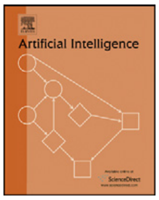
Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint



First-order logical filtering

Afsaneh Shirazi [*], Eyal Amir

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

ARTICLE INFO

Article history:
Available online 3 April 2010

Keywords:
Filtering
First-order logic
Belief update
Situation calculus

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 intractability results.

In this paper we present polynomial-time algorithms for filtering belief states that are encoded as First-Order Logic (FOL) formulas. Our algorithms are exact in many cases of interest. They accept belief states in FOL without functions, permitting arbitrary arity for predicates, infinite universes of elements, and equality. They enable natural representation with explicit references to unidentified objects and partially known relationships, still maintaining tractable computation. Previous results focus on more general cases that are intractable or permit only imprecise filtering. Our algorithms guarantee that belief-state representation remains compact for STRIPS actions (among others) with unbounded-size domains. This guarantees tractable exact filtering indefinitely for those domains. The rest of our results apply to 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 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 [17], and much research is dedicated to its approximation in stochastic domains (*e.g.*, [6,15,20]). 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) [3] 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 numbers of

[*] Corresponding author.
E-mail addresses: hajiamin@cs.uiuc.edu (A. Shirazi), eyal@cs.uiuc.edu (E. Amir).

0004-3702/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.artint.2010.04.015

Algorithm	Assumptions	Time (1 step)	Space
Deduction filter (Section 3.2)	none	unbounded (lazy computation)	unbounded
Factored filter (Section 4)	1:1 actions; Finite FOL filtering of atoms; precompilation stage	precompilation unbounded; $O(\|\varphi\| \cdot C)$	$O(\|\varphi\| \cdot C)$, C depends on precompilation
Unit-case filter (Section 5.1)	1:1 Unit-Cases actions; UNA[a]	$O(Rl(l+\|\varphi\|+p))$	$O(Rl(l+\|\varphi\|+p))$
STRIPS filter (Section 5.2)	STRIPS actions; known success; UNA; φ in clausal form[b]; cases fully instantiated	$O((l\|\psi\|)^{2e})$	$O((l\|\psi\|)^{2e})$;
STRIPS filter (Section 5.2)	STRIPS actions; known success; UNA; φ in 2-clausal[c] form	$O(m^2(R+n)^{2R})$	$O(m^2(R+n)^{2R})$ for any $t > 0$ time steps

Legend: φ: input (initial) belief state; m: number of predicate symbols; R: maximum arity of predicates; n: number of constant symbols (*not objects, which may be infinitely many*); l: total number of cases for action a's successor-state axioms; p: number of distinct atoms in the precondition of a; e: number of distinct affected literals of a.

[a] UNA: Unique-Names Assumption.

[b] Clausal form: Conjunction of disjunctions in $\exists^*\forall^*$ fragment of FOL.

[c] 2-clausal: Every clause has $\leqslant 2$ literals.

Fig. 1. The algorithms presented in this paper, their properties, and their assumptions for correct filtering. *All algorithms assume* no function symbols. *No algorithm assumes* a finite domain or requires a closed-world assumption (CWA). (The *CWA* is made in many planning domains. It assumes that the only objects in the domain are those mentioned explicitly.)

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 filtering 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.

More specifically (detailed results table is presented in Fig. 1), we present the following algorithms and complexity bounds on their performance: *First*, we show that when actions are partial functions that map states 1:1, we can filter arbitrary FOL belief-state formulas in time $O(|\psi| \cdot C)$, for ψ the input belief state representation, after a precompilation stage of the domain (not including ψ). The output size is $O(|\psi| \cdot C)$, for C a constant (possibly large) that depends on our actions' definitions (not on the input sequence of actions or the initial belief state).

Second, we examine in more detail two action representations that permit faster and more compact filtering without precompilation of the domain. Both of those classes of domains represent the effects of actions by cases.

For the first class of actions we show that filtering takes time $O(Rl \cdot (|\psi| + l + p))$ for R the arity of predicate fluents, l the total number of cases into which filtered actions break, and p the number of distinct atoms appearing in preconditions of the action and in cases. The belief state returned has representation size $O(Rl \cdot (|\psi| + l + p))$. To obtain these results we assume that actions are 1:1 and can be broken into cases conditioned on unit clauses (this is defined formally in Section 5.1), and that different constant symbols refer to different elements in our domain (this is the Unique Names Assumption (UNA)).

We present a different result for the second class of action representations, namely, STRIPS actions [18] in two different scenarios. In the first scenario we assume that every action case for a but one instantiates all variables in every affected predicate (thus making it a proposition). Also, we assume the UNA and that actions are executable when they are filtered. Given those, we show that filtering takes time $O((l|\psi|)^{2e})$ per action, for e the number of affected literals for action a. Also, we show that the resulting belief state formula is represented in space bounded by $O((l|\psi|)^{2e})$ (here, $|\psi|$ is the size of the input belief state).

Focusing on filtering sequences of length $T > 0$, and assuming input belief states that have only clauses of $\leqslant 2$ literals, we get a final, better result. Assuming UNA and STRIPS actions, *not assuming* full-instantiation of cases (cases may instantiate subsets of variables), we show that the same algorithm for STRIPS actions takes time $O(m^2(R+n)^{2R})$ per action, for m predicates of arity at most R and n constant symbols. The output belief-state space representation is bounded by $O(m^2(R+n)^{2R})$ regardless of the number of filtering steps. Thus, filtering a sequence of $t > 0$ actions and observations takes time $(tm^2(R+n)^{2R})$.

Notice that the domain may still be large (or infinite) when R, m, n are small, so this result enables tractable filtering for some very large domains. This result guarantees tractable filtering for arbitrary sequence lengths with such domains. It applies to standard STRIPS actions, among others.

These results support a growing belief that FOL can be used efficiently for representing and updating partial knowledge [40,66,53,41,65].

En route to these contributions we relate deterministic Situation Calculus [53] with a FOL transition model [4]. 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 FOL formulas. We re-state results of [40,64] showing that filtering such belief states can be captured exactly by *deduction in FOL* and that deduction can be carried out one time step at a time, if all we wish is to answer queries about a particular future or past state.

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 semantics that we use for FOL filtering. In Section 3 we provide a naive algorithm for filtering and prove its correctness. Section 4 offers a polynomial time algorithm for FOL filtering that requires domain precompilation and is exact for 1:1 actions. Correctness of this algorithm relies on distribution properties of filtering FOL KBs over logical connectives that we prove in the same section. Section 5 presents efficient algorithms for our two kinds of case-based actions.

Some of the results of this paper appeared previously in [56]. In particular, some of the basic algorithms of Sections 4 and 5 appear there. The complexity results, correctness, and examples are new to this manuscript, and so is the perspective relating this work to early and recent research on progression of FOL databases.

2. Semantics of first-order filtering

In this section, we study logical filtering with FOL 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 FOL 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 strings of English characters 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 *FOL 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 FOL languages. A *structure S* for a FOL 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 FOL 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 an n-ary function symbol, then

$$\sigma'\big(f(t_1, \ldots, t_n)\big) = f^S\big(\sigma'(t_1), \ldots, \sigma'(t_n)\big)$$

(2) *Atomic formulas*:
  (a) For term t_1 and t_2,

$$\models_S t_1 = t_2[\sigma] \quad \text{iff} \quad \sigma'(t_1) = \sigma'(t_2)$$

(b) For an n-ary predicate symbol p,

$$\models_S p(t_1, \dots, t_n)[\sigma] \quad \text{iff} \quad \langle \sigma'(t_1), \dots, \sigma'(t_n) \rangle \in p^S$$

(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.

Definition 1. *The language of a set of formulas $\mathcal{D}$, $\mathcal{L}(\mathcal{D})$,* is a set of first-order formulas whose predicate and function symbols appear in of $\mathcal{D}$.

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++: $\text{C++}^S = \{B\}$.

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 $in(\text{C++}, Office)$ is true in S.

We define logical filtering using situation calculus [44] in a way compatible with *basic action theories* of [53]. In the following section we discuss the basics of situation calculus needed for our work.

2.2. Situation calculus

In this section, we specify the language and axiomatization of situation calculus that we will use throughout the paper. The situation calculus is a language for specifying properties of actions. The formal 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 [53]):

- A constant symbol S_0, denoting the initial situation.
- A binary function symbol $do : action \times situation \to situation$. 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 [33]:

(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-names axiom for situations, stating that 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 [52]. This axiom is the only second-order axiom in our action theory. However, Reiter in [33] 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 without this axiom. This subset of the situation calculus is sufficient for filtering, as shown by Reiter.

The above axioms are domain independent. We refer to them as Σ. Combination of these axioms with a specification of axioms holding for the initial situation, successor state axioms, action precondition axioms, and unique-names axioms for actions is called a *basic theory of actions*.

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 FOL there is no intrinsic notion of time. Hence, situation calculus includes relational and functional fluents that 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 not typed. For simplicity, we also assume that all predicates and functions are fluents. Thus, predicates whose truth values do not change from situation to situation are handled with axioms that enforce these constraints.

Definition 2 *(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, *i.e.* A is a function symbol that has arguments of sort object and value of sort action.

Definition 3 *(Action precondition axioms).* An *action precondition axiom* is a sentence of the form:

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

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 4 *(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:

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

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.

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

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.
- $\mathcal{D}_{\text{una}}$ is a set of unique-names axioms for actions. $A(\vec{x}) \neq A'(\vec{y})$, $A(\vec{x}) = A(\vec{y}) \Rightarrow \vec{x} = \vec{y}$ where A and A' are action symbols.
- $\mathcal{D}_{S_0}$ (the initial database) is a finite set of FOL 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 focus more closely on belief states, and situation calculus is used in the proofs of theorems.

2.2.1. Example: Book-keeper robot

Consider a book-keeper robot which 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 omit the others.

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

$$\begin{aligned}&Poss\big(move(b, r_1, r_2), s\big) \quad\Rightarrow\quad \forall b' \forall r' \ \big(in\big(b', r', do\big(move(b, r_1, r_2), s\big)\big)\\ &\quad\Leftrightarrow \big(\big(\big(b' = b\big) \wedge \big(r' = r_1\big)\big) \Rightarrow FALSE \wedge \big(\big(b' = b\big) \wedge \big(r' = r_2\big)\big) \Rightarrow TRUE\\ &\quad\wedge \big(\neg\big(\big(b' = b\big) \wedge \big(r' = r_1\big)\big) \wedge \neg\big(\big(b' = b\big) \wedge \big(r' = r_2\big)\big)\big) \Rightarrow in\big(b', r', s\big)\big)\big)\end{aligned}$$

We write this successor-state axiom in this way to help usage in Section 5. Indeed, the successor-state axiom can be simplified logically.

2.3. Filtering semantics

For filtering we are interested in answering queries over values of fluents after actions and observations occur from time 0. Stated in the language of situation calculus, we are concerned with queries uniform in s, for some ground situation term s.

Thus, in our filtering language there is a predicate corresponding to each relational fluent of our situation calculus language. Those predicates do not take any arguments of sort situation. Roughly speaking, the truth value of a predicate changes from one situation to another. We represent predicates re-using the same symbols of relational fluents but with different arity (one less than the corresponding relational fluent).

