Reasoning about Partially Observed Actions

Megan Nance* **Adam Vogel** **Eyal Amir**
Computer Science Department
University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
mnance@engineering.uiuc.edu {vogel1,eyal}@uiuc.edu

Abstract

Partially observed actions are observations of action executions in which we are uncertain about the identity of objects, agents, or locations involved in the actions (e.g., we know that action $move(?o, ?x, ?y)$ occurred, but do not know $?o, ?y$). *Observed-Action Reasoning* is the problem of reasoning about the world state after a sequence of partial observations of actions and states.

In this paper we formalize Observed-Action Reasoning, prove intractability results for current techniques, and find tractable algorithms for STRIPS and other actions. Our new algorithms update a representation of all possible world states (the belief state) in logic using new logical constants for unknown objects. A straightforward application of this idea is incorrect, and we identify and add two key amendments. We also present successful experimental results for our algorithm in Blocks-world domains of varying sizes and in Kriegspiel (partially observable chess). These results are promising for relating sensors with symbols, partial-knowledge games, multi-agent decision making, and AI planning.

1 Introduction

Agents that act in dynamic, partially observable domains have limited sensors and limited knowledge about their actions and the actions of others. Many such domains include *partially observed actions*: observations of action executions in which we are uncertain about the identity of objects, agents, or locations involved in the actions. Examples are robotic object manipulation (e.g., $push(?x)$ occurs, but we are not sure about $?x$'s identity), card games (e.g., $receive(?agent, ?card)$ occurs, but we do not know $?card$'s identity), and Kriegspiel – partially observable chess (e.g., we observe $capture(?blackPiece, ?x_1, ?y_1, ?whitePiece, ?x_2, ?y_2)$, but know only $?whitePiece, ?x_2, ?y_2$).

Inference is computationally hard in partially observable domains even when the actions are deterministic and fully observed (Liberatore 1997), and we limit ourselves to updating belief states with actions and observations (*filtering*). Still, the number of potential applications has motivated considerable work on exact and approximate solutions in stochastic domains, especially to filtering (e.g., (Murphy 2002)). Most recently, research showed that cases of interest (namely, actions that map states 1:1, and STRIPS actions whose success is known) have tractable algorithms for filtering, if the actions are fully observed (Amir and Russell 2003; Shirazi and Amir 2005).

In this paper we address the problem of *Observed-Action Reasoning* (OAR): reasoning about the world state at time t, after a sequence of partial observations of actions and states, starting from a partially known initial world state at time 0. We are particularly interested in answering questions of the form "*is φ possible at time t?*" for a logical formula, φ.

First (Section 2.1), we formalize the OAR problem using a transition model and possible-states approach. There, we assume that we know what operator was executed (e.g., we know that $move(?o, ?x, ?y)$ occurred), but do not know some of its parameters (e.g., $?o, ?y$).

Then (Section 2.2), we outline two solutions that are based on current technology, and show that they are intractable. These solutions are a SATPlan-like approach that is based in part on (Kautz *et al.* 1996; Kautz and Selman 1996), and an approach based on Logical Filtering (Amir and Russell 2003; Shirazi and Amir 2005). The main caveat for both approaches in a partially observed action setting is their exponential time dependence on the number of time steps, t, and the number of possible values for the unknown action parameters.

Motivated by these results for current techniques, we present a new, tractable algorithm for updating a belief state representation between time steps with partially observed actions (Section 3). It does so using new logical constants for unknown objects. We show that a straightforward application of this idea (adding new object constants) is not complete, and identify and add two key amendments: (1) one must add implied preconditions and effects of equalities to the belief state before further updating, and (2) one must keep the belief state representation in a form that can be processed by subsequent computation.

Our new algorithm is applicable to STRIPS actions that are partially observable. The update with t partially observed actions is done in time $O(t^2 \cdot |\varphi_0|)$, and the size of the resulting belief state representation is linear in $t \cdot |\varphi_0|$, where $|\varphi_0|$ is the size of belief state representation at time 0 (φ_0 is assumed in CNF). This stands in contrast to stan-

*Currently an employee of Google Inc.

dard approaches to belief update and filtering that take time exponential in the number of propositional symbols in φ_0.

Finally, our approach is complemented with efficient inference at time t about the resulting formula. We experimented with equational model finders, and found that the Paradox model finder (Claessen and Orensson 2003) allows specifying a finite domain, and performs orders of magnitude better than propositional reasoners and first-order theorem provers (Riazanov and Voronkov 2001). Our experimental results show a growth in the time of computation that is almost linear with the number of time steps, t.

