Planning II

Introduction to Artificial Intelligence

CSE 150
Lecture 12
May 15, 2007
Administration

• Your second to last Programming Assignment is up - start NOW, you don’t have a lot of time

• The PRIZE: Free lunch at the faculty club for the best checkers player program (with me & Matt)
• Second Prize: Two lunches at the faculty club with me and Matt…;(-)
Last lecture

• Planning
  – Situation calculus

This lecture

• Planning
  – STRIPS language
  – Progression/Regression planning
  – Partially ordered plans

Thursday lecture

• We will go over the midterm, and more on planning

Reading

• Read Chapter 10.1-10.3 to get more background on Elkan’s lecture
• For today and Thursday’s lecture, please read chapter 11.
FOL and Situation Calculus

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

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

• Situations are connected by the \textbf{Result} function
  – \text{Result}(a,s) is the situation that results from doing \text{a} is \text{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 ⇔ 
    - [an action made P true 
    - ∨ P true already and no action made P false]

- Example successor-state axioms for holding the gold:
  - ∀a,s Holding(Gold,Result(a,s)) ⇔ 
    - [(a = Grab ∧ AtGold(s)) ∨ 
    - (Holding(Gold,s) ∧ a ≠ Release)]
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).

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

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

• **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)`

- Positive literals in effect are the “add list”.
- Negative literals in effects are the “delete list”

**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 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.
Searching through the state space

Task: *get milk, bananas, and a cordless drill*

![Task Flow Chart]

- Start
  - Go To Pet Store
    - Buy a Dog
  - Go To School
    - Go To Class
  - Go To Supermarket
    - Buy Tuna Fish
    - Buy Arugula
    - Buy Milk
  - Go To Sleep
  - Read A Book
  - Sit in Chair
  - Etc. Etc. ...
  - ...
  - Finish
A simple STRIPS Planner: Regression Planner

• A plan is sequence of STRIPS operators

• The goal state must unify with at least one of the positive literals in operator’s effect

• Its preconditions must hold in the previous situation, and these become sub-goals which might be satisfied by the initial conditions.

• Perform backward chaining from goal.

• Again, this is just state-space searching using STRIPS operators.
Features/Bugs of STRIPS

• Very simple language: everything is propositional (why?)

• This makes planning efficient

• Unfortunately, it appears to be insufficient for many real-world domains.

• Enter the ADL language:
  – Positive AND negative literals in states
  – Quantified variables in goals (existentially quantified)
  – Goals allow conjunction AND disjunction
  – Conditional effects allowed: when P:E means E is an effect only if P is satisfied
  – Equality predicate built in
  – Variables can have types: p: Plane
State space vs. Plan space

An alternative is to search over the space of plans rather than space of situations.

Standard search: node = concrete world state
Planning search: node = partial plan

Gradually move from incomplete/vague plans to complete, correct plans by
1. Inserting operators to plan
2. Deciding upon order of operators
3. Binding variables to symbols

Principle of least commitment: defer making choice until necessary.
- Defer order of operators
- Defer binding variables
Partially ordered plans: socks & shoes

Operators

LeftSock
Precondition: none
Effect: LeftSockOn

RightSock
Precondition: none
Effect: RightSockOn

LeftShoe
Precondition: LeftSockOn
Effect: LeftShoeOn

RightShoe
Precondition: RightSockOn
Effect: RightShoeOn
From partially ordered plans to fully ordered plans

Partial Order Plan:

- Start
  - Left Sock
    - LeftSockOn
    - Left Shoe
      - LeftShoeOn, RightShoeOn
  - Right Sock
    - RightSockOn
    - Right Shoe
      - Finish

Total Order Plans:

- Start
  - Right Sock
    - RightSockOn
    - Right Shoe
      - Finish
  - Left Sock
    - LeftSockOn
    - Left Shoe
      - Finish
  - Left Sock
    - LeftSockOn
    - Left Shoe
      - Finish
  - Right Sock
    - RightSockOn
    - Right Shoe
      - Finish
  - Left Sock
    - LeftSockOn
    - Left Shoe
      - Finish
  - Right Sock
    - RightSockOn
    - Right Shoe
      - Finish
So, now what is a (partial) plan?

1. A set of plan steps, and each step is one of the operators (note, operators can be used more than once).

2. A set of ordering constraints $S_i < S_j$ --- Step $S_i$ must occur before step $S_j$ but not necessarily immediately before $S_j$.

