CSC384: Lecture 8

- Last time
  - Action Representation; planning as search
- Today
  - STRIPS Planning, Regression planning
- Readings:
  - Today: 8.3 (STRIPS planning in depth, regression planning, briefly resolution-based planning)
  - Next week: uncertainty 10.1, 10.2, start on 10.3

STRIPS Planner

- Last time, discussed intuitive sketch of STRIPS
  - a divide-and-conquer approach
  - tries to find independent plans for individual subgoals and then pieces these plans together
  - recursively tries to achieve necessary preconditions
- We'll sketch a version of the algorithm designed to work with the CWR-D representation
  - contrast with algorithm in text, which is designed to work with the situation calculus representation

STRIPS with CWR-D

- achieve_all(GList,S0,S1,Plan)
  - action sequence Plan applied at state S0 results in state S1, satisfying all goals in GList

```prolog
achieve_all([], S, S, []). achieve_all([Goal|GList], S0, S1, Plan) :- remove(Goal, GList, RestG), achieve(Goal, S0, S1, Plan1), achieve_all(RestG, S1, S2, Plan2), append(Plan1, Plan2, Plan).
```

STRIPS w/ CWR-D: Goal Selection

- remove(G, GList, RestG)
  - selects a goal G from goal list for achievement
  - implementation #1 below always selects first goal
  - note: we'll see that allowing different orderings is important---it should really be a "choose" not "select"
  - implementation #2 allows backtracking

```prolog
remove(G, [G | RestG], RestG).
remove(G, [], []). remove(G, GList, RestG) :- member(G, GList), delete(G, GList, RestG).
```

STRIPS w/ CWR-D: Goal Achievem't

- achieve(G, S0, S1, Plan)
  - action sequence Plan applied at state S0 results in state S1, satisfying all goal G (single goal)
  - all predicates used defined earlier except effect_of
  - effect_of(A,G): action A has G as an effect (exercise)

```prolog
achieve(G, S0, S1, Plan) :- effect_of(A,G), remove(A,P), achieve_all(P, S0, S1, Plan1), append(Plan1, A, Plan2), result(A,S1,S2).
```

STRIPS: Handling Derived Relations

- If we have derived relations, STRIPS can't directly achieve such a fact (not mentioned as effects of any actions)
  - so simply set Body as subgoals to achieve

```prolog
achieve(G, S0, S1, Plan) :- derivedRel(G, Body), achieve_all(Body, S0, S1, Plan).
```
Issues with STRIPS (1)

- Order of goal selection can impact quality/length of plan
  - e.g., we picked \text{mov}(o) to achieve \text{loc}(o) in final plan step; but what if we had picked \text{mov}(m,o)?
  - might have picked \text{mov}(h,m); then \text{mov}(h,l), etc. and taken long way around
  - might have gotten in a cycle
- In general, goal selection ordering can benefit from heuristics: and can even require systematic search/backtracking

Subgoal Protection

- Given \( k \) goals \( g_1, \ldots, g_k \) in this order
  - produce a subplan that achieves \( g_1 \) (say \( p_1 \))
  - produce a subplan \( p_2 \) that produces that achieves \( g_2 \) without affecting \( g_1 \)
  - in general, produce a plan \( p_i \) for \( g_i \) that does not affect any \( g_j \) ordered before \( g_i \)
  - Solution \( p_1; p_2; \ldots; p_k \) guaranteed to achieve all goals

Example of Protection (1)

- If we choose \text{loc}(o) first:
  - we get plan \( p_1 = [ \text{loc}(o) \text{ true in } S_0 \}
  - we protect \text{loc}(o) --- it’s already achieved
  - attempt to find plan to achieve labtidy without altering \text{loc}(o)
  - impossible because of protection
- Once it fails, we retry with labtidy as first goal
  - this will succeed as in original example
  - notice that it’s critical to allow algorithm to backtrack over goal choices so it can try a different ordering

Issues with STRIPS (2)

- STRIPS can return incorrect plans!
  - suppose we chose goal \text{loc}(o) before labtidy
  - plan for \text{loc}(o) is \([ \text{getkeys, mov}(o,l), \text{tidy} \]
  - plan for labtidy is \([ \text{getkeys, mov}(o,l), \text{tidy} \]
  - the second plan destroys or \text{dobbens} the subgoal achieved by the first plan!
  - so returned plan \([ \_ \_ ] + [\text{get}(o,l), \text{tidy}] \) is incorrect
- Subgoal protection:
  - circumvents this problem by protecting achieved subgoals when producing plans for the next subgoals

