A knowledge-based approach to planning with incomplete information and sensing

Ron Petrick    Fahiem Bacchus

Department of Computer Science
University of Toronto

Presented at AIPS-02 in Toulouse, France (April 2002)
Motivation

- Planning with incomplete information and sensing:
  - Correct but incomplete knowledge
  - Sensing actions
  - Conditional plans
- Example: software agent in the UNIX domain, high-level agent control, etc.
- Many recent approaches:
  - Model incomplete knowledge with a set of possible worlds
  - Reason about possible worlds to construct plans
Possible worlds

Incomplete knowledge modelled as a set of worlds, each a possible version of how real world configured

- Each world is a first-order model
- Knowing $\phi$ to be true:
  $\phi$ is true in every possible world
- Not knowing whether or not $\phi$ is true:
  worlds in which $\phi$ is true and $\phi$ is false
- $n$ atomic formulae $\Rightarrow$ potentially $2^n$ possible worlds
Planning at possible world level

Must reason at the possible world level to build plans:

- Preconditions: check truth in all possible worlds
- Actions: update each possible world with effects
- Knowledge state: consider changes across worlds
- Objects generate propositions: # possible worlds grows exponentially with # objects
- Compact representations exist
  - Often limited to propositional case (e.g., BDDs)
  - Not always compact
Planning at the knowledge level

Our approach: build plans based on what is known, model actions as updating knowledge

- Formally: utilize modal logic of knowledge
  - $K$ modal operator added to first-order language
    - $K(\phi)$: $\phi$ is known
  - Semantically understood in terms of PWs
    - $K(\phi)$: $\phi$ true at all worlds considered possible

- Knowledge level: work directly with formulae
- Tractable reasoning: restrict representation

$\Rightarrow$ Goal: efficient planning without manipulating individual possible worlds
Representing knowledge

Represent agent’s knowledge by a collection of four databases: $K_f, K_v, K_w, K_x$

- Each database restricted to a particular type of knowledge
- Knowledge is correct but incomplete
  
  $K(\text{readable}(\text{paper.tex})) \Rightarrow \text{readable}(\text{paper.tex})$

- Contents of databases have fixed translation to formulae in the modal logic of knowledge
- Given set of four databases ($\text{DB}$)
  
  $\Rightarrow$ translation defines agent’s knowledge state ($\text{KB}$)

- Planning: actions update $\text{DB} \Rightarrow$ update $\text{KB}$
$K_f$ database

Contains positive and negative facts:

- Similar to standard STRIPS database, no CWA
- Example: $\text{readable}(\text{paper.tex}) \in K_f$
  $\Rightarrow$ know $\text{paper.tex}$ is readable
- Ground literals (all terms constants): $P(a), \neg Q(c, b)$
- Function mappings of the form:
  $f(c_1, \ldots, c_n) = c_{n+1}$ or $f(c_1, \ldots, c_n) \neq c_{n+1}$
- For $\ell \in K_f$, $\mathbf{KB}$ includes the formula:
  
  $K(\ell)$
$K_w$ database

Contains formulae every instance of which the agent either knows or knows the negation (know whether):

- Model sensing, universal effects
- Example: $\text{readable}(\text{paper.tex}) \in K_w$
  $\Rightarrow$ sense whether $\text{paper.tex}$ is readable or not
  - Plan time: know will come to know readability
  - Run time: definite knowledge
- Entries: conjunctions of atomic formulae
- For $\phi(\vec{x}) \in K_w$, $\text{KB}$ includes the formula:

$$\left( \forall \vec{x} \right). K(\phi(\vec{x})) \lor K(\neg \phi(\vec{x}))$$
$K_v$ database

Contains information about function values that will become known at execution time:

- Model sensors that return constants
- Example: $\text{size}(\text{paper.tex}) \in K_v$
  ⇒ sense the size of $\text{paper.tex}$
  - Plan time: size unknown, will come to know
  - Run time: definite knowledge of size
- Entries: any unnested function term, e.g., $f(x, a)$
- For $f(\vec{x}) \in K_v$, $\text{KB}$ includes the formula:

$$\left( \forall \vec{x} \right) \left( \exists v \right). K(f(\vec{x}) = v)$$
$K_x$ database

