Logic Programming (PLP 11.3)

Prolog: Arithmetic, Equalities, Operators, I/O, Natural Language Parsing

Carlos Varela
Rennselaer Polytechnic Institute

September 10, 2007

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 is 1+2` is used to *assign* numeric value of right-hand-side to variable in left-hand-side.

Loop Revisited

```
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*.

= is not equal to == or =:=

`X=Y` `X\=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` `X\=@=Y`
test whether `X` and `Y` are *structurally identical*.

- `=@=` is weaker than `==` but stronger than `=`.
- Examples:

`a=@=A`	**false**
`A=@=B`	**true**
`x(A,A)=@=x(B,C)`	**false**
`x(A,A)=@=x(B,B)`	**true**
`x(A,B)=@=x(C,D)`	**true**

More on equalities

`X==Y` ⇒ `X=@=Y` ⇒ `X=Y`

but not the other way (⇐).

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

`a=@=A`	**false**
`A=@=B`	**true**
`x(A,A)=@=x(B,C)`	**false**
`x(A,A)=@=x(B,B)`	**true**
`x(A,B)=@=x(C,D)`	**true**

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.
```


Operator precedence/type

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

T	Position	Associativity	Examples
xfx	Infix	Non-associative	`is`
xfy	Infix	Right-associative	`, ;`
yfx	Infix	Left-associative	`+ - * /`
fx	Prefix	Non-associative	`?-`
fy	Prefix	Right-associative	
xf	Postfix	Non-associative	
yf	Postfix	Left-associative	

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

telling(X)
succeeds if X is (or can be) bound to *current output port*.
X = user is screen (standard output.)

tell(X)
opens port for output file bound to X, and makes it *current*.

told
closes current output port, and reverses to screen output (makes user *current*.)

write(X)
writes Prolog expression bound to X into *current* output port.

nl
new line (line feed).

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.
```

Natural Language Parsing

(Example from "Learn Prolog Now!" Online Tutorial)

```
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).
```


Parsing natural language

- *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].`

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 11.26 (pg 655).
14. *PLP Exercise 11.30 (pg 655).