Reasoning with conditional plans in the presence of incomplete knowledge

Ron Petrick    Fahiem Bacchus

Cognitive Robotics Group
University of Toronto

Presented at the ICAPS-03 Workshop on Planning under Uncertainty and Incomplete Knowledge in Trento, Italy (June 2003)
Motivation

- Planning with incomplete but accurate knowledge
- Reason at *plan time* about a plan’s effects and possible executions
  - Validate action preconditions
  - Ensure that a plan achieves goal conditions
- Work within the PKS framework
  ⇒ Planning with Knowledge and Sensing
Previous work: PKS

- Planning at the “knowledge level”
- Actions update the agent’s knowledge state, rather than the state of the world

**Advantages**
- Not a propositional representation (functions)
- Reasoning is “abstracted” \(\Rightarrow\) efficient algorithm

**Disadvantages**
- Representation restricts the types of knowledge that can be modelled
- Inference algorithm is sound but incomplete

\(\Rightarrow\) This work: extend PKS’s reasoning ability
Intuitions

- Example: bottle of liquid, healthy lawn
  - *pour-on-lawn*: if the liquid is poisonous then the lawn becomes dead
  - *sense-lawn*: sense whether lawn is dead or not

- Consider the action sequence:
  \[
  \langle \text{pour-on-lawn}, \text{sense-lawn} \rangle
  \]

- “Intuitive” conclusions
  - If the lawn is dead after execution: liquid is poisonous, liquid was *initially* poisonous
  - Prior to execution: come to know whether the liquid is poisonous (regardless of outcome)
Intuitions...

- How can we automate such inferences?
  - Inferences are intuitive but non-trivial
  - Conclusions don’t follow solely from action effects

- Markov assumption
  - Complete knowledge of action effects and *non-effects*
  - Agent’s actions are the only source of change in the world

⇒ Make use of this additional information to enhance PKS’s reasoning ability
Knowledge is represented by a set of 4 databases, each models a different type of knowledge.

Contents of databases have fixed translation to formulae in a modal logic of knowledge.

Given a set of four databases ($DB$)

\[ \Rightarrow \] translation defines knowledge state ($KB$)
PKS: databases

- $K_f$: knowledge of positive and negative facts
  \[ p(a), \neg q(b, c), f(a) = c, g(b, c) \neq d \]

- $K_w$: plan-time knowledge of sensing actions
  \[ \phi \in K_w : \text{know } \phi \text{ or know } \neg \phi \text{ at execution} \]

- $K_v$: plan-time knowledge of function values
  \[ f(\vec{x}) \in K_v : \text{know } f(\vec{x})'s \text{ value at execution} \]

- $K_x$: exclusive-or knowledge
  \[ (l_1|l_2|\ldots|l_n) : \text{exactly one of the } l_i \text{ must be true} \]
<table>
<thead>
<tr>
<th>Action</th>
<th>Pre</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>pour-on-lawn</td>
<td></td>
<td>(\neg K(\neg \text{poisonous}) \Rightarrow) del(K_f, \neg \text{lawn-dead})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(K(\text{poisonous}) \Rightarrow) add(K_f, \text{lawn-dead})</td>
</tr>
<tr>
<td>sense-lawn</td>
<td></td>
<td>add(K_w, \text{lawn-dead})</td>
</tr>
</tbody>
</table>

- Actions update DB ⇒ update KB
- Inference algorithm examines database contents to evaluate preconditions and goals
PKS: conditional plans

- PKS generates plans by forward-chaining

![Diagram](attachment://pksgeneratedplan.png)

- Original PKS inference algorithm is unable to conclude anything about *poisonous* in this plan

- This work illustrates how PKS’s inference algorithm can be enhanced so that it is able to achieve know-whether knowledge of *poisonous*
1. Generate a conditional plan

\[ \text{pour-on-lawn} \rightarrow \text{sense-lawn} \]

\[ K_f: \neg \text{lawn-dead} \]

2. Form linearizations (possible execution branches)

\[ \text{pour-on-lawn} \rightarrow \text{sense-lawn} \]

\[ K_f: \neg \text{lawn-dead} \]

\[ K_f: \text{lawn-dead} \]

\[ K_f: \neg \text{lawn-dead} \]

3. Augment states by applying 4 new inference rules
Inference rule 1

- Action $a$ cannot have changed the status of $\phi$ between $W$ and $W^+$. 

