Logic Programming

Lecture 8: Term manipulation & Meta-programming

Outline for today

• Special predicates for examining & manipulating terms
  • var, nonvar, functor, arg, =..  
• Meta-programming
  • call/1
  • Symbolic programming
  • Prolog self-interpretation

var/1 and nonvar/1

• var(X) holds
• var(a) does not hold
• nonvar(X) does not hold
• nonvar(a) holds
  • (tutorial questions...)
• None of these affects binding

Other term structure predicates

nonvar/1
atomic/1
number/1
integer/1
float/1

f(X,a)

[ ]

atom/1

a, b, ac, ...

integer/1

..., -1, 0, 1, 2, 3...

float/1

0.1, 10.42, ...
Structural equality

- The ==/2 operator tests whether two terms are **exactly** identical
  - Including variable names! (No unification!)

```
?- X == X.
yes
?- X == Y.
no
```
A difference list implementation

```prolog
bfs_dl([Node|Path], _, [Node|Path]) :-
goal(Node).

bfs_dl([Path|Paths], Z, Solution) :-
extend(Path, NewPaths),
append(NewPaths, Z1, Z),
Paths \== Z1, \% (Paths,Z1) is not empty DL
bfs_dl(Paths, Z1, Solution).

bfs_dl_start(N, P) :- bfs_dl([N|X], X, P).
```

functor/3

- Takes a term T, gives back the atom and arity
  
  ?- functor(a,F,N).
  F = a
  N = 0
  
  ?- functor(f(a,b),F,N).
  F = f
  N = 2

arg/3

- Given a number N and a complex term T
- Produce the Nth argument of T

  ?- arg(1, f(a,b), X).
  X = a

  ?- arg(2, f(a,b), X).
  X = b

=../2 ("univ")

- We can convert between terms and "universal" representation

  ?- f(a,f(b,c)) =.. X.
  X = [f,a,f(b,c)]

  ?- F =.. [g,a,f(b,c)].
  F = g(a,f(b,c))
Variables as goals

• Suppose we do
  \( X = \text{append}([1],[2],Y), X. \)
• What happens? (In Sicstus at least...)
  \( Y = [1,2] \)
• Clearly, this behavior is nonlogical
  • i.e. this does not work:
    \( X, X = \text{append}([1],[2],Y). \)

Call + = ..

• Can do some evil things...
  callwith(P,Args)
    :- Atom =.. [P|Args], call(Atom).
map(P,[],[]).
map(P,[X|Xs],[Y|Ys])
    :- callwith(P,[X,Y]), map(P,Xs,Ys)
plusone(N,M) :- M is N+1.
?- map(plusone,[1,2,3,4,5],L).
  L = [2,3,4,5,6].

call/1

• Given a Prolog term G, solve it as a goal
  \( ?- \text{call}([1],[2],X). \)
  \( X = [1,2]. \)

Symbolic programming

• Logical formulas
  prop(true).
  prop(false).
  prop(and(P,Q)) :- prop(P), prop(Q).
  prop(or(P,Q)) :- prop(P), prop(Q).
  prop(imp(P,Q)) :- prop(P), prop(Q).
  prop(not(P)) :- prop(P).
Formula simplification

simp(and(true,P),P).

simp(or(false,P),P).

simp(imp(P,false), not(P)).

simp(imp(true,P), P).

simp(and(P,Q), and(P1,Q)) :-
    simp(P,P1).

...

Satisfiability checking

• Given a formula, find a satisfying assignment for the atoms in it

• Assume atoms given [p₁,...,pₙ].

• A valuation is a list [(p₁,true|false),...]

• gen([],[]).

• gen([P|Ps], [(P,V)|PVs]) :-
    (V=true;V=false),
    gen(Ps,PVs).

Evaluation

sat(V,true).

sat(V,and(P,Q)) :- sat(V,P), sat(V,Q).

sat(V,or(P,Q)) :- sat(V,P) ; sat(V,Q).

sat(V,imp(P,Q)) :- \+(sat(V,P))
    ; sat(V,Q).

sat(V,not(P)) :- \+(sat(V,P)).

sat(V,P) :- atom(P),
    member((P,true),V).

Satisfiability

• Generate a valuation

• Test whether it satisfies Q

• (On failure, backtrack & try another valuation)
Prolog in Prolog

- Represent definite clauses
  
  ```prolog
  rule(Head,[Body, ...., Body]).
  ```

- A Prolog interpreter in Prolog:
  
  ```prolog
  prolog(Goal) :- rule(Goal,Body),
  prologs(Body)
  prologs([]).
  prologs([Goal|Goals]) :- prolog(Goal),
  prologs(Goals).
  ```

Example

```prolog
rule(p(X,Y), [q(X), r(Y)]).
rule(q(1)).
rule(r(2)).
rule(r(3)).
?- prolog(p(X,Y)).
X = 1
Y = 2
```}

OK, but so what?

- Prolog interpreter already runs programs...
- Self-interpretation is interesting because we can **examine** or **modify** behavior of interpreter.

Rules with "justifications"

```prolog
rule_pf(p(1,2), [], rule1).
rule_pf(p(X,Y), [q(X), r(Y)], rule2(X,Y)).
rule_pf(q(1), [], rule3).
rule_pf(r(2), [], rule4).
rule_pf(r(3), [], rule5).
```
Witnesses

• Produce **proof trees** showing which rules were used

\[
\text{prolog_pf(Goal,[Tag|Proof]) :-}
\text{  \hspace{1cm} rule_pf(Goal,Body,Tag),}
\text{  \hspace{1cm} prologs_pf(Body,Proof).}
\text{prologs_pf([],[]).}
\text{prologs_pf([Goal|Goals],[Proof|Proofs]) :-}
\text{  \hspace{1cm} prolog_pf(Goal,Proof),}
\text{  \hspace{1cm} prologs_pf(Goals,Proofs).}
\]

Other applications

• Tracing
  • Can implement `trace/1` this way

• Declarative debugging
  • Given an error in output, "zoom in" on input rules that were used
  • These are likely to be the ones with problems

Further reading:
• LPN, ch. 9
• Bratko, ch. 23

Next time
• Constraint logic programming
• Course review