First Order Logical Filtering

Afsaneh Shirazi and **Eyal Amir**
Computer Science Department, University of Illinois at U-C
Urbana, IL 61801, USA {hajiamin,eyal}@cs.uiuc.edu

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) formulae. 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. These results apply to most expressive modeling languages, such as partial databases and belief revision in FOL.

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 actions and observations. This computation is called *filtering* (also, state estimation, belief update, and database progression). They use this information to make decisions, answer questions, and explore.

Filtering is intractable in general for discrete domains [Eiter and Gottlob, 1992], and much research is dedicated to its approximation in stochastic domains (e.g., [Boyen and Koller, 1998]). 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) [Amir and Russell, 2003] with actions and observations has shown that *exact* filtering is tractable when belief states are represented as propositional formulae, and certain natural assumptions are met. Still, many domains have propositional encodings that are too large or are not possible (e.g., large numbers of objects, unknown number of objects, and observations with partial knowledge about identity of objects).

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, then we can update FOL belief-state formulae efficiently (linear time in the representation size), after prior compilation. We also show that the representation remains of bounded polynomial size for two classes of actions, including those for which actions have simple case preconditions and actions with STRIPS-like preconditions (non-conditional, 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 [Winslett, 1990; Lin and Reiter, 1997].

On the way to these contributions we form the foundations and provide a theory for FOL belief update. We relate deterministic Situation Calculus [Reiter, 2001] with a first-order transition model [Blass and Gurevich, 2000]. There, every belief state is a set of FOL models over the FOL language of a state. 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 [Lin and Reiter, 1997]). Also, we show that deduction can be carried out one time step at a time.

Most work related to ours is limited to the propositional case (e.g., [Amir and Russell, 2003; Boyen and Koller, 1998]). Important exceptions are [Cravo *et al.*, 2001; Dixon and Wobcke, 1993] (First-Order AGM belief revision), [Winslett, 1990] (belief update and revision in simple subclasses of FOL), and [Lin and Reiter, 1997] (progression in

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.

2 Semantics of First-Order Filtering

In this section, we study logical filtering with first order structures. A *first-order language* has as nonlogical symbols, the variables, the function symbols and the predicate symbols. A 0-ary function symbol is called *constant*. Note that among the binary predicate symbols must be the equality symbol $=$. We define the *terms* and formulas by the generalized inductive definition. Variables and functions are terms. Predicates are atomic formulas.

A *first-order language* is a language in which the symbols and formulas are as described above. 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.
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 $\models_S \psi$. For example, suppose that in structure S, $|S| = \{B, R\}$, for predicate in, $in^S = \{\langle B, R\rangle\}$, for constant *CS-R*, *CS-R*$^S = \{R\}$, and for constant *CS-B*, *CS-B*$^S = \{B\}$. This world has a CS room (*CS-R*), a CS book (*CS-B*), and a predicate in which indicates whether a book is in a room. By this definition, sentence $in($*CS-B*, *CS-R*$)$ is true in S.

We define logical filtering using situation calculus. The one that we use is compatible with the *basic action theory* of [Reiter, 2001]. 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. $\mathcal{A}(\overrightarrow{x}) \neq \mathcal{A}'(\overrightarrow{y})$, $\mathcal{A}(\overrightarrow{x}) = \mathcal{A}(\overrightarrow{y}) \Rightarrow \overrightarrow{x} = \overrightarrow{y}$ where $\mathcal{A}$ and $\mathcal{A}'$ are action symbols.
- $\mathcal{D}_{s_0}$ the initial database, which is a finite set of first order sentences that are *uniform* in s_0

Definition 2.1 (Uniform Formula). *A formula is* uniform *in s if s is the only term of sort situation mentioned by that formula.*

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

Definition 2.2 (Action Precondition Axioms). *An action precondition axiom is a sentence of the form:*[1]

$$Poss(a(x_{1:n}), s) \equiv precond_a(x_{1:n}, s)$$

where a is an n-ary action symbol, and $precond_a(x_{1:n}, s)$ is a formula that is uniform in s and whose free variables are among $x_1, \ldots, x_n, s$.