Thus, in the following definition we drop all situation terms. The values of all predicates and functions are considered in the same situation. Transition from situation s to $do(a, s)$ (performing action a) changes the values of predicates and functions.

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

Let κ be a fixed infinite cardinality $\kappa \geqslant \aleph_0$ (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 FOL structures S over $\langle \mathcal{P}, \mathcal{F}, \mathcal{C} \rangle$ that have cardinality $\|S\|$ at most κ. Progression and filtering semantics are as follows.

Definition 5 *(Transition relation for FOL 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.

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

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

Note that action a is a parameterized action. The parameters are constants (*e.g.*, $move(\text{C++}, \mathit{Office}, \mathit{Lounge})$). 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 for FOL structures.

Definition 6 *(First-order logical filtering semantics).* Let Δ be a set of FOL 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) $Filter[\langle a_i, o_i, \ldots, a_t, o_t\rangle](\Delta) = Filter[\langle a_{i+1}, o_{i+1}, \ldots, a_t, o_t\rangle](Filter[o_i](Filter[a_i](\Delta)))$.

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

In the definition above, filtering is applied to a set of FOL structures. A *belief state formula* is a FOL 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 relational fluent $in(b,r,s)$ 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 first-order 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 are impractical because almost all belief states are of infinite sizes. Even when those are finite, an enumeration approach does not scale to large domains. An alternative approach is to perform logical progression in a form similar to the one described by [40,46]. In our work 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 Sections 4 and 5, 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 (with a compact representation) indefinitely.

From here forth we assume that our first-order language has no functional symbols except constants. This assumption help present our results more concisely, and is does not restrict the applicability of our results.

3.1. Filtering as consequence finding

In this section we show how to 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 $\mathcal{P}' = \{g'_1, \ldots, g'_r\}$. 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 $do(a,s)$. Thus, $g'_i(y_1, \ldots, y_n) = g_i(y_1, \ldots, y_n)_{[\mathcal{P}/\mathcal{P}']}$ where $[\mathcal{P}/\mathcal{P}']$ is a shorthand for $[g_1/g'_1, \ldots, g_r/g'_r]$.

We filter a belief-state formula as follows. (We reuse the notation $Filter[\cdot](\cdot)$ 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 FOL consequences of Ψ (*i.e.*, formulas ϕ in FOL 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 $Cn^{\mathcal{L}(L)}(\Psi)$.

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

When we filter with action a we assert that its precondition held in the last world state. If the action is not executable on the belief state, the new belief state formula 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 define filtering of a set of formulas enabling a recursive use of the previous equations. For a set of formulas Γ,

$$\begin{aligned}
1.\quad & Filter[a](\Gamma) = \Bigg(Cn^{\mathcal{P}'}\Bigg(\Gamma \cup \{precond_a\} \cup \\
& \bigcup_i \big\{\forall y_1, \ldots, y_m, p'_i(y_1, \ldots, y_m) \Leftrightarrow succ_{p_i,a}(y_1, \ldots, y_m)\big\} \cup
\end{aligned}$$

$$\bigcup_i \{\forall z, f_i' = z \Leftrightarrow succ_{f_i,a}(z)\}\Big)\Big)_{[\mathcal{P}'/\mathcal{P}]}$$

2. $$Filter[o](\Gamma) = \Gamma \cup \{o\} \tag{2}$$

We prove in the following theorem that this definition of filtering approximates the semantics of Definition 6.

Theorem 7. *Let ψ be a belief state formula, and let a be a ground action, then*

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

Proof. See Appendix A.1. □

[40] showed that progression is not always first-order definable. They showed that progression always exists as a set of second-order sentences for finite initial databases. Therefore, the two sides of Theorem 7 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 7 would be equal if the progression of a database is FOL definable.

Corollary 8. *Let ψ be a first-order belief state formula. If progression of ψ is first-order definable then*

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

Proof. See Appendix A.2. □

3.2. Deduction-based algorithm for filtering

Our baseline algorithm computes $Filter[\langle a_1, o_1, \ldots, a_t, o_t\rangle](\psi)$ by iteratively applying filtering of a belief-state formula with an action and an observation. It is not too practical, and is presented here for reference and contrast with later methods. Since every step may generate an infinite set of sentences, we reformulate this intuition into the following algorithm.

(1) Set $\psi_0 = \psi$, and $\psi_i = o_i$ for all $0 < i \leqslant t$.
(2) Concurrently for all $0 < i \leqslant t$ do
  (a) Apply a first-order consequence finder (*e.g.*, resolution [32]) to generate sentences in $Filter[a_i](\psi_{i-1})$ using formula (1).
  (b) When a sentence is generated, add it to ψ_i.

This algorithm is correct for filtering, as shown below. We discuss methods for consequence finding that can be used here further in Section 4.2.2. In Sections 4, 5 we provide algorithms for generating the formula that is the result of filtering.

3.3. Sequences of actions and observations

This section shows that iterative application of formula-filtering steps loses no information with respect to answering queries about the outcome of a sequence of actions and observations. It shows that we can discard the previous database after filtering every action and start using the database that we obtain after performing the progression step for that action. To show so we use the first-order situation calculus that we described above. It follows the treatment of situation calculus by Reiter [53].

The development in this section can be seen to follow immediately from the following more general results by [40,64].

Theorem 9. *(See [40, Prop. 4.10].) Let $\phi(s)$ be a formula uniform in s, and let $\vec{a} = \langle a_1, \ldots, a_t\rangle$ be a sequence of ground action terms ($do(\vec{a}, s)$ is a shorthand for $do(a_t, do(a_{t-1}, do(\ldots, do(a_1, s), \ldots))))$. Then, for $\mathcal{D}_{S_\alpha}$ the second-order filtering to S_α and $\mathcal{F}_{S_\alpha}$ the first-order filtering to S_α,*

$$(\mathcal{D} - \mathcal{D}_{S_0}) \cup \mathcal{D}_{S_\alpha} \models S_\alpha \sqsubset do(\vec{a}, S_\alpha) \wedge \phi\big(do(\vec{a}, S_\alpha)\big)$$

iff

$$(\mathcal{D} - \mathcal{D}_{S_0}) \cup \mathcal{F}_{S_\alpha} \models S_\alpha \sqsubset do(\vec{a}, S_\alpha) \wedge \phi\big(do(\vec{a}, S_\alpha)\big)$$

We bring a more complete development below to help explain the foundations of our algorithm in Section 3.2 and our results that follow in Sections 4 and 5.

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.

We use $Cn_{s_t}(\Psi)$ as the set of logical consequences of Ψ uniform in $s_t = do(a_t, do(a_{t-1}, \ldots, do(a_1, S_0)))$.

Suppose that we want to progress our initial database $\mathcal{D}_{S_0}$ with action a_1. Recall that $\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{\text{una}} \cup \mathcal{D}_{S_0}$. To progress $\mathcal{D}$ with action a_1, not all the axioms are required [33]. 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

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

would be replaced by

$$Poss\big(move(\text{C++}, Lounge, Office), S_0\big) \quad \Rightarrow \quad \forall y_1, \ldots, \forall y_m$$
$$\big(p\big(y_1, \ldots, y_m, do(a_1, S_0)\big) \Leftrightarrow succ_{p,move}(\text{C++}, Lounge, Office, y_1, \ldots, y_m, S_0)\big)$$

All other axioms are not needed for progression with a_1. For a ground action term a_1 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.

Define $\mathcal{D}_{s_1}$ as the *first-order progression* of $\mathcal{D}_{S_0}$ with a, the set of sentences uniform in $do(a_1, S_0)$ that are entailed by $\mathcal{D}$. If we show that $\mathcal{D}_g \cup \mathcal{D}_{s_1} \models \psi$ iff $\mathcal{D} \models \psi$ for every first-order sentence ψ uniform in a situation s' such that $do(a_1, S_0) \sqsubseteq s'$, then $\mathcal{D}_{s_1}$ can be used as a new initial database for further filtering. The following theorem states this result for two-step progression.

Theorem 10. *(Similar to [40, Prop. 4.10].) 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 $t \geqslant 1$ 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. See Appendix A.4. □

Theorem 10 is a different derivation of Lin and Reiters' result that applies to filtering. [40] Proposition 4.10 showed that first-order progression is enough for consequences about any specific future ground situation term. Filtering is interested in answering queries about given ground situation terms, so this theorem implies that first-order progression answers filtering queries correctly.

For example, in our book-keeper example if

$$\mathcal{D}_{S_0} = \big\{book(B, S_0), room(R, S_0), in(B, R, S_0)\big\}$$

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

$$\big\{book\big(B, do(a_1, S_0)\big), room\big(R, do(a_1, S_0)\big), \neg in\big(B, R, do(a_1, S_0)\big)\big\}$$

4. Factored filtering

In this section we present one of two main contributions of this paper, namely, a polynomial-time algorithm that computes logical filtering exactly for a significant class of transition systems. For the systems that do not fall within this class our algorithm gives an approximation to the filtering.

Our algorithm in the next section decompose filtering of FOL formulas to the filtering atomic subformulas. Recall that a FOL formula is *atomic* (an *atom*), if it is a predicate (or =) applied to well-founded terms, *i.e.* it includes no quantifiers or connectives. In what follows we prove several decomposition properties of filtering of FOL formulas. Our algorithm uses those properties and filters subformulas of a formula later combining 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 a be an action, and φ and ψ be FOL formulas. Assume that filtering of a is definable in FOL. 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) $Filter[a](\exists x \varphi(x)) \equiv \exists x Filter[a](\varphi(X))_{[X/x]}$.

(*X is a new constant symbol.*)

Proof. See Appendix A.6. □

Thus, filtering $\varphi \vee \psi$ with a 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 logically 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 $\varphi(X)$ with X being a new constant symbol and then replacing every X in the resulted formula with variable x and having an existential quantifier over x.

While 2 and 3 are only approximations (according to Theorem 11), they can be used as an exact answer in some domains. Our following Theorem 13 gives a stronger statement for actions that act as *1:1 partial functions* on the structures in which they are executable.

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

Recall the example from Section 2.2.1. There, action $a = move(\text{C++}, \mathit{Office}, \mathit{Lounge})$ is a one-to-one mapping between FOL structures. Assume that we define another action $moveto(\text{C++}, \mathit{Lounge})$ which moves book C++ to room *Lounge* regardless of where the book is before moving. This new action is not 1:1 because it maps different FOL structures to the same one: *e.g.* it maps structures whose book C++ locations are different and otherwise are identical to the same structure.

In many domains an action which is not one-to-one such as *moveto* can be replaced easily with a 1:1 action such as *move* with no loss in the expressivity of the domain (*e.g. turning on the light* can be replaced by *flipping the light switch*).

Domains that only include 1:1 actions are called *1:1 domains*. The next theorem presents distribution-over-connectives properties for 1:1 domains.

