PSIPLAN: Open World Planning with ψ-forms.

Tamara Babaian
Dept. of Electrical Eng. and
Computer Science
Tufts University
Medford, MA 02155 USA
tbabaian@eecs.tufts.edu

James G. Schmolze
Dept. of Electrical Eng. and
Computer Science
Tufts University
Medford, MA 02155 USA
schmolze@eecs.tufts.edu
http://www.eecs.tufts.edu/∼schmolze

Abstract

We present a new partial order planner called PSIPLAN, which builds on SNLP. We drop the closed world assumption, add sensing actions, add a class of propositions about the agent's knowledge, and add a class of universally quantified propositions. This latter class of propositions, which we call ψ-forms, distinguishes this research. ψ-forms represent *partially closed worlds*, such as "Block A is clear", or "*x.ps* is the only postscript file in directory */tex*." We present our theory of planning with sensing and show how partial order planning is performed using ψ-forms. Noteworthy are the facts that lack of information can be represented precisely and all quantified reasoning has polynomial complexity. Thus, in finite domains where the maximum plan length is bounded, planning with PSIPLAN is NP-complete.

Introduction

Several researchers have examined planning with sensing in an open world, where the agent does not have complete information about the world and must take actions both to acquire knowledge and to change the world (e.g., (Peot & Smith 1992), (Etzioni *et al.* 1992), (Krebsbach, Olawsky, & Gini 1992),(Scherl & Levesque 1997),(Golden 1998)). Incompleteness of the agent's knowledge means that it cannot use the Closed World Assumption (CWA) in the representation of the state and must rely on a different method for representing large quantities of negative information. The most comprehensive of all practical solutions to this problem is the PUCCINI planner (Golden 1998). It uses the LCW sentences in its representation, which we analyze in the next section.

Some works (e.g., (Scherl & Levesque 1997)) use first-order logic (FOL), which easily represents open worlds, but which appears to preclude practical planning algorithms due to the undecidability of entailment in FOL. Conformant Graphplan (Smith & Weld 1998) considers each possible world. SGP (Weld & Anderson 1998) extends Conformant Graphplan to handling sensing actions and uncertain effects.

We have developed a planning formalism called PSIPLAN and a sound and complete partial order planner (POP) called PSIPOP that uses PSIPLAN to plan in open worlds without sensing. Moreover, we have extended both PSIPLAN and PSIPOP to handle sensing actions, knowledge goals, information loss and conditional effects.

In this paper, we focus on demonstrating the power of our ψ-form-based language PSIPLAN, discussing the issues critical to its soundness and completeness in open world planning, and extending the standard POP algorithm to produce PSIPOP.

Representing Open Worlds

We consider the problem of open world planning where the agent does not have complete information about the world. We assume that the world evolves as a sequence of states, where the transitions occur only as the result of deliberate action taken by the single agent.

Since the agent's model of the world is incomplete, we must distinguish between the *world state* (or *state of the world*, or *situation*, in situation calculus terms (McCarthy & Hayes 1969)) and the state of the agent's knowledge of the world, which we call **SOK**. We further assume that the agent's knowledge of the world is *correct*.

While in theory the number of propositions in an SOK can be unlimited, for practical purposes it must be finite and preferably as small as possible. In many domains the number of *negated* propositions that are true in a world state is usually very large, if not infinite. We cannot use the CWA, because we must distinguish propositions that are *false* from those that are unknown. To compactly represent such negated information one solution is to use quantified formulas [1].

[1] We use the term "quantified formula" informally to refer to any formula that can represent a possibly infinite

(Etzioni, Golden, & Weld 1997) define a special class of LCW (for **Local Closed World information**) sentences, to specify the parts of SOK for which information is complete. The agent is said to have Local Closed World information relative to a logical formula Φ if the value of every ground sentence that unifies with Φ is either known to be *true* or known to be *false*. For example, LCW($PS(x) \wedge In(x, /tex)$) states that the agent knows all x's that are postscript files in directory $/tex$, i.e. given any particular x, the agent knows whether $(PS(x) \wedge In(x, /tex))$ is *true* or *false*, but is not *unknown*.

There are drawbacks in the LCW representation. LCW reasoning is incomplete and, in some cases, discards information due to its inability to represent *exceptions*, i.e. the inability to state that the agent knows the value of all instantiations of formula Φ *except* some. As the result, when a value of even one instance of Φ becomes unknown, the entire LCW sentence is no longer true and the agent must discard it, losing information about those instances that were known to be *false*.

In contrast to LCW, we define a class of formulas, called ψ**-forms**, that can represent *Locally Closed World Information with exceptions*, or what we call **Partially Closed Worlds**. We begin with an example before formally defining ψ-forms.

Example 1 Consider a ψ-form
$\psi = [\neg PS(x) \vee \neg In(x, y) \,|\, \neg(x = a.ps) \wedge \neg(x = fig)]$.
ψ represents all clauses of the form $\neg PS(x) \vee \neg In(x, /tex)$ for all values of x *except* for $\neg PS(a.ps) \vee \neg In(a.ps, /tex)$ and $\neg PS(fig) \vee \neg In(fig, /tex)$. Given the following SOK

$$s = \left\{ \begin{array}{l} PS(a.ps), In(a.ps, /tex), In(fig, /tex), \\ [\neg PS(x) \vee \neg In(x, /tex) \,|\, \neg(x = a.ps) \wedge \neg(x = fig)]. \end{array} \right.$$

the agent can conclude that it knows that there are no postscript files in $/tex$ except for the postscript file $a.ps$, and maybe the file fig, whose format is unknown.

Note, we do not have $LCW(PS(x) \wedge In(x, /tex))$ in s, because the format of fig is unknown. This example cannot be represented in LCW representation unless the domain is finite and ground facts are represented explicitly, with no LCW sentences.