[1] $x_{1:n}$ is the abbreviation for $x_1, \ldots, x_n$

Generally, the values of relations and functions in a dynamic world will vary from one situation to the next. Relations 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. Same argument is true about functions.

Definition 2.3 (Successor State Axioms). Successor state axioms *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:*

$$Poss(a(x_{1:n}), s) \Rightarrow \forall y_1, \ldots, \forall y_m$$
$$(p(y_{1:m}, do(a(x_{1:n}), s)) \equiv succ_{p,a}(x_{1:n}, y_{1:m}, s))$$

where a is an action symbol, and $succ_{p,a}(x_{1:n}, y_{1:m}, s)$ 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 in a similar way.

All changes to the world are the result of named actions. An action may by 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. The intended interpretation is that situations are finite sequences of actions, and $do(a, s)$ denotes the sequence formed by adding action a to the sequences s. In other words, $do(a, s)$ is the successor situation resulting from performing the action a. We use 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.

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 between rooms, 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 $move(b, r_1, r_2)$, $return(b, r)$ or $borrow(b, r)$. The actions which are executable in a world state can change the value of different predicates or functions. Predicates are $room(r)$, $book(b)$ or $in(b, r)$. There is no functional fluent except constants.

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

- *Precondition Axiom:*
  $Poss(move(b, r_1, r_2), s) \equiv book(b, s) \wedge room(r_1, s) \wedge room(r_2, s) \wedge in(b, r_1, s)$
- *Successor State Axioms:*

$$\begin{aligned}
&Poss(move(b, r_1, r_2), s) \Rightarrow \\
&\quad \forall b', r' \; (in(b', r', do(move(b, r_1, r_2), s)) \equiv \\
&\qquad (((b = b') \wedge (r_1 = r')) \Rightarrow false \\
&\qquad \wedge((b = b') \wedge (r_2 = r')) \Rightarrow true \\
&\qquad \wedge(\neg((b = b') \wedge (r_1 = r')) \\
&\qquad\quad \wedge\neg((b = b') \wedge (r_2 = r'))) \Rightarrow in(b', r', s)))
\end{aligned}$$

The definition of progression and filtering semantics are as follows.

Definition 2.4 (Transition Relation of Structures). *For an action theory $\mathcal{D}$ and a structure S, we define a transition re-*

lation $\mathcal{R}_{\mathcal{D}}(S, a, S')$ as follows.

$$\mathcal{R}_{\mathcal{D}} = \{\langle S, a(u_{1:n}), S'\rangle \mid \ \models_S precond_a(u_{1:n}), |S'| = |S|, \\ p^{S'} = \{\langle s_{1:m}\rangle \in |S|^m| \ \models_S succ_{p,a}(u_{1:n}, s_{1:m})\}, \\ f^{S'} = \{\langle s_{1:m}, s_r\rangle \in |S|^{m+1}| \\ \models_S succ_{f,a}(u_{1:n}, s_{1:m}, s_r)\}\}$$

We use $[x/v]$ as a notion of substitution in which x is a vector of variables and v is a vector of variables and constants. $[x/v]$ is a shorthand for $[x_1/v_1, \ldots]$ in which $[x_1/v_1]$ means replacing all instances of symbol x_1 by symbol v_1.

Definition 2.5 (Logical Filtering Semantics). *Let σ be a set of first order structures. The* filtering *of a sequence of actions (ground or not) and observations $\langle a_1, o_1, \ldots, a_t, o_t\rangle$ is defined as follows.*

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

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

In the above definition, filtering is applied to a set of first-order structures. A *belief state formula* is a first-order formula that represents a set of belief state structures. A structure satisfies the belief state formula if and only if it is in that set.

3 Filtering of FOL Formulae

In this section we present a straightforward algorithm that filters belief state formulae directly, but does so in worst-case exponential time. From now on, we assume that our first order language has no function symbols except constants.

3.1 Basic Algorithm

In this section, we show how we can 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 progression algorithm can use afterwards.

We define a predicate corresponding to each relational fluent whose truth value does not depend on the situation. The snapshot of system at time t only shows the truth values of predicates. The truth values of predicates would change while moving from one situation to the next. We represent predicates with the same symbol as relational fluents but with different arity.

