First-Order Logical Filtering

Afsaneh Shirazi , Eyal Amir

University of Illinois at Urbana-Champaign, Department of Computer Science, Urbana, IL 61801, USA

Abstract

Logical filtering is the process of updating a belief state (set of possible world states) after a sequence of executed actions and perceived observations. In general, it is intractable in dynamic domains that include many objects and relationships. Still, potential applications for such domains (e.g., semantic web, autonomous agents, and partial-knowledge games) encourage research beyond immediate intractability results.

In this paper we present polynomial-time algorithms for filtering belief states that are encoded as First-Order Logic (FOL) formulas. We sidestep previous discouraging results, and show that our algorithms are exact in many cases of interest. These algorithms accept belief states in full FOL, which allows natural representation with explicit references to unidentified objects, and partially known relationships. Our algorithms keep the encoding compact for important classes of actions, such as STRIPS actions whose success or failure is known. These results apply to most expressive modeling languages, such as partial databases and belief revision in FOL.

Key words: Filtering, First-Order Logic, Belief Update, Situation Calculus

Email addresses: hajiamin@cs.uiuc.edu (Afsaneh Shirazi), eyal@cs.uiuc.edu (Eyal Amir).

Preprint submitted to Elsevier

1 Introduction

Many everyday scenarios are dynamic and partially observable: a robot in one room cannot see the state of another room, a camera overlooking a bookshelf cannot detect the title of a book that is obscured, and one cannot readily observe the amount of money an agent has. Many applications in such domains compute information about the current world state (*belief state*, i.e. set of possible states or a distribution over such a set) after performing actions and perceiving observations. This computation is called *filtering* (also, state estimation, belief update, and database progression). They use this information to make decisions (e.g., "increase the asking price for my car"), answer questions (e.g., "where is my calculus book?"), and explore (e.g., "go to room 2 and sense the state of the circuit break").

Filtering is intractable in general for discrete domains [8], and much research is dedicated to its approximation in stochastic domains (e.g., [4; 7; 9]). Still, these approximations introduce unbounded errors many times, take unbounded computation time in others, and are not usable in most deterministic domains. Recent progress on logical methods for filtering of propositional belief states (sets of states) [2] with actions and observations has shown that *exact* filtering is tractable when belief states are represented as propositional formulas, and certain natural assumptions are met. Still, many domains have propositional encodings that are too large. In some domains, propositional representation is not possible at all. This includes domains with large numbers of objects, unknown number of objects, and observations with partial knowledge about identity of objects. Propositional methods are very inefficient in such domains, and representations are typically large and cumbersome.

In this paper we present tractable algorithms and theoretical results for updating belief states that are represented in First-Order Logic (FOL). These representations permit belief states of infinite sizes, uncertainty about the number of objects and their identity, and observations that do not distinguish between some objects. It also enables more compact representations than those of propositional logic.

We show that when actions map states 1:1, we can update FOL belief-state formulas efficiently (linear time in the representation size), after

prior compilation. We assume that there is no concurrency in actions. We also show that the representation remains of bounded polynomial size for two classes of domains, including those for which actions have simple case effects (we refer to them as unit-case) and STRIPS-like domains (non-conditional effects, and observed success/failure). For those we also present filtering algorithms that do not require precompilation for efficient update. This is in surprising contrast to the common belief that FOL cannot be used efficiently for representing and updating partial knowledge [18; 28].

On the way to these contributions we provide a theory for FOL belief update. We relate deterministic Situation Calculus [21] with a first-order transition model [3]. In the transition model, every belief state is a set of FOL structures over the FOL language of a state. We encode this set of structures with a FOL formulas. We show that filtering such belief states can be captured exactly by *deduction in FOL*, if the result of the filtering is definable in FOL (this is the best we can hope for [18]). Also, we show that deduction can be carried out one time step at a time. This forms the foundations for the rest of our results, which give an efficient (polynomial-time) exact algorithm for computing this deduction, under some common conditions as mentioned above.

The rest of the paper is organized as follows. Section 2 describes the first-order semantics that we use for first-order logical filtering. In section 3 we provide a naive algorithm for filtering and we prove its correctness. Section 4 offers a polynomial time algorithm. This algorithm uses the distribution properties over logical connectives that we prove in the same section. Section 5 presents efficient algorithms for two interesting special cases. Some of the results of this paper appeared previously in [24].

2 Semantics of First-Order Filtering

In this section, we study logical filtering with first-order structures. Logical filtering is the process of updating a set of possible world states after a sequence of executed actions and perceived observations. In this paper, a world state is represented by a first-order structure. Before explaining the details of logical filtering semantics, we present the languages that

we consider.

2.1 First-Order Languages

A *first-order language* is specified by two sets of symbols:

(1) *Logical symbols:*
   (a) Logical connectives: $\wedge, \vee, \neg, \ldots$
   (b) Quantifiers: $\forall, \exists$
   (c) Equality: $=$
(2) *Nonlogical symbols:*
   (a) Variables(infinitely many): $x, y, z, \ldots$
   (b) Function symbols: For each n, a set of symbols called n-ary function symbols. A 0-ary function symbol is called *constant*. We use words starting with capital letters for constants.
   (c) Predicate symbols: For each n, a set of symbols called n-ary predicate symbols.

Note that the equality symbol can be treated as a binary predicate symbol. We define the *terms* and *formulas* by the following generalized inductive definition. Variables and functions are terms. Predicates are atomic formulas. We define *literals* to be atomic formulas or the negation of atomic formulas. Concepts of *free* and *bound* occurrences of a variable in a formula are defined as usual. A *closed formula* is a formula with no free variable.

A *first-order language* is a language in which the symbols and formulas are as described above. It is completely determined by its nonlogical symbols. We now turn to a description of the semantics of first-order languages. A *structure* S for a first-order language consists of:

(1) $|S|$, the nonempty *universe* or *domain* of the structure S. The elements of $|S|$ are called the *individuals* of S.
(2) For each n-ary predicate symbol p, $p^S \subseteq |S|^n$. These tuples of the universe are those tuples on which p is true in the structure.
(3) For each n-ary function symbol f, $f^S : |S|^n \to |S|$. (In particular, for each constant e, e^S is an individual of S)

When a sentence ψ is *true* in a structure S, we denote it by $\models_S \psi$. To define it, we need a more general notion of truth for a formula. Suppose that:

(1) ϕ is a formula of a given first-order language.
(2) S is a structure for the language.
(3) $\sigma : V \to |S|$ is a function, called a variable assignment, from the set V of variables of the language into the universe of S.

We can now define $\models_S \phi[\sigma]$, meaning that the formula ϕ is true in the structure S when its free variables are given the values specified by σ in the universe of S as follows.

(1) *Terms:* Define an extension σ' of the function σ from the set of all terms of the language into the universe.
   (a) For each variable v, $\sigma'(v) = \sigma(v)$
   (b) If $t_1, \ldots, t_n$ are terms and f is a an n-ary function symbol, then
   $$\sigma'(f(t_1, \ldots, t_n)) = f^S(\sigma'(t_1), \ldots, \sigma'(t_n))$$
(2) *Atomic formulas:*
   (a) For term t_1 and t_2,
   $$\models_S t_1 = t_2[\sigma] \text{ iff } \sigma'(t_1) = \sigma'(t_2)$$
   (b) For an n-ary predicate symbol p,
   $$\models_S p(t_1, \ldots, t_n)[\sigma] \text{ iff } \langle \sigma'(t_1), \ldots, \sigma'(t_n) \rangle \in p^S$$
(3) *Well-formed formulas:*
   (a) $\models_S \neg\phi[\sigma]$ iff not $\models_S \phi[\sigma]$
   (b) $\models_S (\phi \Rightarrow \varphi)[\sigma]$ iff $\models_S \neg\phi[\sigma]$ *or* $\models_S \varphi[\sigma]$
   (c) $\models_S (\forall x)\phi[\sigma]$ iff for every $d \in |S|$, $\models_S \phi[\sigma(x/d)]$
   $\sigma(x/d)$ is the function that is exactly like σ except that for the variable x it assigns the value d.

As an example, suppose that in structure S:

- $|S| = \{B, R\}$
- for binary predicate *in*: $in^S = \{\langle B, R \rangle\}$
- for constant *Office*: $Office^S = \{R\}$
- for constant *C++*: $C{+}{+}^S = \{B\}$

This world has the predicate *in* that indicates whether a book is in a room or not, the book *C++* and the room *Office*. By this definition, sentence *in*(*C++, Office*) is true in S.

We define logical filtering using the situation calculus. The one that we use is compatible with the *basic action theory* of [21]. In the next section, we discuss the basics of the situation calculus that we need for our work.

2.2 Situation Calculus

In this section, we specify a language and an axiomatization for the situation calculus. The situation calculus is a form for specifying properties of actions. The language adopted here is a second-order language with equality. It has three disjoint sorts: *action* for actions, *situation* for situations, and *object* for everything else depending on the domain of the application. This language has the following alphabet (this definition is similar to the situation calculus of Reiter [21]):

- A constant symbol s_0, denoting the initial situation.
- A binary function symbol $do : action \times situation \rightarrow situation$. The intended interpretation is that situations are finite sequences of actions, and $do(a, s)$ denotes that sequence formed by adding action a to the sequence s.
- A binary predicate symbol $Poss : action \times situation$. The intended interpretation of $Poss(a, s)$ is to show the possibility of performing the action a in situation s.
- A binary predicate symbol $\sqsubset : situation \times situation$ defining an ordering relation on situations. The intended interpretation of situations is as action histories, in which case $s \sqsubset s'$ means that s is a proper subhistory of s'.

We want to be able to say that a certain sequence of actions is a subsequence of another. To formalize this, we adopt the following four foundational axioms presented in [15]:

(1) $do(a_1, s_1) = do(a_2, s_2) \Rightarrow a_1 = a_2 \wedge s_1 = s_2$
(2) $(\forall P).P(s_0) \wedge (\forall a, s)[P(s) \Rightarrow P(do(a, s))] \Rightarrow (\forall s)P(s)$
(3) $\neg s \sqsubset s_0$
(4) $s \sqsubset do(a, s') \equiv s \sqsubseteq s'$

Axiom 1 is a unique name axiom for situations. Two situations are the same iff they are the same sequence of actions. Axiom 2 is the second-

order induction on situations. The importance of induction for the situation calculus is described by Reiter in [22]. This axiom is the only second-order axiom in our action theory. However, Reiter in [15] showed that the second-order axiom is not needed for projection purposes. In the rest of this paper, we use the first-order situation calculus with the removal of this axiom. This subset of the situation calculus as shown by Reiter is enough for filtering.

The above axioms are domain independent. We refer to them as Σ. Combination of these axioms with a specification of the initial situation, successor state axioms, action precondition axioms, and unique name axioms for actions will be called a *basic theory of actions*. First we should be more precise about what counts as successor state axioms and action precondition axioms.

Generally speaking, the truth value of a predicate and the value of a function in a dynamic world may vary from time to time. In first-order logic there is no notion of time. Here, we introduce relational and functional fluents to capture the dynamic behavior of our world. Predicates whose truth values vary from situation to situation are called *relational fluents*. They are denoted by predicate symbols taking a situation term as their last argument. Functional fluents are defined similarly.

The situation calculus that we use here is non-typed. For simplicity, we also assume that all predicates and functions are fluents. Predicates whose truth values do not change from situation to situation are handled with the axioms that enforce these constraints.

Definition 1 (Uniform Formulas) Let s be a term of sort situation. Inductive definition of a term *uniform* in s is as follows:

(1) Any term that does not mention a term of sort situation is uniform in s.
(2) If f is an n-ary functional fluent other than do, and $t_1, \ldots, t_n$ are terms that are uniform in s, then $f(t_1, \ldots, t_n, s)$ is uniform in s.

The formulas that are *uniform* in s are inductively defined by:

(1) Any formula that does not mention a term of sort situation is uniform in s.

(2) If p is an n-ary relational fluent other than $Poss$, and $t_1, \ldots, t_n$ are terms that are uniform in s, then $p(t_1, \ldots, t_n, s)$ is a formula uniform in s.
(3) If φ_1 and φ_2 are formulas uniform in s, so are $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$ and $\exists v\ \varphi_1$ provided that v is a variable not of sort situation.

Thus, a formula is uniform in s iff it does not mention $Poss$, it does not quantify over variables of sort situation, it does not mention equality of situations, and whenever it mentions a term of sort situation in the situation argument position of a fluent, then that term is s.

We define precondition axioms and successor state axioms as a part of our basic action theory as follows.

From this point on, in our formulas the letter a refers to a ground action, and the letter A refers to the corresponding action symbol.

Definition 2 (Action Precondition Axioms) An *action precondition axiom* is a sentence of the form:

$$Poss(A(x_1, \ldots, x_n), s) \Leftrightarrow precond_A(x_1, \ldots, x_n, s)$$

where A is an n-ary action symbol, and $precond_A(x_1, \ldots, x_n, s)$ is a formula that is uniform in s and whose free variables are among $x_1, \ldots, x_n, s$.

