PSIPOP: Planning with Sensing over Partially Closed Worlds

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

Abstract

We present a new partial order planner called PSIPOP, 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. Note-worthy are the facts that lack of information is can be represented precisely and all quantified reasoning has polynomial complexity. Thus, in finite domains where the maximum plan length is bounded, PSIPOP is NP-complete.

Several researchers have examined planning with sensing in an open world, where the agent must take action 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)). But how does the agent represent a partially closed world (PCW)—i.e., the fact that either (A) it knows everything about a particular question, or (B) it knows precisely what it does not know about that question. For example, how does it represent that “(1) a.ps and b.ps are the only postscript files in directory /tex” (an example of A) or that “(2) c.ps is in /home and d.ps may or may not be in /home, but there are no other postscript files in /home” (an example of B). Such representations are needed for numerous actions as:

- **preconditions**: E.g., when removing a directory in Unix, the directory must be empty.¹
- **effects** of sensing new information: E.g., when performing a Unix “ls”, one learns something about all files, namely, those in the the directory and those not in it. In contrast, SADL (Golden & Weld 1996) uses conditional effects to represent the effects of sensing.

Some works (e.g., (Scherl & Levesque 1997)) use first-order logic (FOL), which easily represents PCWs but which appears to preclude practical planning algorithms due to the undecidability of entailment in FOL. The other works build upon the partial order planning (POP) work of SNLP (McAllester & Rosenblitt 1991) and yield NP-complete planning algorithms when there are a finite number of ground atoms and where the maximum plan length is bounded (Erol, Nau, & Subrahmanian 1992). While NP-complete problems are still intractable in general, applications of stochastic search to propositional satisfiability have made solving many such problems considerably more plausible (Kautz & Selman 1996; Kautz, McAllester, & Selman 1996).

Closely related are locally closed worlds (LCWs) (Golden, Etzioni, & Weld 1994; Etzioni, Golden, & Weld 1997), which are used in the PUCCINI planner (Golden 1998) and others. LCWs allow the agent to represent that it knows everything about a given conjunction of atoms. For example, LCW(PS(x) ∧ In(x, /tex)) states that the agent knows about all x’s that are postscript files and that are in the directory /tex. Coupled with a set of ground atoms that represent all the positive knowledge the agent possesses, the agent can easily determine whether any given file is a postscript file in /tex: test whether or not it is in the set of atoms. However, the LCW framework cannot represent example 2 from above because the agent does not know about all postscript files in /home—i.e., it does not know whether or not d.ps is in /home. In other words, it can represent locally closed world but not what we call partially closed worlds.

This weakness can easily be overcome by the use of exceptions—i.e., represent that the agent knows about all postscript files in /home except for d.ps. However,
We use $\psi$-forms to represent certain sets of propositions. There are two types: a $\psi$-form can represent a set of domain clauses, or a $\psi$-form can represent a set of knowledge propositions that are themselves about domain clauses. Here is the first type.

$$\psi = \{Q(\vec{x}) | \neg \sigma_1 \land \ldots \land \neg \sigma_n\}$$

Here, $Q(\vec{x})$ is a clause of negated literals that uses all and only the variables in $\vec{x}$, i.e., $Q(\vec{x}) = \neg Q_1(x_1) \lor \ldots \lor \neg Q_k(x_k)$ where each $Q_i(x_i)$ is any atom that uses all and only the variables in $x_i$ and $\vec{x} = \bigcup_{i=1}^{k} x_i$. Each $\sigma_i$ represents a set of exceptions and is just $\vec{y}_i = \vec{e}_i$ 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.

$\psi$ denotes all the ground instantiations of $Q(\vec{x})$ minus all the ground instantiations represented by the exceptions.

$$\phi(\psi) = (\phi(Q(\vec{x})) - \phi(Q(\vec{x}) \sigma_1) - \ldots - \phi(Q(\vec{x}) \sigma_n))$$

where $\phi(A)$ to represent all ground instantiations of $A$, i.e., $\phi(A) = \{A \sigma | A \sigma \text{ is ground}\}$.

For a given $\psi$, we call $Q(\vec{x})$ the main part of $\psi$, which we refer to as $M(\psi)$. The exceptions, $\neg \sigma_1 \land \ldots \land \neg \sigma_n$, are written as $\Sigma(\psi)$, and the number of exceptions, $n$, is written as $\#E(\psi)$. We write $E_i(\psi)$ to refer to the set of propositions represented by the $i$-th exception, which is $\phi(M(\psi)) \sigma_i$. Note that this is not necessarily a singleton set. Finally, we refer to the set of all exceptions with $E(\psi)$, which is $\bigcup_{i=1}^{n} E_i(\psi)$.