3. A set of variable bindings for each step of form $v = x$ where $x$ is either a variable or a symbol.

4. A set of causal links $S_i --c--> S_j$ Interpreted as $S_i$ achieves precondition $c$ of $S_j$. 
The Initial Plan

• A plan with two steps (STRIPS operators):
  • **Start Step:**
    – Preconditions: None
    – Effect: Add all propositions that are initially true
  
  • **Finish Step:**
    – Precondition: Goal state
    – Effects: None

• **Steps:** \{ Start, Finish \}
• **Orderings:** \{ Start < Finish \}
• **Bindings:** \{ \}
• **Links:** \{ \}
What is a “solution” to a planning problem?

A solution is complete and consistent plan

• A plan is **complete** iff every precondition is achieved
  – A precondition is **achieved** iff
    1. it is the effect of an earlier step and
    2. no possibly intervening step undoes it

• A plan is **consistent** iff there are no contradictions in ordering or variable binding. Example contradictions:
  – $S_i < S_j$ and $S_j < S_i$
  – $v = A$ and $v = B$

• Every linearization of a solution is a solution

• Our planning procedure only creates consistent (partial) plans and process seeks to find a plan which is also complete
Operators in planning process

Operators on partial plans:
1. add a link from an existing action to an open precondition
2. add a step to fulfill an open condition
3. order one step wrt another

Def: open condition is a precondition of a step not yet fulfilled
Example: Simple Blocks World

Symbols: A, B, C, Table

Predicates:

ON(x,y) – Object x is on top of object y

CL(x) – x doesn’t have anything on top

Start State

On(A, Table)
On(B, Table)
ON(C, Table)
Cl(A)
Cl(B)
Cl(C)

Goal State

ON(B, A)
On(C, B)
Some pseudo-code for nondeterminism

- The POP algorithm is presented as a non-deterministic algorithm, BUT it could just as easily been presented using a more standard search paradigm.

- **Choose([a,b,c])** – each time this is called, another element of the list is selected in some order (random, in order, according to a heuristic, etc.)

- **Fail** – when executed, the pending choice is taken off the agenda and control is resumed at the point where the choice was made. State returns to that prior to the choice.

```plaintext
... foobar = Choose ([A1, B1, C1, D1])
...
bar = Choose([A2, B2, C2])
...
If <something> then Fail
...
```
A Partial Order Planning Algorithm

A partial order regression planner that searchers through plan space.

```
function POP(initial, goal, operators) returns plan
    plan ← MAKE-MINIMAL-PLAN(initial, goal)
    loop do
        if SOLUTION?(plan) then return plan
        S_{need}, c ← SELECT-SUBGOAL(plan)
        CHOOSE-OPERATOR(plan, operators, S_{need}, c)
        RESOLVE-THREATS(plan)
    end

function SELECT-SUBGOAL(plan) returns S_{need}, c
    pick a plan step S_{need} from STEPS(plan)
    with a precondition c that has not been achieved
    return S_{need}, c
```
proc 1. Choose-Operator (plan, operators, S\text{need}, c)

- choose a step S\text{add} from operators or STEPS(plan) that has c as an effect
- if there is no such step then fail
- add the causal link S\text{add} \rightarrow S\text{need} to LINKS(plan)
- add the ordering constraint S\text{add} \prec S\text{need} to ORDERINGS(plan)
- if S\text{add} is a newly added step then:
  - add S\text{add} to STEPS(plan)
  - add Start \prec S\text{add} \prec Finish to ORDERINGS(plan)

proc 2. Resolve-Threats(plan)

- for each S\text{threat} that threatens a link S_i \rightarrow S_j in LINKS(plan) do
  - choose either
    - Demotion: Add S\text{threat} \prec S_i to ORDERINGS(plan)
    - Promotion: Add S_j \prec S\text{threat} to ORDERINGS(plan)
  - if not CONSISTENT(plan) then fail
- end
Example: Blocks World

- Symbols: Table, A, B, C
- Predicates: On (x, y), Cl(x)

Operators

\[
\begin{align*}
\text{Clear}(x) \land \text{On}(x, z) \land \text{Clear}(y) & \quad \text{Clear}(x) \land \text{On}(x, z) \\
\text{PutOn}(x, y) & \quad \text{PutOnTable}(x) \\
\sim \text{On}(x, z) \land \sim \text{Clear}(y) & \quad \sim \text{On}(x, z) \land \text{Clear}(z) \land \text{On}(x, \text{Table}) \\
\text{Clear}(z) \land \text{On}(x, y) &
\end{align*}
\]
Example: Simple Blocks World