ψ-forms are formulas that are used to represent possibly infinite sets of ground clauses that are obtained by instantiating what we call its *main part* in all possible ways, except for certain instantiations, listed as its *exceptions*. In Example 1, the main part of ψ is the formula $\neg PS(x) \vee \neg In(x, /tex)$, while the exceptions are $\neg(x = a.ps)$ and $\neg(x = fig)$. For the efficiency of reasoning we choose the main part of ψ-forms to be a disjunction of negated literals.

We can alternatively view ψ-forms as logical propositions by interpreting them as a conjunction of all ground clauses they represent. We use this duality and define both set-theoretic and logical relations between ψ-forms. This allows us to use them efficiently in partial order planning.

In addition to the suitability of ψ-forms for representing negative information, ψ-form reasoning has nice computational properties: it is sound, complete in what we later define as *sufficiently large* domains, and has only polynomial complexity.

Definitions

The general form of a ψ-form is:

$$\psi = [Q(\vec{x}) \,|\, \neg\sigma_1 \wedge \ldots \wedge \neg\sigma_n] \qquad (1)$$

Here, $Q(\vec{x})$ is a clause of negated literals that uses all and only the variables in $\vec{x}$, i.e., $Q(\vec{x}) = \neg Q_1(\vec{x_1}) \vee \ldots \vee \neg Q_k(\vec{x_k})$ where $k \geq 0$ and each $Q_i(\vec{x_i})$ is any atom that uses all and only the variables in $\vec{x_i}$ and $\vec{x} = \bigcup_{i=1}^{k} \vec{x_i}$. We require, that none of $Q_i(\vec{x_i})$ and $Q_j(\vec{x_i})$ unify, assuming of course $i \neq j$.[2] In addition, we require, that for every atom $Q_i(\vec{x_i})$, every variable in $\vec{x_i}$ occur no more than once in its argument list. Thus, $[\neg P(x, y) \vee \neg Q(y) \,|\, \neg(x = a)]$ is a ψ-form, while $[\neg P(x, x) \vee \neg Q(y) \,|\, \neg(x = a)]$ is not because of multiple occurances of x in $P(x, x)$.

Each σ_i represents a set of *exceptions* and is just a substitution $\vec{y_i} = \vec{e_i}$ for some non-empty vector of variables $\vec{y_i} \subseteq \vec{x}$ and some vector of constants $\vec{e_i}$. Each $\vec{y_i}$ must be the same size as its corresponding $\vec{e_i}$. k and n are, of course, finite.

We call a ψ-form with no exceptions a **simple** ψ-form. A simple ψ-form with no variables is called a **singleton** and represents a single clause. A ψ-form that uses variables is called **quantified**. Given a ψ-form (1), we define the following.

- $\mathcal{M}(\psi)$ is the **main part** of ψ, $Q(\vec{x})$.
- $\mathcal{V}(\psi)$ denotes the **variables** of ψ, $\vec{x}$, though we usually treat it as a set.
- $\Sigma(\psi)$ is the formula that describes the **exceptions**, $\neg\sigma_1 \wedge \ldots \wedge \neg\sigma_n$.
- $\Sigma_i(\psi)$ is the substitution for the i-th exception, σ_i.
- $\#\mathcal{E}(\psi)$ is the number of exceptions, n.
- $\mathcal{E}_i(\psi)$ denotes the instantiation $\mathcal{M}(\psi)\sigma_i$.

set of ground formulas

[2]This requirement is not critical, but it simplifies calculation of entailment and other operations.

For example, let

$$\psi = [\neg P(x,y) \vee \neg Q(y,A) \,|\, \neg(x=A) \wedge \neg(x=C, y=D)]$$

- $\mathcal{M}(\psi) = \neg P(x,y) \vee \neg Q(y,A)$,
- $\mathcal{V}(\psi) = \{x,y\}$,
- $\Sigma(\psi) = \neg(x=A) \wedge \neg(x=C, y=D)$,
- $\Sigma_1(\psi) = \{x=A\}$, $\Sigma_2(\psi) = \{x=C, y=D\}$,
- $\#\mathcal{E}(\psi) = 2$,
- $\mathcal{E}_1(\psi) = \neg P(A,y) \vee \neg Q(y,A)$, $\mathcal{E}_2(\psi) = \neg P(C,D) \vee \neg Q(D,A)$.

A ψ-form is **well-formed** if and only if there is no exception that is a subset of another exception, i.e. $\Sigma_i(\psi) \not\subseteq \Sigma_j(\psi)$, for all i,j where $0 \leq i,j \leq \#\mathcal{E}(\psi)$ and $i \neq j$

ψ-forms as Sets

A ψ-form is a representation of a possibly infinite set of ground clauses. A set represented by a ψ-form is called a **ψ-set**. The ψ-set corresponding to the ψ-form ψ is defined with operation ϕ. We define ϕ recursively as follows:

1. We first define ϕ for a single ψ-form.
   (a) $\phi([\,]) = \emptyset$
   (b) $\phi(\psi) = \{\mathcal{M}(\psi)\sigma \,|\, \mathcal{M}(\psi)\sigma \text{ is ground}\}$, if ψ is simple. Note that $\phi([c]) = \{c\}$, if c is a ground clause.
   (c) $\phi(\psi) = \phi([\mathcal{M}(\psi)]) - \phi([\mathcal{E}_1(\psi)]) - \ldots - \phi([\mathcal{E}_{\#\mathcal{E}(\psi)}(\psi)])$, otherwise.
2. $\phi(\{\psi_1, \ldots, \psi_k\}) = \cup_{i=1}^{k} \phi(\psi_i)$, i.e. a ψ-set of a set of ψ-forms is the union of the ψ-sets of its elements.
3. Let $\Box_1$ and $\Box_2$ denote either a single ψ-form or a set of ψ-forms. Let $*$ denote any of the set operations $\cap, \cup, -, \triangleright$ or $\dot{-}$ (last two operations are defined later)

$$\phi(\Box_1 * \Box_2) = \phi(\Box_1) * \phi(\Box_2).$$

For the simplicity of presentation, since ψ-forms and sets of ψ-forms represent sets of ground clauses, we say that a *ground clause c is in* $\Box$, written $c \in \Box$, instead of saying $c \in \phi(\Box)$. Here $\Box$ denotes either a single ψ-form or a set of ψ-forms.