Definition 3 (Successor State Axioms) A *Successor state axiom* is defined for either a relational fluent or a functional fluent. A successor state axiom for an n-ary relational fluent p is a sentence of the form:

$$Poss(A(x_1, \ldots, x_n), s) \Rightarrow \forall y_1, \ldots, \forall y_m$$
$$(p(y_1, \ldots, y_m, do(A(x_1, \ldots, x_n), s)) \Leftrightarrow succ_{p,A}(x_1, \ldots, x_n, y_1, \ldots, y_m, s))$$

where A is an action symbol, and $succ_{p,A}(x_1, \ldots, x_n, y_1, \ldots, y_m, s)$ is a formula that is uniform in s, and whose free variables are among $x_1, \ldots, x_n$, $y_1, \ldots, y_m, s$. We define a successor state axiom for a functional fluent similarly as follows.

$$Poss(A(x_1,\ldots,x_n),s) \Rightarrow \forall y_1,\ldots,\forall y_m,\forall z$$
$$(f(y_1,\ldots,y_m,do(A(x_1,\ldots,x_n),s)) = z$$
$$\Leftrightarrow succ_{f,A}(x_1,\ldots,x_n,y_1,\ldots,y_m,z,s))$$

The basic action theory has the form $\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{s_0}$ where:

- Σ are the foundational axioms for situations.
- $\mathcal{D}_{ss}$ is a set of successor state axioms.
- $\mathcal{D}_{ap}$ is a set of action precondition axioms.
- $\mathcal{D}_{una}$ is a set of unique name axioms for actions.
  $A(\overrightarrow{x}) \neq A'(\overrightarrow{y}) \quad , \quad A(\overrightarrow{x}) = A(\overrightarrow{y}) \Rightarrow \overrightarrow{x} = \overrightarrow{y}$ where A and A' are action symbols.
- $\mathcal{D}_{s_0}$ (the initial database) is a finite set of first-order sentences that are *uniform* in s_0. $\mathcal{D}_{s_0}$ will function as the initial theory of the world (*i.e.*, the one we start off with, before any action is executed).

All changes to the world are the result of named actions. An action may be parameterized. For example, $move(B, R_1, R_2)$ stands for the action of moving object B from room R_1 to room R_2. We use the situation calculus as foundations and semantics. Later (section 3 onwards) we do not mention it because we always focus on belief states. However, it is used in the proofs of theorems and our results are applicable to it.

2.2.1 Example

Consider a book keeper robot who lives in a world consisting of rooms. When the robot is in a room, it can make observations about the books in that room. It can move a book from one room to the other, it can return a book to the library from an arbitrary room or it can put a borrowed book in a room. So possible actions are $move(b, r_1, r_2)$, $return(b, r)$ and $borrow(b, r)$. The actions which are executable in a world state can change the value of different relational or functional fluents. Relational fluents are $room(r, s)$, $book(b, s)$, and $in(b, r, s)$. There are no functional fluents except constants.

We define the precondition axiom and the successor state axiom for action *move* and skip the others.

- *Precondition Axiom:*
  $Poss(move(b, r_1, r_2), s) \Leftrightarrow book(b, s) \wedge room(r_1, s) \wedge room(r_2, s) \wedge in(b, r_1, s)$
- *Successor State Axiom:*
  $Poss(move(b, r_1, r_2), s) \Rightarrow$
  $\forall b' \forall r' \ (in(b', r', do(move(b, r_1, r_2), s)) \Leftrightarrow$
  $(((b' = b) \wedge (r' = r_1)) \Rightarrow false$
  $\wedge((b' = b) \wedge (r' = r_2)) \Rightarrow true$
  $\wedge(\neg((b' = b) \wedge (r' = r_1)) \wedge \neg((b' = b) \wedge (r' = r_2))) \Rightarrow in(b', r', s)))$

2.3 Filtering Semantics

As mentioned above, there is a predicate corresponding to each relational fluent. The predicate does not take any argument of sort situation. Roughly speaking, the truth value of a predicate changes from one situation to another. We represent predicates with the same symbol as relational fluents but with different arity (one less than the corresponding relational fluent). In the following definition we drop all situation terms. The value of all the predicates and functions are considered in the same situation. Transition from situation s to $do(a, s)$ (performing action a) changes the value of predicates and functions.

The following is a formal definition of transition relations and filtering over first-order structures. The generality of the system demands attention to specific details regarding cardinality of sets and the relationship between first-order formulas and the classes of structures characterized by these formulas. We bring those details first.

Let κ be a fixed infinite cardinality $\kappa \geq \aleph_0$ (we keep this cardinality unspecified to emphasize the generality of the following development). The vocabulary for describing a state of our system is $\langle \mathcal{P}, \mathcal{F}, \mathcal{C} \rangle$, with $\mathcal{P}$ a set of predicate symbols, $\mathcal{F}$ a set of function symbols, and $\mathcal{C}$ a set of constant symbols. Let $\mathbb{S}$ be the set of all first-order structures S over $\langle \mathcal{P}, \mathcal{F}, \mathcal{C} \rangle$ that have cardinality $||S||$ at most κ.

Definition 4 (Transition Relation of First-Order Structures) For

an action theory $\mathcal{D}$, a structure S, and a ground action a, we define a transition relation $\mathcal{R}_\mathcal{D}$ as follows.

$$
\begin{aligned}
\mathcal{R}_\mathcal{D} = \{\langle S, a, S' \rangle \mid\ & \\
& \models_S precond_a, \\
& S, S' \in \mathbb{S} \text{ and } |S| = |S'| \\
& p^{S'} = \{\langle\, v_1, \ldots, v_m \rangle \in |S|^m \mid\ \models_S succ_{p,a}(v_1, \ldots, v_m)\}, \\
& f^{S'} = \{\langle v_1, \ldots, v_m, v_{m+1} \rangle \in |S|^{m+1} \mid\ \models_S succ_{f,a}(v_1, \ldots, v_m, v_{m+1})\}\}
\end{aligned}
$$

In this definition v_i is an individual of $|S|$.

Note that action a is a parameterized action. The parameters are constants (*e.g.*, *move*(*C++, Office, Lounge*)). For $a = A(u_1, \ldots, u_n)$, $precond_a$ is equal to $precond_A(u_1, \ldots, u_n)$ which is defined by the precondition axiom of the action symbol A.

Now, we define filtering semantics using transition relation of first-order structures.

Definition 5 (First-Order Logical Filtering Semantics) Let Δ be a set of first-order structures. The *filtering* of a sequence of actions and observations $\langle a_1, o_1, \ldots, a_t, o_t \rangle$ is defined as follows (ϵ refers to the empty sequence).

(1) $Filter[\epsilon](\Delta) = \Delta$;
(2) $Filter[a](\Delta) = \{S' \mid S \in \Delta,\ \langle S, a, S' \rangle \in \mathcal{R}_\mathcal{D}\}$;
(3) $Filter[o](\Delta) = \{S \in \Delta \mid\ \models_S o\}$;
(4) $Filter[\langle a_i, o_i, \ldots, a_t, o_t \rangle](\Delta) =$
$\quad Filter[\langle a_{i+1}, o_{i+1}, \ldots, a_t, o_t \rangle](Filter[o_i](Filter[a_i](\Delta)))$;

We call Step 2 *progression with* a and Step 3 *filtering with* o.

In the definition above, filtering is applied to a set of first-order structures. A *belief state formula* is a first-order formula in the vocabulary $\langle \mathcal{P}, \mathcal{F}, \mathcal{C} \rangle$, with $\mathcal{P}, \mathcal{F}, \mathcal{C}$ the predicate, function, and constant symbols that correspond to relational fluents, functional fluents, and constants (essentially, they are the same only without a situation argument). For example, the predicate $in(b, r)$ corresponds to $in(b, r, s)$ which is a relational fluent in the situation calculus.

Every belief state formula ψ has a corresponding belief state, $\sigma_\psi = \{S \in \mathbb{S} \mid \models_S \psi\}$. In the opposite direction, every belief state σ has many belief state formulas to which it corresponds, and we distinguish one such formula, the *theory of* σ. The theory of σ, $Th(\sigma)$, is the set of all sentences with vocabulary $\langle \mathcal{P}, \mathcal{F}, \mathcal{C} \rangle$ that all structures in σ satisfy. $Th(\sigma) = \{\phi \mid \phi \text{ in vocabulary } \langle \mathcal{P}, \mathcal{F}, \mathcal{C} \rangle, \ \forall S \in \sigma \ \models_S \phi\}$.

Several delicate points are worth mentioning. First, our choice of κ affects $\mathbb{S}$ and the rest of the sets discussed above. Nonetheless, taking σ_ψ with $\kappa' > \kappa$ does not change $Th(\sigma_\psi)$. Specifically, $Th(\sigma_\psi^\kappa) = Th(\sigma_\psi^{\kappa'})$. This results from the Lowenheim-Skolem theorem. Thus, our choice of κ is not important as long as it is at least countable ($\aleph_0$) and is at least as large as the number of symbols in the vocabulary (when the vocabulary is infinitely large)

3 Filtering of FOL Formulas

Approaches to filtering actions and observations that at any stage enumerate all the structures in a belief state do not scale to large domains. An alternative approach is to perform logical progression in a form similar to the one described by [18; 20]. The difference is that now we wish to do so (efficiently) in the context of nondeterministic actions and observations.

In this section we present a naive algorithm that filters belief state formulas directly. This algorithm serves as a starting point for Section 4, where we propose efficient algorithms. We also present distribution properties of filtering over the logical connectives $\wedge, \vee, \neg$, and examine the theoretical limitations of formula filtering. These will guide us in Section 5, and allow us to present classes of systems that are not subject to those limitations and can be tracked in polynomial time and a compact fashion indefinitely.

From now on, we assume that our first-order language has no functional symbols except constants.

In this section we show how we progress an initial database represented by a logical formula after applying a single action or observation. The result of progression is a new database that the progression algorithm can use afterwards.

Suppose that $\mathcal{P} = \{g_1, \ldots, g_r\}$ is the set of all constants and predicates of our first order language. We define a new set of symbols $\mathcal{P}' = \{g'_1, \ldots, g'_r\}$ such that $g'_i(y_1, \ldots, y_n) = g_i(y_1, \ldots, y_n)_{[\mathcal{P}/\mathcal{P}']}$ where $[\mathcal{P}/\mathcal{P}']$ is a shorthand for $[g_1/g'_1, ..., g_r/g'_r]$. We view $\mathcal{P}$ as the set of constants and predicates in situation s, and $\mathcal{P}'$ as the set of constants and predicates in situation $do(a, s)$.

We filter a belief-state formula as follows. (We reuse $Filter[\cdot](\cdot)$ for filtering a belief-state formula and also a set of formulas.) Let ψ be a belief state formula (a formula that represent all the possible structures of the current state of the world), a be a ground action, $Cn(\Psi)$ be the set of logical consequences of Ψ (*i.e.*, formulas ϕ such that $\Psi \models \phi$), and $Cn^{\mathcal{L}}(\Psi)$ be the set of logical consequences of Ψ in the language $\mathcal{L}$. We write $Cn^{L}(\Psi)$, when L is a *set of symbols* to mean $Cn^{\mathcal{L}(L)}(\Psi)$.

$$
\begin{aligned}
&1.\ \ Filter[a](\psi) = (Cn^{\mathcal{P}'}(\psi \wedge precond_a \wedge \\
&\qquad \bigwedge_i \forall y_1, \ldots, y_m, p'_i(y_1, \ldots, y_m) \Leftrightarrow succ_{p_i,a}(y_1, \ldots, y_m) \wedge \\
&\qquad \bigwedge_i \forall z, f'_i = z \Leftrightarrow succ_{f_i,a}(z)))_{[\mathcal{P}'/\mathcal{P}]} \\
&2.\ \ Filter[o](\psi) = \psi \wedge o
\end{aligned}
\tag{1}
$$

When we filter with action a we assert implicitly that its precondition held in the last world state. If the action is not executable on the belief state, the new belief state would be false which indicates an empty set of structures (meaning that there is no structure of the current belief state in which the action was executable). We also need to provide an equation for filtering a set of formulas to be able to use the previous equations recursively. Following equations provide filtering algorithm for

a set of formulas Γ.

$$
\begin{aligned}
&1.\ \mathit{Filter}[a](\Gamma) = (Cn^{\mathcal{P}'}(\Gamma \cup \{\mathit{precond}_a\} \cup \\
&\quad \bigcup_i \{\forall y_1, \ldots, y_m, p_i'(y_1, \ldots, y_m) \Leftrightarrow \mathit{succ}_{p_i,a}(y_1, \ldots, y_m)\} \cup \\
&\quad \bigcup_i \{\forall z, f_i' = z \Leftrightarrow \mathit{succ}_{f_i,a}(z)\}))_{[\mathcal{P}'/\mathcal{P}]} \\
&2.\ \mathit{Filter}[o](\Gamma) = \Gamma \cup \{o\} \qquad (2)
\end{aligned}
$$

We prove in the following theorem that this definition of filtering approximates the semantics of definition 5.