Theorem 13 *(Distribution properties for 1:1 domains). Let a be a* 1:1 *action, and φ and ψ be formulas. Assume that filtering of a is definable in FOL. 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 1:1 domains. These distribution properties are enough for decomposing the filtering of any FOL formula into the filtering of its atomic subformulas. We can decompose the filtering of any FOL 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)

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

```
PROCEDURE FF($\langle a_i, o_i \rangle_{0<i\leqslant t}$, $\psi$)
$\forall i$, $a_i$ 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($\langle a_i, o_i \rangle_{0<i\leqslant(t-1)}$, $\psi$)).

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

Fig. 2. Factored filtering of a FOL formula in a 1:1 domain.

- Universal quantifier over conjunction: (by rule 2)

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

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

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

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

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

4.2. Factored filtering

Our Factored Filtering (FF) algorithm for 1:1 domains is presented in Fig. 2. It relies on Theorems 8, 11, and 13. 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. (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.) We discuss this preprocessing step below.

Theorem 14. *Let $\mathcal{D}$ be a basic action theory, and ψ an initial belief state. Algorithm FF (Fig. 2) returns a formula that is logically equivalent to $Filter[a](\psi)$, if a is 1:1 in $\mathcal{D}$ and precompilation of $\mathcal{D}$ with respect to a has a finite representation. It runs in time $O(|precond_a \wedge \psi| \times F)$ when the filtering of all the atomic formulas are given and have finite representation, and F is the time needed to retrieve the filtering of an atomic formula.*

Proof. See Appendix A.9. □

4.2.1. Example computation with FF

First, consider a push-button action that permutes the states of two locks, d, e as follows:

$$d, \neg e \quad \Rightarrow \quad d, e \quad \Rightarrow \quad \neg d, e \quad \Rightarrow \quad \neg d, \neg e \quad \Rightarrow \quad d, \neg e$$

Successor-state axioms for this domain are

$$\begin{aligned} &d\big(do(a,s)\big) \equiv a = push \wedge \neg e(s) \vee a \neq push \wedge d(s) \\ &e\big(do(a,s)\big) \equiv a = push \wedge d(s) \vee a \neq push \wedge e(s) \end{aligned} \tag{3}$$

Precompilation of progression of the four literals with $A = push$ yields

$$\begin{aligned} &Filter[A](d) \equiv e \\ &Filter[A](\neg d) \equiv \neg e \\ &Filter[A](e) \equiv \neg d \\ &Filter[A](\neg e) \equiv d \end{aligned} \tag{4}$$

A second example is when we assume that our action has a precondition $\forall w\ up(w)$, *i.e.*, the action changes the world when $\forall w\ up(w)$ and the world stays the same otherwise. We call the new action $A' = push2$. Successor-state axioms for this domain are

$$\begin{aligned} d\big(do(a,s)\big) &\equiv a = push2 \wedge \big(\big(\forall w\ up(w)\big) \wedge \neg e(s)\big) \vee \\ &\quad d(s) \wedge \big(a \neq push2 \vee \big(a = push2 \wedge \neg\big(\forall w\ up(w)\big)\big)\big) \\ e\big(do(a,s)\big) &\equiv a = push2 \wedge \big(\big(\forall w\ up(w)\big) \wedge d(s)\big) \vee \\ &\quad e(s) \wedge \big(a \neq push2 \vee \big(a = push2 \wedge \neg\big(\forall w\ up(w)\big)\big)\big) \end{aligned} \tag{5}$$

(Successor-state axioms are of the form $F(\vec{x}, do(a,s)) \equiv \gamma_F^+(\vec{x}, a) \vee (F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a))$, and $\gamma_F^+(\vec{x}, a), \gamma_F^-(\vec{x}, a)$ are disjunctions of formulas of the form $\exists\vec{z}[a = A(\vec{y}) \wedge \phi(\vec{y})]$ with ϕ quantifier free uniform in s and $\vec{y}$ the exclusive union of $\vec{x}, \vec{z}$.) Precompilation of progression of the three atomic formulas with $A' = push2$ yields

$$\begin{aligned} &Filter\big[A'\big](e) \equiv \big(\big(\forall w\ up(w)\big) \Rightarrow \neg d\big) \wedge \big(\neg\big(\forall w\ up(w)\big) \Rightarrow e\big) \\ &Filter\big[A'\big](d) \equiv \big(\big(\forall w\ up(w)\big) \Rightarrow e\big) \wedge \big(\neg\big(\forall w\ up(w)\big) \Rightarrow d\big) \\ &Filter\big[A'\big]\big(up(w)\big) \equiv up(w) \\ &Filter\big[A'\big](TRUE) \equiv TRUE \end{aligned} \tag{6}$$

($Filter[A'](\neg e), Filter[A'](\neg d), Filter[A'](\neg up(w))$ are computed by noticing that A' is 1:1, so $Filter[A'](\neg\varphi) \equiv Filter[A'](TRUE) \wedge \neg Fitler[A'](\varphi)$.) This precompilation is then used as the results returned by subroutine Single-Literal-Filtering.

4.2.2. Precompilation

Algorithm FF uses a subroutine for filtering atoms (recall, atoms are predicate symbols applied to well-founded terms, *i.e.* including no quantifiers or logical connectives). Such filtering of atomic formulas can be done at a compilation (preprocessing) stage for a domain, if $Filter[a](P(x_1,\ldots,x_l))$ is finitely axiomatizable (in FOL), for every action a and predicate P.

Corollary 15. *Let a be an action term, $P(t_1,\ldots,t_l)$ an atom, and $t_1,\ldots,t_l$ constant symbols. Let $X_1,\ldots,X_l$ be new constant symbols. If $Filter[a](P(X_1,\ldots,X_l))$ has a finite axiomatization in FOL and φ is such an axiomatization, then $Filter[a](P(t_1,\ldots,t_l)) \equiv \varphi_{[X_1/t_1,\ldots,X_l/t_l]}$.*

Proof. Follows the same proof as equivalence (4) in Theorem 11. □

Thus, if the domain description language includes no function symbols and the language is finite (we assume both), then the compilation needs to occur only once per predicate-and-action pair.

Notice that these assumptions are not related to the size of the domain, which may be infinite. Our assumptions permit FOL theories for time step 0 that have only infinite models. For example, consider the example in Section 4.2.1 above. An example belief state for time step 0 is

$$\begin{aligned} &\forall w \exists x\ after(w,x) \wedge \big(up(w) \Rightarrow up(x)\big) \\ &\forall w, x\ after(w,x) \quad \Rightarrow \quad x \neq w \\ &\forall x, y, z\ after(x,y) \wedge after(y,z) \quad \Rightarrow \quad after(x,z) \end{aligned}$$

Here, the predicate *after*(.,.) is a strict order relation and every element has at least one subsequent other element. This forces the universe of every model to be of infinite size. The filtering of action *push*2 holds without change.

Thus, filtering of atoms can be done at a compilation (preprocessing) stage. Such a compilation would be done once for the domain, and would then be used for all filtering needs within that domain.

For n actions and m predicates, the precompilation would be done for $n \cdot m$ pairs. Every such compilation of an atom, $P(\vec{X})$, and action, $a(\vec{Y})$, can be done by applying FOL consequence finding (*e.g.* [32,57,58,42,2]) of sentences uniform in $do(\vec{y}, s)$ from $P(\vec{x}, s) \wedge \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{\text{una}}$.

If our action theory $\mathcal{D}$ also conforms to other restrictions (*e.g.* strictly local actions in [65]), the precompilation can be done directly (in [65,64] the size of the resulting formula per compilation step is $\Omega(2^{p \cdot c^k})$, even when polynomial-size representation is possible, so the result of precompilation may undermine the efficiency and output-compactness of FF).

Thus, any of the above methods can be used to implement Single-Literal-Filtering of Fig. 2 and precompute it in a compilation stage into a table *SLF* that has size $O(n \cdot m)$, storing every compilation of literal and ground action term. Then, the computation with FF of progression of ψ with action $a(\vec{y})$ uses Single-Literal-Filtering as a table lookup, and takes both time and output size in $O(|SLF| \cdot |\psi|)$.

5. Filtering by cases

Our naive filtering algorithm from Section 3 uses consequence finding tools which do not scale to large domains. Theorem 17 below 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 16 *(Derived-formulas).* Let $p_1, \dots, p_n$ be n closed predicates. $DF(p_1, \dots, p_n)$ is the set of derived-formulas of $p_1, \dots, p_n$ which is defined inductively as follows:

(1) $\{p_1, \dots, p_n\} \subset DF(p_1, \dots, p_n)$,
(2) $\neg\phi \in DF(p_1, \dots, p_n)$ if $\phi \in DF(p_1, \dots, p_n)$,
(3) $\phi \wedge \psi \in DF(p_1, \dots, p_n)$ if $\{\phi, \psi\} \subset DF(p_1, \dots, p_n)$,
(4) $\forall x\ \phi(y_1, \dots, y_m)_{[y_1/x_1, \dots, y_m/x_m]}$ if $\phi(y_1, \dots, y_m) \in DF(p_1, \dots, p_n)$.

For example,

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

because $book(x_1) \wedge room(x_2) \wedge in(x_3, x_4)$ 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 17. *Let a be a ground action, ψ be a belief state formula, and $p_1, \dots, p_n$ be n closed predicates in our action theory. Then,*

$$Filter[a](\psi) \equiv \big\{\Phi(p_1, \dots, p_n) \in DF(p_1, \dots, p_n) \bigm| \psi \wedge precond_a \models \Phi(p_1, \dots, p_n)_{[p_1/succ_{p_1,a}, \dots, p_n/succ_{p_n,a}]}\big\} \tag{7}$$

Proof. See Appendix A.10. □

In this formula, every Φ in $DF(p_1, \dots, p_n)$ that is entailed by our belief state and action precondition is in the filtering. Generally, generating all Φs is impossible in practice because there are infinitely many such Φs. In the following sections, we provide simpler closed-form solutions for two special cases of dynamic domains.[1] 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 $a = A(x_1, \dots, x_n)$ and predicate p can be rewritten in the following form:

$$\begin{aligned}
&Poss\big(A(x_1, \dots, x_n), s\big) \quad \Rightarrow \quad \forall y_1, \dots, \forall y_m \ \big(p\big(y_1, \dots, y_m, do(a, s)\big) \\
&\quad \Leftrightarrow \big(case_p^1 \Rightarrow \phi_p^1(x_1, \dots, x_n, y_1, \dots, y_m, s)\big) \\
&\quad \wedge \dots \wedge \big(case_p^{l_p} \Rightarrow \phi_p^{l_p}(x_1, \dots, x_n, y_1, \dots, y_m, s)\big) \\
&\quad \wedge \big(\neg case_p^1 \wedge \dots \wedge \neg case_p^{l_p}\big) \Rightarrow \phi_p^{l_p+1}(x_1, \dots, x_n, y_1, \dots, y_m, s)\big)
\end{aligned}$$

where $case_p^j$ is of the form $(y_p^{j_1} = x_p^{j_1}) \wedge \dots \wedge (y_p^{j_k} = x_p^{j_k})$ (variable $x_p^{j_1}$ is an argument of action a and variable $y_p^{j_1}$ is an argument of predicate p) and *each variable assignment satisfies at most one of the cases*. Note that the set of variables in each case can be a subset of variables in p and a.

Notice that the algorithms presented in this section and the following require *Unique Names Axioms* for objects. This is because we would like to determine which case a ground predicate corresponds to without checking if the constants are equal, *e.g.*, $p(B)$ is not from the case $p(y, do(a, s)) \Leftrightarrow (y = A) \Rightarrow TRUE$ (A and B are constants). Also we would like to be able to say if two instantiated predicates are unifiable without adding equality preconditions, *e.g.*, $p(A, y)$ is not unifiable with $p(B, C)$ with unique name assumption. However, without this assumption any two predicates with the same predicate symbols are unifiable (if $(B = A)$ then $p(B, C)$ and $p(A, y)$ are unifiable).

A successor state axiom is called *unit-case successor state axiom* if it can be rewritten in a form where every $\phi_p^j(x_1, \dots, x_n, y_1, \dots, y_m, s)$ $(1 \leqslant j \leqslant l_p + 1)$ is a unit clause. Our book-carrying robot example from Section 2.2.1 is written in this form. The last term in the precondition assures that the cases are mutually exclusive.

We divide a unit-case successor state axiom into multiple instantiated axioms.

[1] Those domains were shown independently to have finite strong progression in FOL [65].

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

PROCEDURE Filter-True($A(x_1,\dots,x_n)$) [Done once and cashed]
$A(x_1,\dots,x_n)$ an action.
 (1) $S = \emptyset$
 (2) for every predicate symbol $p$, for every case of $p$,
 (3)     [$Poss(A(x_1,\dots,x_n),s) \Rightarrow (cond_p \Rightarrow (p(z_1,\dots,z_m,do(a,s)) \Leftrightarrow$
 (4)       $\phi_p(z_1,\dots,z_{m+n},s)))$ is a successor state axiom of a case[a]]
 (5)       if $\phi_p(z_1,\dots,z_{m+n}) = true$, $S = S \cup \{cond_p \Rightarrow p(z_1,\dots,z_m)\}$
 (6)       elseif $\phi_p(z_1,\dots,z_{m+n}) = false$, $S = S \cup \{cond_p \Rightarrow \neg p(z_1,\dots,z_m)\}$
 (7) for every predicate symbol pair $p, p'$, for every case of each,
 (8)        if unifiable($\phi_p(z_1,\dots,z_{m+n}), \phi_{p'}(z_1,\dots,z_{m+n})$),
 (9)             $S = S \cup \{((cond_p \wedge cond_{p'}) \Rightarrow (p(z_1,\dots,z_m) \Leftrightarrow$
(10)                        $p'(z_1,\dots,z_m)))_{mgu(\phi_p,\phi_{p'})}\}$
(11)       elseif unifiable($\phi_p(z_1,\dots,z_{m+n}), \neg\phi_{p'}(z_1,\dots,z_{m+n})$),
(12)             $S = S \cup \{((cond_p \wedge cond_{p'}) \Rightarrow (p(z_1,\dots,z_m) \Leftrightarrow$
(13)                        $\neg p'(z_1,\dots,z_m)))_{mgu(\phi_p,\phi_{p'})}\}$
(14)       elseif $\phi_p(z_1,\dots,z_{m+n}) = \forall x\, q(x)$, $\phi_{p'}(z_1,\dots,z_{m+n}) = q(T)$,
(15)             $S = S \cup \{(cond_p \wedge cond_{p'}) \Rightarrow (\neg p(z_1,\dots,z_m) \vee p'(z_1,\dots,z_m))\}$
(16)       elseif $\phi_p(z_1,\dots,z_{m+n}) = \forall x\, q(x)$, $\phi_{p'}(z_1,\dots,z_{m+n}) = \neg q(T)$,
(17)             $S = S \cup \{(cond_p \wedge cond_{p'}) \Rightarrow (\neg p(z_1,\dots,z_m) \vee \neg p'(z_1,\dots,z_m))\}$
(18)       elseif $\phi_p(z_1,\dots,z_{m+n}) = \exists x\, q(x)$, $\phi_{p'}(z_1,\dots,z_{m+n}) = q(T)$,
(19)             $S = S \cup \{(cond_p \wedge cond_{p'}) \Rightarrow (p(z_1,\dots,z_m) \vee \neg p'(z_1,\dots,z_m))\}$
(20)       elseif $\phi_p(z_1,\dots,z_{m+n}) = \exists x\, q(x)$, $\phi_{p'}(z_1,\dots,z_{m+n}) = \neg q(T)$,
(21)             $S = S \cup \{(cond_p \wedge cond_{p'}) \Rightarrow (p(z_1,\dots,z_m) \vee p'(z_1,\dots,z_m))\}$
(22) return $\bigwedge_{\varphi\in S}\varphi$.
```