Thus, given 1.and 2.

- $c \in \psi$ iff $\exists \sigma \,.\, c = \mathcal{M}(\psi)\sigma$ and $\forall \sigma, i \,.\, 1 \leq i \leq \#\mathcal{E}(\psi) \Rightarrow c \neq \mathcal{E}_i(\psi)\sigma$, and
- $c \in \Psi$, where Ψ denotes a set of ψ-forms, when there is a ψ-form ψ in the set Ψ, such that $c \in \psi$.

We say that two ψ-forms are **equivalent**, written $\psi_1 = \psi_2$ if their corresponding ψ-sets are equal, i.e. $\psi_1 = \psi_2$ if and only if $\phi(\psi_1) = \phi(\psi_2)$.

In a well-formed ψ-form there are no exceptions such that clauses it denotes are a subset of the clauses denoted by another exception, i.e., $\phi([\mathcal{E}_i(\psi)]) \not\subseteq \phi([\mathcal{E}_j(\psi)])$ for all i,j where $0 \leq i,j \leq \#\mathcal{E}(\psi)$. It is a simple matter to transform a ψ-form that is not well-formed into an equivalent one that is well-formed. Thus, from here on, we only consider well-formed ψ-forms. Note that a well-formed ψ-form representation of a ψ-set is not unique.

From now on we assume that every equation involving ψ-forms or sets of ψ-forms actually denotes an equality between the corresponding ψ-sets. I.e. when A and B are such expressions, $A = B$ is a shorthand for writing $\phi(A) = \phi(B)$.

ψ-form Logic

We can add ψ-forms to propositional logic by providing an interpretation rule for them.

An *interpretation*, $\mathcal{I}$, is an assignment of truth values to all atoms of a theory. Interpretation rules extend an interpretation by defining the truth value of every sentence of a theory, not just atoms. We adopt the standard interpretation rules of the propositional logic and add the following rule for interpreting a ψ-form or a set of ψ-forms, denoted below by $\Box$.

$$\mathcal{I}(\Box) = \bigwedge_{c \in \phi(\Box)} \mathcal{I}(c), \qquad (2)$$

i.e. a single ψ-form or a set of ψ-form is interpreted as a conjunction of all clauses in it.

Now that we have defined an interpretation for ψ-forms we can define *entailment* in a logical language containing ψ-forms in the usual way. A **model** of a logical formula is an interpretation that assigns *true* to that formula. A formula a **entails** formula b if and only if every model of a is also a model of b.

Given the definition of ϕ, we can see that for any two ψ-forms or sets of ψ-forms $\Box_1$ and $\Box_2$, $\phi(\Box_1) = \phi(\Box_1)$ iff ($\Box_1 \models \Box_2$ and $\Box_2 \models \Box_1$).

PSIPLAN Formalism

Worlds and SOKs

A world state is a set of all literals that are true in the world. We define a **domain proposition** or simply a **proposition** as either an atom or a ψ-form. Agent's state of knowledge, or **SOK** is a consistent set of domain propositions.

Representing Actions

Actions are ground and are represented in a fashion similar to STRIPS. Each action a has a *name*, $\mathcal{N}(a)$, a set of *preconditions*, $\mathcal{P}(a)$, and a set of domain literals called the *assert list*, $\mathcal{A}(a)$. Action preconditions identify the domain propositions necessary for executing the action. The propositions in $\mathcal{P}(a)$ can include literals and quantified ψ-forms.[3]

The assert list, also called the *effects* of the domain action, identifies *all and only* the domain propositions whose value may change as a result of the action. We assume that an action is deterministic and can change the truth only finite number of atoms, and thus any ψ-form in the assert list defines a single negated literal.

State Update Procedure.

Actions cause transitions between the world states. The agent's SOKs must evolve in parallel with the world, and must adequately reflect the changes in the world that occur due to an action. *Correctness* of an SOK update guarantees that the SOK is always consistent with the world model, given a consistent initial SOK. The other desirable property of the SOK update is *completeness*: we would like the agent to take advantage of all information that becomes available and not to discard what was previously known and has not changed. Clearly, the correctness and completeness properties of the SOK update, as well as soundness and completeness of entailment within the state language are prerequisites for a sound and complete planning algorithm.

We use symbols $w, w', w_1 \dots w_n$ to refer to states of the world, W, W' to refer to the sets of world states, and $s, s', s_1 \dots s_n$ to refer to the agent's SOK.

The correctness and completeness criteria are best formulated in the context of possible worlds. We define a set of **possible worlds** given a SOK s as the set $\mathcal{B}(s) = \{w \mid w \models s\}$. Let $do(a, W)$ denote the set of world states obtained from performing action a in any of the world states in W, and $update(s, a)$ denote the SOK that results if the agent performs action a from SOK s. We say that the update procedure is **correct** iff

$$\mathcal{B}(update(s, a)) \subseteq do(a, \mathcal{B}(s)) \tag{3}$$

i.e. every possible world after performing the action a has to have a possible predecessor.

[3]Ruling out other forms of non-quantified disjunction is not a limitation, since any action schema that has a non-quantified disjunction as its precondition, can be equivalently split into several actions, each of which is identical to the initial schema, but has only one of the disjuncts for the preconditions.

The update procedure is **complete** iff

$$do(a, \mathcal{B}(s)) \subseteq \mathcal{B}(update(s, a)) \tag{4}$$

i.e. every world obtained from a previously possible world is accessible from the new SOK. This implies that all changes to the world must be reflected in the new SOK.

