Chapter 4
State-Space Planning
Motivation

- Nearly all planning procedures are search procedures
- Different planning procedures have different search spaces
  - Two examples:
  - \textit{State-space planning}
    - Each node represents a state of the world
      - A plan is a path through the space
  - \textit{Plan-space planning}
    - Each node is a set of partially-instantiated operators, plus some constraints
      - Impose more and more constraints, until we get a plan
Outline

- State-space planning
  - Forward search
  - Backward search
  - Lifting
  - STRIPS
  - Block-stacking
Planning: Search Space
Forward-search\((O, s_0, g)\)

\[
\begin{align*}
    s &\leftarrow s_0 \\
    \pi &\leftarrow \text{the empty plan} \\
    \text{loop} &\quad \text{if } s \text{ satisfies } g \text{ then return } \pi \\
    &\quad E \leftarrow \{a|a \text{ is a ground instance an operator in } O, \\
    &\quad \text{and } \text{precond}(a) \text{ is true in } s\} \\
    &\quad \text{if } E = \emptyset \text{ then return failure} \\
    &\quad \text{nondeterministically choose an action } a \in E \\
    &\quad s \leftarrow \gamma(s, a) \\
    &\quad \pi \leftarrow \pi.a
\end{align*}
\]
Algorithm: \textsc{ProgWS}(\text{world-state}, \text{goal-list}, \Lambda, \text{path})

1. If \text{world-state} satisfies each conjunct in \text{goal-list},
2. Then return \text{path},
3. Else let \text{Act} = \text{choose} from \Lambda an action whose precondition is satisfied by \text{world-state}:
   (a) If no such choice was possible,
   (b) Then return failure,
   (c) Else let \text{S} = the result of simulating execution of \text{Act} in \text{world-state}
       and return \text{ProgWS}(\text{S}, \text{goal-list}, \Lambda, \text{Concatenate(path, Act)}).
Properties

- Forward-search is *sound*
  - for any plan returned by any of its nondeterministic traces, this plan is guaranteed to be a solution

- Forward-search also is *complete*
  - if a solution exists then at least one of Forward-search’s nondeterministic traces will return a solution.
Some deterministic implementations of forward search:
- breadth-first search
- depth-first search
- best-first search (e.g., A*)
- greedy search

Breadth-first and best-first search are sound and complete
- But they usually aren’t practical because they require too much memory
- Memory requirement is exponential in the length of the solution

In practice, more likely to use depth-first search or greedy search
- Worst-case memory requirement is linear in the length of the solution
- In general, sound but not complete
  » But classical planning has only finitely many states
  » Thus, can make depth-first search complete by doing loop-checking
Branching Factor of Forward Search

- Forward search can have a very large branching factor
  - E.g., many applicable actions that don’t progress toward goal
- Why this is bad:
  - Deterministic implementations can waste time trying lots of irrelevant actions
- Need a good heuristic function and/or pruning procedure
  - See Section 4.5 (Domain-Specific State-Space Planning) and Part III (Heuristics and Control Strategies)
Backward Search

- For forward search, we started at the initial state and computed state transitions
  - new state = \( \gamma(s,a) \)
- For backward search, we start at the goal and compute inverse state transitions
  - new set of subgoals = \( \gamma^{-1}(g,a) \)
- To define \( \gamma^{-1}(g,a) \), must first define relevance:
  - An action \( a \) is relevant for a goal \( g \) if
    - \( a \) makes at least one of \( g \)'s literals true
      - \( g \cap \text{effects}(a) \neq \emptyset \)
    - \( a \) does not make any of \( g \)'s literals false
      - \( g^+ \cap \text{effects}^-(a) = \emptyset \) and \( g^- \cap \text{effects}^+(a) = \emptyset \)
Inverse State Transitions

- If $a$ is relevant for $g$, then
  - $\gamma^{-1}(g,a) = (g - \text{effects}(a)) \cup \text{precond}(a)$
- Otherwise $\gamma^{-1}(g,a)$ is undefined