[a] $cond_p$ is either $TRUE$ or $(\neg case_p^1 \wedge \cdots \wedge \neg case_p^{l_p})$ using Definition 18.

Fig. 3. Unit-case filtering.

Definition 18. Instantiated successor state axioms for predicate p are:

- $Poss(A(x_1,\dots,x_n),s) \Rightarrow (p(y_1,\dots,y_m,do(a,s)) \Leftrightarrow \phi_p^j(x_1,\dots,x_n,y_1,\dots,y_m,s))_{[y_p^j/x_p^j]}$ for all $1 \leqslant j \leqslant l_p$,
- $Poss(A(x_1,\dots,x_n),s) \Rightarrow \forall y_1,\dots,y_m(\neg case_p^1 \wedge\cdots\wedge \neg case_p^{l_p}) \Rightarrow (p(y_1,\dots,y_m,do(a,s)) \Leftrightarrow \phi_p^{l_p+1}(x_1,\dots,x_n,y_1,\dots,y_m,s))$.

$[y_p^j/x_p^j]$ is the substitution corresponding to $case_p^j$ (y_p^j and x_p^j are sequences of variables). This process is called *breaking into cases*. Note that all instantiated successor state axioms are in the form

$$Poss\big(A(x_1,\dots,x_n),s\big) \quad\Rightarrow\quad \big(cond_p \Rightarrow \big(p\big(z_1,\dots,z_m,do(a,s)\big) \Leftrightarrow \phi_p(z_1,\dots,z_{m+n},s)\big)\big)$$

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

Figs. 3 and 4 show the Unit-Case Filtering (UCF) algorithm. This algorithm is applicable to 1:1 domains whose successor state axioms are unit-case. Algorithm UCF is actually a way to compute every $\Phi(p_1,\dots,p_n)$ in formula (7). Here, $p_1,\dots,p_n$ are the predicates of the instantiated successor state axioms after breaking into cases. Therefore, some of the arguments of these predicates are constant symbols that appear in our ground action.

In 1:1 domains the head of the entailment of formula (7) is an atomic formula since we use distribution properties of Theorem 13 on $\psi \wedge precond_a$. The distribution properties (discussed above) break the conjunction of belief state and precondition into atomic subformulas. Consequently, $\Phi(p_1,\dots,p_n)_{[p_1/succ_{p_1,a},\dots,p_n/succ_{p_n,a}]}$ 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.

Theorem 19. *Let progression be first-order definable, l 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 $O(Rl(l + |\psi| + |precond_a|))$ where R is the maximum arity of all predicates. The length of the returned formula is $O(Rl(l + |\psi| + |precond_a|))$.*

Proof. See Appendix A.11. □

```
PROCEDURE UCStep($A(x_1,\dots,x_n)$, $\psi$)
$A(x_1,\dots,x_n)$ an action, $\psi$ a belief-state formula.
(1) if $\psi$ is an atomic formula then
(2)       $S = \emptyset$
(3)       for every predicate symbol $p$ and every case of $p$,
(4)             [$Poss(A(x_1,\dots,x_n), s) \Rightarrow (cond_p \Rightarrow (p(z_1,\dots,z_m, do(a,s)) \Leftrightarrow$
(5)                $\phi_p(z_1,\dots,z_{m+n}, s)))$ is a successor state axiom of a case[a]]
(6)                if unifiable($\phi_p(z_1,\dots,z_{m+n})$, $\psi$)
(7)                      $S = S \cup \{(cond_p \Rightarrow p(z_1,\dots,z_m))_{mgu(\phi_p,\psi)}\}$
(8)                elseif unifiable($\phi_p(z_1,\dots,z_{m+n})$, $\neg\psi$)
(9)                      $S = S \cup \{(cond_p \Rightarrow \neg p(z_1,\dots,z_m))_{mgu(\phi_p,\psi)}\}$
(10)      return $\bigwedge_{\varphi \in S} \varphi$.
(11) else, use distribution properties as follows:
(12)      if $\psi = \phi \vee \varphi$
(13)            return UCStep($A(x_1,\dots,x_n)$, $\phi$) $\vee$ UCStep($A(x_1,\dots,x_n)$, $\varphi$).
(14)      elseif $\psi = \phi \wedge \varphi$
(15)            return UCStep($A(x_1,\dots,x_n)$, $\phi$) $\wedge$ UCStep($A(x_1,\dots,x_n)$, $\varphi$).
(16)      elseif $\psi = \neg\phi$
(17)            return $\neg$ UCStep($A(x_1,\dots,x_n)$, $\phi$) $\wedge$
(18)                        UCStep($A(x_1,\dots,x_n)$, $TRUE$).
(19)      elseif $\psi = \exists x\ \phi(x)$, return $\exists x$ UCStep($A(x_1,\dots,x_n)s$, $\phi(L))_{[L/x]}$.
```

[a] $cond_p$ is either *TRUE* or $(\neg case_p^1 \wedge \dots \wedge \neg case_p^{l_p})$ using Definition 18.

Fig. 4. One step unit-case filtering.

5.1.1. Example

Consider our book-keeping robot example from Section 2.2.1. Suppose that we have two rooms, office and lounge, and two books, C++ and Java, and our belief state formula is $in(\mathrm{C{+}{+}}, Office) \wedge in(Java, Lounge)$. 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 $move(\mathrm{C{+}{+}}, Office, Lounge)$. It has unit-case successor state axioms.

Our algorithm UCF works as follows. First, we add the precondition to the belief state formula. The new belief state is (since (Office $\neq$ Lounge), the last term of precondition is true):

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

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

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

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

5.2. STRIPS domains

In STRIPS domains [18,37] actions have 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 1:1. Consequently STRIPS successor state axioms cannot be treated by algorithm UCF even though they are unit-case. Successor state axioms in STRIPS domains are of the form:

$$\begin{aligned} Poss(a,s) \quad &\Rightarrow \quad \forall y_1,\dots,\forall y_m \left(p\big(y_1,\dots,y_m, do(a,s)\big) \Leftrightarrow \big(case_p^1 \Rightarrow \phi_p^1(y_1,\dots,y_m,s)\big)\right. \\ &\left.\wedge \dots \wedge \big(case_p^{l_p} \Rightarrow \phi_p^{l_p}(y_1,\dots,y_m,s)\big) \wedge \big(\neg case_p^1 \wedge \dots \wedge \neg case_p^{l_p}\big) \Rightarrow p(y_1,\dots,y_m,s)\right) \end{aligned} \tag{8}$$

where $\phi_p^j(y_1,\dots,y_m,s)$ $(j \leqslant l_p)$ is either TRUE or FALSE. Notice that $case_p^j$ can assign equality to a subset of parameters of p as in the previous section. $case_p^j$ is of the form $(y_p^{j_1} = x_p^{j_1}) \wedge \dots \wedge (y_p^{j_k} = x_p^{j_k})$ where variable $x_p^{j_1}$ is an argument of action a and variable $y_p^{j_1}$ is an argument of predicate p.

