A complete reasoning procedure in the presence of incomplete information

Presented by Tu Phan

Outline

- Reasoning under Incomplete Information
  - Possible World Semantics vs Approximation Semantics
- An alternative approach
- Application to Conformant Planning
- Related work
- Conclusion
A running example

- **Bomb-in-the-toilet domain**
  - one package, one toilet
  - the package may or may not contain a bomb
  - dunking the package causes the bomb to be disarmed if it is armed
  - flushing the toilet makes it unclogged

- **Question:**
  - Whether or not the bomb is disarmed after flushing the toilet and then dunking the package

- **Domain Description D₁** (in language A)
  - **executable** dunk if ¬clogged
  - dunk causes ¬armed if armed
  - dunk causes clogged
  - flush causes ¬clogged

- **Initial condition:** I₁ = ∅

- **Question:** (D₁, I₁) ⊬ ¬armed after [flush;dunk]?

---

Reasoning under Incomplete Information

**Possible World Semantics vs. Approximation Semantics**

**Possible World Semantics**

<table>
<thead>
<tr>
<th>armed</th>
<th>clogged</th>
<th>¬armed</th>
<th>¬clogged</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flush</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dunk</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>¬armed</td>
<td>clogged</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Approximation Semantics**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>flush</td>
<td>¬clogged</td>
</tr>
<tr>
<td>dunk</td>
<td>clogged</td>
</tr>
</tbody>
</table>

(D₁, I₁) ⊬ ¬clogged after [flush]
(D₁, I₁) ⊬ ¬armed after [flush;dunk]
(D₁, I₁) ⊬ ¬clogged after [flush]
(D₁, I₁) not ⊬ ¬armed after [flush;dunk]
## Reasoning under Incomplete Information

**Possible World Semantics vs Approximation Semantics**

<table>
<thead>
<tr>
<th>Possible World Semantics</th>
<th>Approximation Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>more answers</td>
<td>less answers:</td>
</tr>
<tr>
<td>(D1,I1) \models_p \neg armed after [flush;dunk]</td>
<td>(D1,I1) \not\models_p \neg armed after [flush;dunk]</td>
</tr>
<tr>
<td>less efficient: at any time, number of possible worlds may be very big</td>
<td>more efficient: consider only one partial state at a time</td>
</tr>
<tr>
<td></td>
<td>sound but incomplete w.r.t. possible world</td>
</tr>
</tbody>
</table>

## Alternative Approach

- Based on the approximation semantics
- In the beginning, consider a set of partial states rather than a single one
  - done by partitioning over the truth values of some unknown fluents
Alternative Approach

Partition over \{armed\} | Partition over \{clogged\}

Question:
1. What makes it different between \texttt{armed} and \texttt{clogged}?

2. What fluent(s) should be chosen to partition the initial partial state?

\{(D_1, I_1) \models_{A} \neg \text{clogged} \text{ after } \text{[flush]}\} \quad \{(D_1, I_1) \not\models_{A} \neg \text{armed} \text{ after } \text{[flush; dunk]}\}

Decisive sets of fluents

- A set of unknown fluents is \textit{decisive} if it can be used to partition the initial partial state in order for $\models_{A}$ to be complete

  - E.g.:
    - $\{\text{armed}\}, \{\text{armed, clogged}\}$ are decisive
    - $\{\text{clogged}\}$ is not decisive
Algorithm for computing a decisive set of fluents

- **Objectives**
  - computationally efficient
  - returned decisive set should be as small as possible
    - help reduce the search space

- **Method**
  - Based on the concept of dependencies

**Dependencies**

- A literal \( l \) depends on a literal \( l_1 \) if either
  - \( l = l_1 \)
  - \( \neg l \) depends on \( \neg l_1 \)
  - there exists a causes \( l \) if \( p \) s.t. \( l_1 \in p \), or
  - there exists \( l_2 \) such that \( l \) depends on \( l_2 \) and \( l_2 \) depends on \( l_1 \)

**Domain**

- executable dunk if
  - \( \neg \text{clogged} \)
- dunk causes \( \neg \text{armed if armed} \)
- dunk causes clogged
- flush causes \( \neg \text{clogged} \)

**Dependencies**

- \( \Omega(\text{armed}) = \{\text{armed, } \neg \text{armed}\} \)
- \( \Omega(\text{clogged}) = \{\text{clogged}\} \)
Computing a decisive set of fluents

- Decisive(D,I)
  - Initialize $F = \emptyset$
  - Compute the dependency relationship
  - For each unknown fluent $f$
    - if there exists $l$ s.t. $l$ depends on both $f$ and $\neg f$ then
      - $F = F \cup \{f\}$
    - return $F$

- Theorem:
  - Decisive(D,I) is a decisive set of fluents, provided that every action has at most one executability condition

Example

- Domain $D_1$
  - executable: dunk if $\neg$ clogged
  - dunk causes $\neg$ armed if armed
  - dunk causes clogged
  - flush causes $\neg$ clogged

- Initial Partial State: $I_1 = \emptyset$

- Dependencies
  - $\Omega(\text{armed}) = \{\text{armed}, \neg\text{armed}\}$
  - $\Omega(\neg\text{armed}) = \{\neg\text{armed}, \text{armed}\}$
  - $\Omega(\text{clogged}) = \{\text{clogged}\}$
  - $\Omega(\neg\text{clogged}) = \{\neg\text{clogged}\}$

- $\text{DECISIVE}(D_1,I_1) = \{\text{armed}\}$
  - armed depends on both armed and $\neg$ armed
  - no literal $l$ such that $l$ depends on both clogged and $\neg$ clogged
New approach properties

- Sound and complete
- More compact than possible world semantics in many cases
- Computing a decisive set of fluents can be done in polynomial time

What are missing?