Theorem 6 Let ψ be a belief state formula, and let a be a ground action, then

$$\mathit{Filter}[a](\{S \mid \models_S \psi\}) \subseteq \{S' \mid \models_{S'} \mathit{Filter}[a](\psi)\}$$

PROOF See appendix A.1 ■

[18] showed that progression is not always first-order definable. However, they proved that progression always exists as a set of second order sentences for finite initial databases. Therefore, the two sides in theorem 6 are not equivalent since formula (1) is in FOL. In other words, FOL is not strong enough to model the progression of the initial database. However, the following corollary shows that the two sides of theorem 6 would be equal if the progression of a database is FOL definable.

Corollary 7 Let ψ be a first-order belief state formula. If progression of ψ is first-order definable then

$$\mathit{Filter}[a](\{S \mid \models_S \psi\}) = \{S' \mid \models_{S'} \mathit{Filter}[a](\psi)\}$$

PROOF See appendix A.2 ■

From this point, we assume that progression is first-order definable. Our basic algorithm computes $\mathit{Filter}[\langle a_1, o_1, ..., a_t, o_t\rangle](\psi)$ by iteratively applying filtering of a belief-state formula with an action and an observation. It sets $\psi_0 = \psi$ and $\psi_i = \mathit{Filter}[o_i](\mathit{Filter}[a_i](\psi_{i-1}))$ recursively for $i > 0$ using the equations defined above. This algorithm is correct, as

shown by corollary 7. It can be implemented using a first-order consequence finder. Here, we do not talk about how *Filter* generates a sentence, we only postulate the existence. In the next sections, we provide algorithms for generating the formula that is the result of filtering.

3.2 Sequences of Actions and Observations

This section shows that iterative application of progression steps loses no information. It will follow that we can answer queries about the outcome of a sequence of actions and observations by filtering. More specifically, after every action we can discard the previous database and start using the database that we obtain after performing the progression step for that action. In this section we use the first-order situation calculus that we described before. It follows the treatment of Situation Calculus by Reiter [21].

We divide our action theory $\mathcal{D}$ into two parts, the initial database $\mathcal{D}_{s_0}$ and the rest $\mathcal{D}_g$. Therefore, $\mathcal{D} = \mathcal{D}_g \cup \mathcal{D}_{s_0}$. We define the language of an action theory as follows.

Definition 8 *The language of* $\mathcal{D}$, $\mathcal{L}(\mathcal{D})$, is a set of first order formulas whose relational and functional fluents are part of $\mathcal{D}$.

For instance, if $\mathcal{D}_{s_0} = at(A, B, s_0) \wedge \forall x \;\; box(x, s_0)$, then we can say that $at(A, A, s_0)$ and $\forall x \exists y \; at(x, y, s_0)$ are in $\mathcal{L}(\mathcal{D}_{s_0})$ but $box(A, B, s_0)$ is not because it has different arity (it is a different predicate).

We use $Cn_{s_t}(\Psi)$ as the set of logical consequences of Ψ uniform in $do(a_t, do(a_{t-1}, ...do(a_1, s_0)))$.

Suppose that we want to progress our initial database $\mathcal{D}_{s_0}$ with action a_1. Recall that $\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{s_0}$. To progress $\mathcal{D}$ with action a_1, not all the axioms are required [15]. Instead of including all successor state and precondition axioms in the action theory, we just need precondition and successor state axioms related to action a_1 (with the same action symbol). To retrieve these axioms, in all the successor state and precondition axioms with the same action symbol as a_1 we replace their parameters with the constants appearing in ground action

a_1. We refer to this subset of successor state axioms as $\mathcal{D}_{ss_{0,1}}$. The subset of precondition axioms corresponding to action a_1 is $\mathcal{D}_{ap_{0,1}}$. For example, if our first action is $move(C++, Lounge, Office)$, then axiom

$$Poss(move(x_1, x_2, x_3), s) \Rightarrow \forall y_1, \ldots, \forall y_m$$
$$(p(y_1, \ldots, y_m, do(move(x_1, x_2, x_3), s)) \Leftrightarrow succ_{p,move}(x_1, x_2, x_3, y_1, \ldots, y_m, s))$$

would be replaced by

$$Poss(move(C++, Lounge, Office), s_0) \Rightarrow \forall y_1, \ldots, \forall y_m$$
$$(p(y_1, \ldots, y_m, do(a_1, s_0)) \Leftrightarrow succ_{p,move}(C++, Lounge, Office, y_1, \ldots, y_m, s_0))$$

All the other axioms are not required for progression with a_1.

In progression we assume that we are given a ground action term a_1 that refers to the action being performed, and we are looking for a set of sentences $\mathcal{D}_1$ uniform in $do(a_1, s_0)$ that can serve as a new initial database. [18] showed that $\mathcal{D}_1$ is not always first-order definable. Instead of such $\mathcal{D}_1$, we define a first-order database, $\mathcal{D}_{s_1}$, that is enough for first-order queries, as we show below.

We define $\mathcal{D}_{s_1}$ as the set of first-order sentences uniform in $do(a_1, s_0)$ that are entailed by $\mathcal{D}$. For every first-order sentence ψ about the future of $do(a_1, s_0)$ (*i.e.,* ψ is uniform in a situation s' such that $do(a_1, s_0) \sqsubseteq s'$), if we show that $\mathcal{D}_g \cup \mathcal{D}_{s_1} \models \psi$ iff $\mathcal{D} \models \psi$ then $\mathcal{D}_{s_1}$ can be used as a new initial database for all first-order consequences of the action theory. The following theorem states this result for two-step progression. Similar theorem for one step progression is shown in [18].

Theorem 9 Let $\mathcal{D}_{s_0}$ be an initial database for an action theory $\mathcal{D}$, ($\mathcal{D} = \mathcal{D}_g \cup \mathcal{D}_{s_0}$). Define $\mathcal{D}_{s_t}$ for $t \geq 1$ as follows: [1]

$$\mathcal{D}_{s_t} = Cn_{s_t}(\mathcal{D}_{ss_{t-1,t}} \cup \mathcal{D}_{ap_{t-1,t}} \cup \mathcal{D}_{s_{t-1}})$$

[1] Since we rewrote successor state axioms as transition between situations, and we removed the *do* function, we do not need foundational and unique name axioms

Then, for all ψ uniform in $do(a_2, do(a_1, s_0))$,

$$\mathcal{D}_g \cup \mathcal{D}_{s_2} \models \psi \quad \textit{iff} \quad \mathcal{D}_g \cup \mathcal{D}_{s_0} \models \psi$$

PROOF See appendix A.4 ■

Corollary 10 Let $\mathcal{D}_{s_0}$ be an initial database for an action theory $\mathcal{D}$, $(\mathcal{D} = \mathcal{D}_g \cup \mathcal{D}_{s_0})$. Define $\mathcal{D}_{s_t}$ for $t \geq 1$ as follows:

$$\mathcal{D}_{s_t} = Cn_{s_t}(\mathcal{D}_{ss_{t-1,t}} \cup \mathcal{D}_{ap_{t-1,t}} \cup \mathcal{D}_{s_{t-1}})$$

Then, for all ψ uniform in $do(a_t, do(a_{t-1}, \ldots do(a_1, s_0)))$,

$$\mathcal{D}_g \cup \mathcal{D}_{s_t} \models \psi \quad \textit{iff} \quad \mathcal{D}_g \cup \mathcal{D}_{s_0} \models \psi$$

PROOF Similar to the proof of previous theorem. ■

Theorem 9 gives a fine detail of Reiter's result. Reiter showed that the set of models after progression may have no first-order sentence representing it. Our theorem gives a second view of that older result. It shows that for first-order queries we do not need an exact representation of all these models. Instead, we capture a larger set with a first-order database, and proof theory provides the result [2].

For instance, in the book-keeper example if

$$\mathcal{D}_{s_0} = \{book(B, s_0), room(R, s_0), in(B, R, s_0)\}$$

and the first action is $return(B, R)$, then $\mathcal{D}_{s_1}$ is logically equivalent to

$$\{book(B, do(a_1, s_0)), room(R, do(a_1, s_0)), \neg in(B, R, do(a_1, s_0))\}$$

[2] This is an interesting case of Hilbert's agenda for logic in model theory. Reiter's theorem is model theoretic and shows what cannot be done, whereas our theorem is proof theoretic and provides a practical way of living with Reiter's result.

4 Factored Inference

In this section we present the main contribution of this paper, namely, a polynomial-time algorithm that computes logical filtering exactly for a significant class of transition systems. For some special cases (including systems with unit-case actions) we present simpler algorithms that are even more efficient later in the paper. For the systems that do not fall within this class our algorithm gives an approximation to the filtering.

Before getting into the algorithm in the next section, we prove several properties for decomposing the filtering of a FOL formula to the filtering of a set of atomic subformulas. Later in the algorithms we use these properties to perform filtering on the subformulas of a formula and combine the results.

4.1 Distribution Properties

Several distribution properties hold for logical filtering. We can decompose the filtering of a formula φ along logical connectives $\wedge, \vee, \neg, \forall, \exists$.

Theorem 11 Let progression be first-order definable, a be an action, and φ and ψ be first-order formulas. Then,

(1) $Filter[a](\varphi \vee \psi) \equiv Filter[a](\varphi) \vee Filter[a](\psi)$
(2) $\models Filter[a](\varphi \wedge \psi) \Rightarrow Filter[a](\varphi) \wedge Filter[a](\psi)$
(3) $\models Filter[a](\neg\varphi) \Leftarrow \neg Filter[a](\varphi) \wedge Filter[a](TRUE)$
(4) $Filter[a](\exists x\ \varphi(x)) \equiv \exists x\ Filter[a](\varphi(L))_{[L/x]}$

(L is a new constant symbol.)

PROOF See appendix A.6 ■

As the result of this theorem we can have the following discussion. Filtering $\varphi \vee \psi$ can be done by filtering φ and ψ separately and then combining the results. Also, filtering $\varphi \wedge \psi$ can be approximated by filtering φ and ψ separately and then combining the results. The formula that is the conjunction of the filtering of φ and ψ separately is a weaker formula

than the filtering of $\varphi \wedge \psi$. Thus, everything that is entailed by that combination is also true in every structure of the original filtering.

Filtering of $\neg\varphi$ can be approximated in the other direction. The formula that is the negation of the filtering of φ is a stronger formula than the filtering of $\neg\varphi$. Thus, everything that follows from the filtering of $\neg\varphi$ necessarily holds in the negation of the filtering of φ and that of TRUE. Also filtering $\exists x\ \varphi(x)$ can be done by filtering $\varphi(L)$ with L being a new constant symbol and then replacing every L in the resulted formula with an x and having an existential quantifier over x.

While 2 and 3 are only approximations (according to Theorem 10), they can be used as an exact answer in some domains. In the following theorem, we prove a stronger statement for actions that act as *permutation* on the structures in which they are executable.

Definition 12 (Permuting Actions) Action a is *permuting* (*1:1*) if for every structure S' there is at most one S such that $\mathcal{R}_{\mathcal{D}}(S, a, S')$.

For example, , consider $a = move(C{+}{+}, Office, Lounge)$ from previous examples. This action is a one-to-one mapping between structures. Assume that we define another action $moveto(C{+}{+}, Lounge)$ which moves $C{+}{+}$ book to room *Lounge* regardless of where the book is before moving. This new action is not one-to-one because it maps many different structures to the same structure. In many domains an action which is not one-to-one such as *moveto* can easily be replaced by a permuting action such as *move* with no loss in the expressiveness of the domain (turning on the light can be replaced by flipping the switch).

Domains that only include permuting actions are called *permutation domains* which encompass a pool of interesting examples. Next theorem shows distribution properties for permutation domains.

Theorem 13 (Distribution Properties for Permutation Domains) Let progression be first-order definable, a be a permuting action, and φ and ψ be formulas. Then,

(1) $Filter[a](\varphi \vee \psi) \equiv Filter[a](\varphi) \vee Filter[a](\psi)$
(2) $Filter[a](\varphi \wedge \psi) \equiv Filter[a](\varphi) \wedge Filter[a](\psi)$
(3) $Filter[a](\neg\varphi) \equiv \neg Filter[a](\varphi) \wedge Filter[a](TRUE)$

(4) $Filter[a](\exists x\ \varphi(x)) \equiv \exists x\ Filter[a](\varphi(L))_{[L/x]}$

PROOF See appendix A.8 ■

The decomposition strategy offered by this theorem is not only an approximation but an exact computation for permutation domains. These distribution properties are enough for decomposing the filtering of any first-order formula into the filtering of its atomic subformulas. We can decompose the filtering of any first-order formula into the filtering of atomic formulas by using distribution properties proved above. As an example, universally quantified formulas are decomposed as follows:

- Universal quantifier over a formula: (by rule 3 and rule 4)

$$\begin{aligned} Filter[a](\forall x\ \varphi(x)) &\equiv Filter[a](\neg\exists x\ \neg\varphi(x)) \\ &\overset{3}{\equiv} \neg Filter[a](\exists x\ \neg\varphi(x)) \wedge Filter[a](TRUE) \\ &\overset{4}{\equiv} \neg\exists x\ Filter[a](\neg\varphi(L))_{[L/x]} \wedge Filter[a](TRUE) \end{aligned}$$

- Universal quantifier over conjunction: (by rule 2)

$$Filter[a](\forall x\ \varphi(x) \wedge \psi(x)) \overset{2}{\equiv} Filter[a](\forall x\ \varphi(x)) \wedge Filter[a](\forall x\ \psi(x))$$

- Universal quantifier over negation: (by rule 3 and rule 4)

$$\begin{aligned} Filter[a](\forall x\ \neg\varphi(x)) &\equiv Filter[a](\neg\exists x\ \varphi(x)) \\ &\overset{3,4}{\equiv} \neg\exists x\ Filter[a](\varphi(L))_{[L/x]} \wedge Filter[a](TRUE) \end{aligned}$$

- Universal quantifier over disjunction: (by rule 2, rule 3, and rule 4)

$$\begin{aligned} Filter[a](\forall x\ \varphi(x) \vee \psi(x)) &\equiv Filter[a](\neg\exists x\ \neg\varphi(x) \wedge \neg\psi(x)) \\ &\overset{3,4}{\equiv} \neg\exists x\ Filter[a](\neg\varphi(L) \wedge \neg\psi(L))_{[L/x]} \\ &\qquad \wedge\ Filter[a](TRUE) \end{aligned}$$

4.2 Factored Filtering

Our Factored Filtering (FF) algorithm for permutation domains is presented in Figure 1. It relies on theorems 7, 10, and 12. The number of

closed atomic subformulas of a domain is finite, if the number of constants is finite. Therefore, in finite domains we can calculate filtering of all atomic formulas as a preprocessing step and retrieve them later. In this paper, we do not discuss this preprocessing step.