A $\psi$-form of the second type is exactly the same as the first type, except that its main part is of the form $KW(Q(\vec{x}))$, i.e., it represents a set of knowledge propositions where the associated domain propositions are all ground clauses of negated literals.

A $\psi$-form is well-formed if there is no exception such that the clauses it denotes are a subset of the clauses denoted by another exception, i.e., $E_i(\psi) \not\subseteq E_j(\psi)$ for all $i$ and $j$ where $1 \leq i, j \leq n$. It is a simple matter to transform a $\psi$-form that is not well-formed into an equivalent one that is well-formed. Thus, from here on, we only consider well-formed $\psi$-forms.

A singleton $\psi$-form is one that denotes a single formula. A negated literal is a singleton $\psi$-form whose main formula is a domain proposition that has exactly one disjunct. Moreover, the interpretation of a set of propositions, or of a set of sets of propositions, is just the conjunction of all propositions it includes.

**Difference and Image**

For any two sets of ground propositions $A$ and $B$ we define difference: $B - A = \{b | b \in B \land A \not= b\}$, image (the image of $A$ in $B$): $A \triangleright B = \{b | b \in B \land A \vdash b\}$. $B - A$ 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, \((B - A)\) and \((A \triangleright B)\) always partition \(B\). Moreover, we have the following equivalences.

1. \(B - A = B - (A \triangleright B)\) and
2. \(A \triangleright B = B - (B - A)\)

**States and Actions**

Let a *state of knowledge* (SOK) be a consistent set of domain propositions plus a consistent and complete set of knowledge propositions. In an SOK, the truth value of a domain atom may be unknown, but the truth value of \(KW(P)\) for every domain proposition \(P\) is known. An SOK represents the knowledge that our (single) agent has about a particular world state. Our agent, and thus our planner, never has access to complete world states, but only to SOKs. We assume throughout this paper that an agent's SOK is always correct.

An *action* is a ground and represented in a fashion similar to STRIPS. Each action \(a\) has a name \(N(a)\), a set of domain propositions called the *preconditions*, \(P(a)\) (which may include non-singleton \(\psi\)-forms), a set of domain literals called the *assert list*, \(A(a)\) (i.e., all \(\psi\)-forms here must be negated literals only), and a set of knowledge propositions called the *knowledge list*, \(K(a)\) (which may include non-singleton \(\psi\)-forms).

The precondition identifies the domain conditions necessary for executing the action. The assert list, also called the *effects* of the action, identifies all and only the domain propositions that change as a result of the action. The knowledge list is the set of propositions for which the agent will acquire the truth value as a result of the action, but whose truth value will not change as a result of the action. For this paper, we assume that every action is either a *domain action*, which has an empty knowledge list, or a *sensing action*, which has an empty assert list. We also assume there are no knowledge preconditions and no actions that lose information.

Our agent can only execute an action \(a\) if its SOK about the current state is \(S\) and \(S \models P(a)\). When the agent executes an action \(a\), the resulting SOK is determined in two steps. We first calculate \(S^*\), which we call the *base* of the resulting SOK. Then we calculate the resulting SOK, \(S'\). For a *domain action*:

\[
S^*(S, A(a)) = (S - A(a)) \cup \neg A(a)
\]

We define \(S^*\) be the closure of \(S\). For either an atom \(A\) or a negated literal \(\neg A\), the closure is the set \(\{A, \neg A\}\).

For a set of propositions, the closure is the union of the closures of its members. We first remove the closure of the assert list so that we can add the assert list back without conflict. For a *sensing* action: \(S^* = S \cup \Delta\) where \(\Delta\) is the set of propositions that are actually learned from executing \(a\). In both cases, the resulting SOK is simply the completion of \(S^*\), written \(\prod(S^*)\), which is the set of propositions entailed by \(S^*\), i.e.,

\[
S' = \prod(S^*) = \{s \mid S^* \models s\}
\]

Computing all entailments is fairly simple. Computing \(KW(P)\) only requires that we test whether a given set of propositions either entails \(P\) or entails \(\neg P\). In fact, we never need to store physically any \(KW\) propositions. The only entailments that we calculate in advance and cache are resolutions that arise from domain \(\psi\)-forms combined with atoms. Fortunately, the number of such resolutions is limited by the product of the number of \(\psi\)-forms and atoms, both of which are finite. Moreover, we need only perform resolution on the effects of actions that add domain \(\psi\)-forms, which are sensing actions and the START action.