Subgoal Protection (con’t)

- Key to above algorithm:
  - achieve(G, S0, S1, Plan, Protected) is not allowed to construct a subplan that “touches” any literal in the protected list
  - exercise: try it (tricky to do this with derived rel’ns)

Example of Protection (2)

- Same example, but suppose action \text{mov}(l,o) magically makes labtidy false!
- Choose labtidy as first goal
  - we get plan \( p_1 = [ \text{getkeys, mov}(o,l), \text{tidy} \]
  - we protect labtidy
  - attempt to find plan to achieve \text{loc}(o) without altering labtidy
  - try to achieve \text{loc}(o) using \text{mov}(o,l); but this undoes labtidy, so fails due to protection
  - try to achieve \text{loc}(o) using \text{mov}(m,o); this works; sets up subgoal of \text{loc}(m); etc.
  - soln: tidy the lab then go back to office the long way around
- Subgoal protection has desired effect
Is STRIPS with SGP “Complete”?

- STRIPS with subgoal protection is sound
  - if it returns a plan, the plan is correct (achieves goals)
- But STRIPS with SGP is not complete
  - it may not find a plan even if it exists
  - this is true even if it searches over all goal orderings
  - this is due to its notion of achievement
- Why? Let’s consider an example...

Problems with STRIPS (3)

- Example using only two locations -- loc(o), loc(c)
  - but if robot in office and Craig has coffee, if robot
    leaves office, C throws coffee against wall in
    megalomaniacal fit of rage (robot must watch C drink)
  - so action mov(loc,c) has effect neg(chc)
  - Start: neg(cm), neg(chc), neg(rhc), loc(c)
  - Goal: chc, cm
- To solve, STRIPS must solve with
  - ordering #1: cm then chc; or
  - ordering #2: chc then cm

Problems with STRIPS (3)

- Ordering #1: cm then chc is not suitable
  - you could achieve cm by simply making coffee
    - if you did that, any way of achieving chc would clobber cm.
    - Robot must grabcoffee -- neg(cm) -- to give it to Craig
  - Note: you could [makecoffee, grabcoffee, makecoffee] and then take
    coffee to Craig; but STRIPS won’t consider this, since once you
    achieve cm you can’t clobber it. The only reason to consider it is
    if STRIPS looks ahead to next goal
  - Ordering #2: chc then cm is not suitable
    - once you make chc true by the usual plan (make, grab, move, give),
    can’t leave office to make more
    - Note: you could [makecoffee, grabcoffee, makecoffee, givecoffee]; but unless
    it looks ahead to next goal, STRIPS has no reason to consider this

Serializability

- A set of goals G is serializable (wrt s0) if there is some ordering of the goals [g₁, …, gₖ] s.t.
  - you can achieve g₁ from s₀
  - you can achieve g₂ without clobbering g₁ no matter
    what plan you used to achieve g₁
  - you can achieve gₖ without clobbering g₁, g₂ no
    matter what plan you used to achieve g₁, g₂, etc…
- STRIPS-SGP can solve any serializable goal set
  - backtracking over goal orderings must be allowed
  - Note: earlier example is not serializable
  - success depends on the plan chosen
  - but we can’t allow STRIPS to consider arbitrary plans
  - or we lose the benefits of divide and conquer

Regression Planning: Intuitions

- Basic idea behind regression is quite simple:
  - given a goal list G, the regression of G through action A is the weakest set of preconditions WC that ensure
    G is true after A is performed
  - In other words:
    - if WC holds at state S, then G holds at result(A,S)
    - no logically weaker set of conditions satisfies this property
  - This leads to an obvious subgoal strategy
    - given G, find an action A ‘that makes progress’ on G
    - find a plan P’ that achieves WC
    - then return the plan P = [P’, A]

STRAIPS Summary

- STRIPS biggest problem:
  - forced to completely solve one subgoal before
    considering how it affects other goals
  - with subgoal protection we get correct plans, but only
    if subgoal set is serializable
  - but this prevents you from finding plans where goals
    interact strongly
- A different view: regression planning
  - when you insert an action into a plan, you consider
    how it influences all current subgoals
  - but you still focus on achieving one subgoal
Consider a nonserializable example with \( G = [\text{chc}, \text{cm}] \)

\[
\begin{align*}
G &= [\text{chc}, \text{cm}] \\
\text{regress}(A, G, L, W) &= \text{removeffects}(A, L, W) \\
\text{preconds}(A, P) \\
\text{addpreconds}(P, W) \\
\end{align*}
\]