Note that the arguments of these atomic formulas are either the constants associated with existential quantifiers or the constants mentioned in the initial belief state, the set of axioms, or the observations.

PROCEDURE FF($\langle a_i, o_i \rangle_{0<i\leq t}$, ψ)
$\forall i$, a_i an action, o_i an observation, ψ a belief-state formula.
(1) if $t = 0$, return ψ.
(2) return $o_t \wedge$ FF-Step(a_t, $precond_{a_t} \wedge$ FF($\langle a_i, o_i \rangle_{0<i\leq(t-1)}$, ψ)).

PROCEDURE FF-Step(a, ψ)
a an action. ψ a belief-state formula.
(1) if ψ is an atomic formula, then return Single-Literal-Filtering(a, ψ).
(2) else, use distribution properties as follows:
(3) if $\psi = \phi \vee \varphi$, return FF-Step(a, ϕ) $\vee$ FF-Step(a, φ).
(4) elseif $\psi = \phi \wedge \varphi$, return FF-Step($a$, ϕ) $\wedge$ FF-Step(a, φ).
(5) elseif $\psi = \neg\phi$, return $\neg$ FF-Step(a, ϕ) $\wedge$ FF-Step(a, $TRUE$).
(6) elseif $\psi = \exists x\ \phi(x)$, return $\exists x$ FF-Step(a, $\phi(L)$)$_{[L/x]}$.

Fig. 1. Factored filtering of a first-order formula in a permutation domain.

Theorem 14 The algorithm FF is correct when progression is first-order definable. It would run in time $O(|precond_a \wedge \psi| \times F)$ when the filtering of all the atomic formulas are given and has finite representation, ψ is the belief state formula, and F is the time needed to retrieve the filtering of an atomic formula.

PROOF See appendix A.9 ■

Our algorithm FF uses consequence finding tools for filtering the atomic subformulas. Since it is a part of preprocessing, it does not affect the response time of our system. If new constants are added during the system operation, the response time would be different. In that case the filtering of new closed atomic formulas should be computed online. Given an algorithm for computing the progression of atomic formulas then FF shows how we can solve the general problem of progression in polynomial time. In this paper we do not provide any algorithm for atomic subformulas

(we assume that Single-Literal-Filtering of Figure 1 is provided as an input to our algorithm).

5 Filtering Algorithms for Different Domains

Our naive filtering algorithm uses consequence finding tools which do not scale to large domains. Theorem 15 suggests a different reasoning procedure by which updates are broken into smaller local updates. Before that, we need to define the set of *derived-formulas* of a set of predicates.

Definition 15 (Derived-Formulas) Let $p_1, \ldots, p_n$ be n closed predicates. $DF(p_1, \ldots, p_n)$ is the set of derived-formulas of $p_1, \ldots, p_n$ which is defined inductively as follows:

(1) $\{p_1, \ldots, p_n\} \subset DF(p_1, \ldots, p_n)$
(2) $\neg\phi \in DF(p_1, \ldots, p_n)$ iff $\phi \in DF(p_1, \ldots, p_n)$
(3) $\phi \wedge \psi \in DF(p_1, \ldots, p_n)$ iff $\{\phi, \psi\} \subset DF(p_1, \ldots, p_n)$
(4) $\forall x\ \phi(y_1, \ldots, y_m)_{[y_1/x, \ldots, y_m/x]}$ if $\phi(y_1, \ldots, y_m) \in DF(p_1, \ldots, p_n)$

For example,

$$\forall x \forall y\ book(x) \wedge room(y) \wedge in(x, y) \in DF(book(x_1), room(x_2), in(x_3, x_4))$$

Since $book(x_1) \wedge room(x_2) \wedge in(x_3, x_4)$ is a derived-formula and we can perform rule 4 on this formula and substitute x_1 and x_3 by x, and x_2 and x_4 by y.

Theorem 16 Let progression be first-order definable, a be a ground action, ψ be a belief state formula, and $p_1, \ldots, p_n$ be n closed predicates in our action theory. Then,

$$\begin{aligned} Filter[a](\psi) \equiv \{\Phi(p_1, \ldots, p_n) \mid\ & \Phi(p_1, \ldots, p_n) \in DF(p_1, \ldots, p_n), \\ & \psi \wedge precond_a \models \Phi(p_1, \ldots, p_n)_{[p_1/succ_{p_1,a}, \ldots, p_n/succ_{p_n,a}]}\ \} \end{aligned} \tag{3}$$

PROOF See appendix A.10 ■

In this formula, every Φ in $DF(p_1, \ldots, p_n)$ that is entailed by our belief state and action precondition is in the filtering. In general, generating all Φs is impossible because there are infinitely many such Φs. In the following sections, we provide simpler closed-form solutions for two special cases of dynamic domains. These give rise to practical (polynomial) algorithms.

5.1 Unit-Case Successor State Axioms

By definition of successor state axioms, for every pair of actions and predicates exactly one successor state axiom is provided. The successor state axiom for action $a = A(x_1, \ldots, x_n)$ and predicate p_i can be rewritten in the following form:

$$\begin{aligned} &Poss(A(x_1, \ldots, x_n), s) \Rightarrow \forall y_1, \ldots, \forall y_m \ (p_i(y_1, \ldots, y_m, do(a, s)) \Leftrightarrow \\ &\quad (case_i^1 \Rightarrow \phi_i^1(x_1, \ldots, x_n, y_1, \ldots, y_m, s)) \\ &\quad \wedge \ldots \\ &\quad \wedge (case_i^{l_i} \Rightarrow \phi_i^{l_i}(x_1, \ldots, x_n, y_1, \ldots, y_m, s)) \\ &\quad \wedge (\neg case_i^1 \wedge \ldots \wedge \neg case_i^{l_i}) \Rightarrow \phi_i^{l_i+1}(x_1, \ldots, x_n, y_1, \ldots, y_m, s)) \end{aligned}$$

where $case_i^j$ is of the form $(y_i^{j_1} = x_i^{j_1}) \wedge \ldots \wedge (y_i^{j_k} = x_i^{j_k})$ (variable $x_i^{j_1}$ is an argument of action a and variable $y_i^{j_1}$ is an argument of predicate p_i) and each variable assignment satisfies at most one of the cases. A successor state axiom is called *unit-case successor state axiom* if it can be rewritten in a form where every $\phi_i^j(x_1, \ldots, x_n, y_1, \ldots, y_m, s)$ $(1 \leq j \leq l_i + 1)$ is a unit clause.

We divide a unit-case successor state axiom into multiple instantiated axioms. Instantiated successor state axioms for predicate p_i are:

- $Poss(A(x_1, \ldots, x_n), s) \Rightarrow (p_i(y_1, \ldots, y_m, do(a, s))$
  $\Leftrightarrow \phi_i^j(x_1, \ldots, x_n, y_1, \ldots, y_m, s))_{[y_i^j / x_i^j]}$ for all $1 \leq j \leq l_i$
- $Poss(A(x_1, \ldots, x_n), s) \Rightarrow \forall y_1, \ldots, y_m(\neg case_i^1 \wedge \ldots \wedge \neg case_i^{l_i})$
  $\Rightarrow (p_i(y_1, \ldots, y_m, do(a, s)) \Leftrightarrow \phi_i^{l_i+1}(x_1, \ldots, x_n, y_1, \ldots, y_m, s))$

$[y_i^j / x_i^j]$ is the substitution corresponding to $case_i^j$ (y_i^j and x_i^j are sequences

PROCEDURE UCF($\langle a_i, o_i \rangle_{0<i\leq t}$, ψ)
$\forall i$, a_i an action, o_i an observation, ψ a belief-state formula.
(1) if $t = 0$, return ψ.
(2) $\psi_{t-1} = \text{UCF}(\langle a_i, o_i \rangle_{0<i\leq(t-1)}, \psi)$.
(3) return $o_t \wedge$ Filter-True$(a_t) \wedge$ UCStep$(a_t, precond_{a_t} \wedge \psi_{t-1})$.

PROCEDURE Filter-True($A(x_1, \ldots, x_n)$)
$A(x_1, \ldots, x_n)$ an action.
(1) $Poss(A(x_1, \ldots, x_n), s) \Rightarrow (cond_i \Rightarrow (p_i(z_1, \ldots, z_m, do(a, s)) \Leftrightarrow$
(2) $\phi_i(z_1, \ldots, z_{m+n}, s)))$ an instantiated successor state
(3) axiom $1 \leq i \leq k$
(4) $S = \emptyset$
(5) for all $1 \leq i, j \leq k$,
(6) if $\phi_i(z_1, \ldots, z_{m+n}) = true$, add $cond_i \Rightarrow p_i(z_1, \ldots, z_m)$ to S
(7) elseif $\phi_i(z_1, \ldots, z_{m+n}) = false$, add $cond_i \Rightarrow \neg p_i(z_1, \ldots, z_m)$ to S
(8) elseif unifiable$(\phi_i(z_1, \ldots, z_{m+n}), \phi_j(z_1, \ldots, z_{m+n}))$,
(9) add $((cond_i \wedge cond_j) \Rightarrow (p_i(z_1, \ldots, z_m) \Leftrightarrow$
(10) $p_j(z_1, \ldots, z_m)))_{mgu(\phi_i, \phi_j)}$ to S
(11) elseif unifiable$(\phi_i(z_1, \ldots, z_{m+n}), \neg\phi_j(z_1, \ldots, z_{m+n}))$,
(12) add $((cond_i \wedge cond_j) \Rightarrow (p_i(z_1, \ldots, z_m) \Leftrightarrow$
(13) $\neg p_j(z_1, \ldots, z_m)))_{mgu(\phi_i, \phi_j)}$ to S
(14) elseif $\phi_i(z_1, \ldots, z_{m+n}) = \forall x\ q(x), \phi_j(z_1, \ldots, z_{m+n}) = q(T)$,
(15) add $(cond_i \wedge cond_j) \Rightarrow (\neg p_i(z_1, \ldots, z_m) \vee p_j(z_1, \ldots, z_m))$ to S
(16) elseif $\phi_i(z_1, \ldots, z_{m+n}) = \exists x\ q(x), \phi_j(z_1, \ldots, z_{m+n}) = q(T)$,
(17) add $(cond_i \wedge cond_j) \Rightarrow (p_i(z_1, \ldots, z_m) \vee \neg p_j(z_1, \ldots, z_m))$ to S
(18) return $\bigwedge_{\varphi \in S} \varphi$.

Fig. 2. Unit-Case Filtering.

of variables). This process is called *breaking into cases*. Note that all instantiated successor state axioms are in the form

$$Poss(A(x_1, \ldots, x_n), s) \Rightarrow$$

$$(cond_i \Rightarrow (p_i(z_1, \ldots, z_m, do(a, s)) \Leftrightarrow \phi_i(z_1, \ldots, z_{m+n}, s)))$$

where in some of them $cond_i$ is TRUE (i is an enumeration of all instantiated successor state axioms of action a). In this formula z_i is either a variable which is universally quantified or one of the parameters of action $A(x_1, \ldots, x_n)$. The parameters of action $A(x_1, \ldots, x_n)$ are free variables.