Our experiments (Section 4) are for Blocks-world domains of varying sizes (up to 30 blocks) and Kriegspiel, with a complete board of (initially) 32 pieces. They show that we can solve problems of many steps in large domains: $> 10,000$ propositional features and $> 10,000$ actions possible at every step. These results are promising for relating sensors with symbols, partial-knowledge games, multi-agent decision making, and AI planning.

2 Partially Observed Actions

Consider an agent playing and tracking a game of Kriegspiel. Kriegspiel is a variant of chess where each player can only see his/her own pieces, and there is a referee who can see both boards. When a player attempts an illegal move, the referee announces to both players that an illegal attempt has occurred. The referee also announces when captures take place, and other information such as Check.

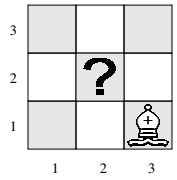

Figure 1: Uncertainty in Kriegspiel

For example, Figure 1 shows a white bishop located at the square $(3,1)$. If white attempts to move the bishop to $(1,3)$, the referee announces "Illegal." Then, white knows that the square $(2,2)$ is occupied by some black piece. When black makes its next move, the parameters of this action are hidden from white. If black moves the piece at $(2,2)$, then that square becomes empty, and white must update its belief state accordingly. Similarly, squares that white previously knew were empty might now be occupied. After just one partially observed move, the world could be in one of many possible states.

2.1 Observed-Action Reasoning: The Formal Problem

In this section we define OAR using a transition model and a process of filtering (updating belief states with actions and observations). Our language is zero-order predicate calculus (with constant symbols and predicates) with equality. We have a set $Objs$ of constant symbols that appear in our language. An *atom* is a ground predicate instance, and a *literal* is an atom or its negation. A *fluent* is an atom whose truth value can change over time. A *clause* is a disjunction of literals. For a formula φ, the number of atoms in φ is written $|\varphi|$.

In what follows, $\mathcal{P}$ is a finite set of propositional fluents, $\mathcal{S} \subseteq Pow(\mathcal{P})$ is the set of all possible world states (each state gives a truth value to every ground atom), and a *belief state* $\sigma \subseteq \mathcal{S}$ is a set of world states. A transition relation $\mathcal{R}$ maps a state s to subsequent states s' following an action a. We are particularly interested in $\mathcal{R}$ that are defined by a relational action schema, such as STRIPS (Fikes *et al.* 1981). There, action $a(\vec{x})$ has an effect that is specified in terms of the parameters $\vec{x}$. Performing action a in belief state σ results in a belief state that includes all the world states that may result from a in a world state in σ. An observation o is a formula in our language. We say an action $a(\vec{x})$ is partially-observable if $\vec{x}$ contains at least one free variable.

Definition 1 (Filtering Partially Observable Actions). *Let $\sigma \subseteq \mathcal{S}$ be a belief state. Let a be a partially-observable action, and let o be an observation. The filtering of a sequence of partial observations of actions and states $\langle a_1(\vec{x}_1), o_1, \ldots a_t(\vec{x}_t), o_t\rangle$, for $\vec{x}_i$ a vector of parameters (some variables and others constants), is defined as follows:*

1. $Filter[\epsilon](\sigma) = \sigma$ *(ϵ: an empty sequence)*
2. $Filter[a](\sigma) = \{s' | \langle s, a, s'\rangle \in \mathcal{R}, s \in \sigma\}$
3. $Filter[\bigvee_i a_i](\sigma) = \bigcup_i Filter[a_i](\sigma)$
4. $Filter[o](\sigma) = \{s \in \sigma | s \models o\}$
5. $Filter[a(\vec{x})](\sigma) =$ $Filter[\bigvee_{\vec{c} \in Objs^{arity(a)}}$ *assign. for the free vars of* $\vec{x}\, a(\vec{c})](\sigma)$
6. $Filter[\langle a_j(\vec{x_j}), o_j\rangle_{i \le j \le t}](\sigma) =$ $Filter[\langle a_j(\vec{x_j}), o_j\rangle_{i+1 \le j \le t}]$ $(Filter[o_i](Filter[a_i(\vec{x_i})](\sigma)))$

OAR concerns answering queries about $Filter[\langle a_i(\vec{x_i}), o_i\rangle_{0 < i \le t}](\sigma)$. Solving it efficiently can involve encoding belief states in logical formulae (called *belief state formulae*), thus avoiding costly updates for (exponentially sized) belief states. A belief state formulae φ represents the set of states that satisfy φ. When we write $Filter[a(\vec{x})](\varphi)$ for some formula φ, we mean $Filter[a(\vec{x})]\,(\{s \in S | s \models \varphi\})$. The following section suggests solutions via straightforward adaptations of current techniques.