Suppose that $\mathcal{P} = \{g_1, \ldots, g_r\}$ is the set of all constants and predicates. We define a new set of symbols $\mathcal{P}' = \{g'_1, \ldots, g'_r\}$ such that $g'_i(y_{1:n}) = g_i(y_{1:n})_{[\mathcal{P}/\mathcal{P}']}$ 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 predicates in situation s, and $\mathcal{P}'$ as the set of 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.) Let ψ be a belief state formula, $a(\overrightarrow{u})$ be a grounded action, $Cn(\Psi)$ be the set of logical consequences of Ψ (i.e. formulae 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 $Cn^{\mathcal{L}(L)}(\Psi)$.

1. $Filter[a(u_{1:n})](\psi) = (Cn^{\mathcal{P}'}(\psi \wedge precond_a(u_{1:n}) \ \bigwedge_i \forall y_{1:m}, p'_i(y_{1:m}) \equiv succ_{p_i,a}(u_{1:n}, y_{1:m}) \ \bigwedge_i \forall y_{1:m} \forall z, f'_i(y_{1:m}) = z \equiv succ_{f_i,a}(u_{1:n}, y_{1:m}, z)))_{[\mathcal{P}'/\mathcal{P}]}$
2. $Filter[o](\psi) = \psi \wedge o$ (1)

We prove in the following theorem that this definition of filtering approximates the semantics of definition 2.5.

Theorem 3.1. *Let ψ be a belief state formula, and let a be an action, then*

$$Filter[a](\{s \mid \ \models_s \psi\}) \subseteq \{s' \mid \ \models_{s'} Filter[a](\psi)\}$$

[Lin and Reiter, 1997] 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 3.1 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 3.1 would be equal if the progression of a database is FOL definable.

Corollary 3.2. *Let ψ be a first order belief state formula. If FOL can represent the progression of ψ after performing action a, then*

$$Filter[a](\{s \mid \ \models_s \psi\}) = \{s' \mid \ \models_{s'} Filter[a](\psi)\}$$

From this point, we assume that progression is first order definable. Our basic algorithm computes $Filter[\langle a_1, o_1, ..., a_t, o_t\rangle](\psi)$ by iteratively applying filtering of a belief-state formula with an action and an observation. It sets $\psi_0 = \psi$ and $\psi_i = Filter[o_i](Filter[a_i](\psi_{i-1}))$ recursively for $i > 0$ using the equations defined above. This algorithm is correct, as shown by corollary 3.2. It can be implemented using a first-order consequence finder.

3.2 Sequences of Actions and Observations

This section shows that iterative applications of progression steps lose no information. Thus, we can throw away the previous database and start working with the new one after performing each action.

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

Definition 3.3. *The language of $\mathcal{D}$, $\mathcal{L}(\mathcal{D})$, is a set of first order formulae whose predicate and function symbols occur in $\mathcal{D}$.*

For instance, if $\mathcal{D}_{s_0} = put(A, B) \wedge \forall x \ \ box(x)$, then $put(A, A)$ and $\forall x \exists y \ put(x, y)$ are in $\mathcal{L}(\mathcal{D}_{s_0})$ but $box(A, B)$ is not.

In progression we assume that a ground action a is performed, and we are looking for a set of sentences $\mathcal{D}_{s_a}$ that can serve as a new initial database (s_a denotes the situation term $do(a, s)$). Unfortunately [Lin and Reiter, 1997] showed that $\mathcal{D}_{s_a}$ is not always first order definable.

We define $\mathcal{F}_{s_a}$ as the set of first-order sentences uniform in s_a entailed by $\mathcal{D}$. If we use $\mathcal{F}_{s_a}$ instead of $\mathcal{D}_{s_a}$, for every first order sentence ψ about the future of s_a, $\mathcal{F}_{s_a} \cup \mathcal{D}_g \models \psi$ iff $\mathcal{D} \models \psi$. The following theorem states this result.

Note that the intersection of all consequences of the action theory with $\mathcal{L}(\mathcal{D}_g \cup \{s_a\})$ is uniform in s_a.