We can instantiate an action by constants $c_1,\dots,c_n$. An instantiated action a is equal to $A(c_1,\dots,c_n)$. A case after instantiation of action a is $case_{[x_1,\dots,x_n/c_1,\dots,c_n]}$ (we omit p and j from $case_p^j$ for simplicity).

Denote $case(a) = case(A(c_1,\dots,c_n)) = case_{[x_1,\dots,x_n/c_1,\dots,c_n]}$.

```
PROCEDURE FOSF($\langle a_i, o_i\rangle_{0<i\leqslant t}$, $\psi$)
$\forall i$, $a_i$ an action, $o_i$ an observation and $\psi$ a belief state formula. $o_i$ and $\psi$
in EAFOL (the form $\exists^*\forall^*\phi$).
 (1) if $t = 0$, return $\psi$.
 (2) return Move-Quan[a]($o_t\wedge$ FO-STRIPS-Step($a_t$,
 (3)                Move-Quan($precond_{a_t}\wedge$ FOSF($\langle a_i, o_i\rangle_{0<i\leqslant(t-1)}$, $\psi$))))
```

[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($a$, $\phi(L))_{[L/x]}$
 (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)         $\psi_{split} \leftarrow$ split every clause in $\psi$ into cases[a]
 (6)         $E \leftarrow \{\varphi \mid \varphi \in \psi_{split}$ and there is $l \in$ atom($\varphi$) and $l' \in$ Eff($a$)
 (7)               such that $l$ is an instance of $l'\}$
 (8)         $S \leftarrow \psi_{split} \setminus E$
 (9)         For all $l \in$ Eff($a$)
(10)               $E \leftarrow$ resolve-out($l$, $E$)[b]
(11)         $\phi = \bigwedge_{c\in E\cup S} c$
(12)         Eff$^+(a) \leftarrow$ literals affected to TRUE
(13)         Eff$^-(a) \leftarrow$ literals affected to FALSE
(14)         return $\phi \wedge \bigwedge_{p(v)\in \mathrm{Eff}^+(a)} p(v) \wedge \bigwedge_{p(v)\in \mathrm{Eff}^-(a)} \neg p(v)$.
```

[a] The definition of such splitting appears in Eq. (9).
[b] Resolve-out: uses resolution on all instances of l in E. After all resolutions are done, we select only those clauses that mention no instance of l or its negation.

Fig. 5. First-order STRIPS filtering.

A STRIPS action affects the truth value of some of the instantiated predicates and keeps the value of the others. An instantiated predicate is a predicate symbol with some arguments constants and others variables. We refer to the set of *affected instantiated predicates* as Eff(a). Eff(a, p) is defined as follows:

$$\mathrm{Eff}(a,p) = \left\{p(y_1,\ldots,y_m)_{[y_{i_1}/v_{i_1},\ldots]} \mid case_p^j(a) = \left((y_{i_1} = c_{i_1}) \wedge \cdots\right) \text{ for case } j \text{ of } a, p\right\}$$

Eff(a) is:

$$\mathrm{Eff}(a) = \bigcup_{p\in \mathrm{Pred}} \mathrm{Eff}(a,p)$$

Therefore, $|\mathrm{Eff}(a,p)|$, the number of elements in the set Eff(a, p), is exactly the number of cases, l of p, a. $|\mathrm{Eff}(a)|$, the number of elements in the set Eff(a), is $\sum_{p\in\mathrm{Pred}} |\mathrm{Eff}(a,p)|$, the total number of cases stated for a and any predicate fluent.

An *affected literal* (an atomic formula or its negation) is any atom in Eff(a), a negation of such an atom, or a literal whose further instantiation (after replacing some variables with terms) without negation is in Eff(a).

Our First-Order STRIPS Filtering (FOSF) algorithm is presented in Fig. 5. A FOL formula is in *EAFOL* if no existential quantifier appears within the scope of a universal quantifier. The input to FOSF is assumed to be a belief state formula $\exists^*\forall^*\phi$ in EAFOL ($\exists*$ refers to any number $\geqslant 0$ of existential quantifications of variables; $\forall*$ refers to any number $\geqslant 0$ of universal quantifications of variables) with no equality, and with observation formulas ϕ provided in clausal form.

Step (5) of our algorithm splits every clause in the clausal form of ϕ into cases. It does so to allow treating instances of predicates which are affected by a differently from those which are not affected by a. We call this step *splitting into cases*. We split every clause φ as in the algorithm described below.

Let $p(\vec{v})$ be an atom in φ ($\vec{v}$ is a vector of variables and constants). Then, let $\vec{v}_j$ be the instantiation of $\vec{v}$ for $case_p^j(a)$ defined as follows ($\vec{v} = \langle w_1,\ldots,w_m\rangle$ and each w_i is a constant or variable):

