Situation Calculus

Introduction to Artificial Intelligence
Cse 150
The Midterm

- Will cover Chapters 1-9
- Yes, it will graded on a curve

Other News

- Charles Elkan will lecture on Tuesday
- What he covers most likely won’t be on the midterm, but will be on the final!
Where have we been?

- Solving problems by search
- Game playing
- Using logic to make inferences

Where are we going?

- Planning
- A bit of uncertainty and probabilistic reasoning
This lecture

• Situation Calculus
• Planning

Reading

• Chapter 10
Planning

Find a sequence of operator instances that transform an initial state into a state in which the goal is satisfied.

Issues:

1. Representation of states
2. Representation of actions
3. Representation of goals
4. Representation of plans
Wumpus world: Building an agent using FOL

• The knowledge base will start with axioms and definitions about the wumpus world.

• Since the agent is perceiving and acting over time, we can introduce the notion of time as an object in the database.

• Suppose a wumpus-world agent is using a FOL KB and perceives a smell and a breeze (but no glitter) at t=5:

  • We can assert this percept (add it to the KB) using:
    \[ \text{Tell(KB, Percept([Smell,Breeze,None], 5))} \]
Wumpus world: Building an agent using FOL

• We can ask the KB what action to do:
  \[ \text{Ask}(KB, \exists a \ \text{Action}(a,5)) \]

• The FOL system would then have to infer whether the KB entails any particular actions for \( t=5 \).
• Answer: Yes, \{a/Shoot\} \leftarrow substitution (binding list)
  – [USING BOOK’S NOTATION!!!]
• Given a sentence \( S \) and a substitution \( \sigma \), \( S\sigma \) denotes the result of substituting (plugging in) \( \sigma \) into \( S \)
  – \( S = \text{Bigger}(x,y) \)
  – \( \sigma = \{x/Cow, y/Lamb\} \)
  – \( S\sigma = \text{Bigger}(\text{Cow, Lamb}) \)

• \( \text{Ask}(KB, S) \) returns some/all \( \sigma \) such that \( KB \models S\sigma \)
Deducing Hidden Properties

• “Squares are breezy near a pit.”

• **Diagnostic rule** – infer cause from effect
  \[ \forall y \text{ Breezy}(y) \Rightarrow [\exists x \text{ Pit}(x) \land \text{Adjacent}(x,y)] \]

• **Causal rule** – infer effect from cause
  \[ \forall x,y \text{ Pit}(x) \land \text{Adjacent}(x,y) \Rightarrow \text{Breezy}(y) \]

• Neither of these is complete – e.g. diagnostic rule doesn’t tell us that if there is no breeze then there isn’t a pit nearby.

• Definition of Breezy predicate:
  \[ \forall y \text{ Breezy}(y) \Leftrightarrow [\exists x \text{ Pit}(x) \land \text{Adjacent}(x,y)] \]
Keeping track of change

• Facts hold in situations, rather than eternally
  E.g., Holding(Gold,Now) rather than just Holding(Gold)

• The **Situation calculus** is one way to represent change in FOL:
  – Adds a situation argument to each non-eternal predicate
  – E.g., Now in Holding(Gold,Now) denotes a situation

• Situations are connected by the **Result** function
  – Result(a,s) is the situation that results from doing a is s
    e.g. $S_1 = \text{Result}(a,S_0)$
Describing actions I

- "Effect" axiom---describe changes due to action
  \[ \forall s \ AtGold(s) \Rightarrow Holding(Gold, Result(Grab, s)) \]

- "Frame" axiom---describe non-changes due to action
  \[ \forall s \ HaveArrow(s) \Rightarrow HaveArrow(Result(Grab, s)) \]

**Frame problem:** find an elegant way to handle non-change
  - (a) representation—avoid frame axioms
  - (b) inference—avoid repeated "copy-overs" to keep track of state

**Qualification problem:** true descriptions of real actions require endless caveats—what if gold is slippery or nailed down or …

**Ramification problem:** real actions have many secondary consequences
  —what about the dust on the gold, wear and tear on gloves, …
Describing actions II

Successor-state axioms solve the representational frame problem
– Combine effect and frame axioms
– List all ways in which a predicate can be true or become false

• Each axiom is ``about'' a **predicate** (not an action per se):
  P true afterwards \(\iff\)
  \[\text{[an action made } P \text{ true } \lor P \text{ true already and no action made } P \text{ false]}\]

• For holding the gold:
  \(\forall a,s \; \text{Holding}(Gold, Result(a,s)) \iff\)
  \[\text{[(a = Grab } \land \text{ AtGold}(s)) \lor \text{ }}\]
  \((\text{Holding}(Gold,s) \land a \neq \text{ Release})]\]
Formulate planning as inferring actions or action sequences on a situation calculus knowledge base
Planning in situation calculus

- Situations are connected by the \textbf{Result} function
  e.g. $s_1 = \text{Result}(a, s_0)$ is the function giving the situation that results from doing action $a$ while in situation $s_0$

Here, we represent an action sequence (plan) by a list $[\text{first} \mid \text{rest}]$ where \textbf{first} is the initial action and \textbf{rest} is a list of remaining actions.