Theorem 3.4. *Let $\mathcal{D}_0$ be an action theory, and define $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}'_2$ as follows.*

$$\begin{aligned}
\mathcal{D}_1 &= Cn(\mathcal{D}_0 \cup \{s_1 = do(a_1, s_0)\}) \cap \mathcal{L}(\mathcal{D}_g \cup \{s_1\}) \\
\mathcal{D}_2 &= Cn(\mathcal{D}_1 \cup \{s_2 = do(a_2, s_1)\}) \cap \mathcal{L}(\mathcal{D}_g \cup \{s_2\}) \\
\mathcal{D}'_2 &= Cn(\mathcal{D}_0 \cup \{s_1 = do(a_1, s_0), s_2 = do(a_2, s_1)\}) \\
&\quad \cap \mathcal{L}(\mathcal{D}_g \cup \{s_2\})
\end{aligned}$$

(a_1 and a_2 are two actions in $\mathcal{D}_0$, not necessarily different. s_0, s_1 and s_2 do not occur in $\mathcal{D}_g$.) Then , $\mathcal{D}_2 = \mathcal{D}'_2$.

For instance in our book keeper example, if $\mathcal{D}_{s_0}$ is $\{book(\mathrm{B}, s_0), room(\mathrm{R}, s_0), in(\mathrm{B}, \mathrm{R}, s_0)\}$ and the first action is $get(\mathrm{B}, \mathrm{R})$, $\mathcal{D}_{s_1}$ would be $\{book(\mathrm{B}, s_1), room(\mathrm{R}, s_1), \neg in(\mathrm{B}, \mathrm{R}, s_1)\}$.

4 Factored Inference

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 4.1. *Let a be an action, and let φ, ψ be first order formulae. Then,*

1. $\models Filter[a](\varphi \vee \psi) \equiv Filter[a](\varphi) \vee Filter[a](\psi)$
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. $\models Filter[a](\exists x\ \varphi(x)) \equiv \exists x\ Filter[a](\varphi(L))_{[L/x]}$

(L is a fresh constant symbol.)

We can say something stronger for actions that act as *permutations* on the structures in which they are executable.

Definition 4.2 (Permuting Actions). *Action a is* permuting *(1:1) if for every structure S' there is at most one S such that $\mathcal{R}_{\mathcal{D}}(S, a, S')$.*

Domains that only include permuting actions are called *permutation domains*.

Theorem 4.3 (Distribution for Permutation Domains). *Let a be a permuting action, and let φ, ψ be formulae. Then,*

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

We can decompose every first order formula into a set of single literals by using distribution properties proved above. For instance, $\forall x\ (\varphi(x) \wedge \psi(x))$ is equivalent to $\forall x\ \varphi(x) \wedge \forall x\ \psi(x)$ so rule 2 can break it into two parts. Also $\forall x\ \neg\varphi(x)$ is equivalent to $\neg\exists x\ \varphi(x)$ so rule 3 and rule 4 can be used, and $\forall x\ (\varphi(x) \vee \psi(x))$ is equivalent to $\neg\exists x\ (\neg\varphi(x) \wedge \neg\psi(x))$ so rule 3 and rule 2 can be used.

In permutation domains, we decompose the formula down to a set of grounded first order single literals, and for filtering a single literal we use formula (1).

Our factored filtering (FF) algorithm for permutation domains is presented in Figure 1. It relies on theorems 3.2, 4.1, and 4.3. The number of different grounded single literals would be finite, if the number of objects is finite. Therefore, we can calculate filtering of all single literals as a preprocessing step and retrieve it later in finite domains.

Note that the arguments of these literals are either the constants associated to existential quantifiers or the constants which are mentioned in the initial belief state, the set of axioms or the observations.

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

```
PROCEDURE FF-Step($a$,$\psi$)
$a$ an action. $\psi$ a belief-state formula.
  1. if $\psi$ is a single literal, then return Single-Literal-Filtering($a$, $\psi$).
  2. else, use distribution properties, call FF-Step recursively on sub-formulas of $\psi$.
```

Figure 1: Filtering of a FOL formula when all the actions are permuting actions.

