating action α_2 *will* set the value of all fluents that *could* have been set by a_1. That is, the context of α_1 implies the context of α_2 for all fluents. To facilitate offline precomputing for this task, we define a procedure *eval*(ϕ), which establishes conditions between variables so that ϕ is a valid formula. Note that *eval*(ϕ) can be determined using an automated theorem prover.

Definition 31. Let ϕ be a quantifier-free formula. *eval*(ϕ) is disjunction of all unifications expressed as equality relations between variables mentioned in ϕ, for which ϕ is a logically valid formula

For example, *eval*(*Oily*$(w_1) \vee \neg$ *Oily*(w_2)) would return $w_1 = w_2$.

Theorem 24. *Let action α_1 and α_2 be consecutive conjunctive resetting action with effect sets Φ_1 and Φ_2 respectively. Let $\phi_1[F(\vec{w})]$ refer to the disjunction of all ϕ_1 for which $\phi_1 \supset F(\vec{w}, z)$ is in Φ_1 for any z, and $\phi_2[F(\vec{w})]$ refer to the disjunction of all ϕ_2 for which $\phi_2 \supset F(\vec{w}, z)$ is in Φ_2 for any z. Then the offline precomputing step is as follows:*

- *For all $\vec{w}$, add eval*$(\phi_1[F(\vec{w})] \supset \phi_2[F(\vec{w})])$ *to the ODS.*

- *For all elements of* Φ_1 *of the form* $\phi_1 \supset F(\vec{w_1}, z_1)$*, for each element of* Φ_2 *of the form* $\phi_2 \supset F(\vec{w_2}, z_2)$ *add* $arg(F, \vec{w_1}, \phi_2)$ *to the ODS.*

Then α_1 *can be eliminated if all formulas in the ODS hold after grounding.*

Proof. Follow the proof of Theorem 14 making use of Definitions 29 and 31 □

Example 19. Using the modified version of Sokoban introduced in Example 10, consider now action symbols $a_1 = \textit{push}(w_{11}, w_{12}, w_{13}, w_{14}, w_{15})$ and $a_2 = \textit{push}(w_{21}, w_{22}, w_{23}, w_{24}, w_{25})$. According to Definition 29, the effect sets for these actions are as follows:

$$\begin{aligned}
\Phi_1 = \{\neg\, \textit{Oily}(w_{13}, w_{15}) &\supset \textit{BlAt}(w_{11}, w_{13}),\\
\textit{Oily}(w_{13}, w_{15}) &\supset \textit{BlAt}(w_{11}, w_{14}),\\
true &\supset \textit{PlAt}(w_{12})\}\\
\Phi_2 = \{\neg\, \textit{Oily}(w_{23}, w_{25}) &\supset \textit{BlAt}(w_{21}, w_{23}),\\
\textit{Oily}(w_{23}, w_{25}) &\supset \textit{BlAt}(w_{21}, w_{24}),\\
true &\supset \textit{PlAt}(w_{22})\}.
\end{aligned}$$

Applying Theorem 24, Definition 30 and Definition 31 then simplifying yields the following ODS:

$$\{w_{11} = w_{21}\}$$

For the online component, let the ground actions be $a_1 = \textit{push}(b_1, pos_1, pos_2, pos_3, T)$ and $a_2 = \textit{push}(b_1, pos_2, pos_3, pos_4, T)$. After grounding, the ODS is:

$$\{b_1 = b_1\}$$

Therefore the ODS holds for these actions. Action a_1 can therefore be elimi-

nated by Theorem 23.

Theorem 25. *Determining whether two actions can be reordered or eliminated can be done in time linear in the size of the ODS after off-line precomputing has been completed.*

Proof. Definitions 29 and 31 require the ODS to be expressed as a set of equality relations between ground object terms. The task of verifying the ODS therefore reduces to that of evaluating each relation. □

4.4 Basic optimization scenarios

We now give some insights about the effectiveness of these technique using two well-known domains, namely Sokoban [105] and the Wumpus World [83, Chapter 7]. Progression and action sequence optimization are performed using a C program that runs a greedy search over the space of possible action sequence manipulations, using a combination of the dominatable pairs and progression speedup heuristics defined earlier, with a horizon of 10 actions. The offline precomputing step is also employed to simplify the effort required to determine action sequence modifications. Logical reasoning is performed using an Indigolog program [24].

We follow the modelling of Sokoban as a planning domain [46] and generate long action sequences of length 5000 actions, by employing a simple agent whose goal is to avoid repeated states. For the Wumpus World we consider action sequences generated by the agent described in [85] in maps of size 8×8 and 10×10. Note that while Sokoban is well suited for using progression for query answering, the Wumpus World, due to the type of incomplete information that needs to be represented, is well suited for using regression. As we want to evaluate our framework over both approaches, we solve projection queries via progression in Sokoban and via regression in the Wumpus World.

