

Situation Calculus

Introduction to Artificial Intelligence

Cse 150

The Midterm

- Will cover Chapters 1-9
- Yes, it will be 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:
`Tell(KB, Percept([Smell,Breeze,None], 5))`

Wumpus world: Building an agent using FOL

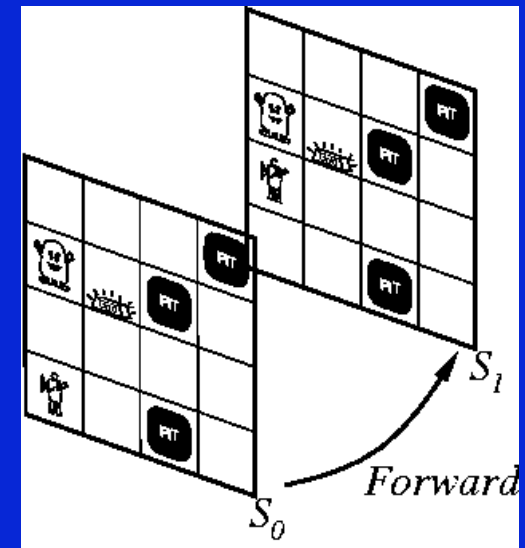
- We can ask the KB what action to do:
 $\text{Ask}(\text{KB}, \exists a \text{ Action}(a,5))$
- The FOL system would then have to infer whether the KB entails any particular actions for $t=5$.
- Answer: Yes, $\{a/\text{Shoot}\} \leftarrow$ substitution (binding list)
 - [USING BOOK'S NOTATION!!!]
- Given a sentence S and a substitution σ , $S\sigma$ denotes the result of substituting (plugging in) σ into S
 - $S = \text{Bigger}(x,y)$
 - $\sigma = \{x/\text{Cow}, y/\text{Lamb}\}$
 - $S\sigma = \text{Bigger}(\text{Cow}, \text{Lamb})$
- $\text{Ask}(\text{KB}, S)$ returns some/all σ such that $\text{KB} \models S\sigma$

Deducing Hidden Properties

- “Squares are breezy near a pit.”
- **Diagnostic rule** – infer cause from effect
$$\forall y \text{ Breezy}(y) \Rightarrow [\exists x \text{ Pit}(x) \wedge \text{Adjacent}(x,y)]$$
- **Causal rule** – infer effect from cause
$$\forall x,y \text{ Pit}(x) \wedge \text{Adjacent}(x,y) \Rightarrow \text{Breezy}(y)$$
- 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:
$$\forall y \text{ Breezy}(y) \Leftrightarrow [\exists x \text{ Pit}(x) \wedge \text{Adjacent}(x,y)]$$

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
 $\forall s \text{ AtGold}(s) \Rightarrow \text{Holding}(\text{Gold}, \text{Result}(\text{Grab}, s))$
- ``Frame" axiom---describe non-changes due to action
 $\forall s \text{ HaveArrow}(s) \Rightarrow \text{HaveArrow}(\text{Result}(\text{Grab}, s))$

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

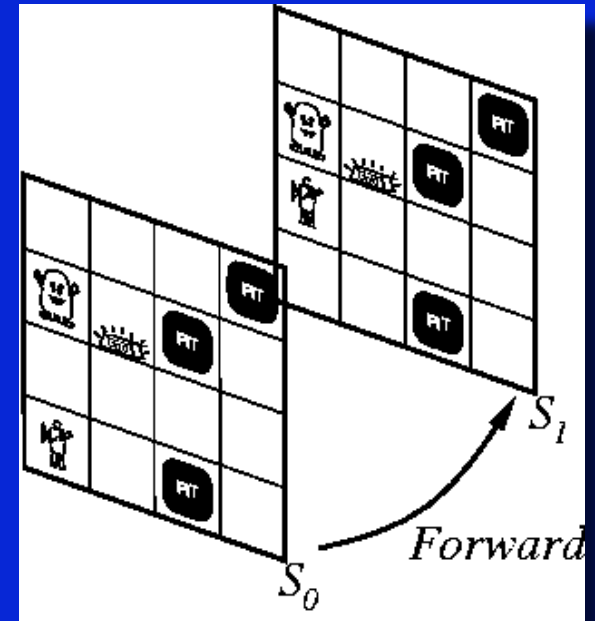
- Combine effect and frame axioms
- List all ways in which a predicate can be true or become false
- Each axiom is "about" a **predicate** (not an action per se):
 P true afterwards \Leftrightarrow
 [an action made P true
 \vee P true already and no action made P false]
- For holding the gold:
 $\forall a,s \text{ Holding}(\text{Gold}, \text{Result}(a,s)) \Leftrightarrow$
 [($a = \text{Grab} \wedge \text{AtGold}(s)$) \vee
 ($\text{Holding}(\text{Gold},s) \wedge a \neq \text{Release}$)]