Theorem 4.4. *The algorithm FF is correct, and if the filtering of all single literals are given, the algorithm FF would run in time $O(|precond_a \wedge \psi|)$, where ψ is a belief state formula.*

Our factored filtering algorithm uses consequence finding tools. Since it is part of preprocessing, it does not affect the runtime of the system. In open systems the time is different since new objects may be added during the operation of the system. In these systems filtering of new single literals should be computed while system is running.

5 Filtering Algorithms for Different Domains

Our naive filtering algorithm uses consequence finding tools which do not scale to large domains. The following theorem suggests a different reasoning procedure.

Theorem 5.1. *Let a be an action, ψ be a belief state formula, and $\Phi(x_{1:n})$ be a first-order logical formula whose atomic subformulas are among $x_1, \ldots, x_n$. Then,*

$$\models Filter[a](\psi) \equiv \bigwedge_{\psi \wedge precond_a \models \Phi(succ_{p_1,a}, \ldots, succ_{p_n,a})} \Phi(p_{1:n}) \qquad (2)$$

In this formula, all possible Φs should be considered which are infinitely many. In general, generating all Φs is impossible. 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

Every successor state axiom can be rewritten as:

$$Poss(a(x_{1:n}), s) \Rightarrow \forall y_1, \ldots, \forall y_m \ (p(y_{1:m}, do(a, s)) \equiv \\ (case_1 \Rightarrow \phi_1) \wedge \ldots \wedge (case_l \Rightarrow \phi_l) \\ \wedge(\neg case_1 \wedge \ldots \wedge \neg case_l) \Rightarrow \phi_{l+1})$$

where $case_i$ is of the form $(y_{i_1} = x_{i_1}) \wedge \ldots \wedge (y_{i_k} = x_{i_k})$ (variable x_{i_j} is an argument of action a and variable y_{i_j} is an argument of predicate p) and each variable assignment satisfies at most one of the cases. A successor state axiom is called *unit-case successor state axiom* when it can be rewritten in a form where every ϕ_i is a unit clause.

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

```
PROCEDURE Unit-Case-Step($a$,$\psi$)
$a$ an action. $\psi$ a belief-state formula. $poss(a) \Rightarrow (case_i \Rightarrow (p_i \equiv \phi_i))$ an instantiated successor state axiom.
  1. if $\psi$ is a literal, then
       $S = \emptyset$
       for all unit clauses $\phi_i$,
           if $\phi_i = true$, add $case_i \Rightarrow p_i$ to $S$
           elseif $\phi_i = false$, add $case_i \Rightarrow \neg p_i$ to $S$
           elseif unifiable($\phi_i, \psi$), add $(case_i \Rightarrow p_i)_{mgu(\phi_i,\psi)}$ to $S$
           elseif unifiable($\phi_i, \neg\psi$), add $(case_i \Rightarrow \neg p_i)_{mgu(\phi_i,\psi)}$
       for all $\phi_i, \phi_j$,
           if unifiable($\phi_i, \phi_j$), add
                   $((case_i \wedge case_j) \Rightarrow (p_i \equiv p_j))_{mgu(\phi_i,\phi_j)}$ to $S$
           elseif unifiable($\phi_i, \neg\phi_j$), add
                   $((case_i \wedge case_j) \Rightarrow (p_i \equiv \neg p_j))_{mgu(\phi_i,\phi_j)}$ to $S$
           elseif $\phi_i = \forall x \ q(x), \phi_j = q(t)$, add
                   $(case_i \wedge case_j) \Rightarrow (\neg p_i \vee p_j)$ to $S$
           elseif $\phi_i = \exists x \ q(x), \phi_j = q(t)$, add
                   $(case_i \wedge case_j) \Rightarrow (p_i \vee \neg p_j)$ to $S$

       return $\bigwedge_{\varphi \in S} \varphi$.
  2. else, use distribution properties, call Unit-Case-Step recursively on sub-formulae of $\psi$.
```

Figure 2: Unit-Case Filtering.