Contains “exclusive or” knowledge of ground literals:

- Know exactly one literal in a set of literals is true
- Example: \(((combo) = c_1 \, | \, (combo) = c_2) \in K_x\)
  \[\Rightarrow\] know one of $c_1$ or $c_2$ is the combination of safe
- Entries are of the form $(l_1 | l_2 | \ldots | l_n)$
- For $(l_1 | l_2 | \ldots | l_n) \in K_x$, $KB$ includes the formula:

\[
K \left( \bigvee_{i=1}^{n} l_i \land (\neg l_1 \land \ldots \land \neg l_{i-1} \land \neg l_{i+1} \land \ldots \land \neg l_n) \right)
\]
Knowledge states

Given a set of databases (DB), fixed translation defines agent’s knowledge state (KB)

- Restrictions on databases contents
  \[ \Rightarrow \text{restrictions on knowledge that can be modelled} \]

- Cannot model certain types of knowledge
  e.g., general disjunctions: \( K(P(a) \lor Q(b, c)) \)

- Cannot model certain planning problems

- Avoid reasoning directly with individual possible worlds
Querying a knowledge state

Require ability to query a knowledge state $\textbf{KB}$: check preconditions, goals, conditional effects

- **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

- **Inference procedure $\textbf{IA}$**: sound, incomplete
  $\Rightarrow$ Check databases to determine truth of query
Planning problems

Planning problem: four tuple \( \langle I, G, A, U \rangle \)

- Initial state \( I \): initial contents of databases (initial knowledge state)
- Goal conditions \( G \): conjunction of primitive queries
  \( \Rightarrow \) must be satisfied in every knowledge state that could arise from executing a plan
- \( A \): non-empty set of action specifications
- \( U \): set of domain specific knowledge update rules
Representing actions ($A$)

Actions modelled as updates to databases:

- **Parameters**: set of variables bound to produce an action instance
- **Preconditions**: conjunctive set of primitive queries, each must evaluate to true to apply action
- **Effects**: conditional effect rules of the form $C \Rightarrow E$
  - Effect preconditions $C$: set of primitive queries
  - Database updates $E$: list of add and del operations to any of the databases

$\Rightarrow$ Easy to compute new knowledge states
### Representing actions: example

<table>
<thead>
<tr>
<th>Action</th>
<th>Precondition</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{drop(}x\text{)}</td>
<td>(K(\text{holding}(x)))</td>
<td>(\text{del}(K_f, \text{holding}(x))) (\text{add}(K_f, \text{onfloor}(x))) (\text{add}(K_f, \text{dropped}(x))) (\text{del}(K_f, \neg \text{broken}(x))) (K(\text{fragile}(x)) \Rightarrow \text{add}(K_f, \text{broken}(x)))</td>
</tr>
<tr>
<td>\textit{inspect(}y\text{)}</td>
<td></td>
<td>(\text{add}(K_w, \text{broken}(y)))</td>
</tr>
</tbody>
</table>
Representing actions: example...

Initial knowledge

\[ K_f \]

- holding(box)
- holding(vase)
- fragile(vase)
Representing actions: example...

Initial knowledge

$K_f$
- holding(box)
- holding(vase)
- fragile(vase)

$K_f$
- holding(box)
- fragile(vase)
- onfloor(vase)
- dropped(vase)
- broken(vase)

drop(vase)
Representing actions: example...

**Initial knowledge**

\[ K_f \]
- holding(box)
- holding(vase)
- fragile(vase)

**\( K_f \)**
- holding(box)
- holding(vase)
- fragile(vase)

**\( K_f \)**
- holding(vase)
- fragile(vase)
- onfloor(box)
- dropped(box)

\[ drop(box) \]

\[ drop(vase) \]
Representing actions: example...

