Part III

Heuristics and Control Strategies
Motivation for Part III of the text Book

- Domain-independent planners suffer from combinatorial complexity
  - Planning is in the worst case intractable
  - Need ways to control the search

- Search heuristics (Chapter 9)
- Pruning rules (Chapter 10)
- HTN decomposition (Chapter 11)
Abstract Search Procedure

Here is a general framework for describing classical and neoclassical planners

◆ Details may differ from one planner to another
◆ e.g., the steps don’t have to be in this order

Abstract-search\(\left(u\right)\)

if Terminal\(\left(u\right)\) then return\(\left(u\right)\)

\(u \leftarrow \text{Refine}\left(u\right)\) ;; refinement step

\(B \leftarrow \text{Branch}\left(u\right)\) ;; branching step

\(B' \leftarrow \text{Prune}\left(B\right)\) ;; pruning step

if \(B' = \emptyset\) then return(failure)
nondeterministically choose \(v \in B'\)

return(\(\text{Abstract-search}\left(v\right)\))

end
Abstract Search

- Abstract search nondeterministically searches a space in which each node $u$ represents a set of solution plans $\Pi_u$, i.e., the set of all solutions that are reachable from $u$

- $u$ is basically a structured collection of actions and constraints
  - In state-space planning, $u$ is simply a set of actions
  - In plan-space planning, $u$ is a set of actions, causal links, orders, and variable binding constraints.
  - In Graphplan algorithm, $u$ is a subgraph of the planning graph
  - In SAT-based planning, $u$ is a set of assigned literals and remaining clauses
Abstract search

- In state-space and plan-space planning, $u$ is a partial plan; every action in $u$ belongs to any solution plan in $\Pi_u$.
- But in the other three, not every action in $u$ appears in a solution plan in $\Pi_u$.
- Abstract-search involves three main steps:
  1. Refinement: modifying the collection of actions and/or constraints associated with a node $u$. In a refinement of $u$, the set $\Pi_u$ remains unchanged.
  2. Branching: generating one or more children of $u$; these are candidates for the next node to visit. Each child $v$ of $u$ represents a subset of solutions $\Pi_v \subseteq \Pi_u$. Branching does not need to generate all children of $u$.
  3. Pruning: removing some nodes that appear to be unpromising from the set of candidates.
Plan-Space Planning

- Branching:
  - select a flaw, find its resolvers

- Refinement:
  - apply a resolver

- Pruning
  - There is no pruning

Abstract-search($u$)
    if Terminal($u$) then return($u$)
    $u \leftarrow$ Refine($u$) ;; refinement step
    $B \leftarrow$ Branch($u$) ;; branching step
    $B' \leftarrow$ Prune($B$) ;; pruning step
    if $B' = \emptyset$ then return(failure)
    nondeterministically choose $v \in B'$
    return(Abstract-search($v$))
end
State-Space Planning

- Forward (or Backward) search
  - Branching: actions
  - Refinement: apply $\gamma$ (or $\gamma^{-1}$) to a goal
  - Pruning: Cycle detection

```
Abstract-search($u$)
    if Terminal($u$) then return($u$)
    $u \leftarrow$ Refine($u$) ;;; refinement step
    $B \leftarrow$ Branch($u$) ;;; branching step
    $B' \leftarrow$ Prune($B$) ;;; pruning step
    if $B' = \emptyset$ then return(failure)
    nondeterministically choose $v \in B'$
    return(Abstract-search($v$))
end
```
Planning-Graph Planning

- Abstract-search with iterative deepening wrapped around it
  - Refinement (planning-graph extension): propagate constraints
  - Branching (in solution extraction): actions that achieve subgoals
  - Pruning (in solution extraction): use mutex and no good info to prune actions

for \( k = 1, 2, \ldots \)

Abstract-search(\( u \))
if Terminal(\( u \)) then return(\( u \))
\( u \leftarrow \text{Refine}(u) \); refinement step
\( B \leftarrow \text{Branch}(u) \); branching step
\( B' \leftarrow \text{Prune}(B) \); pruning step
if \( B' = \emptyset \) then return(failure)
nondeterministically choose \( v \in B' \)
return(Abstract-search(\( v \)))

end
SAT Planning

- Refinement: the Unit-Propagate procedure
- Branching: the selection of some proposition, and calling Davis-Putnam with both possible values for that proposition

for \( k = 1, 2, \ldots \)

... 

Abstract-search(\( u \))

if Terminal(\( u \)) then return(\( u \))

\( u \leftarrow \) Refine(\( u \)) ;; refinement step

\( B \leftarrow \) Branch(\( u \)) ;; branching step

\( B' \leftarrow \) Prune(\( B \)) ;; pruning step

if \( B' = \emptyset \) then return(failure)

nondeterministically choose \( v \in B' \)

return(Abstract-search(\( v \)))

end