Symbols: A, B, C, Table
Predicates:
ON(x,y) – Object x is on top of object y
CL(x) – x doesn’t have anything on top

Start State
- On(A, Table)
- On(B, Table)
- ON(C, Table)
- CL(A)
- CL(B)
- CL(C)

Goal State
- ON(B, A)
- On(C, B)
Example: Simple Blocks World

Steps:
{ Start, Finish, }

Orderings:
{ Start < Finish }

Links:
{}
Example: Simple Blocks World

On(A, Table) On(B, Table) ON(C, Table) Cl(A) Cl(B) Cl(C)

Steps:
{ Start, Finish, S1}

Orderings:
{ Start < Finish, Start < S1, S1 < Finish}

Links:
{S1-c1->Finish}

Start

S1: PutOn(B, A)

~On(B, Table) ~Cl(A) Cl(Table), On(B, A)

ON(B, A) On(C, B)

Finish
Example: Simple Blocks World

Steps:
{ Start, Finish, S1}

Orderings:
{ Start < Finish, Start < S1, S1 < Finish}

Links:
{S1-c1->Finish
Start –c1->S1}

On(A, Table) On(B, Table) ON(C,Table) Cl(A) Cl(B) Cl(C)

Start

Finish

Steps:
{ Start, Finish, S1}

Orderings:
{ Start < Finish, Start < S1, S1 < Finish}

Links:
{S1-c1->Finish
Start –c1->S1}

On(A, Table) On(B, Table) ON(C,Table) Cl(A) Cl(B) Cl(C)

On(B,Table) Cl(A) Cl(B)

S1: PutOn(B,A)

~On(B,Table) ~Cl (A)
Cl (Table), On(B,A)

ON(B, A) On(C, B)
Example: Simple Blocks World

Steps: 
{ Start, Finish, S1 }

Orderings: 
{ Start < Finish, Start < S1, S1 < Finish }

Links: 
{ S1-c1->Finish
Start –c1->S1
Start –c2->S1 }

On(A, Table) On(B, Table) ON(C, Table) Cl(A) Cl(B) Cl(C)

Start

S1: PutOn(B, A)

On(B, Table) Cl(A) Cl(B)

~On(B, Table) ~Cl (A)
Cl (Table), On(B, A)

ON(B, A) On(C, B)

Finish

CSE 150, Spring 2007

Slides edited from ones provided by David Kriegman, who started with Russell & Norvig’s slides…
Example: Simple Blocks World

Steps:
{ Start, Finish, S1 }

Orderings:
{ Start < Finish, Start < S1, S1 < Finish }

Links:
{ S1-c1->Finish
Start – c1->S1
Start – c2->S1
Start – c3->S1 }

On(A, Table) On(B, Table) ON(C, Table) Cl(A) Cl(B) Cl(C)

Start

S1: PutOn(B,A)

~On(B,Table) ~Cl(A) Cl(Table), On(B, A)

ON(B, A) On(C, B)

Finish
Example: Simple Blocks World

On(A, Table) On(B, Table) ON(C, Table) Cl(A) Cl(B) Cl(C)

Steps:
{ Start, Finish, S1, S2 }

Orderings:
{ Start < Finish, Start < S1, S1 < Finish, Start < S2, S2 < Finish }

Links:
{ S1-c1->Finish
Start –c1->S1
Start –c2->S1
Start –c3->S1
S2 –c2->Finish
Start –c1->S2 … }

S1: PutOn(B,A)

~On(B,Table) ~Cl (A)
Cl (Table), On(B,A)

S2: PutOn(C,B)

~On(C,Table) ~Cl (B)
Cl (Table), On(C,B)

Start

Finish

ON(B, A) On(C, B)
But wait!!!

Can you really execute these two steps in either order?
Example: Simple Blocks World

Steps:
\{ \text{Start, Finish, S1, S2} \}

Orderings:
\{ \text{Start < Finish, Start < S1, S1 < Finish, Start < S2, S2 < Finish, S1 < S2} \}

Links:
\{ \text{S1-c1->Finish} \}
\text{Start –c1->S1} \text{ Start –c2->S1} \text{ Start –c3->S1} \text{ S2 –c2->Finish} \text{ Start –c1->S2} \ldots \}

On(A, Table) On(B, Table) ON(C,Table) Cl(A) Cl(B) Cl(C)

On(B,Table) Cl(A) Cl(B)