Initial knowledge

$K_f$

- holding(box)
- holding(vase)
- fragile(vase)

$K_f$

- holding(vase)
- fragile(vase)
- onfloor(vase)
- dropped(vase)
- broken(vase)

$K_f$

- holding(box)
- fragile(vase)
- onfloor(box)
- dropped(box)

$K_w$

- broken(box)
**PKS: Knowledge-based planning**

Initial call: $PlanPKS(I, \emptyset, G)$

$PlanPKS(DB, P, G)$

if \textit{goalsSatisfied}(DB, G) then return $P$

else Choose

pick($A$) : \textit{precondsSatisfied}($A$, DB) ;
applyEffects($A$, DB, DB') ;
return $PlanPKS(DB'$, ($P$, $A$), $G$)

or

pick($\alpha$) : $\alpha$ is a ground instance of an entry in $K_w$ ;
branch($DB$, $\alpha$, $DB_1$, $DB_2$) ;
$C := \{PlanPKS(DB_1, \emptyset, G), PlanPKS(DB_2, \emptyset, G)\}$ ;
return $P, C$
Domain specific update rules ($U$)

Knowledge level state invariants:

- Example: come to know an object is fragile if it has been dropped and is broken

$$K(\text{dropped}(x)) \land K(\text{broken}(x)) \Rightarrow \text{add}(K_f, \text{fragile}(x))$$

- Could be included in action specification

- Including all such updates often cumbersome, include rules independent of action specification

- Conditional effect rules of the form $C \Rightarrow E$
  (primitive queries $C$, database updates $E$)

- Triggered in any knowledge state where $C$ holds
Consistency rules

Domain independent rules ensure databases remain mutually consistent:

- Both $\alpha$ and $\neg\alpha$ cannot be in $K_f$
- $f(c_1, \ldots, c_n)$ in $K_f$ must only map to one constant
- $\ell$ added or deleted from $K_f$ (non-sensing action): remove $K_x$ formulae that mention $\ell$ or $\neg\ell$
- $\ell$ added to $K_f$ (conditional branch):
  for each $\phi \in K_x$, where $\phi = (\ell_1| \ldots |\ell_m)$
  - if $\ell \equiv \ell_i$: delete $\phi$, add $\neg\ell_j$ to $K_f$ for each $j \neq i$
  - if $\ell \equiv \neg\ell_i$: delete $\phi$, add $(\ell_1| \ldots |\ell_{i-1}|\ell_{i+1}| \ldots |\ell_m)$ to $K_x$
Consistency rules...

Example: \( \phi = (\text{infected}(I_1)|\text{infected}(I_2)) \) in \( K_x \)

- \( \text{infected}(I_2) \) added to \( K_f \) by assumption along conditional branch
  - \( \text{infected}(I_2) \) in \( K_w \) as result of sensing action
  - Delete \( \phi \) from \( K_x \), add \( \neg \text{infected}(I_1) \) to \( K_f \)

- \( \text{infected}(I_2) \) added to \( K_f \) by action causing infection
  - Delete \( \phi \) from \( K_x \)
  - No longer necessarily true that only one of \( \text{infected}(I_1) \) or \( \text{infected}(I_2) \) holds
Plan correctness

- Plan correctness relies on two criteria (Levesque)
  - Plan time: agent must know it will have enough information at run time for the plan to achieve the goals
  - Run time: agent must have sufficient knowledge at every step of the plan to execute it

- PKS satisfies both criteria:
  - Goals satisfied along every conditional branch
  - Plan branches based on sensed $K_w$ formulae
    $\Rightarrow$ resolves to definite knowledge at run time
Planning problems
Bomb in the toilet

- $P$ packages, $T$ toilets
- Bomb in one package, toilets possibly clogged
- $dunk(p, t)$ (disarms $p$, clogs $t$), $flush(t)$ (unclogs $t$)
- Possible world level: $P \times 2^T$ different worlds
- Knowledge level: specific location of bomb, which toilets clogged irrelevant

<table>
<thead>
<tr>
<th>Planner</th>
<th>$(#P, #T)$</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMBP</td>
<td>(10, 6)</td>
<td>116.24</td>
</tr>
<tr>
<td>PKS</td>
<td>(10, 10) → (60, 40) (100, 60)</td>
<td>&lt; 1.00 11.54</td>
</tr>
</tbody>
</table>
Opening a safe

- Safe, fixed set of combinations
- Know one is actual combination of the safe \((K_x)\)
- Goal: \(K(open)\)

<table>
<thead>
<tr>
<th>Action</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>(dial(x))</td>
<td>(add(K_w, open))</td>
</tr>
<tr>
<td></td>
<td>(del(K_f, \neg open))</td>
</tr>
<tr>
<td></td>
<td>(add(K_f, justDialled = x))</td>
</tr>
<tr>
<td></td>
<td>(K((combo) = x) \Rightarrow add(K_f, open))</td>
</tr>
</tbody>
</table>