Figures 2 and 3 show the Unit-Case Filtering (UCF) algorithm. This

PROCEDURE UCStep($A(x_1, \ldots, x_n)$, ψ)
$A(x_1, \ldots, x_n)$ an action. ψ a belief-state formula.
(1) if ψ is an atomic formula then
(2) $Poss(A(x_1, \ldots, x_n), s) \Rightarrow (cond_i \Rightarrow (p_i(z_1, \ldots, z_m, do(a, s)) \Leftrightarrow$
(3) $\phi_i(z_1, \ldots, z_{m+n}, s)))$ an instantiated successor state
(4) axiom $1 \leq i \leq k$.
(5) $S = \emptyset$
(6) for all $1 \leq i \leq k$,
(7) if unifiable$(\phi_i(z_1, \ldots, z_{m+n}), \psi)$
(8) add $(cond_i \Rightarrow p_i(z_1, \ldots, z_m))_{mgu(\phi_i, \psi)}$ to S
(9) elseif unifiable$(\phi_i(z_1, \ldots, z_{m+n}), \neg\psi)$
(10) add $(cond_i \Rightarrow \neg p_i(z_1, \ldots, z_m))_{mgu(\phi_i, \psi)}$ to S
(11) return $\bigwedge_{\varphi \in S} \varphi$.
(12) else, use distribution properties as follows:
(13) if $\psi = \phi \vee \varphi$
(14) return UCStep$(A(x_1, \ldots, x_n), \phi) \vee$ UCStep$(A(x_1, \ldots, x_n), \varphi)$.
(15) elseif $\psi = \phi \wedge \varphi$
(16) return UCStep$(A(x_1, \ldots, x_n), \phi) \wedge$ UCStep$(A(x_1, \ldots, x_n), \varphi)$.
(17) elseif $\psi = \neg\phi$
(18) return $\neg$ UCStep$(A(x_1, \ldots, x_n), \phi) \wedge$
(19) UCStep$(A(x_1, \ldots, x_n)$, TRUE).
(20) elseif $\psi = \exists x\ \phi(x)$, return $\exists x$ UCStep$(A(x_1, \ldots, x_n)s, \phi(L))_{[L/x]}$.

Fig. 3. One Step Unit-Case Filtering.

algorithm is applicable to permutation domains whose successor state axioms are unit-case. Algorithm UCF is actually a way to compute every $\Phi(p_1, \ldots, p_n)$ in formula (3). Here, $p_1, \ldots, p_n$ are the predicates of the instantiated successor state axioms after breaking into cases. Therefore, some of the arguments of these predicates are constants appeared in our ground action.

In permutation domains the head of the entailment of formula (3) is an atomic formula since we use distribution properties of Theorem 12 on $\psi \wedge precond_a$. The distribution properties (discussed above) break the conjunction of belief state and precondition into atomic subformulas. Consequently, $\Phi(p_1, \ldots, p_n)_{[p_1/succ_{p_1,a}, \ldots, p_n/succ_{p_n,a}]}$ is either equivalent to that atomic formula or a tautology. The size of a tautology is at most two when unit-case successor state axioms are used. Therefore, we can compute all desired Φs in a finite number of steps which is the foundation of the UCF algorithm.

Theorem 17 Let progression be first-order definable, k be the number of successor state axioms after breaking into cases, and ψ be the belief state formula. Algorithm UCF returns the filtering of ψ with a and o in time $O(Rk^2 + Rk|\psi \wedge precond_a|)$ where R is the maximum arity of all predicates. The length of the returned formula is $O(Rk^2 + Rk|\psi \wedge precond_a|)$.

PROOF See appendix A.11 ■

5.1.1 Example

Consider our previous book-keeping robot example. Suppose that we have two rooms, office and lounge, and two books, C++ and Java, and our belief state formula is $in(C++,\ Office) \wedge in(Java,\ Lounge)$. Assume students studying in the lounge need the C++ tutorial for a while, so the book keeper robot decides to move the book to the lounge. The action is $move(C++,\ Office,\ Lounge)$. This example has unit-case successor state axioms.

First, we add the precondition to the belief state formula. The new belief state is:

$$in(C++,\ Office) \wedge in(Java,\ Lounge) \wedge book(C++) \wedge room(Office) \wedge room(Lounge)$$

We calculate the filtering of all the atomic formulas of the belief state formula separately and compute the result by using distribution properties. What follows is the formula for one of these atomic formulas based on the algorithm presented above.

$$Filter[move(C++,\ Office,\ Lounge)](in(Java,\ Lounge)) \equiv in(C++,\ Lounge) \wedge \neg in(C++,\ Office) \wedge in(Java,\ Lounge)$$

Now suppose that we filter the belief state after perceiving the following observation. The robot enters the office and it observes that there is only one book in the room. A perfect filtering algorithm guarantees that in such cases the book is the same book that the robot has put in the room before.

Assume that the belief state formula is $in(C{+}{+},\ \mathit{Office}) \wedge in(\mathit{Java},\ \mathit{Lounge})$. The observation is $\forall x\ in(x, \mathit{Office}) \Rightarrow x = \mathit{TheBook}$. $Filter[o](\psi) \models \mathit{TheBook} = C{+}{+}$, and we can replace every instance of *TheBook* in the new belief state formula by $C{+}{+}$.

5.2 STRIPS Domains

In STRIPS domains every action has no conditional effects. It means that the value of each predicate either changes to true, changes to false, or remains the same. STRIPS actions are not necessarily permuting. Consequently STRIPS successor state axioms can not be treated by algorithm UCF even though they are unit-case. Successor state axioms in STRIPS domains are of the form:

$$\begin{aligned} \mathit{Poss}(a,s) \Rightarrow \forall y_1,\ldots,\forall y_m\ (p_i(y_1,\ldots,y_m,do(a,s)) \Leftrightarrow \\ (case_i^1 \Rightarrow \phi_i^1(y_1,\ldots,y_m,s)) \\ \wedge \ldots \\ \wedge (case_i^{l_i} \Rightarrow \phi_i^{l_i}(y_1,\ldots,y_m,s)) \\ \wedge (\neg case_i^1 \wedge \ldots \wedge \neg case_i^{l_i}) \Rightarrow p_i(y_1,\ldots,y_m,s)) \end{aligned}$$

where $\phi_i^j(y_1,\ldots,y_m,s)$ $(j \leq l_i)$ is either TRUE or FALSE.

A STRIPS action affects the truth value of some of the instantiated predicates and keeps the value of the others. We refer to the set of affected instantiated predicates as Eff(a).

$$\begin{aligned} \mathrm{Eff}(a) = \{p(v_1,\ldots,v_m) \mid \\ \text{ground action } a \text{ affects every instance of } p(v_1,\ldots,v_m)\} \end{aligned}$$

where v_i is a variable or a constant symbol (l_i instantiated predicates are added to Eff(a) whose predicate symbols are p_i). In other words, Eff(a) is defined as follows:

$$\begin{aligned} \mathrm{Eff}(a) = \{p_i(v_1,\ldots,v_m) \mid p_i(v_1,\ldots,v_m) = p_i(y_1,\ldots,y_m)_{[y_i^j/v_i^j]} \quad j \leq l_i \\ [y_i^j/v_i^j] \text{ is the substitution corresponding to } case_i^j\} \end{aligned}$$

PROCEDURE FOSF($\langle a_i, o_i \rangle_{0<i\leq t}$, ψ)
$\forall i$, a_i an action, o_i an observation and ψ a belief state formula. o_i and ψ in EAFOL

(1) if $t = 0$, return ψ.
(2) return Move-Quan [a] ($o_t \wedge$ FO-STRIPS-Step(a_t,
(3) Move-Quan($precond_{a_t} \wedge$ FOSF($\langle a_i, o_i \rangle_{0<i\leq(t-1)}$, ψ))))

[a] Moves all the quantifiers to the front with fresh variable names

PROCEDURE FO-STRIPS-Step(a, ψ)
a an action, $\psi = \exists^*\forall^* \bigwedge_i c_i$ a belief-state formula.

(1) if $\psi = \exists x\ \phi(x)$, return $\exists x$ FO-STRIPS-Step(a, $\phi(L)$)$_{[L/x]}$
(2) // L is a fresh constant that does not appear in the language
(3) elseif $\psi = \forall x\ \phi(x)$, return $\forall x$ FO-STRIPS-Step(a, $\phi(x)$)
(4) else
(5) $\psi' \leftarrow$ split every clause in ψ into pure literal clauses
(6) $E \leftarrow$ all clauses in ψ' with any literal in Eff(a)
(7) $S \leftarrow$ all clauses in ψ' with no literal in Eff(a)
(8) if $E \neq \emptyset$
(9) for all $l \in$ Eff(a)
(10) $E \leftarrow$ resolve-out(l, E)
(11) $\phi = \bigwedge_{c_i \in E \cup S} c_i$
(12) Eff$^+$(a) $\leftarrow$ literals affected to true
(13) Eff$^-$(a) $\leftarrow$ literals affected to false
(14) return $\phi \wedge \bigwedge_{p(v) \in \text{Eff}^+(a)} p(v) \wedge \bigwedge_{p(v) \in \text{Eff}^-(a)} \neg p(v)$.

Fig. 4. First-Order STRIPS Filtering.

The First-Order STRIPS Filtering (FOSF) algorithm is presented in figure 4. The belief state formula $\exists^*\forall^*\phi$ which is an input to this algorithm should be in EAFOL with no equality as well as all the observation formulas (a first-order formula is in EAFOL if no existential quantifier appears within the scope of a universal quantifier). ϕ is rewritten in a clausal form. A clause in the clausal form of ϕ is splitted into multiple instantiated clauses if one of its predicates has some instances in Eff(a) and some out of Eff(a). This step is called *splitting into pure literal clauses*. We split every clause $p(x) \vee \varphi$ in which some instantiations of $p(x)$ are affected and some are not into:

$$p(v_1) \vee \varphi\ ,\ \ldots\ ,\ p(v_r) \vee \varphi\ ,\ ((x \neq v_1 \wedge \ldots \wedge x \neq v_r) \Rightarrow p(x)) \vee \varphi$$

where $p(v_1), \ldots, p(v_r)$ are in Eff(a). Note that we treat $((x \neq v_1 \wedge \ldots \wedge x \neq$

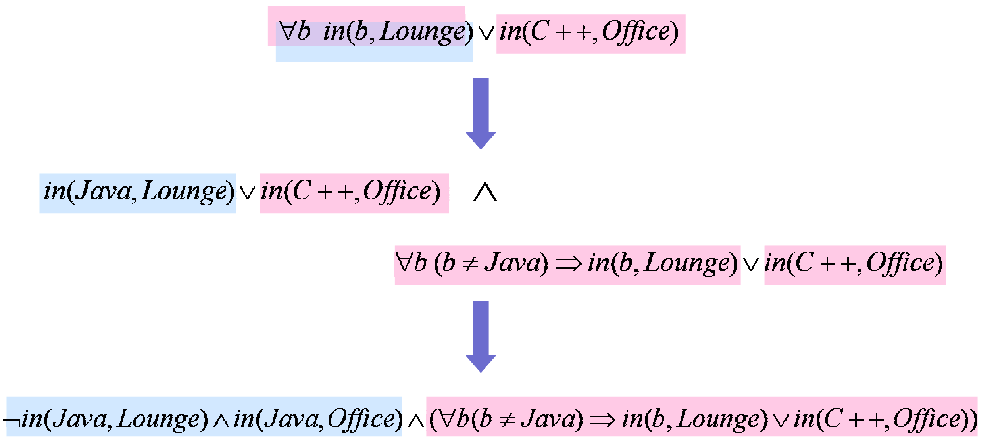

Fig. 5. STRIPS example: boxed literals are affected, filled ones are not.

$v_r) \Rightarrow p(x))$ as a single literal which is not affected by our action.

The new clausal form of ϕ is divided into two parts: clauses with no affected literals and clauses that have at least one affected literal (affected literal is a literal whose predicate is in $\text{Eff}(a)$). The algorithm directly copies the first part to the new belief state formula. It also adds all the consequences of the second part in which no affected literal exists, to the new belief state formula. Then all predicates in $\text{Eff}(a)$ are added to the new belief state formula as positive or negative literals (depending on whether they change to TRUE or FALSE).

An example of how this algorithm works is shown in figure 5. The action is $move(Java, Lounge, Office)$. You can find the successor state axioms of this action in section 2.2.1. Based on this axiom:

$$\text{Eff}(move(Java, Lounge, Office)) = \{in(Java, Lounge), in(Java, Office)\}$$

The top line in the figure is the current belief state. Some instances of the first literal in the belief state are affected and some are not ($in(Java, Lounge)$ is affected, and for every other book $in(b, Lounge)$ is not affected). We split this clause into two clauses. These clauses are pure since there is no literal in them that has both affected and unaffected instances. The middle line of the figure shows the belief state after splitting into pure literal clauses. In the first clause one of the literals is affected, so we try to resolve it with other clauses in our knowledge base (since we have no other

clauses, it is discarded). The second clause has no affected literal. We copy it directly to the new belief state. At the end, all the predicates in Eff($move(Java, Lounge, Office)$) are added as positive or negative literals.

Theorem 18 Let progression be first-order definable. Given action a, observation o, and belief state formula ψ in EAFOL algorithm FOSF returns the filtering of ψ with a and o in time $O(R\,(\frac{|E|}{2})^{2^{|\text{Eff(a)}|}} + |\psi|)$ where E is the set of all clauses with at least one affected literal (after splitting into pure literal clauses) and R is the maximum size of all clauses.

PROOF See appendix A.12 ■

Theorem 19 Let progression be first-order definable. If ψ is in 2-FO-CNF[3], the time complexity of algorithm FOSF after filtering with action a is $O(|\text{Eff}(a)| \times |E|^2 + |\psi|)$. The length of the returned formula is $O(|\text{Eff}(a)| + |E|^2 + |\psi|)$.