We formalise the domains using resetting actions and perform reasoning for answering a projection query that attempts to 'map' the domain by identifying the locations of all known blocks/pits. For each action, the Indigolog program implements a *Poss* axiom, providing the conditions under which the action can be executed, as well as one or more effect axioms. For example, the *moveFwd* action is specified as follows:

```
poss(moveFwd(L), neg(offTheEdge(L))).
causes(moveFwd(X), locRobot, Y, Y=X).
```

The top level control program is implemented as a series of prioritised interrupts which specify the agent behaviour. Each interrupt consists of a guard or condition under which it can be executed, and a small program to run. *locRobot*, *aliveWumpus* and *noGold* are all fluents in the domain.

```
proc(mainControl,
 prioritized_interrupts(
   [interrupt([dir], and(aliveWumpus=true,
      in_line(locRobot,dir,locWumpus)), [shoot(dir)] ),
   interrupt(isGold(locRobot)=true, [pickGold(locRobot)]),
   interrupt(inDungeon=true,
    if(noGold>0,[goto(loc(1,1)),climb],
          [smell(locRobot),
           senseBreeze(locRobot),
           senseGold(locRobot),
           explore_grid
          ]))
    ])
).
```

We report the runtime following two approaches:

- Applying the reasoning method (regression or progression) over the orig-

inal action sequence in order to answer the projection query, and

- Optimizing first the sequence before applying the reasoning method.

The results are reported in Tables 4.1 and 4.2. Columns *opt* and *query* represent the average time in milliseconds taken over 10 sequences.

Sequence	opt.	query	total	length
Original Progressed	-	1.45	1.45	5000
Optimized Progressed	0.15	0.00	**0.15**	**3**

Table 4.1: Experimental results for Sokoban

Sequence	opt.	query	total	length
Original Regressed (8×8)	-	17.2	17.2	104
Optimized Regressed (8×8)	0.3	15.1	**15.4**	**77**
Original Regressed (10×10)	-	41.0	41.0	167
Optimized Regressed (10×10)	0.6	37.2	**37.8**	**125**

Table 4.2: Experimental results for Wumpus World

We observe that the structure of Sokoban permits many elimination operations, with a substantial resultant speedup, that is in fact faster than progression. This is a very interesting result as progression works extremely well for planning domains like Sokoban. For Wumpus World, none of the actions affecting the agent's knowledge of pit locations could be eliminated, hence the performance improvement was far less significant. In a richer setting with overlapping sensing information and physical action, this could prove much more beneficial, drastically and efficiently simplifying the action sequence to a minimal length.

Chapter 5

Conclusion

5.1 Related Work

To the best of our knowledge this is the first approach that looks into formalizing operations over an action history in a rich first-order setting that preserves truth of (simple and generalized) projection queries in the final situation. Nonetheless some related techniques have been explored to improve the performance of reasoning in other settings.

Research has been conducted into eliminating redundant actions in the classical planning domain, in order to improve the efficiency of a satisficing plan found using a tool such as FF [47], Fast Downward [45] or LPG [41]. For example, Chrpa [15] identifies pairs of actions as being either *independent*, *shared*, *nested* or *interleaved*, and goes on to define conditions under which these different types of actions can be removed from a sequence. Balyo [9] encodes the task of determining an improved plan as a weighted MaxSAT problem, using a plan reduction algorithm similar in spirit to our hill-climbing approach. Working in the classical planning domain, however, allows add and delete lists to be used to consider a clear and finite set of effects for each action, which is not the case in more expressive first-order formalisms such as the Situation Calculus. These approaches are also intended to create a realisable plan for execution, thus re-

quiring that the preconditions of each action remain satisfied. By focusing on optimizing an already executed sequence in order to facilitate projection, we are able to bypass these constraints, permitting a much larger range of sequence modifications. Finally, Chrpa does not consider the possibility of reordering actions.

Löwe *et al.* [68] also considers similar properties in the context of Dynamic Epistemic Logic planning. They define event models as being *self-absorbing* or *commutative*, and demonstrate tractability results for models of this type. Löwe's self-absorbing actions corresponds to a special case of our dominatable actions whereby both the dominatable and the dominating action are the same, while commutative actions are similar to our reorderable actions. Yu *et al.* [106] use these same notions to demonstrate the decidability of explanatory diagnosis for multi-agent systems under certain conditions.

Bäckström [6] takes an intuitively similar approach for solving partial-order planning problems. He introduces the ideas of *deorderings* and *reorderings* and examines the properties of plans that enable actions to be reordered or dependencies between actions removed. In more recent work, Aghighi and Bäckström [1] use similar notions of dominating and reordering to optimize action sequences for partial order planning. They introduce algorithms for determining minimum plan length and prove complexity results for their algorithms. Muise et. al [72] implement a similar system and demonstrate its efficiency for finding optimal reorderings. Their work, however, is restricted to classical planning instances and does not consider first-order formalisms such as the situation calculus. In the first-order planning space, both Gretton and Thiébaux [43] and Sanner and Boutilier [84] use regression in the situation calculus to develop policies for solving larger planning instances. Our techniques could potentially be beneficial in improving the performance of this process.

