Extending the knowledge-based approach to planning with incomplete information and sensing

Ron Petrick
(Joint work with Fahiem Bacchus)

Cognitive Robotics Group
Department of Computer Science
University of Toronto

Presented at the Cognitive Robotics Seminar, University of Toronto (15 June 2004)
Motivation

• Planning with incomplete but accurate knowledge

• Reasoning with information available at **plan time**
  ◦ Validate action preconditions
  ◦ Ensure that a plan achieves goal conditions

• Knowledge is dynamic
  ◦ Action effects
  ◦ Outcomes of sensing
  ◦ Assumptions about conditional plans

• Work within the PKS framework
  (Bacchus & Petrick 1998, Petrick & Bacchus 2002-03)

⇒ Planning with Knowledge and Sensing
Previous work: PKS

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

Advantages
- Allow functions and numerical reasoning, unlike propositional approaches
- Reasoning is “abstracted” ⇒ can be very efficient

Disadvantages
- General first-order reasoning is intractable
- Representation restricts the types of knowledge that can be modelled
- Inference algorithm is sound but incomplete

⇒ This work: lift some of the restrictions
PKS: knowledge

- Knowledge is represented by a set of 4 databases
- Each database models a different type of knowledge
- Database contents (DB) have a fixed translation to formulae in a modal logic of knowledge (KB)
- Formal semantics provided by translation

⇒ Given a DB the translation defines a KB
PKS: databases

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

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

- $K_v$: 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
  
  \[ (\ell_1|\ell_2|\ldots|\ell_n) : \text{exactly one of the } \ell_i \text{ must be true} \]
PKS: primitive queries

- An inference algorithm (IA) examines the database contents to evaluate preconditions and goals

- Primitive query language:
  - $K(\alpha)$: is $\alpha$ known to be true?
  - $K(\neg\alpha)$: is $\alpha$ known to be false?
  - $K_w(\alpha)$: is $\alpha$ known to be true or known to be false?
  - $K_v(t)$: is the value of $t$ known?
  - Negation of the above queries
    ($\alpha$ is a ground atomic formula, $t$ is a variable-free term)

- IA is sound, but incomplete
PKS: actions

<table>
<thead>
<tr>
<th>Action</th>
<th>Pre.</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>pour-on-lawn</td>
<td>(-K(\neg\text{poisonous})) ⇒ del(K_f, \neg\text{lawn-dead})) (K(\text{poisonous})) ⇒ add(K_f, lawn-dead)</td>
<td></td>
</tr>
<tr>
<td>sense-lawn</td>
<td>add(K_w, lawn-dead)</td>
<td></td>
</tr>
</tbody>
</table>

- Example: bottle of liquid, healthy lawn
- Actions update DB ⇒ implicitly update KB
PKS: conditional plans

- PKS generates conditional plans by forward-chaining

- $K_w$ knowledge is used to add conditional branches
  - Ensures plan can be executed

- PKS currently employs blind search to find plans
  - No search control
Extending PKS

• This work: extend the representational and inferential capabilities of PKS

• Four extensions to the basic framework
  ◦ Enhanced inference mechanism
  ◦ Temporally-extended goals
  ◦ Finite-range functions
  ◦ Numerical expressions

• We can now represent “richer” planning problems

• Plans we were previously unable to recognize as satisfying the goals, can now be shown to succeed
Reasoning about plans

• PKS previously failed to make certain “intuitive” conclusions

○ If the lawn is dead after execution:
  ⇒ liquid is poisonous and was initially poisonous

○ Prior to execution (regardless of outcome):
  ⇒ come to know whether the liquid is poisonous

• Conclusions don’t follow solely from action effects
  ○ Requires reasoning about action non-effects

⇒ Postdiction (Sandewall 1994)
Postdiction in PKS

1. Generate a conditional plan

2. Form linearizations (possible execution branches)

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

$W$ $\Phi$ $\Rightarrow$ $A$: $\rightarrow \neg \Phi$ $W^+$ $\Phi$

$\Phi$ $\Rightarrow$ $A$ could not have changed the status of $\Phi$ between $W$ and $W^+$. 
Inference rule 2

\[ \Rightarrow \Psi \text{ must be true in } W^+ \text{ as either it was already true or } A \text{ made it true.} \]
Inference rule 3

\[ \Rightarrow A's \text{ conditional effect was activated, so the antecedent of this effect must have been true.} \]
A’s conditional effect was not activated, so the antecedent of this effect must have been false.
Restrictions and efficiency