To achieve correctness of SOK updates, the agent must remove from the SOK all propositions whose truth value might have changed as the result of the performed action.

In order to be complete , the agent must also add to the SOK all facts that become known. The complexity of the SOK update thus depends critically on the process of identifying the propositions that must be retracted to preserve correctness. In our language this computation is reduced to computing the entailment, which has polynomial complexity.

Before defining the world state transition caused by actions, we introduce the operations of *e-difference* and *image* . These operation come handy in planning, as we will show later.

For any two sets of ground propositions A and B we define **e-difference** and **image** (the *image* of A in B), respectively, as follows:

$$\begin{aligned} B \dot{-} A &= \{b \mid b \in B \ \wedge \ A \not\models b\} \\ A \triangleright B &= \{b \mid b \in B \ \wedge \ A \models b\} \end{aligned}$$

$B \dot{-} A$ is the subset of B that *is not* entailed by A while $A \triangleright B$ is the subset of B that *is* entailed by A. Thus, $(B \dot{-} A)$ and $(A \triangleright B)$ always partition B. Moreover, we have the following equivalences.

1. $B \dot{-} A \ = \ B - (A \triangleright B)$ and
2. $A \triangleright B \ = \ B - (B \dot{-} A)$

Determining image and e-difference between sets of atoms is straightforward, and so we will not discuss it. Between ψ-forms, however, it is more complicated. We defer discussing it until the Calculus Section. .

Our agent can only execute an action a if its SOK about the current state is s and $s \models \mathcal{P}(a)$. To obtain the agent's SOK s after performing an action a, we first remove all propositions implied by the negation of the assert list, as only those propositions of s might change their values after a. We also remove from the SOK all redundant propositions, i.e. those that follow from the effects of the action, and then add these effects to the new state. The agent's SOK after executing action a in the SOK s is described by the following formula.

$$update(s, a) = ((s \dot{-} \mathcal{A}^{-}(a)) \dot{-} \mathcal{A}(a)) \cup \mathcal{A}(a) \tag{5}$$

where for a set of propositions P, P^{-} denotes the set of the negations of propositions in P.

Theorem 1 The state of knowledge update procedure (5) is correct and complete.

We do not present proofs in this paper due to space limitations.

Example 2 For an example, we characterize the action $a = mv(fig, /img, /tex)$, which moves the file fig from directory $/img$ into $/tex$. We use $T(x, PS)$ to represent that file x has type postscript. Let $\mathcal{P}(a) = \{In(fig, /img)\}$ which states that fig must be in $/img$. Also, let $\mathcal{A}(a) = [\neg In(fig, /img), In(fig, /tex)]$. We begin with an SOK:

$$s = \begin{cases} In(fig, /img), In(a.tex, /tex), T(a.ps, PS), \\ [\neg In(x, d) \vee \neg T(x, PS) \,|\, \neg(x = a.ps) \wedge \neg(d = /ps)], \\ [\neg In(x, /img) \,|\, \neg(x = fig)] \end{cases}$$

$a = mv(fig, /img, /tex)$ is executable in s, and the resulting SOK is:

$$s' = \begin{cases} In(fig, /tex), In(a.tex, /tex), T(a.ps, PS), \\ [\neg In(x, /img)], [\neg In(x, d) \vee \neg T(x, PS) \,| \\ \neg(x = a.ps) \wedge \neg(d = /ps) \wedge \neg(x = fig, d = /tex)] \end{cases}$$

Note that s contained $\neg In(fig, /tex) \vee \neg T(fig, PS)$ and that we added $In(fig, /tex)$ when determining s'. If our update rule retained $\neg In(fig, /tex) \vee \neg T(fig, PS)$ in s', then in s' we could perform resolution and conclude that $\neg T(fig, PS)$. However, this would be wrong because we have no information on fig being a postscript file or not. Instead, our update rule deletes any clause that is entailed by $\neg In(fig, /tex)$, and so s' does not contain $\neg In(fig, /tex) \vee \neg T(fig, PS)$.

The Planning Problem.

As usual in a planning problem we are given a set of initial conditions $\mathcal{I}$, a set of goals $\mathcal{G}$ and a set of available actions A. $\mathcal{I}$ and are $\mathcal{G}$ sets of domain propositions.

A solution plan is a sequence of actions, that is executable and transforms any state satisfying the initial conditions into a state satisfying the goal. Given a sequence of actions $a_1, \ldots, a_n$, let W_i denote the set of worlds $do(..(do(do(\mathcal{B}(\mathcal{I}), a_1), a_2), \ldots) a_i)$. Then, a sequence of actions $a_1, \ldots, a_n$ is called a **solution plan**, if

1. for all w in W_n, $w \models \mathcal{G}$, and

2. for all values of i, $0 \leq i < n$, and for all w in W_i $w \models \mathcal{P}(a_{(i+1)})$.

A planner is called **sound** if and only if it returns only solution plans, and **complete**, if it returns every solution plan.

Let s_i denote the SOK $update(update(\ldots(update(s_0, a_1), \ldots) a_i))$. We call a sequence $a_1, \ldots, a_n$ a **solution in the agent's theory** if and only if

1. for each i , $0 \leq i < n$, $s_i \models \mathcal{P}(a_{i+1})$, and

2. $s_n \models \mathcal{G}$.

Our planner always returns a solution in the agent's theory, which is guaranteed to be a solution plan, given that the initial SOK s_0 is equal to $\{p \,|\, p \in \mathcal{I}\}$ and the agent's update function is correct.

PSIPOP Algorithm

Figure 1 shows the PSIPOP algorithm as a modified POP algorithm written for a non-deterministic machine. We assume the reader is already familiar with SNLP-style planning (McAllester & Rosenblitt 1991). We made a few changes to the standard algorithm so that it easily generalizes to handling ψ-forms. These changes arise when ψ-forms are added to the state and action description language. Since a link between two ψ-forms actually represents a multitude of links between ground clauses of the source and target ψ-forms, we need to introduce new techniques of establishing and protecting such links. These techniques are based on the theory of ψ-form entailment, which we briefly describe in the Calculus section.