- No search control \(\Rightarrow\) undirected plans, blind search
Opening a safe: search control

- Unnecessary to dial: know whether safe open, combo known not to work
- Add preconditions: \( \neg K(open), \neg K((combo) \neq x) \)
- Simple method of controlling search
- Natural plans generated

Ron Petrick and Fahiem Bacchus / A knowledge-based approach to planning with incomplete information and sensing / Presented at AIPS-02 (April 2002) – p.25
Opening a safe: timings

![Graph showing the relationship between combinations and time for opening a safe with and without preconditions. The graph illustrates that with preconditions, the time required increases significantly as the number of combinations increases.]
Opening a safe: run-time variables

- No knowledge of specific combinations
- Initial knowledge: $haveCombo \in K_f$

<table>
<thead>
<tr>
<th>Action</th>
<th>Precondition</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td>$dial(x)$</td>
<td>$K_v(x)$ $\neg K((combo) \neq x)$</td>
<td>$add(K_w, open)$ $K((combo) = x) \Rightarrow add(K_f, open)$</td>
</tr>
<tr>
<td>$readCombo$</td>
<td>$K(haveCombo)$</td>
<td>$add(K_v, (combo))$</td>
</tr>
</tbody>
</table>

- Plan: $readCombo ; dial((combo))$
- $(combo)$ acts as run-time variable
  $\Rightarrow$ value of $(combo)$ only known at run-time
**UNIX domain**

- Directory structure: $\text{indir}(f, d)$, $(\text{pwd}) = \text{root}$
- File `paper.tex` in directory $kr$ or planning $(K_x)$
- Goal: $K(\text{indir}(\text{paper.tex}, (\text{pwd})))$

<table>
<thead>
<tr>
<th>Action</th>
<th>Precondition</th>
<th>Effects</th>
</tr>
</thead>
</table>
| $\text{cd-down} (x)$ | $K(\text{directory}(x))$  
  $K(\text{indir}(x, (\text{pwd})))$ | $\text{add}(K_f, (\text{pwd}) = x)$ |
| $\text{cd-up} (x)$      | $K(\text{directory}(x))$  
  $K(\text{indir}((\text{pwd}), x))$ | $\text{add}(K_f, (\text{pwd}) = x)$ |
| $\text{ls} (x, y)$      | $K(\text{file}(x))$  
  $K((\text{pwd}) = y)$ | $\text{add}(K_w, \text{indir}(x, y))$ |
UNIX domain...

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

\[
\begin{align*}
\text{papers} & \quad \text{mail} \\
\text{kr} & \quad \text{aips} \\
\text{paper.tex?} & \quad \text{planning}
\end{align*}
\]

Initial knowledge
UNIX domain...

Initial knowledge

(pwd) = root

cd_down(papers)

cd_down(kr)

ls(paper.tex, kr)

cd_up(papers)

cd_down(kr)

cd_down(aips)

cd_down(planning)

One possible plan

cd_down(papers)

papers      mail

kr          aips

kr

paper.tex?  planning
UNIX domain...

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

- papers
- mail

- kr
- aips
- \text{paper.tex}
- planning

\text{Branch on } \text{indir}(\text{papers, kr})

\begin{align*}
\text{cd\_down(papers)} & \quad \text{cd\_down(kr)} \\
\text{ls(paper.tex, kr)} & \quad \text{cd\_up(papers)} \\
& \quad K^+ \\
& \quad K^- \\
\text{cd\_down(kr)} & \quad \text{cd\_down(aips)} \\
& \quad \text{cd\_down(planning)}
\end{align*}

Initial knowledge

One possible plan
Conclusions

- Planning with incomplete knowledge and sensing
  ⇒ Model problems at knowledge level
- Directly model changes to the agent’s knowledge
- Trade-off: restricted representation versus ability to abstract many assertions about knowledge
- Features: functions, run time variables
- Empirical results: blind search (no search control)
  ⇒ Many problems trivial, approach very promising
- Extensions: additional types of knowledge, improved search control, conversion of actions from world-level effects to knowledge-level effects