```
(1) $\vec{v} = \langle w_1,\ldots,w_m\rangle$
(2) For $i = 1$ to $n$
(3)         if $w_i$ is constant, and $y_i = c_j$ is in $case(a)$ then
(4)               If $w_i \neq c_j$, return TRUE
(5)               Else set $u_i = c_j$
(6)         Else $w_i$ is variable,
(7)               If $y_i = c_j$ in $case(a)$ set $u_i = c_j$
(8)               Else set $u_i = w_i$
(9) return $p(u_1,\ldots,u_m)$
```

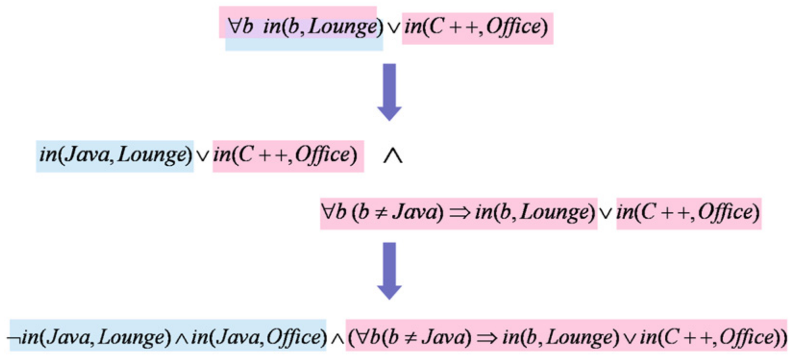

∀b in(b,Lounge) ∨ in(C++,Office)
in(Java,Lounge) ∨ in(C++,Office) ∧
∀b (b ≠ Java) ⇒ in(b,Lounge) ∨ in(C++,Office)
¬in(Java,Lounge) ∧ in(Java,Office) ∧ (∀b(b ≠ Java) ⇒ in(b,Lounge) ∨ in(C++,Office))

Fig. 6. STRIPS example: boxed literals are affected, filled ones are not.

We split every clause φ into:

$$\varphi_{[\vec{v}/\vec{v}_1]},\ \ldots,\ \varphi_{[\vec{v}/\vec{v}_{l_p}]},\ \left(\neg case_p^1 \wedge \cdots \wedge \neg case_p^{l_p} \Rightarrow \varphi\right) \tag{9}$$

Note that we treat $(\neg case_p^1 \wedge \cdots \wedge \neg case_p^{l_p} \Rightarrow \varphi)$ as a single literal which is not affected by our action. Also, as mentioned above, we require Unique-Names Axioms for objects here.

Example. Let $case_1^p = (x_1 = A)$ and $case_2^p = (x_1 \neq A)$. Let $\varphi = p(x_1, B) \vee q(x_1)$. We split this with respect to predicate p into cases. The result would be

$$p(A, B) \vee q(A),\ \left((x_1 \neq A)\ \Rightarrow\ p(x_1, B) \vee q(x_1)\right) \tag{10}$$

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. 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. The literals that are not affected are the only ones who stay the same in the new belief state. Then all predicates in 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 the FOSF algorithm works is shown in Fig. 6. 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:

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

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. There is no literal in these clauses that has both affected and unaffected instances. The middle line of the figure shows the belief state after splitting into cases. 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 20. *Let action a, observation o, and belief state formula ψ in EAFOL be given to FOSF as input. Assume progression is first-order definable, let l be the maximum number of cases in all successor state axioms. Assume that every case of a instantiates all variables. Then, algorithm FOSF returns the filtering of ψ with a and o in time $O((l \cdot |\psi|)^{2|\mathrm{Eff}(a)|})$.*

Proof. See Appendix A.12. □

Theorem 21. *Let progression be first-order definable and let C be the number of constant symbols appearing in the domain description. Let R be the maximum arity of predicates and m be the number of predicates. If ψ is in 2-FO-CNF,[2] the length of formula after filtering with t actions $a_1, \ldots, a_t$ is bounded by $O(m^2 \cdot (R + C)^{2R})$, a term independent of t. The time taken for filtering each action a is $O(m^2 \cdot (R + C)^{2R})$.*

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

Proof. See Appendix A.13. □

Thus, these theorems give a polynomial time and space guarantees for filtering (polynomial in the number of predicates and the length of action sequence). Importantly, Theorem 21 gives a bound on the size of our belief state representation, $|\psi|$, that is maintained after $T > 0$ steps. This bound is important for ensuring tractable (polynomial time) filtering indefinitely, since the inference time per step is polynomial in $|\psi|$. (Notice that without this result it is possible that one step of filtering results in a belief states of size $O(|\psi|^2)$, then compounding over T steps to belief state of size $O(|\psi|^{2T})$.)

6. Related works

Several research streams in AI concern problems that are relevant to our work here. We divide the work relevant to ours into four parts: works within logical AI and Databases that focus on reasoning about actions; works within stochastic filtering and state estimation in Markov Chains; and works between Philosophy, Linguistics, and AI that focus on Belief Revision and Update. We choose not to cover learning in dynamic systems in this scope, but the two are related; see [1] for details.

6.1. Reasoning about actions

Our work is closest to works on reasoning about actions. In its general form, Logical Filtering is a generalization of Database Progression [66,39,40], formal verification with Symbolic Model Checking [7,10], and Belief Update [27,55,28].

Those works that are closest to ours examine progression in FOL theories. In that context, work on filtering is between database progression and stochastic filtering. While applying similar fundamental techniques to progression, filtering's goals are different and so are the types of sought results. It aims at procedures that process a sequence of actions *tractably indefinitely*.

In a series of papers [51,38,40] Lin and Reiter defined progression of FOL theories and showed that sometimes the filtered belief state has no representation within FOL. They also showed that FOL progression (the set of (possibly infinite) FOL consequences uniform in a situation) is enough to characterize progression for correct answers to queries in FOL pertaining to any specified future ground situation term. [63,64] extended this work and showed that FOL progression is further enough to answer correctly entailment of the form "after α, property ϕ will always be true."

Those works are compatible with ours in that they establish correctness of FOL progression for classes of queries that contain ours. They do not concern tractability, which is our focus here.

Somewhat closer to ours are works that find classes of action theories for which progression is tractable or more efficient than in general. [21] studied cases of progression in which one can exploit correspondence between the situation calculus and relational databases to build systems for reasoning about actions. It focuses on tractability results, but is based on standard relational database technology, and assumes the initial knowledge base is complete (the initial belief state includes exactly one state).

Following that line of research, [41,65] developed algorithms for incomplete initial knowledge bases. [41] developed algorithms that can be applied to any *proper* initial knowledge (roughly, one that is consistent) and local actions (successor-state axioms are of the form $F(\vec{x}, do(a,s)) \equiv \gamma_F^+(\vec{x}, a) \vee (F(\vec{x}, s) \wedge \neg\gamma_F^-(\vec{x}, a))$, and $\gamma_F^+(\vec{x}, a), \gamma_F^-(\vec{x}, a)$ are disjunctions of formulas of the form $\exists\vec{z}[a = A(\vec{y}) \wedge \phi(\vec{y})]$ with ϕ quantifier free uniform in s and $\vec{y}$ the exclusive union of $\vec{x}, \vec{z}$). However, their progression algorithm is only complete when applied to context-complete knowledge, *i.e.*, when knowledge is complete about $\phi(.)$ when successor-state actions need to be applied. This differs from our results here in that we effectively restrict the form of successor-state axioms, but do not make assumptions of completeness for the initial or intermediate knowledge.

[65] extended this work on actions of local effects, and showed that progression is always first-order definable for those. They also show that progression is *finite* when $\phi(.)$ has no quantifiers and includes only variables that appear in $\vec{y}$ (*strictly local effects*). Their finiteness result is important, but their formulas grow exponentially with even one progression step; Theorem 2 in [65] computes projection and results in a disjunction of l disjuncts, where l is the number of "J-models" (we omit the definition here) of the progression, a number that is worst-case at least exponential in the number of ground atoms.

In contrast, our results focus on progression that grows at most linearly with every step: Our results in Sections 4 and 5 focus on progression whose output is of size $O(f \cdot |\psi|)$, where ψ is the input databases and where f is a constant that depends only on the form of the action theory, D. Our final result, Theorem 21, gives an even stronger result, bounding the size of progression after T time steps (T actions and observations) by a polynomial $O(T|\psi|^2)$ of the input that grows linearly with T. Such results are critical to practical filtering over long sequences actions and observations.

A final contrast with these previous results is the scope of action domains to which they apply. Specifically, our results in Section 5 *do not assume that the domain is strictly local*. For example, the example action $a' = push2$ shown in Eq. (6) in our Section 4.2.1 is not a strictly local effect action, so it cannot be processed by the procedure of [65], whereas our FF can process it in linear time and with only linear growth in the belief-state representation per progression step.

Finally, a related line of work to the ones above investigates Fluent Calculus [60,61] as a computationally efficient means for updating belief states. [60] introduces a dual representation for basic action theories based on state update axioms that explicitly define the direct effects of each action. It 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. [61] presents an implementation technique for progression based on the state update axioms and the use of a constraint solver. Another work [24] 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.

The main difference that we draw between this work and ours is in that update in this framework trades off accuracy for efficiency. While updates are generally efficient, the results are not guaranteed to be a precise representation of the FOL progression of ones' belief state. These results are useful for our purposes as subroutines that may compute progression (*e.g.*, in compilation of Section 4). However, no computational analysis of time or space is given by the author, so comparison is hard.

6.2. Propositional progression and filtering

Previous works that address tractable algorithms for exact filtering focus on propositional cases and intractability results for the general case of those.

[3] present 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. That work is restricted to the propositional case.

Earlier research about the hardness of belief revision and update [17,35] focused on the properties of different update semantics, showing that finding out if a query follows from the updated belief state is coNP-hard. [8] further showed that many[3] semantics discussed in the literature lead to an exponential growth in the representation size of any belief-state representation scheme, if the polynomial hierarchy does not collapse.

Our contribution and focus are different from earlier work by our emphasis on FOL and on tractable, scalable filtering techniques for indefinite online processing of updates. These would take computation time that depends only linearly on the number of time steps. We focus on tractable cases, and apply belief update semantics that is easier for analysis and computation. Our approach focuses on AI applications in partially observable domains, and is closer in spirit to data-intensive control-theory approaches to tracking of dynamic systems.

6.3. Stochastic filtering

Early work, beginning with Gauss, assumed *stochastic* models. For example, the *Kalman filter* [26] is a ubiquitous device that maintains a multivariate Gaussian belief state over n variables, assuming linear-Gaussian transition and sensor model. Crucially, the $O(n^3)$ update cost and the $O(n^2)$ space requirement *do not depend on the length of the observation sequence*. This type of tractability permits the Kalman filter to run indefinitely and enables well-known applications [59,43]. This type of recursive estimation motivates our term *logical filtering*.

Stochastic filtering and reasoning in probabilistic dynamic models is an area of vast literature and common everyday applications [45]. Progress and research on it is divided between Statistics [30], Artificial Intelligence [47,16], and Control Theory [43,12]. Research on stochastic filtering focuses on Hidden Markov Models [50,19] and Dynamic Bayesian Networks [13,23,48]. Reasoning and estimation with these models is hard when the systems dynamics or sensor model are not Gaussian-linear and the number of state features is of modest size (*e.g.*, a hundred features, thus $\geqslant 2^{100}$ states).

The difficulty of stochastic filtering led early to modern research to focus on reasoning techniques that approximate the model [59,47,30,6,62,31] or approximate inference in the model via sampling [15,16]. Unfortunately, these still require exponential-time computations or produce poor approximations in most cases (approximations work in practice for some applications), especially when the transition model includes states defined from discrete variables [48,5]. This difficulty is a second motivation for our work here (the first are immediate applications [9]).

Logical Filtering is close to stochastic filtering, only without probabilities in belief state, transition model, or sensor model. The filter's first input is a representation for belief state $\sigma_0 \subseteq \mathcal{S}$, where $\mathcal{S} \subseteq \mathit{Pow}(\mathcal{P})$ is the system's set of states defined with propositional fluents $\mathcal{P}$ (fluents are state features that change over time). The filter's second input is a representation for transition relation $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$, with $\langle s, s' \rangle \in \mathcal{R}$ iff state s at time t may transition to state s' at time $t+1$ for any $t \geqslant 0$. The filter's final input is a sequence of representations of observations $o_{0:T}$, for $o_t \subseteq \mathcal{S}$, given in an online fashion, one per time step $t \leqslant T$. The filter's task is to update the belief state representation recursively to σ_t (returning σ_t when requested to) so that it accounts exactly for all the states that are consistent with $\sigma_0, \mathcal{R}, o_{0:t}$.