- What if we have more than one executability condition for an action
  - Solution: Dependencies between actions and literals

- What if we have more than one initial partial state, for example initially \( f \mid g \)
  - Solution:
    - For each partial state, partition it using the same procedure for computing a decisive set

- Static causal laws:
  - At present, we only have the result for domains with static causal laws whose body contains at most one literal
Application – Conformant Planning

- Problem
  - Given: an action theory (D,I), and a set G of literals
  - Find: a sequence of actions that, when executed in any possible initial state, always achieves G

- Our approach
  - If the goal is \{-clogged\} then do not need to partition I
  - Modify algorithm DECISIVE(D,I) so as to take

Computing a decisive set of fluents

- Decisive(D,I,G)
  - Initialize F = ∅
  - Compute the dependency relationship
  - For each unknown fluent f
    - if there exists l ∈ G s.t. l depends on both f and ¬f
      - then
        - F = F ∪ {f}
  - return F

- Theorem:
  - Decisive(D,I,G) is a decisive set of fluents for planning problem (D,I,G)
Computing a decisive set

Example

- **Domain D₁**
  - executable dunk if ¬clogged
  - dunk causes ¬armed if armed
  - dunk causes clogged
  - flush causes ¬clogged

- **Initial Partial State:** \( I₁ = \emptyset \)

- **Dependencies**
  - \( \Omega(\text{armed}) = \{\text{armed}, \neg\text{armed}\} \)
  - \( \Omega(\neg\text{armed}) = \{\neg\text{armed}, \text{armed}\} \)
  - \( \Omega(\text{clogged}) = \{\text{clogged}\} \)
  - \( \Omega(\neg\text{clogged}) = \{\neg\text{clogged}\} \)

- **DECISIVE(D₁,I₁,\{clogged\}) = \emptyset**
  - no fluent \( f \) such that clogged depends on both \( f \) and \( \neg f \)

- **DECISIVE(D₁,I₁,\{\neg\text{armed}\}) = \{\text{armed}\}**
  - \( \neg\text{armed} \) depends on both armed and \( \neg\text{armed} \)

---

**CPA+ - A Conformant Planner**

- Problem Description
- CPA+
  - SCAN AND PARSER MODULE
  - PREPROCESSING MODULE
    - Computing a decisive set of fluents
  - SEARCH MODULE
    - Heuristic: number of fulfilled subgoals
  - Solutions
CPA+ - Performance

<table>
<thead>
<tr>
<th>Problem</th>
<th>KACMBP</th>
<th>CFF</th>
<th>POND</th>
<th>CPA⁺</th>
<th>CPA⁺</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PL</td>
<td>Time</td>
<td>PL</td>
<td>Time</td>
<td>PL</td>
</tr>
<tr>
<td>bomb(10,1)</td>
<td>19</td>
<td>0.01</td>
<td>19</td>
<td>0.05</td>
<td>19</td>
</tr>
<tr>
<td>bomb(50,1)</td>
<td>99</td>
<td>0.51</td>
<td>99</td>
<td>5.33</td>
<td>AB</td>
</tr>
<tr>
<td>bomb(100,1)</td>
<td>199</td>
<td>3.89</td>
<td>199</td>
<td>121.8</td>
<td>AB</td>
</tr>
<tr>
<td>bomb(10,5)</td>
<td>15</td>
<td>0.09</td>
<td>15</td>
<td>0.07</td>
<td>AB</td>
</tr>
<tr>
<td>bomb(50,5)</td>
<td>95</td>
<td>1.66</td>
<td>95</td>
<td>4.7</td>
<td>AB</td>
</tr>
<tr>
<td>bomb(100,5)</td>
<td>195</td>
<td>6.92</td>
<td>195</td>
<td>113.95</td>
<td>AB</td>
</tr>
<tr>
<td>bomb(10,10)</td>
<td>10</td>
<td>0.3</td>
<td>10</td>
<td>0.05</td>
<td>AB</td>
</tr>
<tr>
<td>bomb(50,10)</td>
<td>90</td>
<td>5.39</td>
<td>90</td>
<td>4.04</td>
<td>AB</td>
</tr>
<tr>
<td>bomb(100,10)</td>
<td>190</td>
<td>35.83</td>
<td>190</td>
<td>102.56</td>
<td>AB</td>
</tr>
</tbody>
</table>

Bomb in the toilet domain
CPA⁺: using possible world semantics
CPA⁺: using the new approach
PL: Plan length; TO: Time out; AB: Abnormal Termination
Times are in seconds

Related Work

- Approximations
  - [Son & Chitta, AI 2001]
  - [Son, Tu & Chitta, LPNMR 2004]
  - [Son, Tu, Gelfond & Ricardo, AAAI 2005]
  - [Son, Tu, Gelfond & Ricardo, LPNMR 2005]

- Irrelevant Information
  - [Nebel, Dimopoulos & Koehler, ECP 1997]

- Reduced sets of actions
  - [Haslum & Johnsson, ICAPS 2000]

- Isolated sets of actions and fluents
  - [Lifschitz & Ren 2004]
Conclusion

- New Approach
  - Sound and Complete
  - Efficient

- Application to conformant planning
  - A conformant planner competitive with other state-of-the-art planners