Making plans

Formulate planning as inferring actions or action sequences on a situation calculus knowledge base

Planning in situation calculus

- Situations are connected by the **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 **[first | rest]** where **first** is the initial action and **rest** is a list of remaining actions.

PlanResult(p,s) generalizes **Result** to give the situation resulting from executing **p** starting from state **s**:

$$\forall s \text{ PlanResult}([], s) = s$$

$$\forall s, a, p \text{ PlanResult}([a | p], s) = \text{PlanResult}(p, \text{Result}(a, s))$$

Planning in situation calculus

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

Initial state:

$At(Home, S_0) \wedge \neg Have(Milk, S_0) \wedge \neg Have(Bananas, S_0) \wedge \dots$

Actions as Successor State axioms:

$Have(Milk, Result(a, s)) \Leftrightarrow$
 $[(a = Buy(Milk) \wedge At(Supermarket, s)) \vee (Have(Milk, s) \wedge a \neq Drop(Milk))]$

Query:

$\exists p (s = PlanResult(p, S_0) \wedge At(Home, s) \wedge Have(Milk, s)$
 $\wedge Have(Bananas, s) \wedge \dots$

Solution

$p = [Go(Supermarket), Buy(Milk), Buy(Bananas), Go(HWS), Buy(Drill), \dots]$

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 | p]
[Go to school, Go home, Go to HWS, Buy Drill ...]
 - e.g., actions that are completely irrelevant to goal
[Go to HWS, Read New York Times, Buy Drill, ...]