There are three important changes to the POP.
Change 1. Causal links now have both source and target conditions, which may differ. The source condition must entail the target condition. For example, we may have step S_1 with effect ψ_1 and step S_2 with precondition ψ_2 where

$$\psi_1 = [\neg In(x, /psdir) \vee \neg T(x, y) \,|\, \neg(y = PS)]$$
$$\psi_2 = [\neg In(x, /psdir) \vee \neg T(x, TEX) \vee \neg O(x, Joe)]$$

ψ_1 states that are no files in directory */psdir except* Postscript files. ψ_2 requires that there are no files of type TEX in $/psdir$ owned by Joe. Clearly, $\psi_1 \models \psi_2$ and so we can have a causal link from ψ_1 on S_1 to ψ_2 on S_2.

Thus, causal links between ψ-forms actually represent a set of causal links between each clause that is supported and the clause that support it. This change is reflected in step 2 of PSIPOP.
Change 2. In cases similar to the above where $\psi_1 \not\models \psi_2$ but where ψ_1 *nearly entails* ψ_2, we try *splitting* the goal ψ_2. ψ_1 **nearly entails** ψ_2 iff $[\mathcal{M}(\psi_1)] \models [\mathcal{M}(\psi_2)]$. In such cases, we *split* ψ_2 into two parts:

- The precise portion of ψ_2 that *is entailed* by ψ_1—this is the *image* of ψ_1 in ψ_2, $\psi_1 \triangleright \psi_2$—and

- The remainder of ψ_2—this is precisely $\psi_2 \dot{-} \psi_1$.

Algorithm. PSIPOP-S ($<S, O, L>$, *open*)

1. If *open* is empty, return $<S, O, L>$
2. Pick a goal $< c, S_c >$ from *open* and remove it from *open*. **choose** an existing step S_s from S, or a new step S_s, that has an effect e where $e \models c$ or e *nearly entails* c (if nearly entails, then *Split Goal (e,c)*, goto 4).

   If no such step exists then **fail**.
3. Add link $S_s \overset{e,c}{\rightarrow} S_c$ to L.
4. Add $S_s \prec S_c$ to O.
5. if S_s is a new step:
   - Add START$\prec S_s$ and $S_s \prec$FINISH to L.
   - For each p in $\mathcal{P}(S_s)$ (the preconditions of S_s), add $< p, S_s >$ to *open*.
6. For every step S_t that threatens a link $S_s \overset{e,c}{\rightarrow} S_c$ nondeterministically **choose** either:
   - *Demotion:* Add $S_t \prec S_s$ to O.
   - *Promotion:* Add $S_c \prec S_t$ to O.
   - *Split Link(e,c).*
7. If O is inconsistent then **fail**.
8. Recursively call POP with updated $<S, O, L>$ and *open*.

Triple $< S, O, L >$ denotes a partial plan; S is a set of *steps*, which are (ground) actions, initially contains only START and FINISH; O is a set of *ordering constraints* of the form $S_i \prec S_j$, where S_i and S_j are steps in S, initially contains START$\prec$FINISH; L is a set of (causal) *links* of form $S_i \overset{e,p}{\rightarrow} S_j$, where p is a precondition of S_j, e is an effect of S_i (i.e., e is in the assert list of S_i), and $e \models p$. We call S_i and e the *source* step and proposition, and S_j and p the *target* step and proposition.L is initially empty. *open* is the list of open preconditions and initially contains preconditions of the FINISH step.
We assume that all resolutions have been performed in the effects of the initial step – START.

Figure 1: Modified POP algorithm

Once split, we add a causal link from ψ_1 on S_1 to $(\psi_1 \triangleright \psi_2)$ on S_2, and we are left with $(\psi_2 \dot{-} \psi_1)$ on S_2 that still needs to be linked. Fortunately, calculating both image and e-difference is straightforward and results in a set of ψ-forms, each of which are strictly smaller than the original ψ-form goal in a well founded way.

Split Goal(ψ_e, ψ_c): Perform when
– ψ_e, ψ_c are ψ-forms and
– ψ_e nearly entails ψ_c – i.e.,
$[\mathcal{M}(\psi_e)] \models [\mathcal{M}(\psi_c)]$ but $\psi_e \not\models \psi_c$.

1. Partition ψ_c into $\psi_c^1 = \psi_e \triangleright \psi_c$ and $(\psi_c \dot{-} \psi_e)$.
2. Add $S_s \overset{\psi_e, \psi_c^1}{\rightarrow} S_c$ to L.
3. For each ground clause $c \in (\psi_c \dot{-} \psi_e)$, add $< c, S_c >$ to *open*.

Figure 2: Split Goal

Split Link(ψ_e, ψ_c):Perform when
– effect A on S_t threatens $S_s \overset{\psi_e, \psi_c}{\rightarrow} S_c$ – i.e. $([\neg A] \triangleright \psi_e) \triangleright \psi_c \neq \emptyset$.

1. Add $S_s \prec S_t$ and $S_t \prec S_c$ to O.
2. Partition ψ_e into $[\neg A] \triangleright \psi_e$ and $\psi_e^1 = \psi_e \dot{-} [\neg A]$.
3. Partition ψ_c into $([\neg A] \triangleright \psi_e) \triangleright \psi_c$ and $\psi_c^1 = \psi_c \dot{-} ([\neg A] \triangleright \psi_e)$.
4. Remove original link $S_s \overset{\psi_e, \psi_c}{\rightarrow} S_c$ from L.
5. Add $S_s \overset{\psi_e^1, \psi_c^1}{\rightarrow} S_c$ to L.
6. For each ground clause $c \in (([\neg A] \triangleright \psi_e) \triangleright \psi_c)$, add $< c, S_c >$ to *open*.

