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

Carlos Varela
Rensselaer Polytechnic Institute

January 30, 2012
Backtracking

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

- *Backward chaining* starts from goals and works backwards to prove them with existing axioms.
Backtracking example

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

_C = _X

snowy(C)
snowy(X)

success

X = seattle

rainy(X)

AND

cold(seattle) fails; backtrack.

cold(X)

X = rochester

rainy(rochester)

OR

cold(rochester)

_X

rainy(seattle)
Imperative Control Flow

• Programmer has \textit{explicit control} on backtracking process.

\textbf{Cut} (!)

• As a goal it succeeds, but with a \textbf{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).

GOAL FAILS.

cold(seattle) fails; no backtracking to rainy(X).

C. Varela
Cut (!) Example 2

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 = troy FAILS
snowy(X) is committed to bindings (X = seattle).
GOAL FAILS.
Cut (!) Example 3

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 = _X

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

X = seattle

C. Varela

10
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).
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), !.

\[ \_C = \_X \]

_X = seattle
_X = rochester

\[ \text{success} \]

\[ \text{C. Varela} \]
First-Class Terms

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

- 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 \( \backslash + P \) in some Prolog implementations) can succeed simply because our current knowledge is insufficient to prove P.
More \textbf{not} vs $\neg$

\begin{verbatim}
?- snowy(X).
  X = rochester
  ?- not(snowy(X)).
    no

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

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

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

rather than:

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

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

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

\[
?- \text{not(not(snowy(X)))}.
\]
\[
X=_G147
\]
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), I<=N,
            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]]\text{, or}\n\[[a, b | [c]]\text{, or}\n\[[a,b,c | []]\]
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 “\text{not(not(snowy(City)))}”. When are variables bound/unbound in the search/backtracking process?

10. PLP Exercise 11.6 (pg 571).

11. PLP Exercise 11.7 (pg 571).