Our analysis adopts a view similar to a database, also with disjunctive information about values of tuples. We show how a local search approach can

be used to optimize the size and the effort required to progress the current state after each action step. Sensing actions, that have been incorporated into the previous framework, play an important role in this process. The state representation used is similar in expressiveness to other approaches such as the so-called and *proper*$^+$ KBs [67], but we note that we aimed for a setting that is as close to the database case as possible, which allows to reuse database functionality in the implementation of reasoning systems.

5.2 Future Work

One direct line of research is to incorporate additional classes of action into the reordering and dominating frameworks. For example, many classes of *global-effect* actions are inherently problematic with regards to progression as they may affect infinitely many ground fluent terms. While it may not be possible to positively identify exactly all the possible fluents that will be affected by these actions, it is often possible to identify a wider set of fluents that *may* be affected by them, or at least a set of fluents that will *not* be affected. These properties can facilitate reordering, allowing some global-effect actions to be moved later on in the action sequence. This, in turn, can enable the knowledge base to be progressed further.

On a related note, there often occur situations in which an individual action many not dominate a previous action, but a contiguous sequence of actions performed together may do so. Extending the formalism to handle sequences of actions is not terribly difficult. The sequence may be replaced with a macro-action [44], the effect set of which can be calculated from the individual actions with a little extra syntactic machinery. From an efficiency standpoint, however, comparing all possible sequences is not generally viable. Nonetheless, dominating sequences of actions often occur in predictable patterns. Incorporating domain control knowledge may enable meaningful comparisons to be made,

allowing further redundant actions to be removed from the sequence.

While we are able to show significant improvements to the cost of reasoning by using our planning framework, the h^{e_i} and h^{r_i} heuristics only encapsulate elements of the cost of performing actions. A thorough investigation of the joint use of h^{e_i} and h^{r_i} heuristics would be valuable future work in order to construct combined heuristics. Similarly, an evaluation of search systems other than the hill-climbing algorithm implemented could be conducted to further reduce the costs of the modified action sequences. In practical cases, action selection would be interleaved with sequence reduction to prevent the length of action sequences getting out of hand. There are many parameters to tune, such as the number of action selection steps to take before recalculating the previous sequence and the details of the planning heuristic, all of which effect the total cost of execution. The optimal approach to take is quite domain specific, and care must be taken to ensure that the cost of performing action sequence reduction does not exceed the cost of reasoning over the original action sequence. A thorough tuning of these parameters to popular domains would allow these techniques to be implemented more effectively.

As with most established results in the literature, our work requires reasoning tasks be performed over the entire knowledge base. Amir [2, 3] shows a mechanism for decomposing a situation calculus initial knowledge base into smaller fragments. Reasoning tasks such as projection can be performed within a single decomposed knowledge base, without affecting the consistency of other parts of the decomposition. Ponomarov and Soutchanski [78] extend this work using established mechanisms for performing Δ-decompositions in description logic ontologies[54, 53], in order to develop a progression mechanism that can work over decomposed knowledge bases. The h^{e_i} and h^{r_i} heuristics we have developed could be used to inform the selection of different reasoning techniques for different parts of the decomposition. For example, a segment of the decomposition that involves global-effect actions for which no known progression

mechanism exists could have query answering handled via regression, while other parts of the knowledge base are progressed independently in accordance with the results of Chapter 4. The process for doing so is not obvious, however, as the results of [78] do not imply any particular syntactic form for the decomposed knowledge base, even if the form of the original knowledge base is known. This makes the process of determining the effect sets and the h^{e_i} and h^{r_i} heuristics difficult, and further research is needed to develop restrictions on $\mathcal{D}$ that enable this decomposition to occur in a well-structured manner.

Recently, a number of techniques have been developed to enable tractable reasoning in wider fragments of the situation calculus. These include the so-called *bounded action theories* that incorporate a type of incomplete information [23, 22], approaches designed to handle belief revision [89, 91] and non-deterministic actions [32]. Since our approach is designed to improve the efficiency of reasoning by operating over an action sequence rather than replacing the reasoning module itself, it has great potential to be adapted to work with these new techniques to further reduce the cost of reasoning over long action sequences in the situation calculus.

Taken together, these techniques have significant potential benefits for optimising projection (search) in non-deterministic programming languages, such as Golog. Most current Golog implementations retain a complete action history and use regression to handle projection queries. An interpreter could be designed to make periodic calls to an action sequence optimizer, returning a new action sequence that is easier to reason with. The optimizer could either be called at fixed intervals, or prior to performing queries known to be expensive, such as non-deterministic search operations.