Figure 3: Split Link

For an example, assume again that we have ψ_1 as an effect on step S_1 and ψ_2 as a precondition on step S_2 where

$$\psi_1 = [\neg In(x, /psdir) \vee \neg T(x, y) \mid \neg(y = PS)]$$
$$\psi_2 = [\neg In(x, /psdir) \vee \neg T(x, y) \vee \neg O(x, Joe)]$$

ψ_1 is the same as above. ψ_2 requires that there be no files of any type in */psdir* owned by *Joe*. Clearly, $\psi_1 \not\models \psi_2$ but ψ_1 nearly entailed ψ_2. We split ψ_2 into:

- $\psi_2^1 = \psi_1 \triangleright \psi_2 = [\neg In(x, /psdir) \vee \neg T(x, y) \vee \neg O(x, Joe) \mid \neg(y = PS)]$, which are all the files in */psdir* owned by *Joe* except for Postscript files, and
- $\psi_2^2 = \psi_2 \dot{-} \psi_1 = [\neg In(x, /psdir) \vee \neg T(x, PS) \vee \neg O(x, Joe)]$, which are all the Postscript files in */psdir* owned by *Joe*.

Next, we add a causal link from ψ_1 on S_1 to ψ_2^1 on S_2 and we are left with ψ_2^2 unsupported. Thus, much of ψ_2 is now supported except for ψ_2^2.

This is captured as the Split Goal procedure in Figure 2.

Change 3. The final change to POP adds a new way to resolve threats. A threat is any effect of an action, that results in the removal of the source condition in the SOK during the update (see 5). In our formalism, only a ground atom can threaten a link between ψ-forms, and conversely, only a ψ-form can threaten a link between ground atoms. In the former case, we add a new threat resolution method called *link splitting*.

For an atom to threaten a link, it must remove from the source ψ-form proposition(s) that support some proposition(s) in the target ψ-form. More formally, atom A may pose a threat to the link from ψ_1 to ψ_2 if $([\neg A] \triangleright \psi_1) \triangleright \psi_2 \neq \emptyset$.

Let there be a link from effect ψ_1 on step S_1 to precondition ψ_2 on step S_2. Moreover, let atom A be an effect of step S where A threatens the link. We will refer to a ψ-form constructed out of the negation of A, namely, $\psi_A = [\neg A]$, which is a singleton set.

The threat resolution method does the following.

- ψ_1 on step S_1 is replaced by $\psi_1^1 = \psi_1 \dot{-} \psi_A$ and $\psi_1^2 = \psi_A \triangleright \psi_1$. Note that ψ_1^1 is precisely the subset of ψ_1 that is *not* threatened by A and that ψ_1^2 is the residual of ψ_1.
- ψ_2 is replaced by $\psi_2^1 = \psi_1^1 \triangleright \psi_2$ and $\psi_2^2 = \psi_2 \dot{-} \psi_1^1$. Note that ψ_2^1 is precisely the subset of ψ_2 that is now supported by ψ_1^1 and that ψ_2^2 is precisely the residual of ψ_2.
- The original link from ψ_1 to ψ_2 is replaced by a link from ψ_1^1 to ψ_2^1. Condition A on S no longer is a threat.
- New support must be found for ψ_2^2 on step S_2.

This is presented as the Split Link procedure in Figure 3.

Theorem 2 PSIPOP is sound and complete.

Calculus

In this section we sketch how to determine entailment, image and e-difference. These calculations are somewhat complex and we do not have space to present them fully. A complete description can be found in (Babaian & Schmolze 1999). For the reader who is not interested in these methods, this section can be skipped.

Everywhere below we make a *sufficiently large domain assumption*, i.e. that the object domain contains more objects that are mentioned in all of the participating ψ-forms. Certainly, a domain that is only partially known is sufficiently large.

Determining Entailment

Domain ψ-forms. The critical factor in keeping ψ-form reasoning tractable is given by the following Theorem, that states, essentially, that we do not have to examine combinations of ψ-forms when checking ψ-form entailment.

Theorem 3 Let $\psi_1, \ldots, \psi_n$ be simple ψ-forms and let ψ be an arbitrary ψ-form. $\{\psi_1, \ldots, \psi_n\} \models \psi$ iff $\exists i \,.\, (1 \leq i \leq n) \wedge (\psi_i \models [\mathcal{M}(\psi)])$.

Thus, if we ignore exceptions, then to show that a set of ψ-forms entails ψ, we need to find only one ψ-form in the set that entails $[\mathcal{M}(\psi)]$.

Checking if $\psi_i \models [\mathcal{M}(\psi)]$ is, as it turns out from the next Theorem, is just a matter of finding a subset-match between the the main parts, as both ψ-forms are simple. At the heart of this result is the observation that given two ground clauses, C_1 and C_2, $C_1 \models C_2$ iff $C_1 \subseteq C_2$. Here, we are treating each ground clause as a set of literals.

Theorem 4 Given two simple ψ-forms, ψ_1 and ψ_2, $\psi_1 \models \psi_2$ iff there exists a unifier, σ such that $\mathcal{M}(\psi_1)\sigma \subseteq \mathcal{M}(\psi_2)$.

Note that there can be more than one way a clause can subset-match onto another clause. For example, matching $\neg P(x)$ onto $\neg P(a) \vee \neg P(b) \vee \neg Q(y)$ produces two different substitutions: $(x = a)$ and $(x = b)$.

The next Theorem presents necessary and sufficient conditions for entailment between ψ-forms.

Theorem 5 Let $\psi_1, \ldots, \psi_n$ and ψ be arbitrary ψ-forms. $\{\psi_1, \ldots, \psi_n\} \models \psi$ iff there exists a k,$1 \leq k \leq n$, such that:

- $[\mathcal{M}(\psi_k)] \models [\mathcal{M}(\psi)]$ (i.e., the main part of ψ_k entails $[\mathcal{M}(\psi)]$), and
- $\{\psi_1, \ldots, \psi_{k-1}, \psi_{k+1}, \ldots, \psi_n\} \models \psi \dot{-} \psi_k$.