We use the convention that predicates are written in a lower-case English alphabet, object constants in upper-case English alphabet, and logical variables for objects are preceded by a "?"and are in lower-case English alphabet. For two vectors $\vec{x}$ and $\vec{y}$ of length k, define $(\vec{x} = \vec{y})$ as $(x_1 = y_1 \wedge \ldots \wedge x_k = y_k)$. Let *arity*$(a)$ and *arity*(p) be the arity (number of parameters) of action a and predicate p, respectively. Furthermore, let $r_{pred} = \max_{P \in Predicates}$ *arity*(P), and similarly let $r_{act} = \max_{A \in Actions}$ *arity*(A). Finally, let n be the number of objects in our domain.

2.2 Reasoning with Current Techniques

There are two approaches that are natural to apply to OAR, and we examine them here.

SATPlan (Kautz *et al.* 1996; Kautz and Selman 1996) uses a propositional encoding of a situation calculus theory,

and assumes it can find a plan in t steps. At each time step, every action is propositionalized into *action propositions*. Every predicate is also propositionalized at each time step. These propositions (both action and non-action) include a parameter for time. We can use this representation without filtering, by including the complete set of propositional symbols and axioms for all t steps, using a SAT solver to find answers to queries about propositional variables at time t. For the following theorem, let $O(SAT(k))$ be the time it takes to determine the satisfiability of formula k.

Theorem 2. *A SATPlan-like algorithm takes time* $O(SAT(t \cdot (n^{r_{pred}} + n^{r_{act}})))$. *Furthermore, the number of propositions used is exponential in* $\max(r_{pred}, r_{act})$.

Using propositional or First-Order Logical Filtering (Amir and Russell 2003; Shirazi and Amir 2005) over a disjunction of possible actions is another option. For example, assume we need to filter some action a, and all the parameters of a are unknown. We create all ground instantiations of this action, then filter over the disjunction of those instantiations. Filtering over a disjunction of actions is equivalent to the disjunction of filtering the actions separately.

The disjunction-filtering method causes an exponential blow-up in the size of the belief state. Filtering t partially observed actions (one per time-step) yields a formula of size larger than $n^{r_{act} \cdot t}$ (somewhat smaller if some actions have fewer unknown parameters). This is because each parameter can be any of the n objects, and there are r_{act} parameters.

Theorem 3. *Disjunction-filtering a sequence of* t *actions takes time* $O(t \cdot {r_{pred}}^n)$. *After* t *steps, the belief state takes space* $O((|\varphi_0| + n) \cdot n^{r_{act} \cdot t})$.

3 Partially-Observed Deterministic Actions

In this section we give two algorithms that update a logical representation of belief states with partial action and state observations. The basic insight is an introduction of a single new constant symbol into the belief state formula for each unobserved parameter. Then, as the algorithms filter[1] actions and observations, they maintain the constraints that these new symbols must satisfy, together with the conditions that literals in our belief state formula satisfy.

Because our algorithms introduce new constant symbols for unseen parameters, they actually solve a stronger problem than that defined in Section 2.1. This allows us to answer queries not just about the current belief state, but about the *identity* of the unobserved parameters. We define a new filtering operator to formally represent this change, and relate it to the original semantics.

Definition 4 (Filtering with Constants). *For a belief state* σ *and partially-observed action* $a(\vec{x})$ *define*

$$Filter_c[a(\vec{x})](\sigma) =$$
$$Filter[a(\vec{x}')]\left(\left\{s \cup \left\{\vec{x}' = \vec{A}\right\} \mid s \in \sigma,\ \vec{A} \in Objs^{\text{arity(a)}}\right\}\right)$$

where $\vec{x}'$ *includes all constants of* $\vec{x}$ *and also new constant symbols for all free variables of* $\vec{x}$ *and the filtering* $Filter[a(\vec{x}')]\left(\left\{s \cup \left\{\vec{x}' = \vec{A}\right\} \mid s \in \sigma,\ \vec{A} \in Objs^{\text{arity(a)}}\right\}\right)$ *is defined with the following transition system:*

$$P_{\vec{x}} = P \cup \{x = A \mid A \in \textit{Objs}\ ,\ x \textit{ is a new constant in } \vec{x}'\}$$

$$\mathcal{R}_{\vec{x}} = \{\langle s \cup \{\vec{x}' = \vec{A}\}, a(\vec{x}'), s' \cup \{\vec{x}' = \vec{A}\}\rangle \mid \langle s, a(\vec{A}), s'\rangle \in \mathcal{R}\}.$$

Define $Filter_c[a(\vec{x})](\varphi)$ *as* $Filter_c[a(\vec{x})](\{s \mid s \models \varphi\})$.