We break a unit-case successor state axiom into multiple instantiated axioms. An instantiated successor state axiom would be $Poss(a(x_{1:n}), s) \Rightarrow (p(y_{1:m}, do(a, s))_{[y_i/x_i]} \equiv \phi_{i[y_i/x_i]})$, if we represent $case_i$ with a substitution $[y_i/x_i]$ (y_i and x_i are sequences of variables). However, the last case in successor state axioms can not be rewritten using substitution, but we can leave it in the successor state axiom as $Poss(a(x_{1:n}), s) \Rightarrow \forall \overrightarrow{y} \ (\neg case_1 \wedge \ldots \wedge \neg case_l) \Rightarrow (p(y_{1:m}, do(a, s)) \equiv \phi_{l+1})$, and consider it just when we check whether two formulas are unifiable.

Using formula (2), every $\Phi(subsucc_a^1, \ldots, subsucc_a^k)$ should be considered. In permutation domains, the head of entailment in formula (2) is a single literal because the action precondition can be considered as a conjunct to the belief state formula, and the distribution properties can be used. As a result, $\Phi(subsucc_a^1, \ldots, subsucc_a^k)$ is either equivalent to that literal or a tautology. A tautology is at most of size two when unit-case successor state axioms are divided into multiple instantiated successor state axioms. Therefore, every desired Φ can be computed in finite steps. Figure 2 shows the unit-case filtering algorithm applicable on permutation domains whose successor state axioms are unit-case.

Theorem 5.2. *Let k be the number of successor state axioms after dividing into instantiated successor state axioms, and let l be the length of conjunction of belief state and precondition formulas. If each action has an arity less than some constant number, then the length of formula after filtering is $O(k^2 + lk)$. The time complexity of the algorithm is also $O(k^2 + lk)$.*

5.2 STRIPS Domains

In STRIPS domains every action has no conditional effects, it can not fail, and it is always executable. Consequently, when we filter with action a we assert implicitly that its precondition held in the last world state. The value of each fluent in the following situation is unconditionally determined. It means that the value of a predicate either changes to true, changes to false, or remains the same.

STRIPS domains are not a special case of unit-case domains although STRIPS successor state axioms are unit-case. The reason is that the actions are not necessarily permuting in STRIPS domains. Therefore, distribution properties do not hold on STRIPS domains.

Successor state axioms in STRIPS domains are:

$$Poss(a(x_{1:n}), s) \Rightarrow (p(y_{1:m}, do(a, s)) \equiv \\ (case_1 \Rightarrow \phi_1) \wedge \ldots \wedge (case_l \Rightarrow \phi_l) \\ \wedge(\neg case_1 \wedge \ldots \wedge \neg case_n) \Rightarrow (p(y_{1:m}, s))$$

where ϕ_i ($i \leq l$) is either true or false.

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

$$\text{Eff}(a) = \{p(\overrightarrow{v}) | \text{ action } a \text{ affects every instance of } p(\overrightarrow{v})\}$$

where v is a sequence of variable and constant symbols.

The first order STRIPS filtering algorithm is presented in figure 3. We define $\exists^*\forall^*\phi$ as a first order formula in which there is no existential quantifier inside a universal one.

In FO-STRIPS-Step function, we split every clause $p(x) \vee \varphi$ in which some instantiations of $p(x)$ are affected and some are not, into multiple clauses. The set of new clauses is:

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

where $v_1, \ldots, v_r$ are affected instantiations.

The FO-STRIPS-Step splits instantiated predicates into two sets, those remaining the same after action application and those affected by the action. The new values of affected predicates are added to new belief state formula as their values can be determined unconditionally after applying the action. The algorithm also adds all the consequences of the belief state formula in which no affected predicates exist, to the new belief state formula.

```
PROCEDURE FO-STRIPS-Filter($\langle a_i, o_i\rangle_{0<i\leq t}$,$\psi$)
$\forall i$, $a_i$ an action, $o_i$ an observation and $\psi$ a belief state formula. $o_i$
and $\psi$ in the form $\exists^*\forall^*\phi$, with $\phi$ in clausal form without functions.
  1. if $t = 0$, return $\psi$.
  2. return Move-Quan[a]($o_t\wedge$ FO-STRIPS-Step($a_t$, Move-Quan(
         $precond_{a_t}\wedge$ FO-STRIPS-Filter($\langle a_i, o_i\rangle_{0<i\leq(t-1)}$,$\psi$))))

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