Notice ideas presented here could also be applied in other action formalisms such as the fluent calculus [93] and the event calculus [56, 90], as the relationships between these formalisms are well studied [11, 12, 55]. Indeed, our work shares some similarity to those that utilise constraints solvers, such as the `Flux`

system for implementing fluent calculus theories [94] which uses (among other things) constraints to represent the effects implied to fluents with arithmetic arguments. In `Flux` these constraints specify how the old value of fluents relates to the new one, and are appended to the constraint store as the action history grows. Periodically, `Flux`'s constraint store invokes heuristic techniques for simplifying the history by eliminating redundant constraints. Our work could be also used to advantage in this setting to optimise action histories in more informed ways inspired by temporal action properties, e.g., the effect of sensing, instead of more low-level constraints. From an optimisation perspective, symmetry breaking, and the related issue of planning landmarks, have also been shown to be promising for achieving efficiency in planning domains [26, 104]. Interestingly, dominance relations are known to entail the special case of symmetry in constraint programming [16]. They have been shown to have benefits in the development of partial-order plans [27, 13], model checking [28, 17, 71] and execution monitoring [73]. Dominance relations might therefore form a basis of a range of constraint optimisation techniques for action languages.

5.3 Contributions

This work formalizes the transformation of theories of action in order to optimise frequently executed reasoning tasks. The transformation allows action sequences to be shortened, via the swapping and reordering of independent actions and elimination of dominating actions. This can facilitate easier, more optimised, forms of reasoning. Our techniques are potentially applicable to a wide range of rich first order theories.

We have shown how transforming action sequences facilitates a rewriting and local search setting where the effort for updating can be based on notions of *progression speedup* and *progression slowdown*, and the minimal size of the sequence can be estimated using the notion of *dominatable pairs*. The formal-

isation of these notions allows for a variety of guided search strategies to be employed across the space of action sequence manipulations, enabling efficient alternative sequences to be found.

The framework incorporates sensing and widening actions, the effects of which strongly motivate reordering and elimination of actions. This work sets the ground for developing novel optimization strategies and hybrid approaches in reasoning about action, for utilisation in synchronous, concurrent and even asynchronous reasoning tasks.

We have further demonstrated a process for partially determining action sequences transformations offline, enabling an agent to more efficiently compute desirable transformations. For long running executions, such as agents operating over protracted time periods, our approach provides a mechanism for answering queries efficiently over the large action sequences generated.

Bibliography

[1] AGHIGHI, M., AND BÄCKSTRÖM, C. Plan reordering and parallel execution a parameterized complexity view, 2017.

[2] AMIR, E. (de)composition of situation calculus theories. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence* (2000), AAAI Press, pp. 456–463.

[3] AMIR, E. Projection in decomposed situation calculus. In *Proceedings of the Eights International Conference on Principles of Knowledge Representation and Reasoning* (San Francisco, CA, USA, 2002), KR'02, Morgan Kaufmann Publishers Inc., pp. 315–326.

[4] BACCHUS, F., HALPERN, J. Y., AND LEVESQUE, H. J. Reasoning about noisy sensors and effectors in the situation calculus. *Artificial Intelligence 111*, 1 (1999), 171 – 208.

[5] BACCHUS, F., AND KABANZA, F. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence 22*, 1 (Feb 1998), 5–27.

[6] BÄCKSTRÖM, C. Computational aspects of reordering plans. *J. Artif. Intell. Res. 9* (1998), 99–137.

[7] BAIER, J., AND MCILRAITH, S. On planning with programs that sense. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR06)* (Lake District, UK, June 2006), pp. 492–502.

[8] BAIER, J. A., FRITZ, C., AND MCILRAITH, S. A. Exploiting procedural domain control knowledge in state-of-the-art planners. In *Proceedings of the 17th International Conference onAutomated Planning and Scheduling (ICAPS)* (2007), pp. 26–33.

[9] BALYO, T., CHRPA, L., AND KILANI, A. On different strategies for eliminating redundant actions from plans. In *Proceedings of the Seventh Annual Symposium on Combinatorial Search*, S. Edelkamp and R. Barták, Eds., SoCS 2014. AAAI Press, Prague, Czech Republic, September 2014, pp. 10–18.

[10] BARAL, C., AND SON, T. C. Extending congolog to allow partial ordering. In *Intelligent Agents VI. Agent Theories, Architectures, and Languages* (Berlin, Heidelberg, 2000), N. R. Jennings and Y. Lespérance, Eds., Springer Berlin Heidelberg, pp. 188–204.

[11] BELLEGHEM, K. V., DENECKER, M., AND SCHREYE, D. D. Combining situation calculus and event calculus. In *Proceedings of the 12th International Conference on Logic Programming* (1995), pp. 83 – 97.