Example: suppose that
  - $g = \{\text{on}(b1,b2), \text{on}(b2,b3)\}$
  - $a = \text{stack}(b1,b2)$
- What is $\gamma^{-1}(g,a)$?
Backward-search\((O, s_0, g)\)
\[
\pi \leftarrow \text{the empty plan}
\]
\[
\text{loop}
\]
\[
\text{if } s_0 \text{ satisfies } g \text{ then return } \pi
\]
\[
A \leftarrow \{a | a \text{ is a ground instance of an operator in } O \text{ and } \gamma^{-1}(g, a) \text{ is defined}\}
\]
\[
\text{if } A = \emptyset \text{ then return failure}
\]
\[
\text{nondeterministically choose an action } a \in A
\]
\[
\pi \leftarrow a.\pi
\]
\[
g \leftarrow \gamma^{-1}(g, a)
\]
Algorithm: \texttt{REGWS}(init-state, cur-goals, \Lambda, path)

1. If init-state satisfies each conjunct in cur-goals,
2. Then return path,
3. Else do:
   (a) Let \texttt{Act} = \texttt{choose} from \Lambda an action whose effect matches at least one conjunct in cur-goals.
   (b) Let \texttt{G} = the result of regressing cur-goals through \texttt{Act}.
   (c) If no choice for \texttt{Act} was possible or \texttt{G} is undefined, or \texttt{G} \supset cur-goals,
   (d) Then return failure,
   (e) Else return \texttt{REGWS}(init-state, G, \Lambda, Concatenate(Act, path)).
Efficiency of Backward Search

- Backward search can also have a very large branching factor
  - E.g., many relevant actions that don’t regress toward the initial state
- As before, deterministic implementations can waste lots of time trying all of them
Lifting

Can reduce the branching factor of backward search if we partially instantiate the operators

- this is called \textit{lifting}

foo($x, y$)

precond: p($x, y$)
effects: q(x)

q(a)

p(a,a) \rightarrow

foo(a,a)

p(a,b) \rightarrow

foo(a,b)

p(a,c) \rightarrow

foo(a,c)

... 

foo(a,y) \rightarrow

q(a)

p(a,y) \rightarrow
Lifted Backward Search

- More complicated than Backward-search
  - Have to keep track of what substitutions were performed
- But it has a much smaller branching factor

```
Lifted-backward-search(O, s_0, g)
    \pi \leftarrow \text{the empty plan}
    \text{loop}
        \text{if } s_0 \text{ satisfies } g \text{ then return } \pi
        A \leftarrow \{(o, \theta)|o \text{ is a standardization of an operator in } O, \\
            \theta \text{ is an mgu for an atom of } g \text{ and an atom of effects}^+(o), \\
            \text{and } \gamma^{-1}(\theta(g), \theta(o)) \text{ is defined}\}
        \text{if } A = \emptyset \text{ then return failure}
        \text{nondeterministically choose a pair } (o, \theta) \in A
        \pi \leftarrow \text{the concatenation of } \theta(o) \text{ and } \theta(\pi)
        g \leftarrow \gamma^{-1}(\theta(g), \theta(o))
```
The Search Space is Still Too Large

- Lifted-backward-search generates a smaller search space than Backward-search, but it still can be quite large
  - Suppose \( a, b, \) and \( c \) are independent, \( d \) must precede all of them, and \( d \) cannot be executed
  - We’ll try all possible orderings of \( a, b, \) and \( c \) before realizing there is no solution
  - More about this in Chapter 5 (Plan-Space Planning)

```
\begin{align*}
  d & \rightarrow a \rightarrow b \\
  d & \rightarrow b \rightarrow a \\
  d & \rightarrow b \rightarrow a \\
  d & \rightarrow a \rightarrow c \\
  d & \rightarrow b \rightarrow c \\
  d & \rightarrow c \rightarrow b \\
\end{align*}
```
STRIPS

- $\pi \leftarrow$ the empty plan
- do a modified backward search from $g$
  - instead of $\gamma^{-1}(s,a)$, each new set of subgoals is just $\text{precond}(a)$
  - whenever you find an action that’s executable in the current state, then go forward on the current search path as far as possible, executing actions and appending them to $\pi$
- repeat until all goals are satisfied

\[
\pi = \langle a_6, a_4 \rangle \\
\text{satisfied in } s_0 \\
\text{current search path}
\]
A ground version of the STRIPS algorithm.