```
PROCEDURE FO-STRIPS-Step($a$,$\psi$)
$a$ 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($a$,$\phi(L)$)$_{[L/x]}$ [a]
  2. elseif $\psi = \forall x\ \phi(x)$, return $\forall x$ FO-STRIPS-Step($a$,$\phi(x)$)
  3. else,
       split every clause into a set of pure predicate clauses
       put all clauses with any Eff($a$) predicate in $E$, others in $S$
       if $E \neq \emptyset$
         for all $l \in$ Eff($a$)
           $E \leftarrow$ resolve-out($l$, $E$)
       $\phi = \bigwedge_{c_i\in E\cup S} c_i$
       return $\phi \wedge \bigwedge_{p\in \text{Eff}(a), p\equiv true}\, p \wedge \bigwedge_{p\in \text{Eff}(a), p\equiv false} \neg p$.

[a]L is a fresh constant that does not appear in the language
```

Figure 3: First Order STRIPS Filtering.

Theorem 5.3. *Given action a, observation o, and belief state formula $\psi = \exists^*\forall^*\phi$ with ϕ in clausal form, the first-order STRIPS filtering (FOSF) algorithm in Figure 3 returns the filtering of ψ with a and o in time $O((\frac{|E|}{2})^{2^{\text{Eff}(a)}})$ where E is the set of all clauses in belief state formula after splitting into pure predicate clauses with any Eff(a) predicate.*

Theorem 5.4. *If ψ is in 2-FO-CNF*[2] *then the time complexity of FOSF is $O(|\text{Eff}(a)||E|^2 + |S|)$ where S is the set of all clauses with no Eff(a) predicate. The formula length after filtering is $O(|\text{Eff}(a)| + |E|^2 + |S|)$.*

Extended Example
Consider our previous book-keeping robot. Suppose that we have two rooms, a CS room and an ECE room, and two books, a CS book and an ECE book, and our belief state formula is $in(\textit{CS-B, CS-R}) \wedge in(\textit{ECE-B, ECE-R})$. ECE department needs the CS book for a while, so the book keeper moves it to ECE room. The action is $a = move(\textit{CS-B, CS-R, ECE-R})$. (This example has unit-case successor state axioms.)

First, we add precondition to belief state formula. The new belief state is $in(\textit{CS-B, CS-R}) \wedge in(\textit{ECE-B, ECE-R}) \wedge book(\textit{CS-B} \wedge room(\textit{CS-R} \wedge book(\textit{ECE-R}))$. We calculate the filtering of all the single literals of the belief state formula separately and compute the result by using distribution properties. What follows is the formula for one of these literals based on the algorithm presented before.

$$Filter[move(\textit{CS-B, CS-R, ECE-R})](in(\textit{ECE-B, ECE-R})) \equiv \\ in(\textit{CS-B, ECE-R}) \wedge \neg in(\textit{CS-B, CS-R}) \wedge in(\textit{ECE-B, ECE-R})$$

Now suppose that instead of applying an action we filter the belief state based on an observation. The robot enters the CS room and it observes that there is only one book in the room. The 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 $in(\textit{CS-B, CS-R}) \wedge in(\textit{ECE-B, ECE-R})$. The observation is $\forall x\, in(x, \textit{CS-R}) \Rightarrow x = \textit{TheBook}$. $Filter[o](\psi) \models$ *TheBook = CS-B*, so we can replace every instance of *TheBook* in the new belief state formula by *CS-B*.

[2]A first order formula is in k-FO-CNF if it is the conjunction of clauses of size k.

6 Conclusions

In this paper 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 formulae. We showed that this problem is solvable in polynomial time when actions map states 1:1, or the actions are STRIPS. We showed that 1:1 actions allow us to filter first-order belief state formulae in linear time (in the size of representation), if we can perform a precompilation step. When actions are STRIPS or Unit-Case, we can filter these belief state formulae efficiently without precompilation. In some cases, we showed that the belief state formulae is guaranteed to remain compactly represented. Those cases permit filtering of actions and observations indefinitely in polynomial time (in the number of predicates and objects). As a result, we can use our algorithm for many interesting applications, such as semantic web, autonomous agents, robot motion control, and partial knowledge games.

References

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

[Blass and Gurevich, 2000] 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.

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

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

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

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

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

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

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