1

Logic Programming Proof Procedure

• A goal is either a sequence of literals or the empty goal ❒ (which can always be achieved, and therefore indicates success).

e.g., on(X, Y)
p(X, Y), q(V)

• A sequence of goals

$<G_1, G_2, \ldots G_n>$

is a top-down proof of G_1 iff G_n = ❑ and G_i is a goal inferred from G_{i-1} for i = 2, 3, ...n.

• Let $G_{i-1} = \{l_1, l_2, \ldots, l_m\}$, where l_i is a literal: goal G_i can be inferred from G_{i-1} if one of the following conditions holds:

• There is an atomic clause t in the program such that t and l_j unify with mgu σ and $G_i = \{l_1\sigma, \ldots l_{j-1}\sigma, l_{j+1}\sigma, \ldots l_m\sigma\}$

• There is a clause of the form $t :- t_1, t_2, \ldots, t_r$ such that t and l_j unify with mgu σ and

$G_i = \{l_1\sigma, \ldots l_{j-1}\sigma, t_1\sigma, \ldots, t_r\sigma, l_{j+1}\sigma, \ldots l_m\sigma\}$

©1993 John Mylopoulos

2

An Example

```
on(b4,b5).                              (1)
on(b5,b6).                              (2)
above(X,Y) :- on(X, Y).                 (3)
above(X,Y) :- on(X,Z),above(Z,Y).       (4)
above(c1,c2).                           (5)
```

Inferred goals from `?- above(A, B)`

- `?- on(A, B)  s = {X/A, Y/B}. from (3)`
- `?- on(A,Z),above(Z,B)  s = {X/A, Y/B} from (4)`
- ❒ `s = {A/c1, B/c2}  from (5)`

©1993 John Mylopoulos

3

Another Example

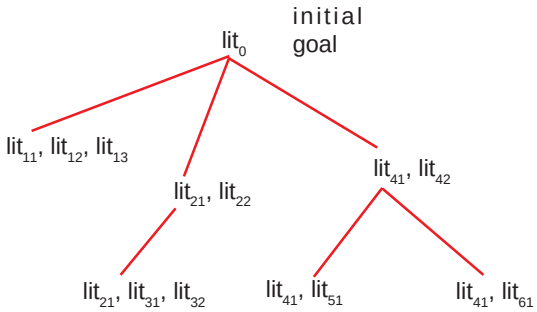
```
on(b4,b5).                          (1)
on(b5,b6).                          (2)
above(X, Y) :- on(X,Y).             (3)
above(X,Y):-on(X,Z),above(Z,Y).     (4)
above(c1,c2).                       (5)
```

Inferred goals from `?-on(A,Z1),above(Z1,B)`

- `?- above(b5,B)`
  σ = {A/b4, Z_1/b5} from (1) and 1st literal
- `?- above(b6,B)`
  σ = {A/b5, Z_1/b6} from (2) and 1st literal
- `?- on(A,c1)`
  σ = {Z_1/c1, B/c2} from (5) and 2nd literal
- ?- `on(Z1, B), on(A, Z1)`
  σ = {X/Z_1, Y/B} from (3) and 2nd
- `?- on(Z1,Z),above(Z,B),on(A,Z1)`
  σ = { X/Z_1, Y/B} from (4) and 2nd

4

Generating a Proof Tree

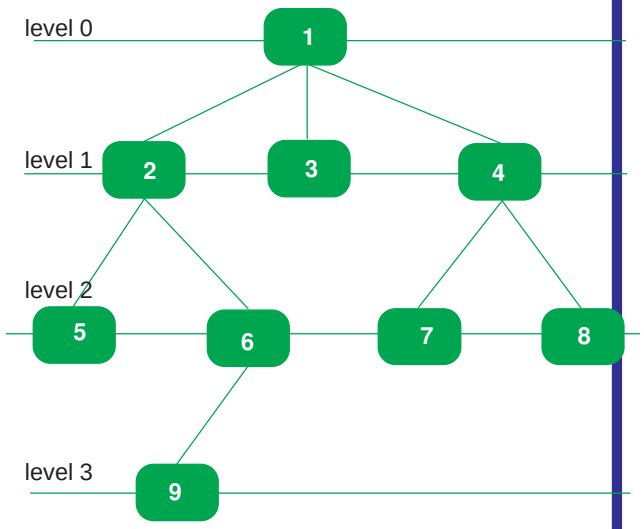


- Which inferred goal to consider next?
- Which literal within that goal?
- Which clause to unify it against?