Let’s look at intuitions before getting into details
- consider nonserializable example with \( G = [\text{chc}, \text{cm}] \)

Basic structure of the algorithm:
- start with subgoal (SG) list equal to goal list
- Loop:
  - choose an action \( A \) that:
    a. achieves at least one subgoal on SG list
    b. doesn’t destroy any other subgoals on list
    c. precons are consistent with other subgoals
  - regress SG list through action \( A \) to obtain SGNew
  - set SG list to SGNew
- until all elements in SG list are true in S0
- Conditions b, c necessary, otherwise A cannot make SG list true (more to come)

Regression Planning
- We first need to define the notion of regression formally (and basic idea behind implementation)
- We then need to define a planner that relies on the notion of regression

Defining Regression for CWR (1)

\[
\begin{align*}
\text{regress}(A, G, L, W) &= \text{removeffects}(A, G, L, W) \\
\text{preconds}(A, P) \\
\text{addpreconds}(P, W, L, W) \\
\end{align*}
\]

Defining Regression for CWR (2)

\[
\begin{align*}
\text{removeffects}(A, G, L, W) &= \text{removeffects}(A, G, L, W) \\
\text{preconds}(A, P) \\
\text{addpreconds}(P, W, L, W) \\
\end{align*}
\]

Why Conditions (b) and (c)
- Why we need condition (b):
  - action \( a \): precond \( x \); effects \( y \), neg(\( z \))
  - subgoals \( S_G = [y, z] \)
  - impossible to do action \( a \) and (immediately) result in a state where \( S_G \) is true: a achieves \( y \), but makes \( z \) false
- Why we need condition (c):
  - action \( a \): precond \( x \), neg(\( z \)); effects \( y \)
  - subgoals \( S_G = [y, z] \)
  - impossible to do action \( a \) and (immediately) result in a state where \( S_G \) is true: a achieves \( y \), but requires \( z \) to be false when executed; since \( a \) doesn’t affect \( z \), \( z \) must be false immediately after doing \( a \)

Note: (b) and (c) ensure regression is “possible”
Defining Regression for CWR (3)

- Simple auxiliary predicate “achieves”

\[
\begin{align*}
\text{achieves}(A,G) & : \text{addlist}(A,\text{GList}), \text{member}(G,\text{GList}). \\
\text{achieves}(A,\neg G) & : \text{deletelist}(A,\text{GList}), \text{member}(G,\text{GList}).
\end{align*}
\]

Defining Regression for CWR (4)

- \text{regress}(A,\text{GL},\text{WP})

  - true if: WP is the weakest precondition such that executing A when WP holds results in goal list GL becoming true; WP is consistent
  - fails if no such consistent WP exists
  - assumes GL is consistent already

\[
\begin{align*}
\text{regress}(A,\text{GL},\text{WP}) & : \\
\text{removeeffects}(A,\text{GL},\text{WGL}), \\
\text{preconds}(A,\text{PC}), \\
\text{addpreconds}(\text{PC},\text{WGL},\text{WP}).
\end{align*}
\]

Defining Regression for CWR (5)

- \text{addpreconds}(\text{PC},\text{WGL},\text{WP})

  - true if: WP results from adding preconditions PC to (weakened) goal list WGL, and result is consistent
  - fails if preconditions conflict with WGL.
  - WGL is assumed consistent.

\[
\begin{align*}
\text{addpreconds}([],\text{WP},\text{WP}). \\
\text{addpreconds}(\text{PC},\text{ResP},\text{WGL},\text{WP}) & : \text{addconsistent}(\text{P},\text{L}), \text{addpreconds}(\text{ResP},\text{L},\text{WP}).
\end{align*}
\]

Regression Planner

- \text{rplan}(\text{GoalList},\text{State},\text{Plan})

  - true if Plan achieves GoalList starting at State
  - Basic intuition: see slide 4

\[
\begin{align*}
\text{rplan}(\text{GoalList},\text{State},[]) & : \text{holdsall}(\text{GoalList},\text{State}). \\
\text{rplan}(\text{GoalList},\text{State},\text{Plan}) & : \text{member}(\text{Goal},\text{GoalList}), \text{achieves}(A,\text{Goal}), \text{regress}(A,\text{Goal},\text{NewGoalList}), \text{rplan}(\text{NewGoalList},\text{State},\text{Plan}1), \text{append}(\text{Plan1},[A],\text{Plan}).
\end{align*}
\]