[12] BELLEGHEM, K. V., DENECKER, M., AND SCHREYE, D. D. On the relation between situation calculus and event calculus. *The Journal of Logic Programming 31*, 1 (1997), 3 – 37. Reasoning about Action and Change.

[13] BOŠNAČKI, D., AND SCHEFFER, M. Partial order reduction and symmetry with multiple representatives. In *NASA Formal Methods* (Cham,

2015), K. Havelund, G. Holzmann, and R. Joshi, Eds., Springer International Publishing, pp. 97–111.

[14] BOUTILIER, C., REITER, R., SOUTCHANSKI, M., AND THRUN, S. High-level agent programming in the situation calculus. In *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence* (2000), pp. 355 – 362.

[15] CHRPA, L., MCCLUSKEY, T. L., AND OSBORNE, H. Determining redundant actions in sequential plans. In *ICTAI* (2012), pp. 484–491.

[16] CHU, G., GARCIA DEL LA BANDA, M., AND STUCKEY, P. J. Exploiting subprogem dominance in constraint programming. *Constraints 1* (2012), 1–38.

[17] CLARKE, E. M., ENDERS, R., FILKORN, T., AND JHA, S. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design 9*, 1 (Aug 1996), 77–104.

[18] CLASSEN, J., ENGELMANN, V., LAKEMEYER, G., AND RÖGER, G. Integrating golog and planning: An empirical evaluation. In *Proceedings of the 12th International Workshop on Nonmonotonic Reasoning (NMR 2008)* (2008), pp. 10–18.

[19] CLASSEN, J., EYERICH, P., LAKEMEYER, G., AND NEBEL, B. Towards an integration of golog and planning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence* (2007), pp. 1846 – 1851.

[20] CLASSEN, J., RÖGER, G., LAKEMEYER, G., AND NEBEL, B. Platas—integrating planning and the action language golog. *KI - Künstliche Intelligenz 26*, 1 (Feb 2012), 61–67.

[21] DE GIACOMO, G., LESPÉRANCE, Y., LEVESQUE, H. J., AND SARDINA, S. *IndiGolog: A High-Level Programming Language for Embedded Reasoning Agents*. Springer US, Boston, MA, 2009, pp. 31–72.

[22] DE GIACOMO, G., LESPÉRANCE, Y., AND PATRIZI, F. Bounded situation calculus action theories. *Artif. Intell. 237* (2016), 172–203.

[23] DE GIACOMO, G., LESPÉRANCE, Y., PATRIZI, F., AND VASSOS, S. Progression and verification of situation calculus agents with bounded beliefs. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014* (2014), pp. 141–148.

[24] DE GIACOMO, G., AND LEVESQUE, H. An incremental interpreter for high-level programs with sensing. In *Logical Foundations for Cognitive Agents*, H. Levesque and F. Pirri, Eds., Artificial Intelligence. Springer Berlin Heidelberg, 1999, pp. 86–102.

[25] DE GIACOMO, G., AND LEVESQUE, H. J. Projection using regression and sensors. In *Proceedings of the 16th International Joint Conference on Artifical Intelligence - Volume 1* (San Francisco, CA, USA, 1999), IJCAI'99, Morgan Kaufmann Publishers Inc., pp. 160–165.

[26] DOMSHLAK, C., KATZ, M., AND SHLEYFMAN, A. Symmetry breaking: Satisficing planning and landmark heuristics. In *International Conference on Automated Planning and Scheduling (ICAPS)* (2013), pp. 298–302.

[27] EMERSON, E. A., JHA, S., AND PELED, D. Combining partial order and symmetry reductions. In *Tools and Algorithms for the Construction and Analysis of Systems* (Berlin, Heidelberg, 1997), E. Brinksma, Ed., Springer Berlin Heidelberg, pp. 19–34.

[28] EMERSON, E. A., AND SISTLA, A. P. Symmetry and model checking. *Formal Methods in System Design 9*, 1 (Aug 1996), 105–131.

[29] EWIN, C., PEARCE, A. R., AND VASSOS, S. Transforming situation calculus action theories for optimised reasoning. In *In Proceedings of the Fourteenth International Conference on Knowledge Representation and Reasoning* (2014).

[30] EYERICH, P., NEBEL, B., LAKEMEYER, G., AND CLASSEN, J. Golog and pddl: What is the relative expressiveness? In *Proceedings of the 2006 International Symposium on Practical Cognitive Agents and Robots* (New York, NY, USA, 2006), PCAR '06, ACM, pp. 93–104.

[31] FAN, Y., CAI, M., LI, N., AND LIU, Y. A first-order interpreter for knowledge-based golog with sensing based on exact progression and limited reasoning. In *AAAI'12* (2012), pp. 734 – 742.