\[
\text{Ground-STRIPS}(O, s, g) \\
\pi \leftarrow \text{the empty plan} \\
\text{loop} \\
\quad \text{if } s \text{ satisfies } g \text{ then return } \pi \\
\quad A \leftarrow \{ a \mid a \text{ is a ground instance of an operator in } O, \text{ and } a \text{ is relevant for } g \} \\
\quad \text{if } A = \emptyset \text{ then return failure} \\
\quad \text{nondeterministically choose any action } a \in A \\
\quad \pi' \leftarrow \text{Ground-STRIPS}(O, s, \text{precond}(a)) \\
\quad \text{if } \pi' = \text{failure then return failure} \\
\quad ;; \text{if we get here, then } \pi' \text{ achieves } \text{precond}(a) \text{ from } s \\
\quad s' \leftarrow \gamma(s, \pi') \\
\quad ;; s \text{ now satisfies } \text{precond}(a) \\
\quad s \leftarrow \gamma(s, a) \\
\quad \pi \leftarrow \pi'. \pi'. a
\]
Quick Review of Blocks World

unstack(x,y)
Pre: on(x,y), clear(x), handempty
Eff: ~on(x,y), ~clear(x), ~handempty, holding(x), clear(y)

stack(x,y)
Pre: holding(x), clear(y)
Eff: ~holding(x), ~clear(y), on(x,y), clear(x), handempty

pickup(x)
Pre: ontable(x), clear(x), handempty
Eff: ~ontable(x), ~clear(x), ~handempty, holding(x)

putdown(x)
Pre: holding(x)
Eff: ~holding(x), ontable(x), clear(?x), handempty
Current state:
- ontable(A), on(C, B), ontable(B), ontable(D), clear(A), clear(C), clear(D), he.

Goal
- on(A, C), on(D, A)
STRIPS Planning

Plan:

Goalstack:

on(A,C), on(D,A)
on(A,C)
Stack(A, C)
holding(A), clear(C)
holding(A)
Pickup(A)
ontable(A), clear(A), he

Current State

ontable(A), on(C, B), ontable(B), ontable(D), clear(A), clear(C), clear(D), he.
STRIPS Planning

Plan:

Goalstack:

on(A,C), on(D,A)
on(A,C)
Stack(A, C)
holding(A), clear(C)
holding(A)

Pickup(A)

Pre: ontable(A), clear(A), he
Del: ontable(A), clear(A), he,
Add: holding(A)

Current State

holding(A), on(C, B), ontable(B), ontable(D), clear(A), clear(C), clear(D)
STRIPS Planning

Plan:
Pickup(A)

Goalstack:
- on(A,C), on(D,A)
- on(A,C)

Stack(A, C)

Stack(A, C)
- Pre: holding(A), clear(C)
- Del: holding(A), clear(C)
- Add: on(A, C), he

Current State

holding(A), on(A,B), on(A,D), on(B,D), clear(A), clear(B), clear(C), clear(D).
Plan:
- Pickup(A)
- Stack(A, C)

Goalstack:
- on(A,C), on(D,A), on(D, A)
- Stack(D,A)
- holding(D), clear(A)
- holding(D)
- Pickup(D)
- ontable(D), clear(D), he

Current State:
- on(A,C), on(C, B), ontable(B), holding(D), clear(A), clear(D), he.
STRIPS Planning

Plan:

Pickup(A)
Stack(A, C)
Pickup(D)

Goalstack:  on(A, C), on(D, A)
on(D, A)
Stack(D, A)
holding(D), clear(A)
holding(D)

Current State:
on(A, C), on(C, B), ontable(B), bo(D, A), clear(D, A)
STRIPS Planning: Getting it Wrong!

Plan:

Goalstack: on(A,C), on(D,A)
on(D,A)
Stack(D, A)
holding(D), clear(A)
holding(D)
Pickup(D)
ontable(D), clear(D), he
STRIPS Planning: Getting it Wrong!