PROOF See appendix A.13 ■

6 Related Works

Most work related to ours is limited to exact filtering (belief update) in propositional cases or approximation to the true belief state in first-order cases. [2] presents efficient logical filtering algorithms that maintain a compact belief state representation indefinitely, for a broad range of environment classes including nondeterministic, partially observable successful-STRIPS environments (when success/failure of the action is known) and environments in which actions act as 1:1 mapping on states. Still, their work is restricted to the propositional case. Most of the works in first-order belief update approximate the true belief state. An example of such a work is [4] that investigates the idea of maintaining a compact approximation to the true belief state, and analyzes the conditions under which the errors due to the approximations do not accumulate to make answers completely irrelevant.

[3] An EAFOL is in k-FO-CNF if it is in clausal CNF, and the size of each clause is at most k.

There are other significant works that used FOL for filtering. However, they made different assumptions to make it tractable. For example, [10] studies how to exploit correspondence between the situation calculus of [21] and relational databases to build systems for reasoning about actions based on standard relational database technology. However, it works only when the initial knowledge base has complete knowledge. Follow up to this work is [19] which uses database techniques in the incomplete case. They propose a tractable solution to the projection problem (determining if a formula holds after a sequence of actions has been performed) in the presence of limited form of incomplete first-order knowledge and context-dependent actions. They restrict their work to the actions with local effects. Also, the query should be in a certain normal form and the agent should have complete knowledge about the context of any context-dependent actions. There is also a very recent work [27] that presents a method for progression based on updating directly the set of affected fluents similar to our methods but under different assumptions for the form of the successor state axioms. They assume that if an action changes the truth value of a fluent then the arguments of the fluent is contained in the arguments of the action, also if the change depends on another fluent then the same restriction holds for this new fluent. They also assume uniqueness of names for constants (unlike us).

Some other related works that used FOL for belief update are [5; 6] (First-Order AGM belief revision), [28] (belief update and revision in simple subclasses of FOL), and [18] (progression in the Situation Calculus). The important difference that we draw with these works is that ours provides efficient inference procedures, while others focus on the use of general-purpose theorem provers, and intractability results.

Another work related to ours is [16] which presents an approach for incorporating a new piece of knowledge into knowledge bases about actions and change. Their approach is different from all other belief revision operations since they consider another criterion: A possible cause explaining the new fact and all the effects of the possible cause should be added and all cause-effect relations should be maintained. Their approach is called possible cause approach (PCA).

Another group of related works show how belief update and progression are correlated with each other. [17] show how several different semantics for belief update can be expressed in a framework for reasoning about

actions which can be considered as a common core of all these update formalisms. On the other hand, [14] shows that belief update is a specific case of feedback-free action progression. It also investigates new issues, especially reverse update, which is to regression is what update is to progression.

Other works that are between Situation Calculus and Fluent Calculus are [25] and [26]. [25] introduces a dual representation for basic action theories based on state update axioms that explicitly define the direct effects of each action. The paper starts from the Situation Calculus where successor state axioms are used to solve the frame problem and it shows how the Fluent Calculus can be viewed as the result of gradually improving this logic to improve inferential aspects. [26] presents an implementation technique for progression based on the state update axioms and the use of a constraint solver. Another work ([12]) extends the fluent calculus by a method for belief change, which allows agents to revise their internal model upon making observations that contradict this model.

Another work that is based on dynamic logic is [11] which defines a logical framework for belief change. Dynamic logic is a viable alternative to situation calculus approaches because of its simplicity and existence of proof procedures for it. This paper presents the semantics of belief change (when an action actually has a behavior that is different from what we always believed it had) and its syntactical counterpart.

There are a few works on iterated belief revision which is related to ours. [23] provide a new approach to belief change associated with performing actions which is based on a theory of action in the situation calculus extended to deal with belief. Their approach also handles nested belief, belief introspection, mistaken belief, and belief revision and belief update together with iterated belief change. Their belief revision corresponds to our observations, where belief update corresponds to update with actions. [13] also provide a base belief revision operator which is used for iterated revision. They show that this base revision operator is almost optimal in terms of computational complexity.

Another related work ([1]) provides translations of some of the most known action description languages into a logic program language (Evolp).

7 Conclusions

We presented semantics and methodology for filtering in domains that include many objects whose identity is not certain. We generalized this problem to filtering with FOL formulas. Assuming that progression is first-order definable, we showed that this problem is solvable in polynomial time when actions map states 1:1, or the actions are STRIPS and we know that the action succeeded. We showed that 1:1 actions allow us to filter first-order belief state formulas in linear time (in the size of representation), given an algorithm for computing the progression of atomic formulas. When actions are successful-STRIPS or Unit-Case, we can filter these belief state formulas efficiently without precompilation. We showed that in some cases the belief state formulas is guaranteed to remain represented compactly. Those cases permit filtering of actions and observations indefinitely in polynomial time (in the number of predicates and constants). We expect to use these algorithms for many interesting applications, such as semantic web, autonomous agents, robot motion control, and partial knowledge games.

8 Acknowledgements

We wish to thank Megan Nance for useful discussions on related topics. We also wish to acknowledge support from DAF Air Force Research Laboratory Award FA8750-04-2-0222 (DARPA REAL program).

References

[1] José Júlio Alferes, Federico Banti, and Antonio Brogi. From logic programs updates to action description updates. In *CLIMA V*, pages 52–77, 2004.

[2] Eyal Amir and Stuart Russell. Logical filtering. In *IJCAI '03*, pages 75–82. MK, 2003.

[3] A. Blass and Y. Gurevich. Background, Reserve, and Gandy Machines. In P. Clote and H. Schwichtenberg, editors, *Computer Science Logic*

(Proceedings of CSL 2000), volume 1862 of *LNCS*, pages 1–17. Springer, 2000.

[4] Xavier Boyen and Daphne Koller. Tractable inference for complex stochastic processes. In *Proc. UAI '98*, pages 33–42. MK, 1998.

[5] Maria R. Cravo, João P. Cachopo, Ana C. Cachopo, and João P. Martins. Permissive belief revision. In *EPIA*, pages 335–348, 2001.

[6] Simon Dixon and Wayne Wobcke. The implementation of a first-order logic agm belief revision system. In *ICTAI*, pages 40–47, 1993.

[7] Arnaud Doucet, Nando de Freitas, Kevin Murphy, and Stuart Russell. Rao-Blackwellised particle filtering for dynamic bayesian networks. In *Proc. UAI '00*, pages 176–183. MK, 2000.

[8] T. Eiter and G. Gottlob. On the complexity of propositional knowledge base revision, updates, and counterfactuals. *AIJ*, 57(2-3):227–270, 1992.

[9] Zoubin Ghahramani. An introduction to Hidden Markov Models and Bayesian networks. *Intl J. of Pattern Recog. and AI*, 15(1):9–42, 2001.

[10] G. De Giacomo and T. Mancini. Scaling up reasoning about actions using relational database technology. In *Proc. AAAI '04*, pages 245–250, 2004.

[11] Andreas Herzig, Laurent Perrussel, and Ivan Varzinczak. Elaborating domain descriptions. In *In Proc. 17th Eur. Conf. on Artificial Intelligence (ECAI06), Riva del Garda*. IOS Press, 2006.

[12] Yi Jin and Michael Thielscher. Representing beliefs in the fluent calculus. In *ECAI*, pages 823–827, 2004.

[13] Yi Jin and Michael Thielscher. Actions and belief revision: A computational approach. In *Belief Change in Rational Agents*, 2005.

[14] Jérôme Lang. Belief update revisited. In *IJCAI*, pages 2517–2522, 2007.

[15] Hector Levesque, Fiora Pirri, and Ray Reiter. Foundations for the situation calculus. *ETAI*, 3:nr 18, December 1998.

[16] Renwei Li and Lu'is Moniz Pereira. What is believed is what is explained (sometimes. In *In Proc. of AAAI'96*, pages 550–555, 1996.

[17] Paolo Liberatore. A framework for belief update. In *JELIA '00: Proceedings of the European Workshop on Logics in Artificial Intelligence*, pages 361–375, London, UK, 2000. Springer-Verlag.

[18] Fangzhen Lin and Ray Reiter. How to Progress a Database. *AIJ*, 92(1-2):131–167, 1997.

[19] Yongmei Liu and Hector J. Levesque. Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions. In *IJCAI '05*, pages 522–527, 2005.

[20] Sheila McIlraith. Explanatory diagnosis: Conjecturing actions to explain observations. In Anthony G. Cohn, Lenhart Schubert, and Stuart C. Shapiro, editors, *Proc. KR '98*, pages 167–177. MK, San Francisco, California, 1998.

[21] Raymod Reiter. *Knowledge In Action: Logical Foundations for Describing and Implementing Dynamical Systems*. MIT Press, 2001.

[22] Raymond Reiter. Proving properties of states in the situation calculus. *Artificial Intelligence*, 64(2):337–351, 1993.

[23] Steven Shapiro and Maurice Pagnucco. Iterated belief change and exogeneous actions in the situation calculus. In *ECAI*, pages 878–882, 2004.

[24] Afsaneh Shirazi and Eyal Amir. First-order logical filtering. In *IJCAI '05*, pages 589–595, 2005.

[25] Michael Thielscher. From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, 111:277–299, 1999.

[26] Michael Thielscher. Flux: A logic programming method for reasoning agents. *Theory Pract. Log. Program.*, 5(4-5):533–565, 2005.

[27] Stavros Vassos, Gerhard Lakemeyer, and Hector. J. Levesque. First-order strong progression for local-effect basic action theories. In *11th International Conference on the Principles of Knowledge Representation and Reasoning*.

[28] Mary-Anne Winslett. *Updating Logical Databases*. Cambridge U. Press, 1990.

A Proofs

A.1 Proof of Theorem 6: Filtering Algorithm

We show that the left-hand side is contained in the right-hand side.

$$Filter[a](\{S \mid \models_S \psi\}) \subseteq \{S' \mid \models_{S'} Filter[a](\psi)\}$$

Take $S' \in Filter[a](\{S \mid \models_S \psi\})$. From Definition 5, there is an S such that $S \in \{S \mid \models_S \psi\}$ and $\langle S, a, S' \rangle \in \mathcal{R}_\mathcal{D}$. In other words, there is an S such that $\models_S \psi$ and $\langle S, a, S' \rangle \in \mathcal{R}_\mathcal{D}$.

To prove the theorem, we need to show that $\models_{S'} Filter[a](\psi)$ (S' is in the right-hand side). Note that S and S' are restricted to our first order language L. Let $\mathcal{P}$ be the set of all constants and predicates in L. For each element of $\mathcal{P}$ ($g \in \mathcal{P}$) we define a new symbol g'. The set of all these new symbols is called $\mathcal{P}'$. We define structure S'' on language $L \cup \mathcal{P}'$ such that $|S''| = |S|$, for each $g \in \mathcal{P}$, $g^{S''} = g^S$ and for each $g' \in \mathcal{P}'$, $g'^{S''} = g^{S'}$ where g' is the new symbol for g.

We claim that $\models_{S''} \xi$ where ξ is

$$\begin{array}{l} \psi \wedge precond_a \wedge \\ \quad \bigwedge_i \forall y_1 \ldots \forall y_m \;\; p_i'(y_1, \ldots, y_m) \Leftrightarrow succ_{p_i,a}(y_1, \ldots, y_m) \wedge \\ \quad \bigwedge_i \forall z \;\; f_i' = z \Leftrightarrow succ_{f_i,a}(z) \end{array}$$

S'' does not model ξ only if one of the conjuncts is falsified. This in not the case for ψ or $precond_a$ by our choice of S. Assume by contradiction that this is the case for some i (*i.e.*, $p_i'(y_1, \ldots, y_m) \Leftrightarrow succ_{p_i,a}(y_1, \ldots, y_m)$ does not hold). From the way we defined $\mathcal{R}_\mathcal{D}$ this is never the case. The way we define $p_i^{S'}$ based on structure S always guarantees that $p_i'(y_1, \ldots, y_m)$ and $succ_{p_i,a}(y_1, \ldots, y_m)$ are equivalent. This contradicts our assumption. Thus, we get $\models_{S''} \xi$. Therefore $\models_{S''} Cn^{\mathcal{P}'}(\xi)$. Since the language of $Cn^{\mathcal{P}'}(\xi)$ is $\mathcal{L}(\mathcal{P}')$, the restriction of S'' to $\mathcal{L}(\mathcal{P}')$ (we refer to it as S''') also models $Cn^{\mathcal{P}'}(\xi)$.