If $a$ cannot make $\phi$ false (similarly, true) then if $\phi$ becomes newly known in $W$ ($W^+$), make $\phi$ known in $W^+$ ($W$).
Action $a$ cannot have changed the status of $\phi$ between $W$ and $W^+$. If $a$ cannot make $\phi$ false (similarly, true) then if $\phi$ becomes newly known in $W$ ($W^+$), make $\phi$ known in $W^+$ ($W$).
Inference rule 1

Action $a$ cannot have changed the status of $\phi$ between $W$ and $W^+$.

If $a$ cannot make $\phi$ false (similarly, true) then if $\phi$ becomes newly known in $W$ ($W^+$), make $\phi$ known in $W^+$ ($W$).
Inference rule 1

- Action $a$ cannot have changed the status of $\phi$ between $W$ and $W^+$. 

If $a$ cannot make $\phi$ false (similarly, true) then if $\phi$ becomes newly known in $W$ ($W^+$), make $\phi$ known in $W^+$ ($W$).
Inference rule 2

\[
\begin{align*}
\text{If } \phi \text{ becomes newly known in } W \text{ and } a \text{ has the conditional effect } \phi \rightarrow \psi, \text{ make } \psi \text{ known in } W^+.
\end{align*}
\]
\( \psi \) must be true in \( W^+ \) as either it was already true or \( a \) made it true.

If \( \phi \) becomes newly known in \( W \) and \( a \) has the conditional effect \( \phi \rightarrow \psi \), make \( \psi \) known in \( W^+ \).
Inference rule 3

Action $a$’s conditional effect was activated, so the antecedent of this effect must have been true.

If $a$ has the conditional effect $\psi \rightarrow \phi$ and it becomes newly known that $\phi$ holds in $W^+$ and $\neg \phi$ holds in $W$, make $\psi$ known in $W$. 

\[ W \xrightarrow{a : \psi \rightarrow \phi} W^+ \]
Action $a$’s conditional effect was activated, so the antecedent of this effect must have been true.

If $a$ has the conditional effect $\psi \rightarrow \phi$ and it becomes newly known that $\phi$ holds in $W^+$ and $\neg\phi$ holds in $W$, make $\psi$ known in $W$. 
Action $a$’s conditional effect was not activated, so the antecedent of this effect must have been false.

If $a$ has the conditional effect $\psi \rightarrow \phi$ and it becomes newly known that $\neg\phi$ holds in $W^+$, make $\neg\psi$ known in $W$. 
Inference rule 4

- Action $a$’s conditional effect was not activated, so the antecedent of this effect must have been false.

If $a$ has the conditional effect $\psi \rightarrow \phi$ and it becomes newly known that $\neg \phi$ holds in $W^+$, make $\neg \psi$ known in $W$. 
Initiation and restrictions

- Start with states that result from branching on know-whether knowledge
- Apply rules recursively (i.e., stack-based)
- To achieve efficient implementation:
  - Restrict $\phi$, $\psi$ to literals, no free parameters
  - Actions cannot add or delete a fluent $F$ with more than one conditional effect

\[ b_1 \rightarrow F \\
\qquad b_2 \rightarrow F \Rightarrow b_1 \lor b_2 \]

$\Rightarrow$ Avoid generating disjunctions
Example: poisonous liquid

$Kf: \neg \text{lawn-dead}$

pour-on-lawn \rightarrow sense-lawn \rightarrow

$pour-on-lawn$ \rightarrow $sense-lawn$ \rightarrow

$Kf: \neg \text{lawn-dead}$

$Kf: \text{lawn-dead}$
Example: poisonous liquid

\[
pour\text{-on}\text{-lawn} \rightarrow sense\text{-lawn} \\
Kf: \neg \text{lawn-dead} \quad Kf: \text{lawn-dead} \quad Kf: \text{lawn-dead}
\]