Some Notes on rplan

- We assume that initial goal list is consistent
  - we ensure subgoal lists remain consistent in “regress”
  - Search occurs with goal and action choices
    - \text{member}(\text{Goal},\text{GoalList}) chooses a goal to achieve
    - \text{achieves}(A,\text{Goal}) chooses action to achieve it
    - backtracking taken care of by Prolog
  - This implementation will never work in practice!
    - by allowing Prolog to do the search, we’re committing to DFS without cycle checking!
  - Exercise (asu?): fix this by controlling search yourself (don’t hand it off to Prolog)
    - e.g., use BFS or iterative deepening
Wrap up of Regression Planning

- Main idea: we are reasoning backward from the goal conditions to S0
  - choose a goal and an action that achieves it
  - search space is not the set of states, but the set of subgoal lists (nbs are subgoal lists we can reach by regressing consistently through some "useful" action)

```
g = \{ \text{clean, cm} \}
givecoffee \rightarrow \text{makecoffee} \rightarrow \text{\[\text{in, loc(a, cm)}\], \text{\text{clean, loc(c)}}\]
```

"Goal" in this search space is any subgoal list that is true in the initial state

Planning

- Most modern planners more sophisticated than STRIPS/ regression
  - but most rely on basic ideas of decomposition and the idea of "regressing" (reasoning backward) from goal
- Partial-order planning (see 8.3 of text)
  - exploits "least commitment" idea by choosing actions without committing to their order right away
  - nice ideas, but computationally expensive in practice
- Planning as search quite common (fast)
  - use backchaining ideas to guide search/generate heuristics
  - sophisticated search used (e.g., stochastic search)

Situation Calculus

- SC an alternative representation for actions
- A logical language in which
  - situations are terms (e.g., init, s27, S0)
    - init a special constant referring to initial state
  - actions are terms (e.g., mov(X,Y), mov(a, loc))
  - do(A,S) refers to situation that results from doing A in situation S (do a special function symbol)
  - domain predicate all have a situation argument (e.g., rhc(s27), or loc(m,init)

```
\text{SitCalc: Example expressions}
```

```
\text{Situations (states of world, but w/ action history)}
\text{init}
\text{do(grab, init)}
\text{do(grab, S)}
\text{do(give, do(mov(c, o), do(grab, init)))}
```

```
\text{Statements about what's true}
\text{lo(c, init)}
\text{cm(init)}
\text{rhc(do(grab, init))}
\text{rhc(do(give, do(mov(c, o), do(grab, init)))}
```

Actions in the SitCalc

- We specify actions by specifying preconditions and effects
- Preconditions specified using the \text{poss} predicate

```
\text{poss(give, S)} \leftarrow \text{rhc(S), loc(a, S)}.
\text{poss(mov(X,Y), S)} \leftarrow \text{adj(X, Y, S), acc(Y, S), loc(X, S)}.
```

Effect Axioms

- Action effects specified using effect axioms

```
\text{\text{rhc(do(give, S)) \leftarrow poss(give, S),}}
\text{\text{loc(Y, do(mov(X,Y), S)) \leftarrow poss(mov(X,Y), S).}}
```

- Sadly, specifying effects alone logically insufficient
  - how do we know if labtidy is true after doing givecof?
  - must explicitly specify \text{frame axioms} stating that unaffected things remain unchanged after an action

```
\text{\text{btdy(do(give, S)) \leftarrow poss(give, S), btdy(S),}}
\text{\text{rhc(do(give, S)) \leftarrow poss(give, S), btdy(S).}}
```

CSC 384 Lecture Slides (c) 2002, C. Boutilier
Reasoning in the SitCalc

- You can use these axioms to prove that certain things are true or false after performing a sequence of actions
  - provide action specifications (incl. frame axioms)
  - state what is true at init (e.g., loc(c,init), cm(init),...)
  - ask query, e.g.,
  - ?- chc(do(givecf, do(mov(c,o), do(grabcf, init))).

- If axioms allow proof, then you know chc is true after this sequence from init
  - but must rely on negation as failure for false things
  - no way to prove “neg(cm)” after grabcf otherwise

Planning as Theorem Proving

- You can now use SLD to construct a plan
  - given init specification, poss and effect axioms
  - given a goal G such as [chc, cm]
  - ask the query: ?- chc(S), cm(S).

- SLD will return an answer in which variable S is bound to a situation term from which plan can be extracted; e.g.
  - S= do(givecf, do(mov(c,o), do(maked, do(grabcf, do(maked, init))))

- Computationally: this relies on SLD/Prolog doing usual DFS (so may not work very well)