The last requirement in Theorem 5 requires some explanation. While $[\mathcal{M}(\psi_k)] \models [\mathcal{M}(\psi)]$, the exceptions of ψ_k weaken ψ_k. Thus, each clause in $\psi \dot{-} \psi_k$ must be entailed by some other ψ-form in $\{\psi_1, \ldots, \psi_{k-1}, \psi_{k+1}, \ldots, \psi_n\}$.

We discuss methods of computing the e-difference in a later section. Here we would just like to notice that entailment in PSIPLAN is easily decided and has a time complexity that is polynomial in the number of propositions, maximum number of exceptions and the maximum number of variables used in a ψ-form.

Note also the following simple facts.

Given two ground atoms A and B, A entails B, written $A \models B$, iff $A = B$.

Given an atom A and ψ-form ψ, A can never entail ψ and ψ can never entail A.

Determining Image and E-Difference Among ψ-forms

The image and difference operators between ψ-forms, in general, are complex. Important for the algorithms presented in this paper are the facts that both operations produce sets of ψ-forms, the time complexity of ψ-form image and e-difference is polynomial the maximum number of variables used in a ψ-form and the maximum number of exceptions. Here we only state the key results to provide the reader with some intuition about ψ-form calculus. The full account of this calculus can be found in (Babaian & Schmolze 1999).

We define $MGU(A, B, V)$ as the most general unifier of A and B using only the variables in V. I.e., For $\sigma = MGU(A, B, V)$, $A\sigma = B\sigma$ and σ is a most general such unifier. In a similar fashion, we define $MGU_{\subseteq}(A, B, V)$ as the set of all σ such that $A\sigma \subseteq B\sigma$ and σ is a most general such unifier. $MGU(A, B)$ and $MGU_{\subseteq}(A, B)$ are defined similarly, except they do not restrict the bindings to any particular set of variables.

Theorem 6 For any ψ_1, ψ_2 - simple ψ-forms $\psi_1 \triangleright \psi_2 = \{[\mathcal{M}(\psi_2)\sigma] \mid \sigma \in MGU_{\subseteq}(\mathcal{M}(\psi_1), \mathcal{M}(\psi_2))\}$

So computing the image of one simple ψ-form in another simple ψ-form amounts to finding either a unifier, or a set of subset unifiers of their main parts and appropriately instantiating the main part of the second ψ-form.

As we will see shortly, to compute e-difference we first perform the set or subset unification procedure and then add the resulting substitution(s) to the set of exceptions of the second ψ-form. Before adding those substitutions, we preprocess them to conform to the syntax of ψ-forms, i.e. we remove all bindings on variables in $\mathcal{V}(\psi_1)$, add a $\neg$ sign in front of each σ. That is the purpose of the Σ' in the next Theorem.

Theorem 7 For any ψ_1, ψ_2 -ψ-forms such that ψ_1 and is simple

$$\psi_2 \dot{-} \psi_1 = \begin{cases} \emptyset, & \text{if } \psi_1 \models \psi_2, \\ \{[\mathcal{M}(\psi_2) \mid \Sigma(\psi_2) \vee \Sigma']\} & \text{otherwise.} \end{cases}$$

where $\Sigma' = \{\neg\sigma' \mid \sigma \in MGU_{\subseteq}(\mathcal{M}(\psi_1), \mathcal{M}(\psi_2))$ and $\sigma' = MGU_{\subseteq}(\mathcal{M}(\psi_1), \mathcal{M}(\psi_2)\sigma, \mathcal{V}(\psi_2))$.

Note, that the first case is actually subsumed by the second, because if $\psi_1 \models \psi_2$, $MGU_{\subseteq}(\mathcal{M}(\psi_1), \mathcal{M}(\psi_2))$ contains the substitution σ which subset matches $\mathcal{M}(\psi_1)$ onto $\mathcal{M}(\psi_2)$, i.e. σ only uses variables from $\mathcal{V}(\psi_1)$. In such case the added exception $\sigma' = \neg\emptyset$ denotes the whole $[\mathcal{M}(\psi_2)]$, thus leaving the resulting ψ-form equal to $\emptyset$.

Also, when $MGU_{\subseteq}(\mathcal{M}(\psi_1), \mathcal{M}(\psi_2))$ is empty, Σ' is empty and $\psi_2 \dot{-} \psi_1 = \psi_2$.

The image and e-difference computations in the general case are reduced to computing the operations on the simple ψ-forms. We do not present it here, but demonstrate it in the following example.

Example 3 Let $\psi_i = [\neg In(f, d) \mid \neg(d = /tmp) \wedge \neg(f = a.ps, d = /tex) \wedge \neg(f = b.ps)]$, $\psi_g = [\neg In(f_g, d_g) \vee \neg T(f_g, PS) \mid \neg(d_g = /ps)]$. f, d, f_g, d_g denote variables.