Theorem 5 (Equivalence of Filtering with Constants). *For a state* $s \in S$, *a formula* φ *and an action* $a(\vec{x})$

$$Filter[a(\vec{x})](\{s \mid s \models \varphi\}) = \{s \in S \mid s \subseteq s' \textit{ for some } s' \in Filter_c[a(\vec{x})](\varphi)\}$$

Accordingly, we present algorithms that compute $Filter_c$ for two classes of actions.

3.1 Domain Description Language

Our algorithms, *Param-STRIPS-1:1* and *Param-STRIPS-Term*, accept as input a domain description in an extension of the STRIPS (Fikes *et al.* 1981) action specification language, an initial belief state formula, and a sequence of partially-specified actions and observations.

In this language, each action a is defined by a single *causal rule* of the form "a **causes** F if G", with F a condition that is met after a is executed, if G holds before executing a. Here, F is limited to be a conjunction of literals; G is limited to a formula in CNF in Section 3.2, and is limited to a conjunction of literals in Section 3.3.

We can link these causal rules to our transition model in a straightforward fashion. For an action a, let F_a and G_a be from its single causal rule. Furthermore, let $I(a, s)$ be those propositions which are not changed by executing a in s. We then define

$$\mathcal{R} = \{\langle s, a, s'\rangle \mid s \models G_a,\ s' \cap I(a,s) = s \cap I(a,s),\ s' \models F_a\}$$

The precondition of a, denoted *pre*(a), is the set of literals that appear in G in the rule for a. The effect of a, denoted *eff*(a), is the set of literals that appear in F in the rule for a. We assume that these actions are successful, i.e. when we observe an action a then we know that the precondition of a held when a was executed, and now the effect of a holds.

3.2 STRIPS Domains with CNF Preconditions

In this section we present an algorithm, *Param-STRIPS-1:1* (Figure 2) that solves OAR in a generalization of STRIPS domains. We allow the preconditions of each action to be a formula in CNF, but restrict our actions to be 1:1, which allows us to factor the filtering problem.

Definition 6 (1:1 Actions). *Grounded action* $a(\vec{c})$ *is* 1:1 *if for every state* s' *there is at most one* s *such that* $\langle s, a(\vec{c}), s'\rangle \in \mathcal{R}$. *A partially-observed action* $a(\vec{x})$ *is 1:1 if each of its ground instances is 1:1.*

[1] This is not *filtering* per-se because our new representation includes features (object names) that are not part of the current state, but we use this name for lack of a better one.

PROCEDURE Param-STRIPS-1:1($(a_i, obs_i)_{0<i\leq t}, \varphi$)
$\forall_i$, a_i an action, obs_i an observation, φ a belief-state formula
1: **if** t=0 **then**
2: **return** φ
3: **return** Param-STRIPS-Filter-1:1(a_t, obs_t, Param-STRIPS-1:1((a_i,obs_i)$_{0<i\leq(t-1)}, \varphi$))

PROCEDURE Param-STRIPS-Filter-1:1(a, obs, φ)
a an action, obs an observation, φ a belief-state formula
1: **for** free parameter p in PARAMETERS(a) **do**
2: REPLACE-WITH-CONSTANT(a, p)
3: $\phi \leftarrow \varphi \wedge$ PRECONDITIONS(a)
4: **for** clause $c \in \phi$ **do**
5: **for** clause c' in PRECONDITIONS(a) **do**
6: $\phi \leftarrow \phi \wedge$UNIFY-RESOLVE($c$,$c'$)
7: **for** literal $l(\vec{x})$ in EFFECTS(a) **do**
8: **if** $\neg l(\vec{y})$ or $l(\vec{y})$ occurs in c **then**
9: $c \leftarrow (\vec{x} = \vec{y}) \vee c$
10: **return** $\phi \wedge$ **EFFECTS(**a**)** $\wedge obs$