If we replace symbols in $\mathcal{P}'$ by their corresponding symbols in $\mathcal{P}$, S''' would be equal to S'. Therefore, $\models_{S'} Filter[a](\psi)$. ∎

A.2 Proof of Corollary 7

We show that the two sets of structures have the same elements. By theorem 6 we know that the left-hand side of the equality is contained in the right-hand side.

$$Filter[a](\{S \mid \models_S \psi\}) \subseteq \{S' \mid \models_{S'} Filter[a](\psi)\}$$

For the opposite direction (showing the right-hand side is contained in the left-hand side), suppose that we can encode the left-hand side with a first order formula φ. Formula φ holds in all the members of the left-hand side. To prove the opposite direction we need to show that,

$$Filter[a](\psi) \models \varphi$$

Every model of $Filter[a](\psi)$ (right-hand side of equation) is also a model of φ (left-hand side of the equation). In other words, right-hand side is a subset of left-hand side.

To prove this statement we use the following lemma:

Lemma 20 Let φ' be the formula computed by replacing every predicate $p(y_1, \ldots, y_m)$ in φ with $succ_{p,a}(y_1, \ldots, y_m)$. We prove that:

$$\psi \wedge precond_a \models \varphi'$$

See Section A.3 ■

We know that $\varphi' = \varphi_{[p_i(y_1,\ldots,y_m)/succ_{p_i,a}(y_1,\ldots,y_m)]}$ and we use this lemma $\psi \wedge precond_a \models \varphi'$, Therefore,

$$\begin{array}{l}\psi \wedge precond_a \wedge \\ \bigwedge_i \forall y_1 \ldots \forall y_m \;\; p'_i(y_1, \ldots, y_m) \Leftrightarrow succ_{p_i,a}(y_1, \ldots, y_m) \wedge \\ \bigwedge_i \forall y_1 \ldots \forall y_m \forall z \;\; f'_i = z \Leftrightarrow succ_{f_i,a}(z) \models \varphi_{[\mathcal{P}/\mathcal{P}']}\end{array}$$

and the proof is done ($Filter[a](\psi) \models \varphi$). ■

A.3 Proof of Lemma 19

By contradiction assume that this is not the case. There is a structure S which $\models_S \psi \wedge precond_a$ but S is not a model of φ'. Since $\models_S \psi$ and $\models_S precond_a$, observing the way $\mathcal{R}_\mathcal{D}$ is defined, there should be a S' such that $\langle S, a, S' \rangle \in \mathcal{R}_\mathcal{D}$ and $S' \in Filter[a](\{S \mid \models_S \psi\})$. We encode $Filter[a](\{S \mid \models_S \psi\})$ with formula φ. So $\models_{S'} \varphi$.

Since we have replaced every predicate p in φ with $succ_{p,a}$ for getting φ', with $\models_{S'} \varphi$ and $\langle S, a, S' \rangle \in \mathcal{R}_\mathcal{D}$ we conclude that $\models_S \varphi'$. This contradicts our assumption. ■

A.4 Proof of Theorem 9: Progression Possibility

We want to prove:

$$\mathcal{D}_g \cup \mathcal{D}_{s_2} \models \psi \quad \textit{iff} \quad \mathcal{D}_g \cup \mathcal{D}_{s_0} \models \psi$$

Proving the forward direction is easy. We assume that $\mathcal{D}_g \cup \mathcal{D}_{s_2} \models \psi$ and we prove that $\mathcal{D}_g \cup \mathcal{D}_{s_0} \models \psi$. By definition:

$$\mathcal{D}_{s_2} = Cn_{s_2}(\mathcal{D}_{ss_{1,2}} \cup \mathcal{D}_{ap_{1,2}} \cup \mathcal{D}_{s_1})$$

$$\mathcal{D}_{s_1} = Cn_{s_1}(\mathcal{D}_{ss_{0,1}} \cup \mathcal{D}_{ap_{0,1}} \cup \mathcal{D}_{s_0})$$

Therefore, $\mathcal{D}_g \cup \mathcal{D}_{ss_{1,2}} \cup \mathcal{D}_{ap_{1,2}} \cup \mathcal{D}_{s_1} \models \psi$ (consequence finding procedure is sound). Now, we replace $\mathcal{D}_{s_1}$ by its definition. We conclude that

$$\mathcal{D}_g \cup \mathcal{D}_{ss_{1,2}} \cup \mathcal{D}_{ap_{1,2}} \cup \mathcal{D}_{ss_{0,1}} \cup \mathcal{D}_{ap_{0,1}} \cup \mathcal{D}_{s_0} \models \psi$$

and since $\mathcal{D}_{ss_{t-1,t}}$ and $\mathcal{D}_{ap_{t-1,t}}$ are part of $\mathcal{D}_g$, we can conclude that $\mathcal{D}_g \cup \mathcal{D}_{s_0} \models \psi$.

For the opposite direction, we should show that for a formula ψ uniform in $do(a_2, do(a_1, s_0))$ if $\mathcal{D}_g \cup \mathcal{D}_{s_0} \models \psi$, then $\mathcal{D}_g \cup \mathcal{D}_{s_2} \models \psi$. We can rewrite $\mathcal{D}_g$ as in the next formula.

$$\mathcal{D}_g \cup \mathcal{D}_{s_0} \models \psi$$
$$\mathcal{D}_g \cup \mathcal{D}_{ss_{0,1}} \cup \mathcal{D}_{ap_{0,1}} \cup \mathcal{D}_{ss_{1,2}} \cup \mathcal{D}_{ap_{1,2}} \cup \mathcal{D}_{s_0} \models \psi$$

From deduction theorem

$$\mathcal{D}_{ss_{0,1}} \cup \mathcal{D}_{ap_{0,1}} \cup \mathcal{D}_{s_0} \models (\mathcal{D}_g \wedge \mathcal{D}_{ss_{1,2}} \wedge \mathcal{D}_{ap_{1,2}}) \Rightarrow \psi$$

We use the following lemma:

Lemma 21 Let φ and ψ be two first-order formulas, and $f(T)$ be a ground function in $\mathcal{L}(\varphi) \cup \mathcal{L}(\psi)$. Let $\varphi_{[f(T):R]}$ be the formula that is obtained after replacing every $\forall x\ \phi$ where $f(x)$ is in ϕ with $(\forall x\ (x \neq T) \Rightarrow \phi) \wedge \phi_{[x/T]}$ and also replacing every instance of $f(T)$ with R. Then $\varphi \models \psi$ if and only if $\varphi_{[f(T):R]} \models \psi_{[f(T):R]}$ where R is a new constant symbol.

Using the lemma, we replace $do(a_1, s_0)$ with s_1 and $do(a_2, s_1)$ with s_2 in the above equation (note that a_1, a_2, s_0, s_1 , and s_2 are constants). After this replacement, the intersection of the language of the two sides of $\models$ does not include s_0, s_2, and function *do*.

Now, by applying Craig's interpolation theorem for FOL, we get that there should be a φ in $\mathcal{L}(\mathcal{D}_{ss_{0,1}} \cup \mathcal{D}_{ap_{0,1}} \cup \mathcal{D}_{s_0}) \cap \mathcal{L}((\mathcal{D}_g \wedge \mathcal{D}_{ss_{1,2}} \wedge \mathcal{D}_{ap_{1,2}}) \Rightarrow \psi)$ such that

$$\mathcal{D}_{ss_{0,1}} \cup \mathcal{D}_{ap_{0,1}} \cup \mathcal{D}_{s_0} \models \varphi$$
$$\varphi \models (\mathcal{D}_g \wedge \mathcal{D}_{ss_{1,2}} \wedge \mathcal{D}_{ap_{1,2}}) \Rightarrow \psi$$

Next, we replace back s_1 and s_2 with $do(a_1, s_0)$ and $do(a_2, do(a_1, s_0))$, respectively (using the result of the previous lemma, we know that this replacement does not affect the entailment). Since φ does not include any term of sort situation except $do(a_1, s_0)$, it is uniform in $do(a_1, s_0)$. From our definition of $\mathcal{D}_{s_1}$ we infer that $\varphi \in \mathcal{D}_{s_1}$. Therefore

$$\mathcal{D}_{s_1} \models (\mathcal{D}_g \wedge \mathcal{D}_{ss_{1,2}} \wedge \mathcal{D}_{ap_{1,2}}) \Rightarrow \psi$$
$$\mathcal{D}_{ss_{1,2}} \cup \mathcal{D}_{ap_{1,2}} \cup \mathcal{D}_{s_1} \models \mathcal{D}_g \Rightarrow \psi$$

With the same argument as before we can prove that

$$\mathcal{D}_{s_2} \models \mathcal{D}_g \Rightarrow \psi$$

and the opposite direction is done. ■

A.5 Proof of Lemma 20

By contradiction assume that $\varphi \models \psi$ but $\varphi_{[f(T):R]} \not\models \psi_{[f(T):R]}$. There is a structure M which $\models_M \varphi_{[f(T)|R]}$ but M is not a model of $\psi_{[f(T)|R]}$. Define M' equal to M except that $f(T)^{M'} = R^M$ and also M' does not have constant symbol R. With the definition of M', it is clear that $\models_{M'} \varphi$. Given our assumption $\models_{M'} \psi$.

We define M'' equal to M' except that it has a new constant symbol R for which $R^{M''} = f(T)^{M'}$. We can conclude that this new M'' is a model of $\psi_{[f(T):R]}$. M'' and M differ only in the interpretation of $f(T)$, and $\psi_{[f(T):R]}$ does not have any evaluation of $f(T)$. Therefore, $\models_M \psi_{[f(T)|R]}$.

The other direction is proved similarly. ■

A.6 Proof of Theorem 10: Distribution Over Connectives

The proof for the first three parts is similar to the proof given by [2] for propositional filtering. We use the result stated by Corollary 7 in our proof.

(1) Take a structure S' that satisfies $Filter[a](\varphi \vee \psi)$. Then, there is a structure S that satisfies $\varphi \vee \psi$ such that $\mathcal{R}_{\mathcal{D}}(S, a, S')$. Thus, S satisfies one of φ or ψ because S is a complete setting of the fluents. Therefore, S' is in one of $Filter[a](\varphi)$ or $Filter[a](\psi)$. It follows that $Filter[a](\varphi \vee \psi) \Rightarrow Filter[a](\varphi) \vee Filter[a](\psi)$.
For the other direction take S' that satisfies $Filter[a](\varphi) \vee Filter[a](\psi)$. Then, it satisfies one of $Filter[a](\varphi)$ or $Filter[a](\psi)$. Thus, there is a structure S such that $\mathcal{R}_{\mathcal{D}}(S, a, S')$ and S satisfies one of φ or ψ.

Thus, S satisfies $\varphi \vee \psi$ and S' satisfies $Filter[a](\varphi \vee \psi)$. It follows that $Filter[a](\varphi \vee \psi) \Leftarrow Filter[a](\varphi) \vee Filter[a](\psi)$.

(2) Take a structure S' that satisfies $Filter[a](\varphi \wedge \psi)$. Then, there is a structure S that satisfies $\varphi \wedge \psi$ such that $\mathcal{R}_{\mathcal{D}}(S, a, S')$. Thus, S satisfies both of φ and ψ. Thus, S' is in both of $Filter[a](\varphi)$ and $Filter[a](\psi)$. We conclude that every S' that satisfies $Filter[a](\varphi \wedge \psi)$ also satisfies $Filter[a](\varphi) \wedge Filter[a](\psi)$. It follows $\models Filter[a](\varphi \wedge \psi) \Rightarrow Filter[a](\varphi) \wedge Filter[a](\psi)$.

(3) Take S' that satisfies $\neg Filter[a](\varphi) \wedge Filter[a](TRUE)$. Then, there is no structure S that $\mathcal{R}_{\mathcal{D}}(S, a, S')$ and S satisfies φ. Thus, for every structure S that $\mathcal{R}_{\mathcal{D}}(S, a, S')$, S satisfies $\neg\varphi$. Since S' satisfies $Filter[a](TRUE)$ there is a structure S such that $R_D(S, a, S')$. This S satisfies $\neg\varphi$ so S' satisfies $Filter[a](\neg\varphi)$. It follows that $\models Filter[a](\neg\varphi) \Leftarrow \neg Filter[a](\varphi) \wedge Filter[a](TRUE)$.

(4) We used Skolemization for this part. We show that the two sets of world structures are equal. We first show that the left-hand side of the equation is contained in the right-hand side.

$$Filter[a](\exists x\ \varphi(x)) \equiv \exists x\ Filter[a](\varphi(L))_{[L/x]}$$

Take structure M such that $\models_M Filter[a](\exists x\ \varphi(x))$. We show that $\models_M \exists x\ Filter[a](\varphi(L))_{[L/x]}$.

Based on theorem 6:

$\models_M Filter[a](\exists x\ \varphi(x))$ iff $M \in Filter[a](\{S| \models_S \exists x\ \varphi(x))\}$

Based on definition 5:

$M \in \{S'| \models_S \exists x\ \varphi(x), \langle S, a, S'\rangle \in \mathcal{R}_{\mathcal{D}}\}$

Based on definition 4, there exists a S that:

$\models_S \exists x\ \varphi(x)$

$\models_S precond_a$

$|S| = |M|$

$p^M = \{\langle v_1, \ldots, v_m\rangle|\ \models_S succ_{p,a}(v_1, \ldots, v_m)\}$

$f^M = \{\langle v_1, \ldots, v_r\rangle|\ \models_S succ_{f,a}(v_1, \ldots, v_r)\}$

We define $S_{L,\varphi}$ equal to S with additional mapping of new constant symbol L to $v \in |S|$ such that $\models_S \varphi(v)$. In this case:

$\models_{S_{L,\varphi}} \varphi(L)$

We can progress $S_{L,\varphi}$ with action a.

