Logic Programming (PLP 11)
Prolog Imperative Control Flow:
Backtracking, Cut, Fail, Not

Carlos Varela
Rensselaer Polytechnic Institute

February 5, 2015
Backtracking

- *Forward chaining* goes from axioms forward into goals.

- *Backward chaining* starts from goals and works backwards to prove them with existing axioms.
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X).

Backtracking example

snowy(C)

\_C = \_X

success

cold(seattle)
fails;
backtrack.

cold(X)

X = rochester

OR

cold(rochester)

snowy(X)

AND

cold(rochester)

rainy(X)

rainy(seattle)

OR

X = seattle

X = rochester
Imperative Control Flow

- Programmer has *explicit control* on backtracking process.

**Cut (!)**

- As a goal it succeeds, but with a *side effect*:
  - Commits interpreter to choices made since unifying parent goal with left-hand side of current rule.
Cut (!) Example

rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), !, cold(X).
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), !, cold(X).

Cut (!) Example

cold(seattle) fails; no backtracking to rainy(X).
GOAL FAILS.
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), !, cold(X).
snowy(troy).
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), !, cold(X).
snowy(troy).

_C = _X

C = troy FAILS
snowy(X) is committed to bindings (X = seattle).
GOAL FAILS.

X = seattle
rainy(seattle)

OR

X = seattle

OR

C = troy

snowy(troy)
cold(troy)
cold(rochester)

C = troy

snowy(X)
cold(X)

AND

C = troy

FAILS

snowy(X) is committed to bindings (X = seattle).
GOAL FAILS.

C = troy

snowy(troy)
cold(troy)
cold(rochester)

C = troy

FAILS

snowy(X) is committed to bindings (X = seattle).
GOAL FAILS.
rainy(seattle) :- !.
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X).
snowy(troy).
Cut (!) Example 3

rainy(seattle) :- !.
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X).
snowy(troy).

C = troy succeeds
Only rainy(X) is committed to bindings (X = seattle).

C = _C = _X

X = seattle

rainy(seattle)
nasty(rochester)
cold(rochester)
Cut (!) Example 4

rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- !, rainy(X), cold(X).
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- !, rainy(X), cold(X).

_C = _X

success

cold(seattle)
fail;
backtrack.

cold(X)

X = rochester

OR

rainy(rochester)
cold(rochester)

X = seattle

OR

rainy(seattle)
x

AND

snowy(X)
snowy(C)

C. Varela
Cut (!) Example 5

rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X), !.
Cut (!) Example 5

rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X), !.

\_C = \_X

\text{success}

\text{AND}

\text{OR}

\text{OR}

\text{rainy(seattle)}

\text{rainy(rochester)}

\text{cold(rochester)}
## First-Class Terms

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>call(P)</td>
<td>Invoke predicate as a goal.</td>
</tr>
<tr>
<td>assert(P)</td>
<td>Adds predicate to database.</td>
</tr>
<tr>
<td>retract(P)</td>
<td>Removes predicate from database.</td>
</tr>
<tr>
<td>functor(T,F,A)</td>
<td>Succeeds if T is a term with functor F and arity A.</td>
</tr>
<tr>
<td>findall(F,P,L)</td>
<td>Returns a list L with elements F satisfying predicate P.</td>
</tr>
</tbody>
</table>
**not P is not ¬P**

- In Prolog, the database of facts and rules includes a list of things assumed to be **true**.
- It does not include anything assumed to be **false**.
- Unless our database contains everything that is **true** (the *closed-world assumption*), the goal **not P** (or \(+ P\) in some Prolog implementations) can succeed simply because our current knowledge is insufficient to prove \(P\).
More not vs ¬

?- snowy(X).
X = rochester
?- not(snowy(X)).
no

Prolog does not reply: \textit{X = seattle}.

The meaning of \texttt{not(snowy(X))} is:

\[ \neg \exists X \ [ \text{snowy}(X) ] \]

rather than:

\[ \exists X \ [ \neg \text{snowy}(X) ] \]
## Fail, true, repeat

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>fail</td>
<td>Fails current goal.</td>
</tr>
<tr>
<td>true</td>
<td>Always succeeds.</td>
</tr>
<tr>
<td>repeat</td>
<td>Always succeeds, provides infinite choice points.</td>
</tr>
</tbody>
</table>

repeat.

repeat :- repeat.
**not Semantics**

\[
\text{not}(P) \ :- \ \text{call}(P), !, \text{fail}.
\]
\[
\text{not}(P).
\]

Definition of **not** in terms of failure (**fail**) means that variable bindings are lost whenever **not** succeeds, e.g.:

\[
?- \ \text{not} (\text{not}(\text{snowy}(X))).
\]
\[
X = \_G147
\]
Conditionals and Loops

statement :- condition, !, then.
statement :- else.

natural(1).
natural(N) :- natural(M), N is M+1.
my_loop(N) :- N>0,
            natural(I),
            write(I), nl,
            I=N,
            !, fail.

Also called generate-and-test.
Prolog lists

• \([a,b,c]\) is syntactic sugar for:

\[. (a, . (b, . (c, [])))\]

where \([]\) is the empty list, and \(\).\ is a built-in cons-like functor.

• \([a,b,c]\) can also be expressed as:

\([a | [b,c]]\), or
\([a, b | [c]]\), or
\([a,b,c | []]\)
Prolog lists append example

append([], L, L).
append([H|T], A, [H|L]) :- append(T, A, L).
8. What do the following Prolog queries do?

?- repeat.

?- repeat, true.

?- repeat, fail.

Corroborate your thinking with a Prolog interpreter.

9. Draw the search tree for the query “\texttt{not(not(snowy(City)))}”. When are variables bound/unbound in the search/backtracking process?

10. PLP Exercise 11.7 (pg 571).