These are questions of search!

5

Breadth-First Search



$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9$

©1993 John Mylopoulos

6

Breadth-First Search

Definition: Visit all nodes at level i before visiting any nodes at level (i+1)

Definition of Level:
- initial goal is at level 0
- given goal at level i, its subgoals are all at level (i+1)

Breadth-First Search

Let $\gamma \leftarrow G_0$ (the initial goal);
q $\leftarrow$ [] (the queue of goals to be visited;

repeat:
If γ is the success goal, return(success);

Generate the list of all subgoals of γ, call it L;
q $\leftarrow$ append(q, L);

If q is empty, return(failure);
$\gamma \leftarrow$ top(q);
Goto repeat;

©1993 John Mylopoulos

7

Breadth-First Search

The good news: If there is a proof, breadth-first search is quaranteed to find it; in fact, it is quaranteed to find the proof of shortest length.

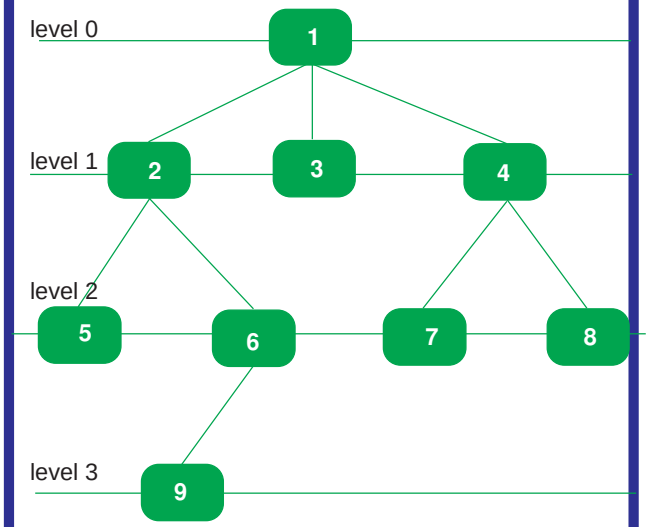
The bad news: The nodes of an entire level need to be stored on a queue; space requirements grow exponentially with the length of a proof

Variations

Recompute the entire goal subtree at depth 1, then 2, then 3,...

8

Depth-First Search



$1 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 9 \rightarrow 3 \rightarrow 4 \rightarrow 7 \rightarrow 8$

9

Depth-First Search

Depth-First Search

Let g ¨ G_0 (the initial goal);
q ¨ [] (the stack of goals to be visited;

repeat:
If g is the success goal, return(success);

Generate the list of all subgoals of g, call it L;
q ¨ append(L, q); /* new goals added at the top of the stack */

If q is empty, return(failure);
g ¨ top(q);
Goto repeat;

©1993 John Mylopoulos

10

Depth-First Search

The bad news: Depth-first search may not find a proof, even if one exists; nor is it quaranteed to find the shortest one.

The good news: The number of nodes that needs to be stored on the search stack is proportional to the depth of the search; for a proof of length n, O(n) space is required.

Variations

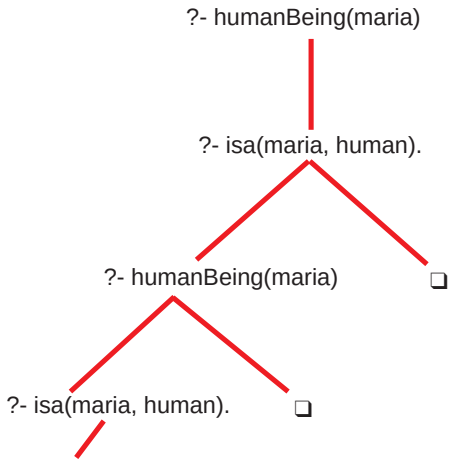
Recompute the entire goal subtree at depth 1, then 2, then 3,...