\text{S1: PutOn(B, A)}

\text{~On(B,Table)} \text{ ~Cl (A)}
\text{Cl (Table), On(B,A)}

\text{ON(B, A) On(C, B)}

On(C,Table) Cl(C) Cl(B)

\text{S2: PutOn(C, B)}

\text{~On(C,Table)} \text{ ~Cl (B)}
\text{Cl (Table), On(C,B)}

Start

Finish

C C B B A A
A Partial Order Planning Algorithm

A partial order regression planner that searches through plan space.

function POP(initial, goal, operators) returns plan

    plan ← MAKE-MINIMAL-PLAN(initial, goal)

    loop do
        if SOLUTION?(plan) then return plan
        S_{need}, c ← SELECT-SUBGOAL(plan)
        CHOOSE-OPERATOR(plan, operators, S_{need}, c)
        RESOLVE-THREATS(plan)
    end

function SELECT-SUBGOAL(plan) returns S_{need}, c

    pick a plan step S_{need} from STEPS(plan)
    with a precondition c that has not been achieved

    return S_{need}, c
Clobbering and promotion/demotion

A clobberer is a potentially intervening step that destroys the condition achieved by a causal link.

Consider two steps of plan with goal

To meet precondition of goal, add step Go(Home)
But, this threatens (clobbers) precondition of Buy(Drill)

Options to protect Buy(Drill)
1. Demotion: put before Go(HWS)
2. Promotion: put after Buy(Drill)
**POP Algorithm Continued**

```plaintext
procedure CHOOSE-OPERATOR(plan, operators, S\_need, c)
    choose a step \( S_{add} \) from operators or STEPS(plan) that has \( c \) as an effect
    if there is no such step then fail
    add the causal link \( S_{add} \xrightarrow{c} S_{\text{need}} \) to LINKS(plan)
    add the ordering constraint \( S_{add} \prec S_{\text{need}} \) to ORDERINGS(plan)
    if \( S_{add} \) is a newly added step from operators then
        add \( S_{add} \) to STEPS(plan)
        add Start \prec S_{add} \prec Finish to ORDERINGS(plan)

procedure RESOLVE-THREATS(plan)
    for each \( S_{\text{threat}} \) that threatens a link \( S_i \xrightarrow{c} S_j \) in LINKS(plan) do
        choose either
            Demotion: Add \( S_{\text{threat}} \prec S_i \) to ORDERINGS(plan)
            Promotion: Add \( S_j \prec S_{\text{threat}} \) to ORDERINGS(plan)
    if not CONSISTENT(plan) then fail
end
```
Example: Blocks World

"Sussman anomaly" problem

Start State

ON(C,A)
On(A, Table)
On(B, Table)
Cl(B)
Cl(C)

Goal State

ON(A,B)
On(B, C)

Symbols: A, B, C, Table
Predicates:
ON(x,y) – Object x is on top of object y
CL(x) – x doesn’t have anything on top

+ several inequality constraints
Example: Blocks World

"Sussman anomaly" problem

Start State

Goal State

Operators

\[\text{Clear}(x) \land \text{On}(x, z) \land \text{Clear}(y)\]

\[\text{PutOn}(x, y)\]

\[\sim\text{On}(x, z) \land \sim\text{Clear}(y)\]

\[\text{Clear}(z) \land \text{On}(x, y)\]

\[\text{Clear}(x) \land \text{On}(x, z)\]

\[\text{PutOnTable}(x)\]

\[\sim\text{On}(x, z) \land \text{Clear}(z) \land \text{On}(x, \text{Table})\]
Initial and Final conditions

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

On(A,B) On(B,C)

START

FINISH
Insert operation to achieve goal precondition

\[ On(C,A) \quad On(A,\text{Table}) \quad Cl(B) \quad On(B,\text{Table}) \quad Cl(C) \]

\[ Cl(B) \quad On(B,z) \quad Cl(C) \]

PutOn(B,C)

\[ On(A,B) \quad On(B,C) \]

FINISH
PutOn(A,B) is a threat to PutOn(B,C)

Dashed line denotes order
PutOnTable(C) meets precondition but must be in right order

Dashed line denotes order

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

On(C,z) Cl(C)

PutOnTable(C)

Cl(A) On(A,z) Cl(B)

PutOn(B,C)

Cl(B) On(B,z) Cl(C)

PutOn(A,B)

On(A,B) On(B,C)

Finish

PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)

PutOn(B,C) clobbers Cl(C) => order after PutOnTable(C)