- Restrict $\Phi$, $\Psi$ to literals; no free parameters
- A fluent cannot appear in multiple conditional effects

\[ c_1 \rightarrow F \]
\[ c_2 \rightarrow F \quad \Rightarrow \quad c_1 \vee c_2 \quad X \]

⇒ Avoid generating disjunctions we cannot represent

- Soundness: we assume we have complete information about action effects
- Worst case: $O(nd^2)$ testings of the inference rules
  - Conditional plan with $n$ leaves, depth $d$
  - Performance is much better in practice
Example: poisonous liquid

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

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

\[ \Rightarrow \text{Linearizations of conditional plan} \]
Example: poisonous liquid...(2)

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

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

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

\[\Rightarrow \text{States augmented by postdiction}\]
Temporally extended goals

- Since postdiction examines all execution paths in a plan, it becomes possible to check “path formulae.”

- Goal queries are extended with a temporal component:
  - \( Q^N \): query final state (e.g., classical goals)
  - \( Q^0 \): query initial state (e.g., restore goals)
  - \( Q^* \): query all states (e.g., “hands-off” goals)

Cf. (Weld & Etzioni 1994)

- Queries can be combined using conjunction, disjunction, negation, and limited quantification.

- Goal is satisfied if it is satisfied in every execution path.

\[
\begin{align*}
&\quad (K^0(\text{poisonous}) \land K^N(\text{poisonous})) \lor \\quad (K^0(\neg\text{poisonous}) \land K^N(\neg\text{poisonous}))
\end{align*}
\]
Finite-range functions

- $K_x$ is extended to represent functions with a finite and known range, e.g.,

$$\text{colour}(x) = \text{red} \mid \text{colour}(x) = \text{green} \mid \text{colour}(x) = \text{blue}$$

- PKS can reason about sets of function mappings, e.g.,

$$f(x) = \{c_1 | c_2 | \ldots | c_n\}, \ g(x) = \{c_1 | d_1 | \ldots | d_m\}$$

$\Rightarrow$ If $f(a) = g(b)$ then conclude $f(a) = g(b) = c_1$.

- PKS can construct a multi-way branch in a plan
  - Requires $K_v$ knowledge of a finite-range function
  - Build a branch for each possible mapping
Example: open safe domain

<table>
<thead>
<tr>
<th>Action</th>
<th>Pre.</th>
<th>Effects</th>
</tr>
</thead>
</table>
| `dial(x)` |      | `add(K_w, open)`  
             |      | `del(K_f, ¬open)`  
             |      | $K(combo() = x) \Rightarrow add(K_f, open)$ |

⇒ Construct a branch for each possible mapping of `combo()` and continue planning.
Example: open safe domain...(2)

<table>
<thead>
<tr>
<th>Action</th>
<th>Pre.</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>dial(x)</td>
<td></td>
<td>add(K_w, open)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>del(K_f, ¬open)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K(combo() = x) ⇒ add(K_f, open)</td>
</tr>
</tbody>
</table>

$\Rightarrow$ This problem can also be solved without reasoning about specific combinations (Petrick & Bacchus 2002)
Numerical evaluation

- Planning often requires the ability to reason about numbers (e.g., constraints, resource management)
- PKS can evaluate a subset of C language expressions
  - Standard arithmetic operations
  - Logical connectives, temporary variables
  - Control structures (e.g., if-else, iterative loops)
- Expressions are permitted in queries and DB updates
- Restriction: must be able to evaluate at plan time

  e.g., \( \text{size}(\text{paper.tex}) > 1024 \)

  \( \Rightarrow \text{size}(\text{paper.tex}) \) must evaluate to a number
Example: UNIX domain

- Actions: $\text{cd}(d)$, $\text{cd-up}(d)$, $\text{ls}(f, d)$
- Goal: count the # of copies of a file in a directory tree
- Directory tree is initially known, but not necessarily the directory contents
- There is at most one copy of a file in each directory
- Encoding uses a function $\text{count}()$, numerical exprs.
- If a directory has not been “processed,” and the file is known to be in the directory, increment the count
### Example: UNIX domain ...(2)