©1993 John Mylopoulos

11

Infinite Loop, Prolog-Style

```
humanBeing(X) :- isa(X, human).
isa(X, human) :- humanBeing(X).
isa(maria, human).
```



12

Heuristic Search

Use a function f(G) which evaluates goals

$$f(G_1) \leq f(G_2)$$

means that G_1 is "more promising" than G_2, or the cost of a solution path through G_1 is less than that of a solution path through G_2

Often it makes sense to define f as an additive function

$$f(G) = p(G) + h(G)$$

where p(G) measures the cost of a path from the initial goal G_0 to G and h(G) estimates the cost of a path to a success node.

A conservative search strategy emphasizes p(G), thereby leading to a breadth-first-like search; an optimistic search strategy emphasizes h(G) and leads to a depth-first-like search.

13

Heuristic Search

Let $\gamma \leftarrow G_0$ (the initial goal);
q ← [] (the list of goals to be visited;

repeat:
If γ is the success goal, return(success);

Generate the list of all subgoals of γ, call it L;
q ← append(L, q);
q ← sort(q, f) /* sort the nodes in q according to a heuristic function f */

If q is empty, return(failure);
$\gamma \leftarrow$ top(q);
Goto repeat;

14

Prolog =
Logic Programming + Prolog Search Strategy

Prolog Search Strategy

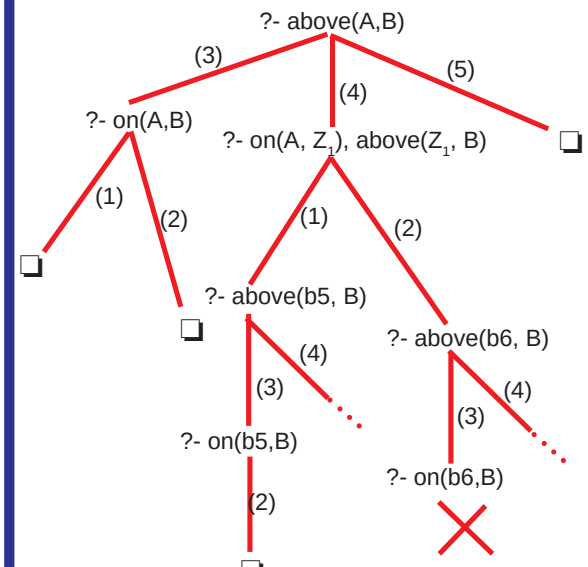
- To select subgoal, use depth-first search strategy
- To select literal within a subgoal, start at the left, move right
- To select clause against which to unify chosen literal, assume chronological ordering for all clauses (oldest ones are tried out first)

Prolog can miss proofs;
Prolog inference engine requires little storage

15

An Example

```
on(b4,b5).                          (1)
on(b5,b6).                          (2)
above(X, Y) :- on(X,Y).             (3)
above(X,Y):-on(X,Z),above(Z,Y).     (4)
above(c1,c2).                       (5)
```



16

Prolog Search vs Procedure Calling

Consider

```
p(X) :- <body1>.
p(X) :- <body2>.
```

and goal

```
?- p(a)
```

`p(X)` is like a procedure, but it has two definitions which will be tried in order to find one that succeeds

• Automatic Backtracking for Prolog "procedure calls"

Consider

```
q(a, b).   q(a, c).      q(a, d).
r(c).   r(d).
p(X) :- q(X, Y), r(Y).
```

Call `?- p(a)`

assign `Y/b` then fail
try `Y/c` then succeed
(if more proofs needed)
try `Y/d` then succeed

17

Prolog Search vs Procedure Calling

• No need for arguments in Prolog "calls". For example,

```
?- p(X)
```

is acceptable in Prolog, values will be bound to `X` during the search for a proof

The values of variables are constrained by goal, clauses rather than given explicitly

```
p(X, Y) :- q(X), r(Y).
q(b).   r(a).  r(b).  r(c).
```

`?- p(Z, Z)` doesn't give values to `X, Y`, but constrains them to be equal

Likewise, in

```
r(s(X),Y) :- ...      ?- r(Z,p(Z))
```

`Z` is forced to be `s(...)` and `Y` to be `p(Z)` = `p(s(...))`

©1993 John Mylopoulos