For an example, we characterize the action \(a_D = \text{mv}(\text{fig}, /\text{img}, /\text{tex})\), which moves the file \text{fig} from directory /\text{img} into /\text{tex}. We use \(PS(x)\) to represent that file \(x\) is in postscript format. Let \(P(a_D) = \{\text{In}(\text{fig}, /\text{img})\}\) which states that \text{fig} must be in /\text{img}. Also, let \(A(a_D) = \{\neg\text{In}(\text{fig}, /\text{img}), \text{In}(\text{fig}, /\text{tex})\}\). \(K(a_D)\) is empty. We begin with the base of our initial SOK:

\[
S_0 = \left\{ \text{In}(\text{aps}, /\text{tex}), \text{In}(\text{fig}, /\text{img}), \text{PS}(\text{aps}), \{\neg\text{In}(x, /\text{img}) \lor -\text{PS}(x) \mid -(x = \text{fig})\}, \{\text{In}(x, /\text{tex}) \lor -\text{PS}(x) \mid -(x = \text{aps})\} \right\}
\]

Our initial SOK is just \(S_0 = \prod(S_0^*)\). The only entailments here are the obvious ones regarding \(KW\) formulas.

\(a_D = \text{mv}(\text{fig}, /\text{img}, /\text{tex})\) is executable from \(S_0\), and the base of the resulting SOK is:

\[
S_1^* = \left\{ \text{In}(\text{fig}, /\text{tex}), \text{In}(\text{aps}, /\text{tex}), \text{PS}(\text{aps}), \{\neg\text{In}(x, /\text{img})\}, \{\neg\text{In}(x, /\text{tex}) \lor -\text{PS}(x) \mid -(x = \text{aps})\} \right\}
\]

and \(S_1 = \prod(S_1^*)\).

Note that \(S_0\) contained \(\neg\text{In}(\text{fig}, /\text{tex})\lor -\text{PS}(\text{fig})\) and that we added \(\text{In}(\text{fig}, /\text{tex})\) when determining \(S_1\). If our update rule retained \(\neg\text{In}(\text{fig}, /\text{tex})\lor -\text{PS}(\text{fig})\) in \(S_1\), then in \(S_1\) we could perform resolution and conclude that \(-\text{PS}(\text{fig})\). However, this would be wrong because we have no information on \text{fig} being a postscript file or not. Instead, our update rule deletes any clause that is entailed by \(\neg\text{In}(\text{fig}, /\text{tex})\) thanks to our special definition of difference, and so \(S_1\) does not contain \(\neg\text{In}(\text{fig}, /\text{tex})\lor -\text{PS}(\text{fig})\).

There is one exception to the rule disallowing quantified \(\psi\)-forms in assert lists. We allow them in the assert list of the START action, which describes the base of the initial SOK for a planning problem.

**\(\psi\)-form Entailment**

We briefly review the techniques for determining entailment for sets of \(\psi\)-forms. Essentially, these tests
are translated into operations of unification, instantiation, identity, and simple set operations. The full theory of $\psi$-form entailment will be presented in another paper. Entailment in the class of less expressive $\psi$-forms is presented in (Babaian & Schmolze 1998).

The critical factor in keeping the $\psi$-form reasoning tractable is that for a set of $\psi$-forms $\{\psi_1, \ldots, \psi_n\}$ to entail another $\psi$-form $\psi$, there must be a $\psi$-form in the set whose main part entails the main part of $\psi$, so there’s no need to examine combinations of $\psi$-forms when determining entailment.

The properties of $\psi$-forms rely on the following simple observations about ground clauses.

A clause can be considered simply as a set of disjuncts. For two clauses $c_1$ and $c_2$ we write $c_1 \subseteq c_2$ iff the set of disjuncts of $c_1$ is a subset of the set of disjuncts of $c_2$. Therefore, for any two ground clauses $c_1$ and $c_2$, $c_1 \subseteq c_2$ iff $c_1 \subseteq c_2$. In the presence of variables in the clauses, entailment becomes a matter of finding a substitution that satisfies the above condition.

