Logic Programming
(PLP 11, CTM 9.3)
Prolog Imperative Control Flow:
Backtracking, Cut, Fail, Not
Lists, Append

Carlos Varela
Rensselaer Polytechnic Institute

November 15, 2016
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).

\[\begin{align*}
_\text{C} &= _\text{X} \\
\text{snowy(C)} \\
\text{snowy(X)} \\
\text{AND} \\
\text{rainy(X)} \\
\text{OR} \\
\text{X = seattle} \\
\text{rainy(seattle)} \\
\text{OR} \\
\text{X = rochester} \\
\text{rainy(rochester)} \\
\text{AND} \\
\text{cold(X)} \\
\text{cold(rochester)} \\
\text{cold(seattle)} &\text{ fails; backtrack.} \\
\text{success}
\end{align*}\]

C. Varela
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. Choices include variable unifications and rule to satisfy the parent goal.
Cut (!) Example

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

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

\textbf{C = troy FAILS}

\textit{snowy(X) is committed to bindings (X = seattle).}

\textbf{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 = _\_X

\_C = \_X

X = seattle

rainy(seattle)

OR

OR

rainy(rochester)

cold(rochester)

C = troy SUCCEEDS

Only rainy(X) is committed to bindings (X = seattle).
Cut (!) Example 4

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

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

The diagram illustrates the execution of the `snowy(X)` rule, showing the possible states and outcomes of the query for different values of `X`. The query succeeds for `X = rochester` and fails for `X = seattle` due to the cut (!) operator preventing further exploration of the query for `X = seattle`.
Cut (!) Example 5

\begin{verbatim}
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X), !.
\end{verbatim}
rainy(seattle).
rainy(rochester).
cold(rochester).
snowy(X) :- rainy(X), cold(X), !.

Cut (!) Example 5
# First-Class Terms

<table>
<thead>
<tr>
<th>Predicate</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 $\neg$

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

Prolog does not reply: X = 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>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

not(P) :- call(P), !, fail.
not(P).

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

?- 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),
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:

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

append([],L,L).
append([H|T], A, [H|L]) :- append(T,A,L).
Oz lists (Review)

• \([a \ b \ c]\) is syntactic sugar for:
  
  `'|'(a '|'(b '|'(c nil)))
  
  where nil is the empty list, and '|' is the tuple’s functor.

• A list has two components:
  a head, and a tail

  declare L = [6 7 8]
  L.1 gives 6
  L.2 give [7 8]
Oz lists append example

```
proc {Append Xs Ys Zs}
    choice Xs = nil Zs = Ys
    [] X Xr Zr in
        Xs=X|Xr
        Zs=X|Zr
        {Append Xr Ys Zr}
    end
end

% new search query
proc {P S}
    X Y in
        {Append X Y [1 2 3]} S=X#Y
    end

% new search engine
E = {New Search.object script(P)}

% calculate and display one at a time
{Browse {E next($)}}

% calculate all
{Browse {Search.base.all P}}
```
Exercises

79. What do the following Prolog queries do?

?- repeat.

?- repeat, true.

?- repeat, fail.

Corroborate your thinking with a Prolog interpreter.

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

81. PLP Exercise 11.7 (pg 571).

82. Write the students example in Oz (including the has\_taken(Student, Course) inference).