PROCEDURE UNIFY-RESOLVE(c,c')
c and c' clauses
1: **for** every two literals $l(\vec{A}) \in c$, $\neg l(\vec{B}) \in c'$, for some constant symbols (possibly new) $\vec{A}$, $\vec{B}$ **do**
2: $\psi \leftarrow \psi \wedge [(c \setminus \{l(\vec{A})\} \cup c' \setminus \{\neg l(\vec{B})\}) \vee \neg(\vec{A} = \vec{B})]$
3: **return** ψ

Figure 2: Filtering 1:1 STRIPS actions with CNF preconditions

The algorithm progresses as follows: Procedure PARAMETERS returns the parameter list of a, and REPLACE-WITH-CONSTANT replaces free parameters of a with new constant symbols. Next, the algorithm conjoins the preconditions of a, representing the assumption that a was successful and the preconditions of a held in φ. Then, for each clause in the belief-state formula and each clause in the precondition, we perform all possible "resolutions". These resolutions are conditioned on the equality of the arguments of the complementary literals we resolve on. If the arguments are different, then the result evaluates to TRUE; if the arguments are equal, then we are left with the correct resolution. The intuition behind this step is that we must generate all consequences of the new knowledge obtained through the preconditions of the current action before the effects of that same action change the state of the world. Then, steps 7-9 add conditions (in the form of equality statements) to clauses that contain literals from the effects of a. These equality statements are added to specify whether the clause still holds after a has been executed; the clause holds only if the arguments of l in the clause, $\vec{y}$, are different from the arguments of l in the effect, $\vec{x}$.

Theorem 7 (Correctness of Param-STRIPS-1:1). *For any formula φ representing a belief state σ, a world state s', and a sequence of 1:1 partially-observed actions and observations, $\langle a_i(\vec{x_i}), o_i\rangle_{0<i\leq t}$, where φ_t = Param-STRIPS-1:1($\langle a_i(\vec{x_i}), o_i\rangle_{0<i\leq t}, \varphi$),*

$$s' \in \textit{Filter}_c[\langle a_i(\vec{x_i}), o_i\rangle_{0<i\leq t}](\varphi) \textit{ iff } s' \textit{ satisfies } \varphi_t$$

We next analyze the running time of our algorithm.

Theorem 8. *Let φ_t be a belief-state formula, a is an action. Then, procedure* Param-STRIPS-1:1 *returns φ_{t+1} in time $O(|\varphi_t| \cdot r_{pred} \cdot (|pre(a)| + |eff(a)|))$. Also, $|\varphi_{t+1}| = O(\varphi_t + |eff(a)| + |pre(a)| \cdot |\varphi_t|)$.*

3.3 STRIPS Domains with Term Preconditions

In this section we present an algorithm, *Param-STRIPS-Term* (Figure 3), that solves OAR in STRIPS domains. In STRIPS domains, for a given causal rule a **causes** F if G, we limit F and G to be conjunctions of literals. Our algorithm maintains a compact belief state formula which grows linearly in t. The key insight behind this compactness is that we only need to add *equality* statements to our formula, which do not have to be updated in future filtering steps.

PROCEDURE Param-STRIPS-Term($(a_i, obs_i)_{0<i\leq t}, \varphi$)
$\forall_i$, a_i an action, obs_i an observation, φ a belief-state formula
1: **if** t=0 **then**
2: **return** φ
3: **return** Param-STRIPS-Filter-Term(a_t, obs_t, Param-STRIPS-Term((a_i,obs_i)$_{0<i\leq(t-1)}, \varphi$))

PROCEDURE Param-STRIPS-Filter-Term(a, obs, φ)
a an action, obs an observation, φ a belief-state formula
1: **for** free parameter p in PARAMETERS(a) **do**
2: REPLACE-WITH-CONSTANT(a, p)
3: $\phi \leftarrow \varphi \wedge$ PRECONDITIONS(a)
4: **for** clause c in ϕ **do**
5: **for** complex-literal $l_c(\vec{y})$ in clause c **do**
6: literal $l(\vec{y}) \leftarrow$ literal from $l_c(\vec{y})$
7: **for** literal $l_p(\vec{x})$ in PRECONDITIONS(a) **do**
8: **if** UNIFY($l(\vec{y}), \neg l_p(\vec{x})$) **then**
9: ADD-CONCLUSION($l_c(\vec{y}), l_p(\vec{x})$)
10: **for** literal $l_e(\vec{x})$ in EFFECTS(a) **do**
11: **if** UNIFY($l(\vec{y}), l_e(\vec{c})$) $\vee$ UNIFY($l(\vec{y}), \neg l_e(\vec{x})$) **then**
12: ADD-CONDITION($l_c(\vec{y}), l_e(\vec{x})$)
13: **return** $\phi \wedge$ **EFFECTS(**a**)** $\wedge obs$