Given a set of clauses $C$ and a single clause $c_1$, $C \models c_1$ iff $\exists \sigma \in C. C_2 \models c_1$.

To test whether $\{M(\psi_1)\} \models \{M(\psi_2)\}$, we simply test whether there is a subset-match between $M(\psi_1)$ and $M(\psi_2)$.

To determine whether or not $\{\psi_1, \ldots, \psi_n\} \models \psi$ in general, we first find a $\psi_i$ whose main part entails the main part of $\psi$. However, the exceptions of $\psi_i$ weakens $\psi_i$. Therefore, we must also account for every clause in $\psi$ not implied by $\psi_i$, i.e. $\psi - \psi_i$. The difference of two $\psi$-forms is by itself a set of $\psi$-forms all of which are strictly smaller than $\psi_i$ in a well founded sense, i.e. it has strictly fewer quantified variables.

**POP Algorithm with $\psi$-forms**

We have implemented the algorithm for partial order planning with $\psi$-forms called PSIPOP. It is based on SNLP (McAllester & Rosenblitt 1991). Figure 1 shows the PSIPOP algorithm written for a non-deterministic machine. We assume that the reader is already familiar with SNLP-style planning and will rely upon the tests that are defined in the previous section.

Each branch of the non-deterministic algorithm either fails or returns a pair - a plan and a set of subgoals. If this set is empty, then the returned plan is a complete solution to the planning problem. Otherwise, the returned plan contains a sensing step that must be performed in order to achieve one of the subgoals, and the returned set represents all open subgoals of the incomplete plan.

We are currently investigating sound and efficient strategies of interleaving planning with execution, but we do not combine planning with execution in the pre-sent algorithm. However, we can interleave planning with execution as in planners such as (Golden 1998; Golden, Etzioni, & Weld 1996). If $p$ is a partial plan

---

**Algorithm.** PSIPOP ($D, K, < S, O, L >$, open, open$_d$, open$_s$, open)

1. If open is empty, return ($< S, O, L >$, open$_d$).
2. Pick a goal $< c, S_c >$ from open. Remove $< c, S_c >$ from open. choose an existing step $S_t$ from $S$, or a new step $S_t$ from $D$, that has an effect $e$ where $e \models c$ or $e$ nearly entails $c$ (if nearly entails, then Split Goal $(e, c)$, goto 5).
3. If no such step exists, $S$ does not contain a sensing action, and $\neg KW(c)$ holds immediately before $S_t$; then choose a sensing action $S_k$ from $K$ that has a knowledge effect $k$, such that $k \models KW(c)$, if $c$ is a literal, or $k \models \{KW(M(c))\}$, if $c$ is a quantified $\psi$-form. Add $< c, S_k >$ back to open. Set open = open$_s$, goto 5. If no such sensing action exists then fail.
4. Add link $S_t \notin S_k$ to $L$.
5. Add $S_t < S_k$ to $O$. If $S_t$ is a new step:
   - Add START-< $S_t$ and $S_t$ -< FINISH to $L$.
   - For each $p$ in $P(S_t)$ (the preconditions of $S_t$), add $< p, S_t >$ to open.
6. For every step $S_t$ that threatens a link $S_t \notin S_k$ non-deterministically choose either:
   - Demotion: Add $S_t < S_k$ to $O$.
   - Promotion: Add $S_k < S_t$ to $O$.
   - Split Link $(e, c)$.
7. If $O$ is inconsistent then fail. Otherwise, recursively call PSIPOP with an updated $< S, O, L >$ and open.

$D$ and $K$ are sets of domain and sensing actions respectively. A 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 $S_i < S_j$, where $S_i$ and $S_j$ are steps in $S$, initially contains START-<FINISH; $L$ is a set of (causal) links of form $S_i \notin S_j$, where $p$ is a precondition of $S_j$, $e$ is an effect of $S_i$, and $e \models p$. We call $S_i$ and $S_j$ the source and the target steps. $L$ is initially empty. open$_d$ is the list of open preconditions and initially contains preconditions of the FINISH step. open$_s$ is a list of open preconditions of steps preceding and including the sensing action. Initially empty. open points to the open list currently in use. Initially points to open$_d$.