The main part of ψ_i implies the main part of ψ_g, and $MGU_{\subseteq}(\mathcal{M}(\psi_i), \mathcal{M}(\psi_g), \mathcal{V}(\psi_i)) = \{\{f = f_g, d = d_g\}\}$To calculate the difference $\psi_g \dot{-} \psi_i$ we first find the images of ψ_i exceptions, $\mathcal{E}_1(\psi_i) = [\neg In(f, /tmp)], \mathcal{E}_2(\psi_i) = [\neg In(a.ps, /tex)], \mathcal{E}_3(\psi_i) = [\neg In(b.ps, d)]$, on $\mathcal{M}(\psi_g)$.

$$\begin{aligned} [\mathcal{E}_1(\psi_i)] \triangleright [\mathcal{M}(\psi_g)] &= [\neg In(f_g, /tmp) \vee \neg T(f_g, PS)] \\ [\mathcal{E}_2(\psi_i)] \triangleright [\mathcal{M}(\psi_g)] &= [\neg In(a.ps, /tex) \vee \neg T(a.ps, PS)] \\ [\mathcal{E}_3(\psi_i)] \triangleright [\mathcal{M}(\psi_g)] &= [\neg In(b.ps, d_g) \vee \neg T(b.ps, PS)] \end{aligned}$$

The clauses of ψ_g that aren't entailed by ψ_i are exactly those that are entailed by ψ_i's exceptions and are not themselves exceptions of ψ_g, i.e.

$$\psi_g \dot{-} \psi_i = \begin{cases} [\neg In(f, /tmp) \vee \neg T(f, PS)], \\ [\neg In(a.ps, /tmp) \vee \neg T(a.ps, PS)], \\ [\neg In(b.ps, d) \vee \neg T(b.ps, PS) \mid \neg(d = /ps)] \end{cases}$$

Related Work

We have already briefly discussed the work of (Golden, Etzioni, & Weld 1994; Etzioni, Golden, & Weld 1997; Golden 1998) in the introduction. PUCCINI **??** has a richer action and goal languages, handles sensing actions and execution. But due to the incompleteness of the LCW reasoning, it is incomplete even when it does no sensing.

Conformant Graphplan (Smith & Weld 1998) and SGP (Weld & Anderson 1998) are propositional open world planners that consider every possible world and thus rely on the domain of objects being sufficiently small.

In another related work, Levy (Levy 1996) presents a method for *answer-completeness*, which determines whether the result of a database query is correct when the underlying database is incomplete. For this work, the database is relational and incompleteness means that it may be missing tuples. The incompleteness is expressed with LC constraints, which are based on LCWs but which are considerably more expressive. In fact, Levy's LCs can easily represent the information in LCWs with exceptions, which is roughly the expressive power of ψ-forms. (Friedman & Weld 1997) expands on (Levy 1996) with a richer set of LC constraints and an algorithm that determines whether a given information-gathering plan subsumes another.

Both of these works address the answering of queries in an unchanging database. They do not,however, address a changing world, much less the planning of an agent to change it.Also, PSIPLAN needs several operations—e.g., entailment, e-difference and image—for planning that do not arise when only answering queries.

Conclusions and Future Work

We have presented PSIPOP, a sound and complete partial order planning algorithm that does not make the closed world assumption and that can represent partially closed worlds. The key idea is the use of ψ-forms to represent quantified negative information and to integrate ψ-forms into POP such that it adds only polynomial cost algorithms. We thus argue informally that, even with our expanded representation, we can keep the complexity of planning within NP if we have a finite language and if we bound the length of plans. We have developed an extended formalism and planning system PSIPLAN-S, that uses an extension of the PSIPOP algorithm to handle information goals, sensing actions, information loss, conditional effects and execution. The system has been implemented in Common Lisp and has performed successfully on numerous examples. The extension of the presented algorithm to a lifted version with conditional planning and execution is straightforward.

Future work is already in progress. We will continue to explore richer representations for ψ-forms. We will also incorporate into the formalism actions that both sense and change the world. We are investigating methods for efficient integration sensing, conditional planning and plan execution. We will also examine application of PSIPLAN to SAT planning (Kautz & Selman 1996).

References

Babaian, T., and Schmolze, J. G. 1999. PSIPOP: Planning with sensing over partially closed worlds . Extended version of the paper that appears in the Notes for 1999 AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information. Available from http://www.eecs.tufts.edu/ tbabaian.

Etzioni, O.; Hanks, S.; Weld, D.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An Approach to Planning with Incomplete Information. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning (KR-92)*, 115–125.

Etzioni, O.; Golden, K.; and Weld, D. 1997. Sound and efficient closed-world reasoning for planning. *Artificial Intelligence* 89(1–2):113–148.

Friedman, M., and Weld, D. S. 1997. Efficiently Executing Information-Gathering Plans. In *Proceedings of the Fifthteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*.

Golden, K.; Etzioni, O.; and Weld, D. 1994. Omnipotence Without Omniscience: Efficient Sensor Management for Planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, 1048–1054. Seattle, WA: American Association for Artificial Intelligence.

Golden, K. 1998. Leap Before You Look: Information Gathering in the PUCCINI Planner. In *Artificial Intelligence Planning Systems: Proceedings of the Fourth International Conference (AIPS-98)*, 70–77.

Kautz, H., and Selman, B. 1996. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 1194–1201. Portland, OR: American Association for Artificial Intelligence.

Krebsbach, K.; Olawsky, D.; and Gini, M. 1992. An Empiracal Study of Sensing and Defaulting in Planning. In *Artificial Intelligence Planning Systems: Proceedings of the First International Conference (AIPS-92)*, 136–144.

Levy, A. 1996. Obtaining Complete Answers from Incomplete Databases. In *Proceedings of the 22nd VLDB Conference*.

McAllester, D., and Rosenblitt, D. 1991. Systematic Nonlinear Planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, 634–639.

McCarthy, J., and Hayes, P. J. 1969. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In Meltzer, B., and Michie, D., eds., *Machine intelligence 4*. New York: American Elsevier.

Peot, M. A., and Smith, D. E. 1992. Conditional Nonlinear Planning. In *Artificial Intelligence Planning Systems: Proceedings of the First International Conference (AIPS-92)*, 189–197.

Scherl, R. B., and Levesque, H. J. 1997. The Frame Problem and Knowledge-Producing Actions. Submitted to *Artificial Intelligence.*.

Smith, D., and Weld, D. S. 1998. Conformant Graphplan. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*.

Weld, D. S., and Anderson, C. R. 1998. Extending Graphplan to Handle Uncertainty and Sensing Actions. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*.