An inquiry in logical filtering is typical of inference from logical knowledge bases, *e.g.*, satisfiability checking (*e.g.*, *is it possible that A is on B?*), entailment (*e.g.*, *must it be that A is on B?*), and checking the feasibility of a state (*e.g.*, *is state* $\{\mathit{inHand}(B), \mathit{onTable}(A)\}$ *possible*?).

[3] They also show that the WIDTIO (When In Doubt Throw It Out) semantics has a compact representation. This is so because one can always discard information to achieve a compact representation.

6.4. Belief revision and update in FOL

Important research that is related to ours is on the usage of FOL for belief update, either as holder of a belief formula or as supplying some of the semantics involved with different revision tasks. Examples of the first are [11,14] (first-order AGM belief revision) and [66] (belief update and revision in simple subclasses of FOL).

An important difference that we draw with these works is that ours provides efficient inference procedures, while those lines of work focus on the use of general-purpose theorem provers and provide only intractability results.

[34] 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. This is called *possible cause approach* (PCA).

Interestingly, a set of works examined the question of how belief update and progression are related with each other. [36] shows that several different semantics for belief update can be expressed in a single framework for reasoning about actions. That framework can be considered as a common core of all these update formalisms. In the other direction, [29] shows that belief update is a specific case of feedback-free action progression. It also presented *reverse update*, which is to regression as update is to progression.

Similarly, works on iterated belief revision [54,25] examine the belief change associated with performing actions. They assume a theory of action represented in the situation calculus and extended to deal with belief. Their belief revision corresponds to our observations, where belief update corresponds to update with actions.

7. Conclusions

We presented semantics and methodology for efficient filtering in domains that include many objects. Some of our algorithms allow objects whose identity is not certain. We generalized this problem to filtering FOL formulas with ground actions and FOL observations.

Previous results guarantee that FOL representations are sufficient for filtering. We showed that such filtering is solvable in polynomial time when actions map states 1:1 (after precompilation of the domain), or the actions have a case-based representation (this includes STRIPS actions whose success or failure are observed). We showed that 1:1 actions allow us to filter FOL belief state formulas in linear time per time step in the size of belief-state representation, given an algorithm for computing the progression of atomic formulas, and assuming this algorithm compiles our domain (and terminates) before filtering commences.

When actions are successful-STRIPS or Unit-Case, we can filter arbitrary belief state formulas efficiently without pre-compilation. We showed that for a class of actions and belief states (roughly, corresponding to conjunctions of first-order clauses of size 2) belief state formulas are guaranteed to remain represented compactly (polynomial size in the initial belief state representation and the number of time steps T) for arbitrary action-observation sequence lengths, T. Those cases permit filtering of actions and observations indefinitely in polynomial time (in the number of predicates and constants of our domain).

Tractable filtering is important for many practical, everyday life applications. Technologies such as the ones presented in this paper are important for practical natural-language processing (NLP), autonomous agents in robotics and WWW domains [9], game playing [49], and state estimation more generally [22]. In those applications it is crucial that processing be efficient for large numbers of states and large domains. The efficient processing algorithms reported in this paper and those that we hope will be derived in the future are key to new applications in these domains. We expect uses of our algorithms in these domains, and hope to help introducing FOL-based tractable applications into mainstream AI, including NLP, autonomous agents, robot motion planning and world-state estimation, and partial knowledge games.

Acknowledgements

We wish to thank Megan Nance and Adam Vogel for useful discussions on related topics. We also thank the diligent reviewers and editors of Artificial Intelligence Journal for helping improve this manuscript significantly. Finally, we wish to acknowledge support from DAF Air Force Research Laboratory Award FA8750-04-2-0222 (DARPA REAL program) and National Science Foundation CAREER grant 05-46663.

Appendix A. Proofs

A.1. Proof of Theorem 7: Filtering algorithm

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

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

Take $S' \in Filter[a](\{S \mid \models_S \psi\})$. From Definition 6, 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

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

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 8

We show that the two sets of structures have the same elements. By Theorem 7 we know that the left-hand side of the equality is contained in the right-hand side.

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

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 22. *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*:

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

Proof. 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,

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

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

A.3. Proof of Lemma 22

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 an 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 10: Progression possibility

We want to prove:

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

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 23. *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 23

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 11: Distribution over connectives

The proof for the first three parts is similar to the proof given by [3] for propositional filtering. We use the result stated by Corollary 8 in our proof.

(1) Take a structure S' that satisfies $\mathit{Filter}[a](\varphi \vee \psi)$. 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 $\mathit{Filter}[a](\varphi)$ or $\mathit{Filter}[a](\psi)$. It follows that $\mathit{Filter}[a](\varphi \vee \psi) \Rightarrow \mathit{Filter}[a](\varphi) \vee \mathit{Filter}[a](\psi)$.
For the other direction take S' that satisfies $\mathit{Filter}[a](\varphi) \vee \mathit{Filter}[a](\psi)$. Then, it satisfies one of $\mathit{Filter}[a](\varphi)$ or $\mathit{Filter}[a](\psi)$. 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 $\mathit{Filter}[a](\varphi \vee \psi)$. It follows that $\mathit{Filter}[a](\varphi \vee \psi) \Leftarrow \mathit{Filter}[a](\varphi) \vee \mathit{Filter}[a](\psi)$.
(2) Take a structure S' that satisfies $\mathit{Filter}[a](\varphi \wedge \psi)$. 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 $\mathit{Filter}[a](\varphi)$ and $\mathit{Filter}[a](\psi)$. We conclude that every S' that satisfies $\mathit{Filter}[a](\varphi \wedge \psi)$ also satisfies $\mathit{Filter}[a](\varphi) \wedge \mathit{Filter}[a](\psi)$. It follows $\models \mathit{Filter}[a](\varphi \wedge \psi) \Rightarrow \mathit{Filter}[a](\varphi) \wedge \mathit{Filter}[a](\psi)$.
(3) Take S' that satisfies $\neg\mathit{Filter}[a](\varphi) \wedge \mathit{Filter}[a](\mathit{TRUE})$. 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 $\mathit{Filter}[a](\mathit{TRUE})$ there is a structure S such that $R_D(S, a, S')$. This S satisfies $\neg\varphi$ so S' satisfies $\mathit{Filter}[a](\neg\varphi)$. It follows that $\models \mathit{Filter}[a](\neg\varphi) \Leftarrow \neg\mathit{Filter}[a](\varphi) \wedge \mathit{Filter}[a](\mathit{TRUE})$.
(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.

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

Take structure M such that $\models_M \mathit{Filter}[a](\exists x\, \varphi(x))$. We show that $\models_M \exists x\, \mathit{Filter}[a](\varphi(L))_{[L/x]}$.
Based on Theorem 7:

$$\models_M \mathit{Filter}[a](\exists x\, \varphi(x)) \quad \text{iff} \quad M \in \mathit{Filter}[a]\big(\{S \mid \models_S \exists x\, \varphi(x)\}\big)$$

Based on Definition 6:

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

Based on Definition 5, there exists an S that:

$$\models_S \exists x\, \varphi(x)$$
$$\models_S \mathit{precond}_a$$
$$|S| = |M|$$
$$p^M = \{\langle v_1, \ldots, v_m\rangle \mid \models_S \mathit{succ}_{p,a}(v_1, \ldots, v_m)\}$$
$$f^M = \{\langle v_1, \ldots, v_r\rangle \mid \models_S \mathit{succ}_{f,a}(v_1, \ldots, v_r)\}$$

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.

$$M' \in \mathit{Filter}[a]\big(\{S_{L,\varphi}\}\big) \quad \text{iff}$$
$$|M'| = |S_{L,\varphi}|$$
$$p^{M'} = \{\langle v_1, \ldots, v_m\rangle \mid \models_{S_{L,\varphi}} \mathit{succ}_{p,a}(v_1, \ldots, v_m)\}$$
$$f^{M'} = \{\langle v_1, \ldots, v_r\rangle \mid \models_{S_{L,\varphi}} \mathit{succ}_{f,a}(v_1, \ldots, v_r)\}$$

We conclude that $\models_{M'} \mathit{Filter}[a](\varphi(L))$.
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 24. *Take $\psi \in \mathcal{L}(\mathcal{D}_L)$, we can prove that*:

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

Proof. 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 24

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

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

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 X to M such that $X = v$ and call this new structure M''. $\models_{M''} \psi$ and M is a restriction of M'' to $\mathcal{L}(\psi)\backslash\{X\}$. 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 X would satisfy the existential quantifier and the proof is done.

A.8. Proof of Theorem 13

Theorem 11 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 [3]).

For "⇐" of 2, let S' be a structure that satisfies $\mathit{Filter}[a](\varphi) \wedge \mathit{Filter}[a](\psi)$. Then, it satisfies both of $\mathit{Filter}[a](\varphi)$ and $\mathit{Filter}[a](\psi)$. For $\mathit{Filter}[a](\varphi)$ there is a structure S such that $\mathcal{R}_{\mathcal{D}}(S, a, S')$ and S satisfies φ. Similarly, for $\mathit{Filter}[a](\psi)$ 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 $\mathit{Filter}[a](\varphi \wedge \psi)$. It follows that $\models \mathit{Filter}[a](\varphi \wedge \psi) \Leftarrow \mathit{Filter}[a](\varphi) \wedge \mathit{Filter}[a](\psi)$.

For "⇒" of 3, let S' be a structure that satisfies $\mathit{Filter}[a](\neg\varphi)$. 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 $\mathit{Filter}[a](\varphi)$ and therefore it satisfies $\neg\mathit{Filter}[a](\varphi)$. Clearly, S' also satisfies $\mathit{Filter}[a](\mathit{TRUE})$. So S' satisfies $\neg\mathit{Filter}[a](\varphi) \wedge \mathit{Filter}[a](\mathit{TRUE})$. It follows that $\models \mathit{Filter}[a](\neg\varphi) \Rightarrow \neg\mathit{Filter}[a](\varphi) \wedge \mathit{Filter}[a](\mathit{TRUE})$.

A.9. Proof of Theorem 14

By Theorem 13, we know that algorithm FF is correct. It decomposes $\psi \wedge \mathit{precond}_a$ 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 $O(|\psi \wedge \mathit{precond}_a| \times F)$.

A.10. Proof of Theorem 17

Using formula 1 we should show that every clause in formula 1 is also in the set of 7 and every Φ in 7 is a consequence of formula 1. We repeat formula 1 here.

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

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 $\psi \wedge \mathit{precond}_a \models \Phi(\mathit{succ}_{p_1,a}, \ldots, \mathit{succ}_{p_n,a})$. By the above definition

$$\psi \wedge \mathit{precond}_a \wedge \bigwedge_i \forall y_1 \ldots \forall y_m p'_i(y_1, \ldots, y_m) \quad \Leftrightarrow \quad \mathit{succ}_{p_i,a}(y_1, \ldots, y_m)$$
$$\bigwedge_i \forall z, f'_i = z \quad \Leftrightarrow \quad \mathit{succ}_{f_i,a}(z) \models \Phi\big(p'_1, \ldots, p'_n\big)$$

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 $\mathit{succ}_{p_i,a}$ in M, we conclude that $\models_M \Phi(\mathit{succ}_{p_1,a}, \ldots, \mathit{succ}_{p_n,a})$. As a result,

$$\xi \models \Phi(\mathit{succ}_{p_1,a}, \ldots, \mathit{succ}_{p_n,a})$$

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 7, $\xi \models \Phi(p'_1,\ldots,p'_n)$. Since $\psi \wedge precond_a \models \Phi(succ_{p_1,a},\ldots,succ_{p_n,a})$, this part is also trivial.

A.11. Proof of Theorem 19

Correctness. To prove that the algorithm UCF returns the filtering of ψ correctly we use the result of Theorem 17. 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_ps are unit clauses, the size of tautologies is at most 2. The algorithm checks the unifiability of every pair of ϕ_p and $\phi_{p'}$ 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 13 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. (6) 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

$$Poss\bigl(A(x_1,\ldots,x_n),s\bigr) \quad\Rightarrow\quad \bigl(cond_p \Rightarrow \bigl(p_p\bigl(z_1,\ldots,z_m,do(a,s)\bigr) \Leftrightarrow \phi_p(z_1,\ldots,z_{m+n},s)\bigr)\bigr)$$

where $l = l_1 + 1 + \cdots + l_t + 1$. 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 $2l_1 + Rl_1 + \cdots + 2l_t + Rl_t$ which is $O(Rl)$.

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 $O(Rl|\psi \wedge precond_a|)$. 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 $x_1,\ldots,x_l$. The size of filtering of TRUE would be less than $\Sigma_i \Sigma_j (x_i + x_j) = \Sigma_i (l x_i + \Sigma_j x_j) = O(Rl^2)$. The size of the formula is $O(Rl^2 + Rl|\psi \wedge precond_a|)$.

