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

Carlos Varela
Rensselaer Polytechnic Institute

September 2, 2014
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

- \( X = \text{seattle} \)
- \( \text{cold(seattle)} \) fails; backtrack.

- \( X = \text{rochester} \)
- \( \text{cold(rochester)} \)
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. Varela

GOAL FAILS.
Cut (!) Example 2

rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), !, cold(X).
snowy(troy).
Cut (!) Example 2

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 = _X

snowy(X)
AND

X = seattle

C = troy

snowy(troy)
OR

cold(X)

rainy(rochester)
OR

cold(rochester)

_ C = _ X

rainy(seattle)
rainy(rochester)
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

_x = seattle

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

C = troy

SUCCEEDS
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).

Cut (!) Example 4

X = seattle

Or

rainy(seattle)

AND

C = X

success

cold(seattle) fails; backtrack.

X = rochester

Or

rainy(rochester)
cold(rochester)
cold(rochester)
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), !.

_C = _X

success

snowy(C)
snowy(X)

AND

rainy(X)

OR

X = seattle

X = rochester

cold(X)
cold(rochester)
# First-Class Terms

<table>
<thead>
<tr>
<th>call(P)</th>
<th>Invoke predicate as a goal.</th>
</tr>
</thead>
<tbody>
<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>
• 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 \( \\neg P \) in some Prolog implementations) can succeed simply because our current knowledge is insufficient to prove \( P \).
More \textit{not} vs $\neg$

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

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

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

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

rather than:

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

<table>
<thead>
<tr>
<th></th>
<th>Description</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>

```prolog
repeat.  
repeat  :- repeat.  
```
not Semantics

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

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

\[
?\quad \text{not(\texttt{not(snowy(X))}).} \\
X=\_G147
\]
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),
            write(I), nl,
            I=N,
            !, fail.

Also called *generate-and-test*.
Prolog lists

• \([a, b, c]\) is syntactic sugar for:

\[\text{.}(a, \text{.}(b, \text{.}(c, [])))\]

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

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

\[[a \mid [b, c]]\], or
\[[a, b \mid [c]]\], or
\[[a, b, c \mid []]\]
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).