$\text{PlanResult}(p, s)$ generalizes \textbf{Result} to give the situation resulting from executing $p$ starting from state $s$:

- $\forall s \ \text{PlanResult}([], s) = s$
- $\forall s, a, p \ \text{PlanResult}([a \mid p], s) = \text{PlanResult}(p, \text{Result}(a, s))$
Planning in situation calculus

Consider the task: *get milk, bananas, and a cordless drill*

Initial state:

\[ \text{At(Home,}S_0) \land \neg \text{Have(Milk,}S_0) \land \neg \text{Have(Bananas,}S_0) \land \ldots \]  

Actions as Successor State axioms:

\[ \text{Have(Milk,Result(a,}s)) \iff \]  

\[ [(a=\text{Buy(Milk)} \land \text{At(Supermarket,}s)) \lor (\text{Have(Milk,}s) \land a \neq \text{Drop(Milk)})] \]

Query:

\[ \exists p \ (s=\text{PlanResult(p,}S_0)) \land \text{At(Home,}s) \land \text{Have(Milk,}s) \land \text{Have(Bananas,}s) \land \ldots \]

Solution

\[ p = [\text{Go(Supermarket)}, \text{Buy(Milk)}, \text{Buy(Bananas)}, \text{Go(HWS)}, \text{Buy(Drill)}, \ldots ] \]
Problems of using inference procedure & situation calculus

- Branching factor can be high
- Inference procedure finds a valid sequence of actions, but might contain actions that are irrelevant
  - e.g., \([ A^{-1}, A \mid p ]\)
    - \[[ \text{Go to school, Go home, Go to HWS, Buy Drill} \ldots\]\n  - e.g., actions that are completely irrelevant to goal
    - \[[ \text{Go to HWS, Read New York Times, Buy Drill,} \ldots\]\n
The alternative

1. Restrict language when defining problems.
2. Use special purpose algorithm (i.e. a planner).
Search vs. Planning

Consider the task: \textit{get milk, bananas, and a cordless drill}

- Standard search algorithms seem to fail miserably:

- Why? Huge branching factor & heuristics & goal tests are \textit{inadequate}
Search vs. planning contd.

Planning systems do the following:

1) open up action and goal representation to allow selection
2) divide-and-conquer by subgoaling
3) relax requirement for sequential construction of solutions

<table>
<thead>
<tr>
<th>Search</th>
<th>Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>States</strong></td>
<td>Logical sentences</td>
</tr>
<tr>
<td><strong>Actions</strong></td>
<td>Preconditions/outcomes</td>
</tr>
<tr>
<td><strong>Goal</strong></td>
<td>Logical sentence (conjunction)</td>
</tr>
<tr>
<td><strong>Plan</strong></td>
<td>Constraints on actions</td>
</tr>
</tbody>
</table>

- **States**: Lisp data structures vs. Logical sentences
- **Actions**: Lisp code vs. Preconditions/outcomes
- **Goal**: Lisp code (func of state) vs. Logical sentence (conjunction)
- **Plan**: Sequence from $S_0$ vs. Constraints on actions
The STRIPS language

- More restrictive way to express states, plans and goals, but leads to more efficient plan determination.

- **States**: Represented as conjunctions of function-free ground literals (Predicates applied to constant symbols, possibly negated).

- **Initial state**: just a state description as above, e.g.
  \[\text{At(Home)} \land \neg \text{Have(Milk)} \land \neg \text{Have(Bananas)} \land \neg \text{Have(Drill)}\ldots\]

- **Goals**: Represented as conjunctions of literals
  \[\text{At(Home)} \land \text{Have(Milk)} \land \text{Have(Bananas)}\]

  May contain variables (existentially quantified), hence goal may represent more than one state.
  \[\text{At(x)} \land \text{Sells(x, Bananas)}\]

- **Actions**: Strips operators
STRIPS operators: A restrictive language

**Action description:** name & variable symbols
**Precondition:** Conjunction of positive literals
**Effect:** conjunction of literals (positive or negative)

Example:

**Action:** Buy(x)
**Precondition:** At(p), Sells(p, x)
**Effect:** Have(x)

- **Operator Schema:** an operator with variables
- An operator is **applicable** in state s if there is some way to instantiate the variables so every precondition is true in s
- In STRIPS, No situation variables – situation variables are implicit.
A note ...

STRIPS: STanford Research Institute Problem Solver

Original notation in STRIPS language found elsewhere:
- Precondition
- Delete List
- Add List

Shakey

Borrowed/edited from David Kriegman, slides
A simple STRIPS Planner: Progression Planner

• A plan is a sequence of STRIPS operators

• From initial state, search forward by selecting operators whose preconditions can be unified with the literals in the state.

• New state includes positive literals of effect; the negated literals of effect are “deleted” from state.

• Search forward until goal unifies with resulting state

• This is just state-space searching using STRIPS operators.