---

Figure 1: PSIPOP algorithm
returned by PSIPOP and selected for execution, the agent executes the sensing action and all preceding actions that establish preconditions for the sensing, and calls PSIPOP with an updated initial state, the chosen incomplete plan \( p \) (with executed actions removed) and a set of \( p \)'s open subgoals.

When all the actions in the domain are reversible (Golden, Etzioni, & Weld 1996), choosing any of the partial plans does no harm to achieving the final goals, as we can backtrack over executions.

We recognize that a link between two \( \psi \)-forms is actually a multitude of links between the clauses of one and the clauses of the other. Thus, we sometimes split \( \psi \)-forms and/or links as needed.

In addition, we suppose that all possible resolutions have been performed on the initial state prior to calling PSIPOP, and after executing a sensing action. We need not perform any resolutions while planning, neither do we need an explicit representation of knowledge propositions, since they can always be deduced easily.

Below we present the details of our algorithm. We first focus on the issue of sensing, and then handling \( \psi \)-form propositions. Unless stated otherwise, we use the term actions in place of domain actions.

**Step 3: Adding a Sensing Step**

To avoid redundant information gathering, before using a sensing step to possibly support a subgoal \(<c, S_c>\) the planner must show that \( \neg KW(c) \) holds prior to step \( S_c \), i.e. there is no step ordered before \( S_c \) that has \( c \) or \( \neg c \) among its effects. The sensing action's knowledge list must entail \( KW(c) \) or \( KW(\neg c) \) in case \( c \) is a literal, or a \( \psi \)-form \{\( KW(M(c)) \)\}, if \( c \) is a quantified \( \psi \)-form.

A sensing step is inserted only if there is no knowledge about the value of the proposition \( c \). In the latter case, however, when \( c \) is a \( \psi \)-form, then it must be \( c \). In the latter case, however, we also seek steps where \( e \) does not entail \( c \) but where \( e \) nearly entails \( c \), i.e. the main part of \( e \) entails the main part of \( c \) and the image of \( e \)'s exceptions on main part of \( c \) does not contain \( c \). In such cases, we perform goal splitting.

If we have two \( \psi \)-forms, \( \psi_e \) and \( \psi_e \), where both are not single clauses and where \( \{M(\psi_e)\} \models \{M(\psi_e)\} \), then "most" of \( \psi_e \) is entailed by \( \psi_e \). The only "leftovers" are the clauses of \( \psi_e \) that are not entailed by any clause of \( \psi_e \). These clauses are a subset of those clauses implied by exceptions of \( \psi_e \). In fact, this set of "leftovers" is precisely \( \psi_e - \psi_e \). The result of this difference is a set of \( \psi \)-forms. Thus, goal splitting is the act of taking such a \( \psi_e \) and \( \psi_e \) and splitting \( \psi_e \) into a set of \( \psi \)-forms, \( \Psi = \{\psi\} \cup (\psi_e - \psi_e) \), where \( \psi = \psi_e \lor \psi_e \).

Here, \( \psi \) is just \( \psi_e \) after we add more exceptions to it. Then, we add a link from \( \psi_e \) to \( \psi_e \), namely \( S_{s \rightarrow} \psi \rightarrow \psi_e \). We add the \( \psi \)-forms of \( (\psi_e - \psi) \) to open. In this way, we have split \( \psi_e \) into a quantified \( \psi \)-form, \( \psi \), which is linked from \( \psi_e \), plus a set of \( \psi \)-forms, namely \( \psi_e - \psi_e \), that still need links.

Note that goal splitting is an equivalence-preserving transformation.

**Example 2.** Suppose after the execution, the \( LSall \) action from the previous example produced the following set of propositions:

\[
\begin{align*}
\{&In(a.tex, /home), In(a.ps, /home), In(b.ps, /ps), \\
&\psi_e = \{\neg In(f, d) \lor \neg PS(f) | \neg (d = /ps) \} \\
&\neg (f = b.ps, d = /ps) \land \neg (f = a.tex, d = /home) \} \\

\psi_e &\text{ nearly entails } \psi_e, \text{ and after the goal splitting we obtain the link } S_{s \rightarrow} \psi \rightarrow \psi_e, \text{ where } S_{s} \text{ is the START step, } \\

\psi &= \psi_e \lor \psi_e = \{\neg In(f, d) \lor \neg PS(f) | \neg (d = /ps) \lor \neg (f = a.ps, d = /home) \land \neg (f = a.tex, d = /home) \} \\
&\land \neg PS(a.ps) \}, \{\neg In(a.tex, /home) \lor \neg PS(a.tex) \}
\end{align*}
\]
Step 6: Handle Threats

For a step to be a threat, its effect must remove a supporting proposition for the precondition from the SOK. Thus, $\psi$-forms can only be threatened by atoms, and vice versa.