[32] FANG, L., LIU, Y., AND WEN, X. On the progression of knowledge and belief for nondeterministic actions in the situation calculus. In *Proceedings of the 24th International Conference on Artificial Intelligence* (2015), IJCAI'15, AAAI Press, pp. 2955–2963.

[33] FARINELLI, A., FINZI, A., AND LUKASIEWICZ, T. Team programming in golog under partial observability. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence* (San Francisco, CA, USA, 2007), IJCAI'07, Morgan Kaufmann Publishers Inc., pp. 2097–2102.

[34] FIKES, R. E., AND NILSSON, N. J. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence 2*, 3 (1971), 189 – 208.

[35] FINZI, A., AND LUKASIEWICZ, T. Game-theoretic agent programming in golog. In *Proceedings ECAI-2004* (2004), IOS Press, pp. 23–27.

[36] FINZI, A., AND LUKASIEWICZ, T. Game-theoretic agent programming in golog under partial observability. In *KI 2006: Advances in Artificial Intelligence* (Berlin, Heidelberg, 2007), C. Freksa, M. Kohlhase, and K. Schill, Eds., Springer Berlin Heidelberg, pp. 113–127.

[37] FOX, M., AND LONG, D. Pddl2.1: An extension to pddl for expressing temporal planning domains. *J. Artif. Int. Res. 20*, 1 (Dec. 2003), 61–124.

[38] FRITZ, C., BAIER, J. A., AND MCILRAITH, S. A. Congolog, sin trans: compiling congolog into basic action theories for planning and beyond (extended version. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning 2018* (Sydney, Australia, 2008), pp. 600–610.

[39] GABALDON, A. Non-markovian control in the situation calculus. In *Proc. AAAI-02* (Menlo Park, CA, USA, 2002), pp. 519–524.

[40] GABALDON, A. Programming hierarchical task networks in the situation calculus. In *In AIPS02 Workshop on On-line Planning and Scheduling* (2002).

[41] GEREVINI, A., SAETTI, A., AND SERINA, I. Planning through stochastic local search and temporal action graphs in lpg. *J. Artif. Int. Res. 20*, 1 (Dec. 2003), 239–290.

[42] GIACOMO, G. D., LESPRANCE, Y., AND LEVESQUE, H. J. Congolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence 121*, 1 (2000), 109 – 169.

[43] GRETTON, C., AND THIÉBAUX, S. Exploiting first-order regression in inductive policy selection. In *Proceedings of the 20th Conference on*

Uncertainty in Artificial Intelligence (Arlington, Virginia, United States, 2004), UAI '04, AUAI Press, pp. 217–225.

[44] GU, Y. Macro-actions in the situation calculus. In *Proceedings of IJCAI-03 Workshop on Non-monotonic Reasoning, Action, and Change (NRAC-03)* (2003), IJCAI-03.

[45] HELMERT, M. The fast downward planning system. *J. Artif. Int. Res. 26*, 1 (July 2006), 191–246.

[46] HELMERT, M. Domains - ipc-2008, deterministic part, 2010. [Online; accessed 13-February-2015].

[47] HOFFMANN, J., AND NEBEL, B. The ff planning system: Fast plan generation through heuristic search. *J. Artif. Int. Res. 14*, 1 (May 2001), 253–302.

[48] J. LEVESQUE, H., PIRRI, F., AND REITER, R. Foundations for the situation calculus. 159–178.

[49] KAUTZ, H., AND SELMAN, B. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence* (New York, NY, USA, 1992), ECAI '92, John Wiley & Sons, Inc., pp. 359–363.

[50] KELLY, R. F. *Asynchronous multi-agent reasoning in the situation calculus*. Thesis, 2008.

[51] KELLY, R. F., AND PEARCE, A. R. Towards high-level programming for distributed problem solving. In *2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology* (Dec 2006), pp. 490–497.

[52] KIM, T.-W., LEE, J., AND PALLA, R. Circumscriptive event calculus as answer set programming. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)* (2009), pp. 823–829.

[53] KONEV, B., LUTZ, C., PONOMARYOV, D., AND WOLTER, F. Decomposing description logic ontologies, 2010.

[54] KONEV, B., LUTZ, C., WALTHER, D., AND WOLTER, F. Modular ontologies. Springer-Verlag, Berlin, Heidelberg, 2009, ch. Formal Properties of Modularisation, pp. 25–66.

[55] KOWALSKI, R., AND SADRI, F. Reconciling the event calculus with the situation calculus. *The Journal of Logic Programming 31*, 1 (1997), 39 – 58. Reasoning about Action and Change.

[56] KOWALSKI, R., AND SERGOT, M. A logic-based calculus of events. *New Gen. Comput. 4*, 1 (1986), 67–95.

