Logic Programming (PLP 11)
Prolog: Arithmetic, Equalities, Operators, I/O,
Natural Language Parsing

Carlos Varela
Rensselaer Polytechnic Institute

September 5, 2014
Arithmetic Goals

- $N > M$
- $N < M$
- $N =< M$
- $N >= M$

- $N$ and $M$ must be bound to numbers for these tests to succeed or fail.

- $X \text{ is } 1+2$ is used to assign numeric value of right-hand-side to variable in left-hand-side.
natural(1).

natural(N) :- natural(M), N is M+1.

my_loop(N) :- N>0,
natural(I),
write(I), nl,
I=N,
!.

my_loop(_).

Also called *generate-and-test*. 

C. Varela
\( = \) is not equal to \( == \) or \( =:= \)

\( X=Y \) \( X\neq Y \)
test whether \( X \) and \( Y \) can be or cannot be unified.

\( X==Y \) \( X\\=\=Y \)
test whether \( X \) and \( Y \) are currently co-bound, i.e., have been bound to, or share the same value.

\( X=:=Y \) \( X\\=\=Y \)
test arithmetic equality and inequality.
More equalities

\[ X = @= Y \quad \text{and} \quad X \backslash = @= Y \]

test whether \( X \) and \( Y \) are \textit{structurally identical}.

- \( =@= \) is weaker than \( == \) but stronger than \( = \).

- Examples:

\[
\begin{align*}
\text{a} &= @= \text{A} & \text{false} \\
\text{A} &= @= \text{B} & \text{true} \\
x (\text{A}, \text{A}) &= @= x (\text{B}, \text{C}) & \text{false} \\
x (\text{A}, \text{A}) &= @= x (\text{B}, \text{B}) & \text{true} \\
x (\text{A}, \text{B}) &= @= x (\text{C}, \text{D}) & \text{true}
\end{align*}
\]
More on equalities

\[ \text{More on equalities} \]

\[ X == Y \Rightarrow \quad x = @= Y \Rightarrow \quad X = Y \]

but not the other way (\( \Leftarrow \)).

- If two terms are currently co-bound, they are structurally identical, and therefore they can unify.
- Examples:

\[
\begin{align*}
  a &= @= A & \text{false} \\
  A &= @= B & \text{true} \\
  x (A, A) &= @= x (B, C) & \text{false} \\
  x (A, A) &= @= x (B, B) & \text{true} \\
  x (A, B) &= @= x (C, D) & \text{true}
\end{align*}
\]
Prolog Operators

:- op(P,T,O)

    declares an operator symbol O with precedence P and type T.

• Example:

  :- op(500,xfx,'has_color')

  a has_color red.
  b has_color blue.

then:

  ?- b has_color C.
  C = blue.
  ?- What has_color red.
  What = a.

C. Varela
## Operator precedence/type

- **Precedence** $P$ is an integer: the larger the number, the less the precedence (*ability to group*).
- **Type** $T$ is one of:

<table>
<thead>
<tr>
<th>$T$</th>
<th>Position</th>
<th>Associativity</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>$xfx$</td>
<td>Infix</td>
<td>Non-associative</td>
<td>$is$</td>
</tr>
<tr>
<td>$xfy$</td>
<td>Infix</td>
<td>Right-associative</td>
<td>$, ;$</td>
</tr>
<tr>
<td>$yfx$</td>
<td>Infix</td>
<td>Left-associative</td>
<td>$+ - * /$</td>
</tr>
<tr>
<td>$fx$</td>
<td>Prefix</td>
<td>Non-associative</td>
<td>$? -$</td>
</tr>
<tr>
<td>$fy$</td>
<td>Prefix</td>
<td>Right-associative</td>
<td></td>
</tr>
<tr>
<td>$xf$</td>
<td>Postfix</td>
<td>Non-associative</td>
<td></td>
</tr>
<tr>
<td>$yf$</td>
<td>Postfix</td>
<td>Left-associative</td>
<td></td>
</tr>
</tbody>
</table>
Testing types

atom(X)

tests whether X is an atom, e.g., ‘foo’, bar.

integer(X)

tests whether X is an integer; it does not test for complex terms, e.g., integer(4/2) fails.

float(X)

tests whether X is a float; it matches exact type.

string(X)

tests whether X is a string, enclosed in ` ` ... ` `.
**Prolog Input**

- **seeing**(X)
  
  succeeds if X is (or can be) bound to *current read port*.
  
  X = user is keyboard (standard input.)

- **see**(X)
  
  *opens* port for input file bound to X, and makes it *current*.

- **seen**
  
  *closes* current port for input file, and makes user *current*.

- **read**(X)
  
  *reads* Prolog type expression from *current* port, storing value in X.

- **end-of-file**
  
  is returned by **read** at <end-of-file>.
Prolog Output

\textbf{telling}(X)

succeeds if $X$ is (or can be) bound to \textit{current output port}.

$X = \text{user}$ is screen (standard output.)

\textbf{tell}(X)

\textit{opens} port for output file bound to $X$, and makes it \textit{current}.

\textbf{told}

\textit{closes} current output port, and reverses to screen output
(makes $\text{user}$ \textit{current}.)

\textbf{write}(X)

\textit{writes} Prolog expression bound to $X$ into \textit{current} output port.

\textbf{nl}

new line (line feed).

\textbf{tab}(N)

writes $N$ spaces to current output port.
I/O Example

browse(File) :-
    seeing(Old), /* save for later */
    see(File), /* open this file */
repeat,
read(Data), /* read from File */
process(Data),
seen, /* close File */
see(Old), /* prev read source */
!. /* stop now */

process(end_of_file) :- !.
process(Data) :- write(Data), nl, fail.
# First-Class Terms Revisited

<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 all elements F satisfying predicate P.</td>
</tr>
<tr>
<td><code>clause(H,B)</code></td>
<td>Succeeds if the clause H :- B can be found in the database.</td>
</tr>
</tbody>
</table>
word(article,a).
word(article,every).
word(noun,criminal).
word(noun,'big kahuna burger').
word(verb,eats).
word(verb,likes).

sentence(Word1,Word2,Word3,Word4,Word5) :-
    word(article,Word1),
    word(noun,Word2),
    word(verb,Word3),
    word(article,Word4),
    word(noun,Word5).
• *Definite Clause Grammars (DCG)* are useful for natural language parsing.

• Prolog can load DCG rules and convert them automatically to Prolog parsing rules.
DCG Syntax

-->  

  DCG operator, e.g.,
  
  sentence --> subject, verb, object.

  Each goal is assumed to refer to the head of a DCG rule.

{prolog_code}

  Include Prolog code in generated parser, e.g.,
  
  subject --> modifier, noun, {write('subject')}.

[terminal_symbol]

  Terminal symbols of the grammar, e.g.,
  
  noun --> [cat].

C. Varela 16
Natural Language Parsing
(example rewritten using DCG)

sentence --> article, noun, verb, article, noun.

article --> [a] | [every].

noun --> [criminal] | ['big kahuna burger'].

verb --> [eats] | [likes].
Exercises

12. How would you translate DCG rules into Prolog rules?
13. PLP Exercise 11.8 (pg 571).