Plan:
Pickup(D)

Goalstack:
on(A,C), on(D,A)
on(D,A)
Stack(D, A)

Current State:
ontable(A), on(C, B), ontable(B), holding(D), clear(C), clear(D), he
STRIPS Planning: Getting it Wrong!

Plan:
- Pickup(D)
- Stack(D, A)

Goalstack: on(A,C), on(D,A)

Now What?
- We chose the wrong goal first
- A is no longer clear.
- Stacking D on A messes up the preconditions for actions to accomplish on(A, C)
- Either have to backtrack, or else we must undo the previous actions

Current State
- ontable(A), on(C, B), ontable(B), on(D,A), clear(C), clear(D), he.
The Sussman Anomaly

- On this problem, STRIPS can’t produce an irredundant solution
  - Try it and see
The Register Assignment Problem

- State-variable formulation:

  Initial state: \{\text{value}(r_1)=3, \text{value}(r_2)=5, \text{value}(r_3)=0\}

  Goal: \{\text{value}(r_1)=5, \text{value}(r_2)=3\}

  Operator: assign(\text{}\text{r},v,\text{}\text{r}',v')

    precond: \text{value}(\text{}\text{r})=v, \text{value}(\text{}\text{r}')=v'

    effects: \text{value}(\text{}\text{r})=v'

- STRIPS cannot solve this problem at all
How to Fix?

Several ways:

- Do something other than state-space search
  » e.g., Chapters 5–8

- Use forward or backward state-space search, with *domain-specific* knowledge to prune the search space
  » Can solve both problems quite easily this way
  » Example: block stacking using forward search
A blocks-world planning problem $P = (O, s_0, g)$ is solvable if $s_0$ and $g$ satisfy some simple consistency conditions:

- $g$ should not mention any blocks not mentioned in $s_0$
- A block cannot be on two other blocks at once
- Etc.

- Can check these in time $O(n \log n)$

If $P$ is solvable, can easily construct a solution of length $O(2m)$, where $m$ is the number of blocks:

- Move all blocks to the table, then build up stacks from the bottom
- Can do this in time $O(n)$

With additional domain-specific knowledge can do even better …
Additional Domain-Specific Knowledge

- A block $x$ needs to be moved if any of the following is true:
  - $s$ contains $\text{ontable}(x)$ and $g$ contains $\text{on}(x,y)$ - see a below
  - $s$ contains $\text{on}(x,y)$ and $g$ contains $\text{ontable}(x)$ - see d below
  - $s$ contains $\text{on}(x,y)$ and $g$ contains $\text{on}(x,z)$ for some $y \neq z$
    » see c below
  - $s$ contains $\text{on}(x,y)$ and $y$ needs to be moved - see e below
Domain-Specific Algorithm

loop
  if there is a clear block $x$ such that
    $x$ needs to be moved and
    $x$ can be moved to a place where it won’t need to be moved
  then move $x$ to that place
else if there is a clear block $x$ such that
  $x$ needs to be moved
  then move $x$ to the table
else if the goal is satisfied
  then return the plan
else return failure
repeat
Easily Solves the Sussman Anomaly

\[
\begin{array}{l}
\text{loop} \\
\quad \text{if there is a clear block } x \text{ such that } \\
\qquad x \text{ needs to be moved and } \\
\qquad x \text{ can be moved to a place where it won’t need to be moved } \\
\qquad \text{then move } x \text{ to that place} \\
\quad \text{else if there is a clear block } x \text{ such that } \\
\qquad x \text{ needs to be moved } \\
\qquad \text{then move } x \text{ to the table} \\
\quad \text{else if the goal is satisfied } \\
\qquad \text{then return the plan} \\
\quad \text{else return failure} \\
\end{array}
\]

repeat
Properties

- The block-stacking algorithm:
  - Sound, complete, guaranteed to terminate
  - Runs in time $O(n^3)$
    » Can be modified to run in time $O(n)$
  - Often finds optimal (shortest) solutions
  - But sometimes only near-optimal
    » Recall that PLAN LENGTH for the blocks world is NP-complete