[57] LAKEMEYER, G. Evaluation-based reasoning with disjunctive information in first-order knowledge bases. In *Proceedings of the Eights International Conference on Principles of Knowledge Representation and Reasoning* (San Francisco, CA, USA, 2002), KR'02, Morgan Kaufmann Publishers Inc., pp. 73–81.

[58] LAKEMEYER, G., AND LEVESQUE, H. J. Situations, si! situation terms, no! In *Proceedings of the Ninth International Conference on Principles of Knowledge Representation and Reasoning* (2004), KR'04, AAAI Press, pp. 516–526.

[59] LEE, J., AND PALLA, R. Situation calculus as answer set programming. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence* (2010), AAAI'10, AAAI Press, pp. 309–314.

[60] LESPRANCE, Y., AND NG, H.-K. Integrating planning into reactive high-level robot programs, 2000.

[61] LEVESQUE, H. J., REITER, R., LESPRANCE, Y., LIN, F., AND SCHERL, R. B. Golog: A logic programming language for dynamic

domains. *The Journal of Logic Programming 31*, 1 (1997), 59 – 83. Reasoning about Action and Change.

[62] LIN, F. Compiling causal theories to successor state axioms and strips-like systems. *CoRR abs/1106.4867* (2011).

[63] LIN, F., AND REITER, R. Forget it! In *Working Notes, AAAI Fall Symposium on Relevance* (1994), R. Greiner and D. Subramanian, Eds., American Association for Artificial Intelligence, pp. 154–159.

[64] LIN, F., AND REITER, R. State constraints revisited. *Journal of Logic and Computation 4*, 5 (1994), 655–677.

[65] LIN, F., AND REITER, R. How to progress a database. *Artificial Intelligence 92*, 1-2 (1997), 131–167.

[66] LIU, Y., AND LAKEMEYER, G. On first-order definability and computability of progression for local-effect actions and beyond. In *Cognitive Robotics* (Dagstuhl, Germany, 2010), G. Lakemeyer, H. J. Levesque, and F. Pirri, Eds., no. 10081 in Dagstuhl Seminar Proceedings, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[67] LIU, Y., AND LEVESQUE, H. J. Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions. In *Proceedings of the 19th international joint conference on Artificial intelligence* (San Francisco, CA, USA, August 2005), IJCAI'05, Morgan Kaufmann Publishers Inc., pp. 522–527.

[68] LÖWE, B., PACUIT, E., AND WITZEL, A. Del planning and some tractable cases. In *Logic, Rationality, and Interaction*, H. van Ditmarsch, J. Lang, and S. Ju, Eds., vol. 6953 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, pp. 179–192.

[69] MCCARTHY, J. Situations, actions, and causal laws. 14.

[70] MCILRAITH, S. A., AND SON, T. C. Adapting golog for composition of semantic web services. In *Proceedings of the Eights International Conference on Principles of Knowledge Representation and Reasoning* (San Francisco, CA, USA, 2002), KR'02, Morgan Kaufmann Publishers Inc., pp. 482–496.

[71] MILLER, A., DONALDSON, A., AND CALDER, M. Symmetry in temporal logic model checking. *ACM Comput. Surv. 38*, 3 (Sept. 2006).

[72] MUISE, C., BECK, J. C., AND MCILRAITH, S. A. Optimal partial-order plan relaxation via maxsat. *J. Artif. Int. Res. 57*, 1 (Sept. 2016), 113–149.

[73] MUISE, C., MCILRAITH, S. A., AND BECK, J. C. Monitoring the execution of partial-order plans via regression. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three* (2011), IJCAI'11, AAAI Press, pp. 1975–1982.

[74] NEBEL, B. On the compilability and expressive power of propositional planning formalisms. *J. Artif. Int. Res. 12*, 1 (May 2000), 271–315.

[75] 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* (San Francisco, CA, USA, 1989), Morgan Kaufmann Publishers Inc., pp. 324–332.

[76] PIRRI, F., AND REITER, R. Some contributions to the metatheory of the situation calculus. *J. ACM 46*, 3 (May 1999), 325–361.

[77] PLEXOUSAKIS, D. Simulation and analysis of business processes using golog. In *Proceedings of Conference on Organizational Computing Systems* (New York, NY, USA, 1995), COCS '95, ACM, pp. 311–322.

[78] PONOMARYOV, D. K., AND SOUTCHANSKI, M. Progression of decomposed local-effect action theories. *CoRR abs/1705.04712* (2017).

[79] REITER, R. Artificial intelligence and mathematical theory of computation. Academic Press Professional, Inc., San Diego, CA, USA, 1991, ch. The Frame Problem in Situation the Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression, pp. 359–380.

[80] REITER, R. *Knowledge in Action. Logical Foundations for Specifying and Implementing Dynamical Systems*. 2001.

[81] RÖGER, G., HELMERT, M., AND NEBEL, B. On the relative expressiveness of adl and golog: The last piece in the puzzle. In *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning* (2008), KR'08, AAAI Press, pp. 544–550.