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
</table>
| \( cd(d) \) | \( K(\text{dir}(d)) \)  \
\( K(\text{indir}(d, \text{pwd}())) \) | \( \text{add}(K_f, \text{pwd}() = d) \) |
| \( cd-up(d) \) | \( K(\text{dir}(d)) \)  \
\( K(\text{indir}(\text{pwd}(), d)) \) | \( \text{add}(K_f, \text{pwd}() = d) \) |
| \( ls(f, d) \) | \( K(\text{pwd}() = d) \)  \
\( K(\text{file}(f)) \)  \
\( \neg K_w(\text{indir}(f, d)) \) | \( \text{add}(K_w, \text{indir}(f, d)) \) |
Example: UNIX domain...(3)

Domain specific update rules

\[ \neg K(processed(f, d)) \land K(indir(f, d)) \land K_v(size(f, d)) \Rightarrow \]
\[ t = [(size-max() > size(f, d))? size-max() : size(f, d)], \]
\[ add(K_f, size-max() = t), \]
\[ add(K_f, count() = count() + 1), \]
\[ add(K_f, processed(f, d)) \]

\[ \neg K(processed(f, d)) \land K(indir(f, d)) \land \neg K_v(size(f, d)) \Rightarrow \]
\[ add(K_f, size-unk() = size-unk() + 1), \]
\[ add(K_f, processed(f, d)) \]

\[ \neg K(processed(f, d)) \land K(\neg indir(f, d)) \Rightarrow \]
\[ add(K_f, processed(f, d)) \]
Example: UNIX domain...(4)

How many instances of paper.tex?

\[ \text{pwd()} = \text{root} \]

\[ \text{icaps} \quad \text{kr} \]

\[ \text{paper.tex} \]

\[ \text{planning} \]

\[ \text{ls(paper.tex,root)} \]

\[ \text{cd(icaps)} \]

\[ \text{cd(planning)} \]

\[ \text{ls(paper.tex,planning)} \]

\[ \text{indir(paper.tex,planning)} \]

\[ K_+ \quad K_- \]

\[ \text{indir(paper.tex,root)} \]

\[ \text{count()} = \]

\[ 4 \quad 3 \quad 3 \quad 2 \]
Example: UNIX domain...(5)

How many instances of *paper.tex*?

```
    pwd() = root ?
     |
    icaps ?
     |
    planning ?
```

Conditional plan

- 16 branches
- DFS: 0.01 sec.
- BFS: 30.1 sec.
Example: UNIX domain... (6)

Configuration at execution time

\[ \text{pwd}() = \text{root} \]

\[
\begin{align*}
\text{icaps} & \quad \text{kr} \\
\text{planning} & \quad \text{paper.tex} \\
\text{paper.tex} & 
\end{align*}
\]

\[
\begin{align*}
\text{ls}(\text{paper.tex}, \text{root}) & \\
\text{cd}(\text{icaps}) & \\
\text{ls}(\text{paper.tex}, \text{icaps}) & \\
\text{cd}(\text{planning}) & \\
\text{ls}(\text{paper.tex}, \text{planning}) & \\
\text{cd-up}(\text{icaps}) & \\
\text{cd-up}(\text{root}) & \\
\text{cd}(\text{kr}) & \\
\text{ls}(\text{paper.tex}, \text{kr}) & \\
\text{indir}(\text{paper.tex}, \text{root})? & (K-) \\
\text{indir}(\text{paper.tex}, \text{icaps})? & (K-) \\
\text{indir}(\text{paper.tex}, \text{planning})? & (K+) \\
\text{indir}(\text{paper.tex}, \text{kr})? & (K+) \\
\text{count}() = 2 & 
\end{align*}
\]
Conclusions and future work

• Enhanced the planning abilities of PKS in four areas
  ◦ Postdiction
  ◦ Temporal goal conditions
  ◦ Finite-range functions (e.g., multi-way branches)
  ◦ Numerical expressions

• PKS now solves a more interesting range of problems
  ⇒ illustrates utility of the knowledge-based approach

• Future work
  ◦ Extend plan-time reasoning about numerical expressions (e.g., $f(a) < c$)
  ◦ Progress/regress more complex formulae
  ◦ Search control
References