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

A.12. Proof of Theorem 20

Correctness. To prove that the algorithm FOSF returns the filtering of ψ correctly we use the result of Theorem 17. 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_ps 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 that 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 $c'(p_1,\ldots,p_n)$.

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

t_i cannot be TRUE or FALSE otherwise it would be subsumed by the last two terms of (14) in FO-STRIPS-Step. t_i cannot be in Eff(a) and since the resolution is complete c' should be computed by (11).

Complexity. Suppose that the length of the belief state formula is $|\psi|$. The first two lines (recursive calls) of the algorithm take $O(|\psi|)$ time. Splitting into pure literal clauses takes $O(|E| + |\psi|)$ time because the size of the resulted formula is $|E| + |S|$ and $|S| \leqslant |\psi|$ (a clause in the original belief state is either broken into multiple clauses, in which case the number of ones added to E is more than the number of ones added to S, or is not broken and is added directly to E or S and the number of such clauses is less than $|\psi|$).

After splitting into cases, every literal that is an instance of a member of Eff(a) is resolved out from E. This process may not terminate in general, but we assume in the theorem that cases leave no variable uninstantiated. This reduces resolution over those variables to propositional resolution, which is guaranteed to terminate. More specifically, there are at most l different literals that are instantiated this way, for every affected literal in Eff(a).

Thus, the size of E before resolution is $\leqslant l \cdot |\psi|$. After resolution E's size increases to at most $(l \cdot |\psi|)^{2|\mathrm{Eff}(a)|}$. The time complexity of the resolution depends on the size of the resulting set, so the algorithm takes $O((l \cdot |\psi|)^{2|\mathrm{Eff}(a)|})$, and the resulting belief state is of the same size.

A.13. Proof of Theorem 21

The number of different literals possible with m predicates of maximum arity R and C constants is bounded by $2m \cdot (R+C)^R$, for m predicates. Thus, the total number of 2-clauses possible with m predicates of arity at most R and C constants (and unbounded number of variables) is bounded by $(2m \cdot (R+C)^R)^2$. Since all clauses in the algorithm are 2-clauses, the number of clauses maintained in the belief state at time t is of size $O(m^2 \cdot (R+C)^{2R})$.

The time complexity follows from this bound on the size of the resulting formula and noticing that the majority of time spent in the procedure is on resolution and deriving the resolvents in that bound.

References

[1] Eyal Amir, Allen Chang, Learning partially observable deterministic action models, Journal of Artificial Intelligence Research 33 (2008) 349–402.
[2] Eyal Amir, Sheila McIlraith, Partition-based logical reasoning for first-order and propositional theories, Artificial Intelligence 162 (1–2) (2005) 49–88.
[3] Eyal Amir, Stuart Russell, Logical filtering, in: Proc. Eighteenth International Joint Conference on Artificial Intelligence (IJCAI '03), Morgan Kaufmann, 2003, pp. 75–82.
[4] A. Blass, Y. Gurevich, Background, reserve, and Gandy machines, in: P. Clote, H. Schwichtenberg (Eds.), Computer Science Logic (Proceedings of CSL 2000), in: LNCS, vol. 1862, Springer, 2000, pp. 1–17.
[5] Georg Böker, Jan Lunze, Stability and performance of switching Kalman filters, International Journal of Control 75 (16–17) (2002) 1269–1281.
[6] Xavier Boyen, Daphne Koller, Tractable inference for complex stochastic processes, in: Proc. Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI '98), Morgan Kaufmann, 1998, pp. 33–42.
[7] Jerry R. Burch, Ed.M. Clarke, Ken L. McMillan, David L. Dill, L.J. Hwang, Symbolic model checking: 10^{20} states and beyond, in: Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, Washington, DC, 1990, pp. 1–33.
[8] Marco Cadoli, Francesco M. Donini, Paolo Liberatore, Marco Schaerf, The size of a revised knowledge base, Artificial Intelligence 115 (1) (1999) 25–64.
[9] Allen Chang, Eyal Amir, Goal achievement in partially known, partially observable domains, in: Proceedings of the 16th Int'l Conf. on Automated Planning and Scheduling (ICAPS '06), AAAI Press, 2006.
[10] Edmund M. Clarke, Armin Biere, Richard Raimi, Yunshan Zhu, Bounded model checking using satisfiability solving, Formal Methods in System Design 19 (1) (2001) 7–34.
[11] Maria R. Cravo, João P. Cachopo, Ana C. Cachopo, João P. Martins, Permissive belief revision, in: EPIA, 2001, pp. 335–348.
[12] Jon H. Davis, Foundations of Deterministic and Stochastic Control, Birkhäuser, Boston, 2002.
[13] Tom Dean, Keiji Kanazawa, A model for reasoning about persistence and causation, Computational Intelligence 5 (3) (1989) 142–150.
[14] Simon Dixon, Wayne Wobcke, The implementation of a first-order logic AGM belief revision system, in: ICTAI, 1993, pp. 40–47.
[15] Arnaud Doucet, Nando de Freitas, Kevin Murphy, Stuart Russell, Rao-Blackwellised particle filtering for dynamic Bayesian networks, in: Proc. Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI '00), Morgan Kaufmann, 2000, pp. 176–183.
[16] Arnaud Doucet, Nando De Freitas, Neil Gordon (Eds.), Sequential Monte Carlo Methods in Practice, Statistics for Engineering and Information Science, Springer-Verlag, 2001.
[17] Thomas Eiter, Georg Gottlob, On the complexity of propositional knowledge base revision, updates, and counterfactuals, Artificial Intelligence 57 (2–3) (October 1992) 227–270.
[18] Richard Fikes, Peter Hart, Nils Nilsson, Learning and executing generalized robot plans, in: Bonnie Webber, Nils Nilsson (Eds.), Readings in Artificial Intelligence, Morgan Kaufmann, 1981, pp. 231–249.
[19] Zoubin Ghahramani, Michael I. Jordan, Factorial hidden Markov models, Machine Learning 29 (1997) 245–275.
[20] Zoubin Ghahramani, An introduction to Hidden Markov Models and Bayesian networks, International Journal of Pattern Recognition and Artificial Intelligence 15 (1) (2001) 9–42.
[21] G. De Giacomo, T. Mancini, Scaling up reasoning about actions using relational database technology, in: Proc. National Conference on Artificial Intelligence (AAAI '04), 2004, pp. 245–250.
[22] Hannaneh Hajishirzi, Eyal Amir, Sampling first-order logical particles, in: Proc. Twenty Fourth Conference on Uncertainty in Artificial Intelligence (UAI '08), 2008.
[23] Finn V. Jensen, Steffen L. Lauritzen, Kristian G. Olesen, Bayesian updating in recursive graphical models by local computation, Computational Statistics Quarterly 4 (1990) 269–282.
[24] Yi Jin, Michael Thielscher, Representing beliefs in the fluent calculus, in: ECAI, 2004, pp. 823–827.
[25] Yi Jin, Michael Thielscher, Actions and belief revision: A computational approach, in: Belief Change in Rational Agents, 2005.
[26] Emil Kalman Rudolph, A new approach to linear filtering and prediction problems, Transactions of the ASME – Journal of Basic Engineering Series D 5 (1960) 35–45.
[27] Hirofumi Katsuno, Ken Satoh, A unified view of consequence relation, belief revision and conditional logic, in: John Mylopoulos, Ray Reiter (Eds.), Principles of Knowledge Representation and Reasoning: Proc. Second International Conference (KR '91), Morgan Kaufmann, Sydney, Australia, August 1991, pp. 406–412.
[28] Jérôme Lang, Belief update revisited, in: Proc. Twentieth International Joint Conference on Artificial Intelligence (IJCAI '07), 2007.
[29] Jérôme Lang, Belief update revisited, in: IJCAI, 2007, pp. 2517–2522.
[30] Steffen L. Lauritzen, Propagation of probabilities, means and variances in mixed graphical association models, Journal of the American Statistical Association 87 (1992) 1098–1108.
[31] Joseph LaViola, A comparison of unscented and extended Kalman filtering for estimating quaternion motion, in: Proceedings of the American Control Conference, vol. 3, IEEE, 2003, pp. 2435–2440.
[32] Richard Char-Tung Lee, A completeness theorem and a computer program for finding theorems derivable from given axioms, PhD thesis, University of California, Berkeley, 1967.
[33] Hector J. Levesque, Fiora Pirri, Ray Reiter, Foundations for the situation calculus, Electronic Transactions on Artificial Intelligence 3 (18) (December 1998), http://www.etaij.org.
[34] Renwei Li, Lu'is Moniz Pereira, What is believed is what is explained sometimes, in: Proc. of AAAI '96, 1996, pp. 550–555.
[35] Paolo Liberatore, The complexity of belief update, in: Proc. Fifteenth International Joint Conference on Artificial Intelligence (IJCAI '97), 1997, pp. 68–73.
[36] Paolo Liberatore, A framework for belief update, in: JELIA '00: Proceedings of the European Workshop on Logics in Artificial Intelligence, Springer-Verlag, London, UK, 2000, pp. 361–375.
[37] Vladimir Lifschitz, On the semantics of STRIPS, in: Michael P. Georgeff, Amy Lansky (Eds.), Reasoning About Actions and Plans, Morgan Kaufmann, Los Altos, California, 1986, pp. 1–9.
[38] Fangzhen Lin, Raymond Reiter, State constraints revisited, Journal of Logic and Computation 4 (5) (1994) 655–678.

[39] Fangzhen Lin, Raymond Reiter, How to progress a database II: The STRIPS connection, in: Proc. Fourteenth International Joint Conference on Artificial Intelligence (IJCAI '95), Montreal, Canada, 1995, pp. 2001–2007.
[40] Fangzhen Lin, Ray Reiter, How to progress a database, Artificial Intelligence 92 (1–2) (1997) 131–167.
[41] Yongmei Liu, Hector J. Levesque, Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions, in: Proc. Nineteenth International Joint Conference on Artificial Intelligence (IJCAI '05), 2005, pp. 522–527.
[42] Bill MacCartney, Sheila McIlraith, Eyal Amir, Tomas Uribe, Practical partition-based theorem proving for large knowledge bases, in: Proc. Eighteenth International Joint Conference on Artificial Intelligence (IJCAI '03), Morgan Kaufmann, 2003, pp. 89–96.
[43] Peter S. Maybeck, Stochastic Models, Estimation, and Control, Mathematics in Science and Engineering, vol. 141, Academic Press, 1979.
[44] John McCarthy, Patrick J. Hayes, Some philosophical problems from the standpoint of artificial intelligence, in: B. Meltzer, D. Michie (Eds.), Machine Intelligence, vol. 4, Edinburgh University Press, 1969, pp. 463–502.
[45] Leonard A. McGee, Stanley F. Schmidt, Discovery of the Kalman filter as a practical tool for the aerospace industry, Technical Memorandum 86847, NASA, 1985.
[46] Sheila McIlraith, Explanatory diagnosis: Conjecturing actions to explain observations, in: Anthony G. Cohn, Lenhart Schubert, Stuart C. Shapiro (Eds.), Principles of Knowledge Representation and Reasoning: Proc. Sixth Int'l Conference (KR '98), Morgan Kaufmann, San Francisco, California, 1998, pp. 167–177.
[47] Thomas Minka, A family of algorithms for approximate Bayesian inference, PhD thesis, MIT, 2001.
[48] Kevin Murphy, Dynamic Bayesian networks: Representation, inference and learning, PhD thesis, University of California at Berkeley, 2002.
[49] Megan Nance, Adam Vogel, Eyal Amir, Reasoning about partially observed actions, in: Proc. National Conference on Artificial Intelligence (AAAI '06), AAAI Press, 2006.
[50] Lawrence R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proceedings of the IEEE 77 (2) (February 1989) 257–285.
[51] Raymond Reiter, The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression, in: V. Lifschitz (Ed.), Artificial Intelligence and Mathematical Theory of Computation (Papers in Honor of John McCarthy), Academic Press, 1991, pp. 359–380.
[52] Raymond Reiter, Proving properties of states in the situation calculus, Artificial Intelligence 64 (2) (1993) 337–351.
[53] Raymod Reiter, Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems, MIT Press, 2001.
[54] Steven Shapiro, Maurice Pagnucco, Iterated belief change and exogeneous actions in the situation calculus, in: ECAI, 2004, pp. 878–882.
[55] Steven Shapiro, Maurice Pagnucco, Yves Lespérance, Hector J. Levesque, Iterated belief change in the situation calculus, in: Principles of Knowledge Representation and Reasoning: Proc. Seventh Int'l Conference (KR '2000), Morgan Kaufmann, 2000.
[56] Afsaneh Shirazi, Eyal Amir, First order logical filtering, in: Proc. Nineteenth International Joint Conference on Artificial Intelligence (IJCAI '05), International Joint Conferences on Artificial Intelligence, 2005, pp. 589–595.
[57] James R. Slagle, Chin-Liang Chang, Richard C.T. Lee, Completeness theorems for semantic resolution in consequence-finding, in: Proc. First International Joint Conference on Artificial Intelligence (IJCAI '69), 1969, pp. 281–285.
[58] James R. Slagle, Interpolation theorems for resolution in lower predicate calculus, Journal of the ACM 17 (3) (July 1970) 535–542.
[59] Gerald L. Smith, Stanley F. Schmidt, The application of statistical filter theory to optimal trajectory determination on-board a circular vehicle, in: American Astronautical Society Meeting, 1961, pp. 61–92.
[60] Michael Thielscher, From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem, Artificial Intelligence 111 (1999) 277–299.
[61] Michael Thielscher, Flux: A logic programming method for reasoning agents, Theory Pract. Log. Program. 5 (4–5) (2005) 533–565.
[62] Rudolph van der Merwe, Eric A. Wan, The square-root unscented Kalman filter for state and parameter-estimation, in: Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2001), vol. 6, 2001, pp. 3461–3464.
[63] Stavros Vassos, Hector Levesque, Progression of situation calculus action theories with incomplete information, in: Proc. Twentieth International Joint Conference on Artificial Intelligence (IJCAI '07), 2007.
[64] Stavros Vassos, Hector Levesque, On the progression of situation calculus basic action theories: Resolving a 10-year-old conjecture, in: Proc. National Conference on Artificial Intelligence (AAAI '08), 2008.
[65] Stavros Vassos, Gerhard Lakemeyer, 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, AAAI Press, 2008.
[66] Mary-Anne Winslett, Updating Logical Databases, Cambridge University Press, 1990.