PROCEDURE ADD-CONCLUSION($l_c(\vec{y})$, $l_p(\vec{x})$)
l_c a complex-literal, l_p a literal
$l_c(\vec{y}) \leftarrow l_c(\vec{y}) \wedge (\vec{x} \neq \vec{y})$

PROCEDURE ADD-CONDITION($l_c(\vec{y})$, $l_e(\vec{x})$)
l_c a complex-literal, l_e a literal
$l_c(\vec{y}) \leftarrow ((\vec{y} = \vec{x}) \vee l_c(\vec{y}))$

Figure 3: Filtering STRIPS actions with term preconditions

Equality statements are added in two different cases, reflected by the ADD-CONCLUSION and ADD-CONDITION subroutines in Figure 3. Note that two vectors $\vec{X}$ and $\vec{Y}$ unify if they are of equal length, and if X_i = Y_i for all ground constants in $\vec{X}$ and $\vec{Y}$. ADD-CONCLUSION is invoked when a literal in the belief state unifies with the negation of a literal in the preconditions of the current action. Equality statements are added to indicate that the parameters of these two literals cannot be equal, since we know the preconditions must hold. ADD-CONDITION is invoked when a literal in the effects of the current action unifies with a literal (or its negation) in

the belief state. In this case, equality statements are added to indicate when the literal in the belief state still holds.

These equality statements are kept locally for each literal. We call a literal together with its surrounding equality statements a *complex-literal*. In general, a complex-literal is of the form

$$[(\vec{E_1} = \vec{A}) \vee \ldots ((\vec{E_{\text{t-1}}} = \vec{A}) \vee l(\vec{A})) \wedge (\vec{P_{\text{t-1}}} \neq \vec{A})) \ldots] \wedge (\vec{P_1} \neq \vec{A}) \quad (1)$$

where $\vec{P_i}$ are arguments from literals in the preconditions of actions (added by ADD-CONCLUSION), and $\vec{E_i}$ are arguments from literals in the effects of actions (added by ADD-CONDITION). A *complex-clause* is a disjunction of complex-literals.

We present our algorithm with the example from Section 2. Suppose the literal $l = \neg$*empty*$(2,2)$ is part of our belief state φ_t, indicating that a black piece occupies the square $(2,2)$. We now observe that black moves a piece, but we know neither the piece nor its initial or final destination squares. The partially observed action a is *move*$(?p, ?x, ?y, ?a, ?b)$, where $?p$ is the unobserved piece, $(?x, ?y)$ is the initial square $?p$ occupies, and $(?a, ?b)$ is the destination square. PARAMETERS(a) returns $(?p, ?x, ?y, ?a, ?b)$, and REPLACE-WITH-CONSTANT returns $move(c_p, c_x, c_y, c_a, c_b)$. A precondition of the *move* action is that the destination must be empty: *empty*$(c_a, c_b) \in$ *pre*(a). Similarly, an effect of the *move* action is that the initial square becomes empty: *empty*$(c_x, c_y) \in$ *eff*(a). Since $\neg$*empty*$(2,2)$ unifies with the precondition *empty*(c_a, c_b), subroutine ADD-CONCLUSION adds $\neg$*empty*$(2,2) \wedge ((c_a \neq 2) \vee (c_b \neq 2))$. These conclusions reflect that $(2,2)$ cannot be the destination square since it was already occupied in φ_t. Next, since $\neg$*empty(2,2)* unifies with the effect *empty*(c_x, c_y), subroutine ADD-CONDITION adds $(((c_x = 2) \wedge (c_y = 2)) \vee \neg$*empty*$(2,2))$. These conditions reflect that $(2,2)$ is still occupied after action a has occurred only if it was not the initial square for the piece that was moved. Thus, the final update of l in φ_{t+1} is

$$[((c_x = 2) \wedge (c_y = 2)) \vee (\neg \textit{empty}(2,2))] \wedge ((c_a \neq 2) \vee (c_b \neq 2)) \quad (2)$$

Subsequent updates would only change l, replacing it with a form similar to Equation 2.

Unlike *Param-STRIPS-1:1*, *Param-STRIPS-Term* does not require that the input actions be 1:1, as long as the initial belief state is a CNF formula that contains all of its prime implicates, which we term PI-CNF[2].