\[
pour\text{-on}\text{-lawn} \rightarrow sense\text{-lawn} \\
Kf: \neg \text{lawn-dead} \quad Kf: \neg \text{lawn-dead}
\]
Example: poisonous liquid

\[ Kf \vdash \neg \text{lawn-dead} \quad Kf \vdash \text{lawn-dead} \quad Kf \vdash \text{lawn-dead} \]

\[ Kf \vdash \text{poisonous} \]

\[ Kf \vdash \neg \text{lawn-dead} \quad Kf \vdash \neg \text{lawn-dead} \]
Example: poisonous liquid

\[
\begin{align*}
\text{pour-on-lawn} & \quad \text{sense-lawn} \\
Kf: \neg \text{lawn-dead} & \quad Kf: \text{lawn-dead} & \quad Kf: \text{lawn-dead} \\
Kf: \text{poisonous} & \quad \text{Kf: poisonous} & \quad \text{Kf: poisonous} \\
\text{pour-on-lawn} & \quad \text{sense-lawn} \\
\text{Kf: } \neg \text{lawn-dead} & \quad \text{Kf: } \neg \text{lawn-dead} \\
\end{align*}
\]
Example: poisonous liquid

\[\text{pour-on-lawn} \rightarrow \text{sense-lawn} \]

\[Kf: \neg \text{lawn-dead} \quad Kf: \text{lawn-dead} \quad Kf: \text{lawn-dead}\]

\[Kf: \text{poisonous} \quad Kf: \text{poisonous} \quad Kf: \text{poisonous}\]
Example: poisonous liquid

1. Start with a poisonous liquid.
2. Pour the liquid on the lawn.
3. Sense the lawn.

- If the lawn is not dead, repeat step 2.
- If the lawn is dead, the liquid is poisonous.

Kf: ¬ lawn-dead  Kf: lawn-dead  Kf: lawn-dead
Kf: poisonous  Kf: poisonous  Kf: poisonous

Kf: ¬ lawn-dead  Kf: ¬ lawn-dead
Example: poisonous liquid

pour-on-lawn  sense-lawn

\( Kf: \neg \text{lawn-dead} \quad Kf: \text{lawn-dead} \quad Kf: \text{lawn-dead} \)

\( Kf: \text{poisonous} \quad Kf: \text{poisonous} \quad Kf: \text{poisonous} \)
Efficiency

- $O(nd^2)$ testings of the inference rules (worst case)
  - Conditional plan with $n$ leaves, depth $d$
  - Rule evaluation has the same complexity as action application
- In practice, few effects applied at each state
Extensions to planning ability

- Plans previously rejected may now be proven to satisfy goal conditions.
- Potential to solve more complicated temporal goals conditions, expressed in terms of:
  - Final state (e.g., classical goals)
  - Initial state (e.g., restore goals)
  - Every state (e.g., “hands-off” goals)
- Note: PKS still employs blind search to find plans.
Example: UNIX domain

<table>
<thead>
<tr>
<th>Action</th>
<th>Pre</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>chmod+x(d)</td>
<td>K(dir(d))</td>
<td>add(K_f, exec(d))</td>
</tr>
<tr>
<td>chmod-x(d)</td>
<td>K(dir(d))</td>
<td>add(K_f, ¬exec(d))</td>
</tr>
<tr>
<td>cp(f, d)</td>
<td>K(file(f))</td>
<td>exec(d) ⇒ add(K_f, indir(f, d))</td>
</tr>
<tr>
<td></td>
<td>K(dir(d))</td>
<td>add(K_w, indir(f, d))</td>
</tr>
</tbody>
</table>

- Init: dir(icaps), file(paper.tex), ¬indir(paper.tex, icaps) ∈ K_f
- Goal: K(indir(paper.tex, icaps), restore(exec(icaps)))

PKS can solve this example in time < the resolution of the timer
Conclusions and future work

- Simple mechanism for reasoning about the knowledge effects of conditional plans
  \[ \implies \text{a form of postdiction} \]
- Extensions allow PKS to solve more interesting range of problems, more complex goal conditions
- Future work
  - Function terms with unknown range
  - Progress/regress more complex formulae
  - Search control