CS 313 Lecture 30

Prolog:
[removing from BST]
Unification
Prolog control
remove(N, T, Result)

should remove the number N from tree T and return the result in Result. If N is not contained in T, nothing should happen (i.e., Result = T). The algorithm for deletion is as follows (let N be the node to be deleted):

- if N is a leaf, return nil
- if N has only one child, return that child
- if N has two children, replace N's value with the maximum value in the left subtree, and remove that value from the left subtree.
HW 9: removing from a BST

• Ex: remove node 3
• case 1: leaf
HW 9: removing from a BST

• Ex: remove node 3
• case 1: leaf
• case 2: only left child
HW 9: removing from a BST

- Ex: remove node 3
- case 1: leaf
- case 2: only left child
- case 3: only right child
HW 9: removing from a BST

- Ex: remove node 30
- case 1: leaf
- case 2: only left child
- case 3: only right child
- case 4: two children
HW 9: removing from a BST

• Ex: remove node 30
• case 1: leaf
• case 2: only left child
• case 3: only right child
• case 4: two children
  • find predecessor P
  • replace node with P
  • left child of P moves up
Unification

• How does Prolog solve queries like

?- X = p(a).
 X = p(a).

?- p(X) = p(a).
 X = a.

?- [1, 2, 3] = [H|T].
 H = 1,
 T = [2, 3].

?- [A, B, 3] = [1|T].
 A = 1,
 T = [B, 3].
Unification  (Sethi 11.5)

• Substitution  $\sigma =$

  function from variable to terms

  Ex: $\sigma = \{X \rightarrow f(a), \ Y \rightarrow [2, \ 3]\}$

• $T\sigma = \text{apply } \sigma \text{ to term } T$

  Ex: $f(X)\sigma = f(f(a))$

  $[Z|Y]\sigma = [Z, \ 2, \ 3]$
Unification (Sethi 11.5)

• U is *instance* of T if \( U = T\sigma \) for some \( \sigma \)

• \( T_1 \) and \( T_2 \) *unify* if they have common instance
  \[ \exists \sigma \text{ s.t. } T_1\sigma = T_2\sigma \]

• Prolog will use *most general unifier* to unify two terms

\[
\text{?- [H|T] = [A, B].}
\]
\[
H = A, \quad H = 1, \quad A = 1
\]
\[
T = [B], \quad T = [2], \quad B = 2.
\]
Prolog control

• Depth-first search of goal tree

\[
\text{memb}(X, \left[ X | _ \right]). \\
\text{memb}(X, \left[ _ | T \right]) :- \\
\quad \text{memb}(X, T).
\]
Prolog control

Search algorithm

• Start with query as goal \( G \):
  
  visit(\( G \)),
  fail

\[\text{visit}(G) :\]

if \( G = \emptyset \): succeed

else:

let \( G_1, \ldots, G_k = G \)

choose first subgoal \( G_1 \)

for each matching rule \( A : - B_1, \ldots, B_j \)

let \( \sigma \) be most general unifier of \( G_1, A \)

new goal \( G' = B_1\sigma, ..., B_j\sigma, G_2\sigma, ..., G_k\sigma \)

visit(\( G' \))

backtrack using recursion (try other rules, subgoals)
Rule order vs goal order

• Rule order determines order of solutions (permutes branches)

• Goal order affects solutions found (changes shape of tree)
Rule order

member.pl:

\[
\text{memb1}(X, [X|\_]). \\
\text{memb1}(X, [_|T]) : - \text{memb1}(X, T). \\
\text{memb2}(X, [_|T]) : - \text{memb2}(X, T). \\
\text{memb2}(X, [X|\_]).
\]

?- \text{memb}(X, [1, 2, 3])
\text{X=1} \quad \text{T=[2, 3]}
\text{memb}(1, [1|[2, 3]]) \quad \text{memb}(X, [2, 3])
true.
\text{X=2} \quad \text{T=[3]}
\text{memb}(2, [2|[3]]) \quad \text{memb}(X, [3])
true.
\text{X=3} \quad \text{T=[]} \\
\text{memb}(3, [3|[]]) \quad \text{memb}(X, [])
true.
\text{memb}(X, [])
false.