We add link splitting to the arsenal of threat resolution techniques. Link splitting applies when an atom threatens a causal link that is supported by a quantified $\psi$-form.

An atom $A$ that is an effect of step $S_s$ is a threat to a causal link $S_e \psi_e \rightarrow \psi_s$ if it removes a clause of $\psi_e$ that supports some clause(s) in $\psi_c$. In particular, the clauses of $\psi_e$ that may lose support from $\psi_e$ as the result of the threat, $A$, are exactly $\Psi \equiv (\neg \neg A) \triangleright \psi_e \triangleright \psi_e$, and therefore $A$ is a threat, iff $\Psi \neq \emptyset$. Note that $\Psi$ is a set of $\psi$-forms, which is empty if $\neg \neg A \triangleright \psi_e = \emptyset$.

Here, $(\neg \neg A \triangleright \psi_e)$ are those clauses that will be "lost" from $\psi_e$, due to effect $A$ of $S_s$, and $(\neg \neg A \triangleright \psi_e) \triangleright \psi_c$ are those clauses in $\psi_c$ that are entailed by those "lost" supports.

We first separate from the original goal the clauses that may lose support due to the threat, and also separate supporting them propositions that the threat "removes" from the source. Thus, both the goal and the source are partitioned into sets consisting of a quantified $\psi$-form and a set of ground clauses. We partition $\psi_e$ into $\psi^1_e = \psi_e - (\neg \neg A) \triangleright \psi_e$ and a set of $\psi$-forms $(\neg \neg A) \triangleright \psi_e$ are those clauses in $\psi_c$ that are entailed by those "lost" supports.

We also partition $\psi_c$ into $\Psi^1_c = \psi_c - (\neg \neg A) \triangleright \psi_c$ and $(\neg \neg A \triangleright \psi_e) \triangleright \psi_e$. $\Psi^1_c$ is a set of $\psi$-form subsets of $\psi_e$ that did not lose support due to the threat. The remainder, $(\neg \neg A \triangleright \psi_e) \triangleright \psi_e$, is the set of all $\psi$-forms that are entailed by the clauses that we’ve taken out of $\psi_e$, i.e. $(\neg \neg A) \triangleright \psi_e$.

We now have $\forall \psi \in \Psi^1_c \cdot \psi \models \psi$, but $A$ is not a threat to any of the links $S_s \psi^1_s \rightarrow \psi_c$, since $(\neg \neg A) \triangleright \psi^1_s = \emptyset$.

We replace the original link with a set of links $S_s \psi^1_s \rightarrow \psi_c$. However, we must find new links to support the remaining $\psi$-form goals - $((\neg \neg A \triangleright \psi_e) \triangleright \psi_c)$, which we add to the open.

Applying link splitting we have replaced the original link with a new one, which is not threatened and which supports all clauses of original goal $\psi_e$ except for a finite number of ground clauses which we post to the open.

Related Work

Continuing with our comments on the LCW representation (Golden, Etzioni, & Weld 1994; Etzioni, Golden, & Weld 1997), due to the inability to represent exceptions, information sometimes gets discarded. The LCW reasoning in itself is incomplete even without sensing and information loss. The query mechanism is also incomplete—it cannot deduce all ground facts that are implied by the agent’s SOK.

The language of $\psi$-forms doesn't have any of these problems and thus is more adequate in representing incomplete knowledge. It allows us to express statements with exceptions, while keeping the reasoning complete and tractable.

In (Levy 1996), Levy 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 LC constraints can easily represent the information in LCWs with exceptions, which is roughly the expressive power of $\psi$-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.

Conclusions and Future Work

We have presented PSIPLAN, a partial order planner that does not make the closed world assumption and that can represent partially closed worlds. The key idea is the use of $\psi$-forms to represent quantified negative information and to integrate $\psi$-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. PSIPOP 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 effects is straightforward.

Future work is already in progress. We will continue to explore richer representations for $\psi$-forms. Also, we will develop methods for integrating sensing, conditional planning and plan execution. Finally, we will formally examine the issues of soundness, completeness and complexity of the planning algorithm.
References