Theorem 9 (Correctness of Param-STRIPS-Term). *For any PI-CNF formula φ representing a belief state σ, a world state s', and a sequence of partially observed actions with 1-CNF preconditions and observations $\langle a_i(\vec{x_i}), o_i \rangle_{0<i\leq t}$, where $\varphi_t =$ Param-STRIPS-Term$(\langle a_i(\vec{x_i}), o_i \rangle_{0<i\leq t}, \varphi)$,*

$$s' \in \textit{Filter}_c[\langle a_i(\vec{x_i}), o_i \rangle_{0<i\leq t}](\varphi) \textit{ iff } s' \textit{ satisfies } \varphi_t$$

[2]If 1:1 actions are used, PI-CNF is not needed. Otherwise, an initial belief state can be converted to PI-CNF using resolution.

Algorithm *Param-STRIPS-Term* (Figure 3) updates an initial belief state with t action-observation steps, returning a belief state representation of time t. We now show that it takes time that is proportional to t^2, with an output belief state representation of size proportional to t. Let $\#$*lit*(φ_t) be the the number of instances of non-equality literals in φ_t. Let $\#$*eff* $= \max_a |$*eff*$(a)|$ and $\#$*pre* $= \max_a |$*pre*$(a)|$. Our first theorem concerns *Param-STRIPS-Term* for a single time step, and shows that the algorithm takes time that is linear in the input size, and outputs a formula that has at most $\#$*eff* $+ \#$*pre* more non-equality literals.

Theorem 10 (Time for Update and Size of Result). *Let φ_t be a parametrized belief state formula, a is an action. Then, procedure* Param-STRIPS-Term *returns $\varphi_{t+1} \equiv$ Filter$_c[a](\varphi_t)$ in time $O(\#$lit$(\varphi_t) \cdot r_{pred} \cdot (|eff(a)| + |$pre(a)$|))$. Also, the returned formula's number of literals is $\#$lit$(\varphi_{t+1}) \leq \#$lit$(\varphi_t) + |$pre$(a)| + |$eff$(a)|$.*

Our update format and belief state format (Equation 1) for *Param-STRIPS-Term* imply that every non-equality literal accumulates at most *arity*$(a)^{r_{pred}}$ equality literals after an update with action a. Thus, after t steps, every non-equality literal accumulates a total of $t \cdot r_{act}^{r_{pred}}$ equality literals. We conclude a linear bound for the size of a belief state formula after t steps.

Corollary 11 (Iterating Param-STRIPS-Term Filter). *Let φ_0 be a parametrized belief state formula. For t STRIPS actions and observations, procedure* Param-STRIPS-Term *returns $\varphi_t \equiv$ Filter$_c[\langle a_i(\vec{x_i}), o_i \rangle_{0<i\leq t}](\varphi_0)$ in time $O(t^2 \cdot \#$lit$(\varphi_0) \cdot r_{act}^{r_{pred}} \cdot r_{pred} \cdot (\#$eff $+ \#$pre$))$. The belief state representation size is $|\varphi_t| \leq |\varphi_0| \cdot t \cdot r_{act}^{r_{pred}}$.*

Unlike the SATPlan or the Disjunction-filtering approaches, our representation does not grow exponentially as time progresses, and the total time is quadratic in t. *Param-STRIPS-Term* gives a more *compact* solution to tracking dynamic worlds with partially observed actions and fluents while not compromising efficiency.

4 Experimental Evaluation

We implemented our Param-STRIPS algorithms and tested them in both the well known Blocks-world and in Kriegspiel.

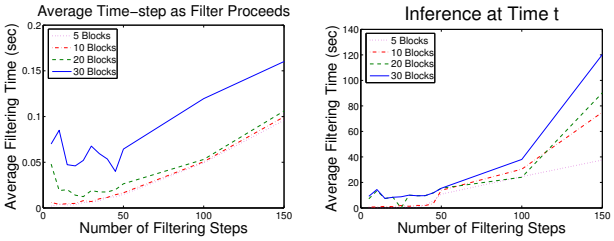

Figure 4: Param-STRIPS-Term in the Blocks-world

Our first experimental results are from the Blocks-world domain. We experimented with our Param-STRIPS-Term

algorithm over domains of sizes 5 to 30 blocks and action sequences between 5 and 150 actions in length. The timing results are shown in Figure 4. We note that the size of the domain has little effect on the running time, so our algorithm scales well to larger domains. Also, the filter time-step grows quadratically with the number of performed actions, as predicted by our theoretical results.