[82] RÖGER, G., AND NEBEL, B. Expressiveness of adl and golog: Functions make a difference. In *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2* (2007), AAAI'07, AAAI Press, pp. 1051–1056.

[83] RUSSELL, S., AND NORVING, P. *Artificial Intelligence: A Modern Approach*, second ed. Prentice Hall, 2003.

[84] SANNER, S., AND BOUTILIER, C. Practical linear value-approximation techniques for first-order mdps. In *Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence* (Arlington, Virginia, United States, 2006), UAI'06, AUAI Press, pp. 409–417.

[85] SARDINA, S., AND VASSOS, S. The Wumpus World in IndiGolog: A Preliminary Report. In *Proceedings the Nonmonotonic Reasoning, Action and Change Workshop at IJCAI (NRAC-05)* (2005), pp. 90–95.

[86] SCHERL, R., AND LEVESQUE, H. J. Knowledge, action, and the frame problem. *Artificial Intelligence 144*, 1–2 (2003), 1–39.

[87] SCHIFFEL, S., AND THIELSCHER, M. Interpreting golog programs in flux. In *Proceedings of the 7th International Symposium on Logical Formalizations of Commonsense Reasoning* (2005).

[88] SCHIFFEL, S., AND THIELSCHER, M. Reconciling situation calculus and fluent calculus. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1* (2006), AAAI'06, AAAI Press, pp. 287–292.

[89] SCHWERING, C., LAKEMEYER, G., AND PAGNUCCO, M. Belief revision and progression of knowledge bases in the epistemic situation calculus. In *Proceedings of the 24th International Conference on Artificial Intelligence* (2015), IJCAI'15, AAAI Press, pp. 3214–3220.

[90] SHANAHAN, M. The event calculus explained. *Lecture Notes in Computer Science 1600* (1999).

[91] SHAPIRO, S., PAGNUCCO, M., LESPRANCE, Y., AND LEVESQUE, H. J. Iterated belief change in the situation calculus. *Artificial Intelligence 175*, 1 (2011), 165 – 192. John McCarthy's Legacy.

[92] SON, T. C., BARAL, C., , AND TUAN, L.-C. Adding time and intervals to procedural and hierarchical control specifications. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04)* (2004), pp. 92 – 97.

[93] THIELSCHER, M. From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence 111*, 1-2 (July 1999), 277–299.

[94] THIELSCHER, M. FLUX: A logic programming method for reasoning agents. *Theory and Practice of Logic Programing 5*, 4-5 (2004), 533–565.

[95] THIELSCHER, M. Logic-based agents and the frame problem: A case for progression. In *First-Order Logic Revisited: Proceedings of the Conference* (2004), vol. 75, pp. 323–336.

[96] THIELSCHER, M. Flux: A logic programming method for reasoning agents. *Theory Pract. Log. Program. 5*, 4-5 (July 2005), 533–565.

[97] THIELSCHER, M. A unifying action calculus. *Artificial Intelligence 175*, 1 (2011), 120 – 141. John McCarthy's Legacy.

[98] VASSOS, S., LAKEMEYER, G., AND LEVESQUE, H. First-order strong progression for local-effect basic action theories. In *Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning* (2008), KR'08, AAAI Press, pp. 662–272.

[99] VASSOS, S., LAKEMEYER, G., AND LEVESQUE, H. J. First-order strong progression for local-effect basic action theories. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR08)* (2008), p. 662 671.

[100] VASSOS, S., AND LEVESQUE, H. On the progression of situation calculus basic action theories: Resolving a 10-year-old conjecture. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence* (Chicago, Illinois, USA, 2008), AAAI'08, AAAI Press, pp. 1004–1009.

[101] VASSOS, S., AND LEVESQUE, H. J. Progression of situation calculus action theories with incomplete information. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007* (2007), pp. 2029–2024.

[102] VASSOS, S., AND LEVESQUE, H. J. How to progress a database III. *Artificial Intelligence 195* (Feb. 2013), 203–221.

[103] VASSOS, S., AND PATRIZI, F. A classification of first-order progressable action theories in situation calculus. In *Proceedings of the 23rd international joint conference on Artificial intelligence* (August 2013), IJCAI'13, pp. 1132–1138.

[104] WEHRLE, M., HELMERT, M., SHLEYFMAN, A., AND KATZ, M. Integrating partial order reduction and symmetry elimination for cost-optimal classical planning. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015* (2015), pp. 1712–1718.

[105] WIKIPEDIA. Sokoban - Wikipedia, the free encyclopedia, 2015. [Online; accessed 13-February-2015].

[106] YU, Q., WEN, X., AND LIU, Y. Multi-agent epistemic explanatory diagnosis via reasoning about actions. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence* (2013), IJCAI '13, AAAI Press, pp. 1183–1190.