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

Carlos Varela
Rensselaer Polytechnic Institute

January 31, 2013
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

success

cold(seattle) fails; backtrack.

\_C = \_X

X = seattle

OR

X = rochester

OR

AND

cold(X),

cold(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).

_C = _X

_x = seattle

rainy(X)

AND

!?

OR

rainy(seattle) rainy(rochester)

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

GOAL FAILS.
cold(rochester)
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.

C = troy

x = seattle

_C = _X

snowy(C)

snowy(X)
snowy(troy)

AND

rainy(X)
cold(X)

OR

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

C = troy

_\_C = _\_X

X = seattle

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

rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- !, rainy(X), cold(X).
Cut (!) Example 4

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

\[
\begin{align*}
\text{snowy}(C) & \quad \text{snowy}(X) \\
\text{\_C = \_X} & \quad \text{success} \\
\text{AND} & \\
\text{rainy}(X) & \quad \text{cold}(\text{seattle}) \\
\text{OR} & \quad \text{fails; \ backtrack.} \\
\text{x = seattle} & \\
\text{rainy}(\text{seattle}) & \\
\text{x = rochester} & \quad \text{cold}(\text{rochester}) \\
\text{rainy}(\text{rochester}) &
\end{align*}
\]
Cut (!) Example 5

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

\[ \text{X = seattle} \]
\[ \text{X = rochester} \]

\[ \text{success} \]
# First-Class Terms

<table>
<thead>
<tr>
<th>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 \( + P \) in some Prolog implementations) can succeed simply because our current knowledge is insufficient to prove P.
More not vs \( \neg \)

\[
\begin{align*}
? &- \text{ snowy}(X) . \\
X & = \text{ rochester} \\
? &- \text{ not} (\text{ snowy}(X)) . \\
\text{ no}
\end{align*}
\]

Prolog does not reply: \( X = \text{ seattle} . \)

The meaning of \( \text{ not} (\text{ snowy}(X)) \) 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><strong>fail</strong></td>
<td>Fails current goal.</td>
</tr>
<tr>
<td><strong>true</strong></td>
<td>Always succeeds.</td>
</tr>
<tr>
<td><strong>repeat</strong></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 \textit{not} in terms of failure (\texttt{fail}) means that variable bindings are lost whenever \textit{not} succeeds, e.g.:

\[
?- \text{not}(\text{not}(	ext{snowy}(X))).
\]
\[
X=_G147
\]
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 \(\cdot\) is a built-in cons-like functor.

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

\[ [a | [b,c]] \text{, or} \]
\[ [a, b | [c]] \text{, or} \]
\[ [a,b,c | []] \]
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 11.6 (pg 571).

11. PLP Exercise 11.7 (pg 571).