Figure 4 (right) also shows timing results for queries over the result of Param-STRIPS-Term that were computed using the Paradox model finder. Interestingly, while inference takes longer at the result of longer sequences, it grows slower than t^2. One would expect inference to grow exponentially with the number of time steps (the number of new object constants), so the slow growth is surprising and merits further investigation.

Next, we constructed random Kriegspiel sequences of 5 to 35 moves. In these sequences, the parameters of white's moves are observed, but the parameters of black's moves are hidden. All move sequences are on a full 32 piece 64 square chess board. This domain brings a number of challenges that space constraints prevent us from describing in length. It suffices to say that the game has an extremely large state space.

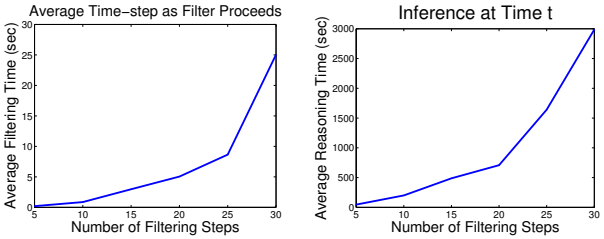

Figure 5: Param-STRIPS-Term in Kriegspiel

We filtered using *Param-STRIPS-Term*, with its efficiency guarantees, which gave us an approximation to the belief state. We assume that we are given the type of piece moved, but not the actual piece nor its initial or final location. Thus, example actions would be $moveKnight(?p, ?x, ?y, ?a, ?b)$ or $rookCapture(?c, ?p, ?x, ?y, ?a, ?b)$. Figure 5 details our filtering results. Our results are exciting because current technology cannot track exactly even a few steps (Russell and Wolfe 2005). Note also the difficulty of using the propositional STRIPS filter over a disjunction of actions. For each unspecified move action, each containing 5 parameters, the propositional filter would have to filter over $16 \cdot 8^4 = 65536$ grounded actions. Thus, in practical application, our algorithms render a new class of problems tractable.

5 Conclusion and Discussion

In this paper we presented the problem of Observed-Action Reasoning and described four algorithms for its solution. Our main contribution, Algorithm *Param-STRIPS-1:1* and Algorithm *Param-STRIPS-Term*, update belief-state representations using new object constant symbols for unobserved parameters. This representation allows the tracking of sequences of partially known actions in practically sized applications. Our immediate application for this work is in tracking full games of Kriegspiel.

Beyond our experimental evaluation, there are many real-world situations to which our algorithms apply. For example, consider problems from interpreting narratives. As the narrative progresses, pronouns are introduced. The verbs that these pronouns are associated with are represented by partially observed actions. Together with NLP techniques, our algorithms can reason about both the end state of world (after the narrative is finished), and about the unknown parameters in the actions (i.e., who did what in the story).

Acknowledgments

We wish to acknowledge support from DAF Air Force Research Laboratory Award FA8750-04-2- 0222 (DARPA REAL program).

References

Eyal Amir and Stuart Russell. Logical filtering. In *Proc. Eighteenth International Joint Conference on Artificial Intelligence (IJCAI '03)*, pages 75–82. Morgan Kaufmann, 2003.

K. Claessen and N. Orensson. New techniques that improve mace-style finite model finding. In *CADE-19, Workshop W4. Model Computation*, 2003.

Richard Fikes, Peter Hart, and Nils Nilsson. Learning and executing generalized robot plans. In Bonnie Webber and Nils Nilsson, editors, *Readings in Artificial Intelligence*, pages 231–249. Morgan Kaufmann, 1981.

Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *AAAI-96*, 1996.

Henry Kautz, David McAllester, and Bart Selman. Encoding plans in propositional logic. In J. Doyle, editor, *Proceedings of KR'96*, pages 374–384, Cambridge, Massachusetts, November 1996. KR, Morgan Kaufmann.

Paolo Liberatore. The complexity of belief update. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 68–73, 1997.

Kevin Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California at Berkeley, 2002.

Alexander Riazanov and Andrei Voronkov. Vampire 1.1 (system description). In *Proc. First International Joint Conference on Automated Reasoning (IJCAR '01)*, number 2083 in Lecture Notes in Computer Science, pages 376–380. Springer, 2001.

Stuart Russell and Jason Wolfe. Finding checkmates in Kriegspiel. In *Proc. Nineteenth International Joint Conference on Artificial Intelligence (IJCAI '05)*. Morgan Kaufmann, 2005.

Afsaneh Shirazi and Eyal Amir. First order logical filtering. In *Proc. Nineteenth International Joint Conference on Artificial Intelligence (IJCAI '05)*. Morgan Kaufmann, 2005.