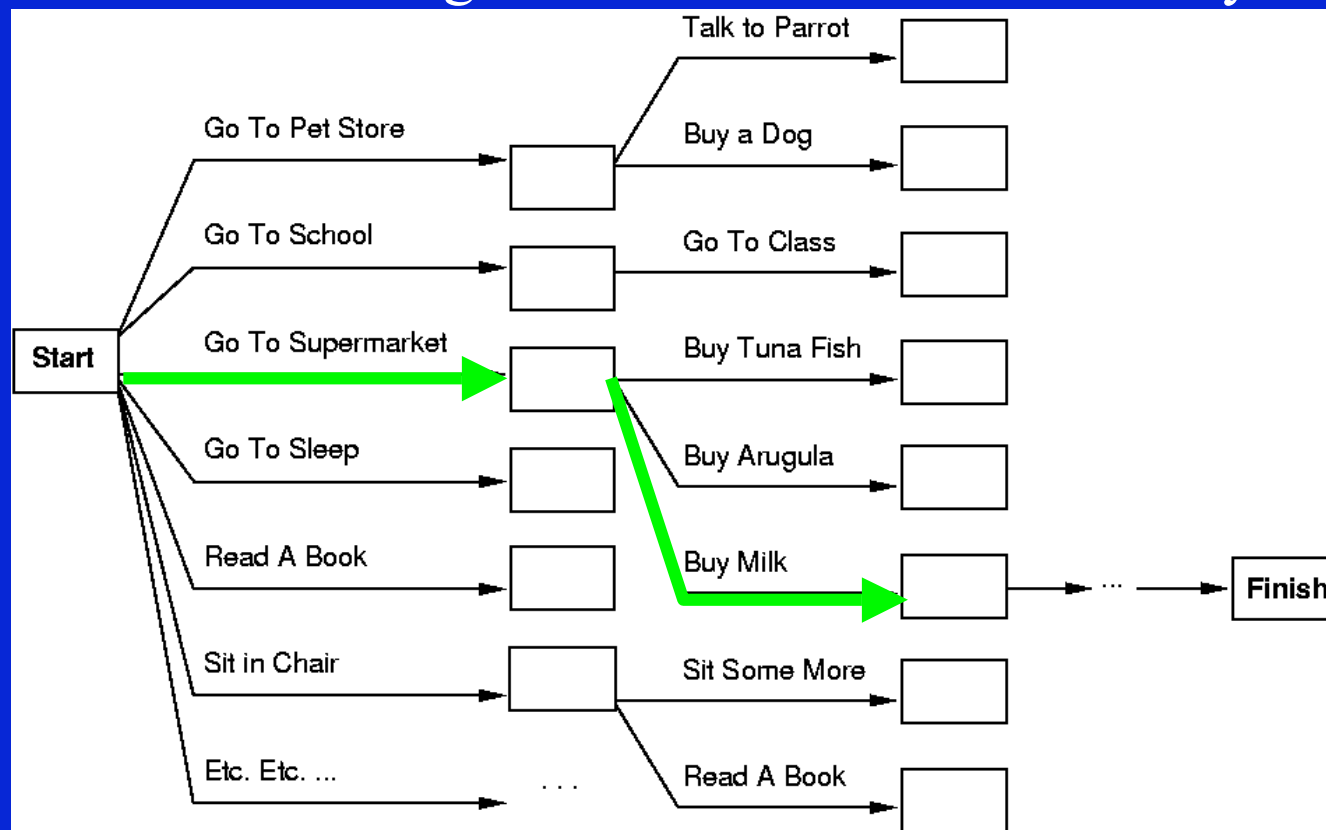
The alternative

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

Search vs. Planning

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

- Standard search algorithms seem to fail miserably:



- Why? Huge branching factor & heuristics & goal tests are 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

	Search	Planning
States	Lisp data structures	Logical sentences
Actions	Lisp code	Preconditions/outcomes
Goal	Lisp code (func of state)	Logical sentence (conjunction)
Plan	Sequence from S_0	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.

$At(Home) \wedge \neg Have(Milk) \wedge \neg Have(Bananas) \wedge \neg Have(Drill) \wedge \dots$

- **Goals:** Represented as conjunctions of literals

$At(Home) \wedge Have(Milk) \wedge Have(Bananas)$

May contain variables (existentially quantified), hence goal may represent more than one state.

$At(x) \wedge Sells(x, Bananas)$

- **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)

A depiction of an
operator schema

At(p) Sells(p,x)

Buy(x)

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.

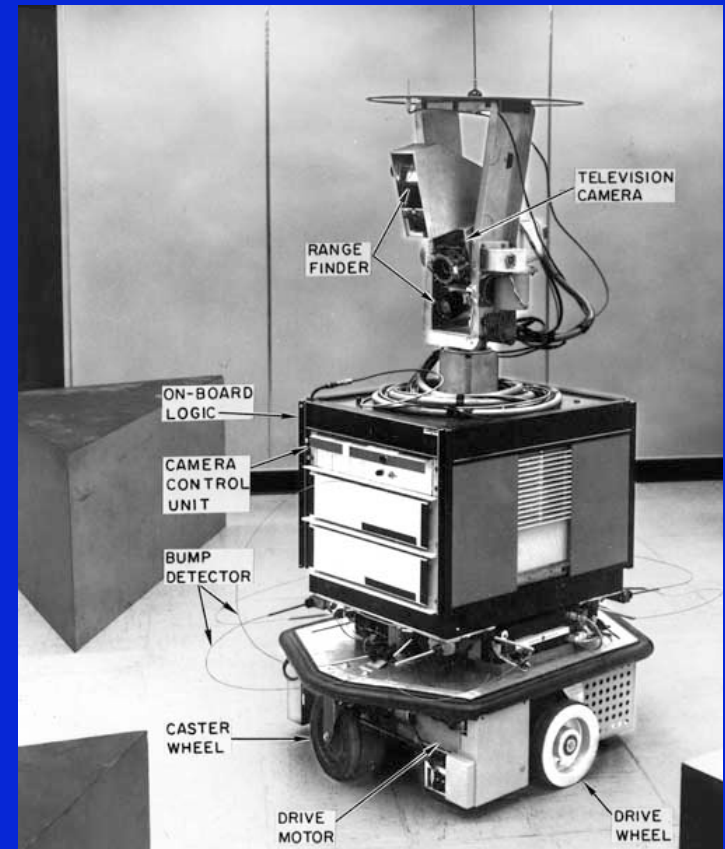
A note ...

STRIPS: STanford Research Institute Problem Solver

Original notation in STRIPS language found elsewhere:

- Precondition
- Delete List
- Add List

Shakey



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.