$M' \in Filter[a](\{S_{L,\varphi}\})$ iff

$|M'| = |S_{L,\varphi}|$

$p^{M'} = \{\langle v_1, \ldots, v_m\rangle |\ \models_{S_{L,\varphi}} succ_{p,a}(v_1, \ldots, v_m)\}$

$f^{M'} = \{\langle v_1, \ldots, v_r\rangle |\ \models_{S_{L,\varphi}} succ_{f,a}(v_1, \ldots, v_r)\}$

We conclude that $\models_{M'} Filter[a](\varphi(L))$.

Define $\mathcal{D}_L$ equal to $\mathcal{D}$ with an extra constant symbol L. We can say that M is the restriction of M' to symbols in $\mathcal{D}$ because it behaves like M' on every element in the domain except that it does not have the symbol L.

We can use the following lemma to complete the proof.

Lemma 22 Take $\psi \in \mathcal{L}(\mathcal{D}_L)$, we can prove that:

$\models_M \exists x\ \psi_{[L/x]}$ iff $\exists M'\ \models_{M'} \psi$ such that M is a restriction of M' to $\mathcal{L}(\psi)\backslash\{L\}$

See Section A.7 ■

With this lemma the proof is complete. The other side is the same. All conclusions are if and only if so the other side is also proved.

■

A.7 Proof of Lemma 21

Suppose that the left-hand side is true but the right-hand side is not.

$\models_M \exists x\ \psi_{[L/x]}$ iff $\exists M'\ \models_{M'} \psi$ such that M is a restriction of M' to $\mathcal{L}(\psi)\backslash\{L\}$

It means there is no structure M' that models ψ and M is a restriction of M'. Assume that $v \in |M|$ is the x that makes the existential quantifier true. Add a constant L to M such that $L = v$ and call this new structure M''. $\models_{M''} \psi$ and M is a restriction of M'' to $\mathcal{L}(\psi)\backslash\{L\}$. We can conclude that the left-hand side implies the right-hand side.

Now for the other side, suppose that the right-hand side is true but the left-hand side is not. The element in the domain associated with L would

satisfy the existential quantifier and the proof is done. ■

A.8 Proof of Theorem 12

Theorem 10 supplies the proof of 1, the "⇒" direction of 2, the "⇐" direction of 3, and the proof of 4. We are left to prove the "⇐" direction of 2 and the "⇒" direction of 3 (similar to propositional case in [2]).

For "⇐" of 2, let S' be a structure that satisfies $Filter[a](\varphi) \wedge Filter[a](\psi)$. Then, it satisfies both of $Filter[a](\varphi)$ and $Filter[a](\psi)$. For $Filter[a](\varphi)$ there is a structure S such that $\mathcal{R}_{\mathcal{D}}(S, a, S')$ and S satisfies φ. Similarly, for $Filter[a](\psi)$ there is a structure S_1 such that $\mathcal{R}_{\mathcal{D}}(S_1, a, S')$ and S_1 satisfies ψ. However, since a acts as a one-to-one mapping between structures, there is only one structure that maps to S'. Thus, $S = S_1$, and S satisfies ψ. S satisfies $\varphi \wedge \psi$ and S' satisfies $Filter[a](\varphi \wedge \psi)$. It follows that $\models Filter[a](\varphi \wedge \psi) \Leftarrow Filter[a](\varphi) \wedge Filter[a](\psi)$.

For "⇒" of 3, let S' be a structure that satisfies $Filter[a](\neg\varphi)$. Then, there is a structure S that satisfies $\neg\varphi$ such that $\mathcal{R}_{\mathcal{D}}(S, a, S')$. Thus, S does not satisfies φ. Since a acts as a one-to-one mapping, there is only one structure that maps to S' after a. There is no structure S_1 that satisfies φ and for which $\mathcal{R}_{\mathcal{D}}(S_1, a, S')$. So, S' does not satisfy $Filter[a](\varphi)$ and therefore it satisfies $\neg Filter[a](\varphi)$. Clearly, S' also satisfies $Filter[a](TRUE)$. So S' satisfies $\neg Filter[a](\varphi) \wedge Filter[a](TRUE)$. It follows that $\models Filter[a](\neg\varphi) \Rightarrow \neg Filter[a](\varphi) \wedge Filter[a](TRUE)$.
■

A.9 Proof of Theorem 13

By theorem 12, we know that algorithm FF is correct. It decomposes $\psi \wedge precond_a$ into atomic sub-formulas by using distribution properties. Since we assume that we have the filtering of atomic formulas (as a precomputed table) we just retrieve it in time F and combine the results. Therefore, the time complexity of this algorithm is $O(|\psi \wedge precond_a| \times F)$.
■

A.10 Proof of Theorem 15

Using formula 1 we should show that every clause in formula 1 is also in the set of 3 and every Φ in 3 is a consequence of formula 1. We repeat formula 1 here.

1. $Filter[a](\psi) = (Cn^{\mathcal{P}'}(\psi \wedge precond_a$
   $\bigwedge_i \forall y_1, \ldots, y_m, p'_i(y_1, \ldots, y_m) \Leftrightarrow succ_{p_i,a}(y_1, \ldots, y_m)$
   $\bigwedge_i \forall z, f'_i = z \Leftrightarrow succ_{f_i,a}(z)))_{[\mathcal{P}'/\mathcal{P}]}$
2. $Filter[o](\psi) = \psi \wedge o$

Suppose that $\Phi(p'_1, \ldots, p'_n)$ is a consequence of formula 1 before applying $[\mathcal{P}'/\mathcal{P}]$. We want to show that $\psi \wedge precond_a \models \Phi(succ_{p_1,a}, \ldots, succ_{p_n,a})$. By the above definition

$$\psi \wedge precond_a \wedge \bigwedge_i \forall y_1 \ldots \forall y_m \; p'_i(y_1, \ldots, y_m) \Leftrightarrow succ_{p_i,a}(y_1, \ldots, y_m)$$
$$\bigwedge_i \forall z, f'_i = z \Leftrightarrow succ_{f_i,a}(z) \models \Phi(p'_1, \ldots, p'_n)$$

We refer to left-hand side as ξ. Therefore, for every structure M, if $\models_M \xi$ then $\models_M \Phi(p'_1, \ldots, p'_n)$ and since the value of every p'_i is equivalent to $succ_{p_i,a}$ in M, we conclude that $\models_M \Phi(succ_{p_1,a}, \ldots, succ_{p_n,a})$. As a result,

$$\xi \models \Phi(succ_{p_1,a}, \ldots, succ_{p_n,a})$$

Since p'_i does not appear in $\Phi(succ_{p_1,a}, \ldots, succ_{p_n,a})$ we conclude that $\psi \wedge precond_a \models \Phi(succ_{p_1,a}, \ldots, succ_{p_n,a})$ (the rest of ξ consists of some equivalence formulas about p'_i). One direction of the proof is done.

Now for the other direction, we show that for every $\Phi(p_1, \ldots, p_n)$ in the set of 3, $\xi \models \Phi(p'_1, \ldots, p'_n)$. Since $\psi \wedge precond_a \models \Phi(succ_{p_1,a}, \ldots, succ_{p_n,a})$, this part is also trivial. ■

Correctness. To prove that the algorithm UCF returns the filtering of ψ correctly we use the result of Theorem 15. We want to find all $\Phi(p_1, \ldots, p_n)$s such that $\psi \wedge precond_a \models \Phi(succ_{p_1,a}, \ldots, succ_{p_n,a})$.

The algorithm Filter-True finds all Φs that $true \models \Phi(succ_{p_1,a}, \ldots, succ_{p_n,a})$. In other words, $\Phi(succ_{p_1,a}, \ldots, succ_{p_n,a})$ should be a tautology. Since all ϕ_is are unit clauses, the size of tautologies is at most 2. The algorithm checks the unifiability of every pair of ϕ_i and ϕ_j and finds all the tautologies of size 2.

Next, we need to find all Φs that $\psi \wedge precond_a \models \Phi(succ_{p_1,a}, \ldots, succ_{p_n,a})$. With Theorem 12 we divide $\psi \wedge precond_a$ into atomic formulas. Let q be an atomic formula, we want to find all Φs that $q \models \Phi(succ_{p_1,a}, \ldots, succ_{p_n,a})$. We rewrite it as

$$true \models \neg q \vee \Phi(succ_{p_1,a}, \ldots, succ_{p_n,a})$$

Therefore, $\neg q \vee \Phi(succ_{p_1,a}, \ldots, succ_{p_n,a})$ should be a tautology. 1(c) of UCStep finds all such Φs.

Complexity. Suppose that we have t successor state axioms and we divide them into k instantiated successor state axioms of the form

$$Poss(A(x_1, \ldots, x_n), s) \Rightarrow$$
$$(cond_i \Rightarrow (p_i(z_1, \ldots, z_m, do(a, s)) \Leftrightarrow \phi_i(z_1, \ldots, z_{m+n}, s)))$$

where $k = l_1{+}1{+}\ldots{+}l_t{+}1$. The first successor state axiom after breaking into instantiated successor state axioms results in $l_1 + 1$ new axioms. We can easily verify that the size of the first l_1 axioms is 2 and the size of $l_1 + 1$th axiom is $O(Rl_1)$. Therefore, the total size of all instantiated successor state axioms is $2l_1 + Rl_1 + \ldots + 2l_t + Rl_t$ which is $O(Rk)$.

Since we have $|\psi \wedge precond_a|$ atomic formulas and for every atomic formula the size of the resulted formula after filtering is less than the size of all successor state axioms, the size of the returned formula would be $O(Rk|\psi \wedge precond_a|)$. In addition there is a constant formula (filtering of

TRUE) that we add to the resulted formula. Suppose that the size of the instantiated successor state axioms are $x_1, \ldots, x_k$. The size of filtering of TRUE would be less than $\Sigma_i \Sigma_j (x_i + x_j) = \Sigma_i (k x_i + \Sigma_j x_j) = O(Rk^2)$. The size of the formula is $O(Rk^2 + Rk|\psi \wedge precond_a|)$.

For computing the time complexity, we run Filter-True once which takes $O(k^2)$ and we run UCStep once for each single literal and it takes $O(k)$ each time. Therefore, the time complexity is the same as the space complexity $O(Rk^2 + Rk|\psi \wedge precond_a|)$. ■

A.12 Proof of Theorem 17

Correctness. To prove that the algorithm FOSF returns the filtering of ψ correctly we use the result of Theorem 15. We want to find all $\Phi(p_1, \ldots, p_n)$s such that $\psi \wedge precond_a \models \Phi(succ_{p_1,a}, \ldots, succ_{p_n,a})$.

FO-STRIPS-Step finds some Φs that $\psi \wedge precond_a \models \Phi(succ_{p_1,a}, \ldots, succ_{p_n,a})$ by using resolution (since all ϕ_is are either TRUE, FALSE, or the same literal). We refer to the set of Φs that this part returns as ξ.

We need to show if $\psi \wedge precond_a \models c'(succ_{p_1,a}, \ldots, succ_{p_n,a})$ where c' is a clause, then a clause in ξ subsumes $c'(succ_{p_1,a}, \ldots, succ_{p_n,a})$.

$$c'(succ_{p_1,a}, \ldots, succ_{p_n,a}) = t_1 \vee \ldots \vee t_m$$

t_i can not be TRUE or FALSE otherwise it would be subsumed by the last two terms of 3(h) in FO-STRIPS-Step. t_i can not be in Eff(a) and since the resolution is complete c' should be computed by 3(e).

Complexity. Suppose that the length of the belief state formula is $|\psi|$. The first two lines (recursive calls) of the algorithm take $O(|\psi|)$. Splitting into pure literal clauses takes $O(|E| + |\psi|)$ because the size of the resulted formula is $|E| + |S|$ and $|S| \leq |\psi| + |E|$ (A clause in the original belief state either is broken into multiple clauses which in that case the number of ones added to E is more than the number of ones

added to S or is not broken which is added directly to E or S and the number of such clauses is less than $|\psi|$).

After splitting into pure clauses, every predicate in Eff(a) is resolved out from E. For every predicate the size of E after resolution increases at most to $(|E|/2)^2$. The time complexity of the resolution depends on the size of the set. The algorithm takes $O(|E| + |\psi| + R(\frac{|E|}{2})^{2^{|\mathrm{Eff(a)}|}} + |S|) = O(R(\frac{|E|}{2})^{2^{|\mathrm{Eff(a)}|}} + |\psi|)$. ■

A.13 Proof of Theorem 18

There are no more than $2|E|$ literals in E. Therefore, no more than $O(|E|^2)$ different clauses of size two can be obtained after each step of resolution. Since the size of E can not be more than $O(|E|^2)$, the time complexity of the algorithm is $O(|\mathrm{Eff(a)}| \times |E|^2 + |\psi|)$.

The size of the formula is $O(|E|^2+|S|+|\mathrm{Eff(a)}|) = O(|\mathrm{Eff(a)}|+|E|^2+|\psi|)$. ■