Logic programming III

Henrik Boström
Stockholm University

- Arithmetics
- Meta-logical predicates
- Cut
- Extra-logical predicates
- Predicates to find all solutions

Arithmetics

X is Y	- The arithmetic expression Y is evaluated and unified with X.
X =:= Y	- The values of X and Y are the same.
X =\= Y	- The values of X and Y are different.
X < Y	- The value of X is less than the value of Y.
X > Y	- The value of X är is greater than the value of Y.
X =< Y	- The value of X is less than or equal to the value of Y.
X >= Y	- The value of X is greater than or equal to the value of Y.

Example

```
sum([],0).
sum([X|L],Sum):-
    sum(L,RestSum),
    Sum is RestSum+X.
```

```
sum(L,Sum):- sum(L,0,Sum).

sum([],Sum,Sum).
sum([X|L],PreviousSum,Sum):-
    SumSoFar is PreviousSum+X,
    sum(L,SumSoFar,Sum).
```

Meta-logical predicates

Term1 == Term2	- Term1 and Term2 are identical
Term1 \== Term2	- Term1 and Term2 are not identical
Term1 = Term2	- Term1 and Term2 are unified
var(X)	- X is an (uninstantiated) variable
nonvar(X)	- X is not an (uninstantiated) variable
ground(X)	- X does not contain a variable
atom(X)	- X is a constant and not a number
number(X)	- X is a number
atomic(X)	- X is a constant or a number
compound(X)	- X is a term with arity > 0

Metalogical predicates

functor(T,N,A) - The term T has the term name N and the arity A.

```
?- functor(p(a,X),N,A).
       N = p A = 2
?- functor(T,f,3).
       T = f(X,Y,Z)
```

arg(No,T,Arg) - Argument no. No in the term T is unified with Arg.

```
?- arg(3,p(0,s(0),s(s(0))),X).
       X = s(s(0))
?- arg(2,p(a,b),a).
       no
```

Meta-logical predicates

T =.. L - L is a list where the first element is the term name of T and the rest of the list concists of of the arguments of T.

```
?- p(a,s(X),Y) =.. L.
    L = [p,a,s(X),Y]

?- T =.. [a,b,c,d].
    T = a(b,c,d)
```

Unification with occur's check

```
unify(X,Y):- var(X), var(Y), X=Y.
unify(X,Y):- atomic(X), atomic(Y), X=Y.
unify(X,Y):- var(X), nonvar(Y), not_in(X,Y), X=Y.
unify(X,Y):- nonvar(X), var(Y), not_in(Y,X), X=Y.
unify(X,Y):- compound(X),compound(Y), term_unify(X,Y).

not_in(X,Y):- var(Y), X \== Y.
not_in(X,Y):- nonvar(Y), functor(Y,_,N), not_in(X,N,Y).

not_in(_,0,_).
not_in(X,N1,Y):- N1 > 0, arg(N1,Y,Arg), not_in(X,Arg),
                 N2 is N1-1, not_in(X,N2,Y).
```

Unification with occur's check (cont.)

```
term_unify(X,Y):-
        functor(X,F,N),
        functor(Y,F,N),
        unify_args(N,X,Y).

unify_args(0,_,_).
unify_args(N1,X,Y):-
        N1 > 0,
        arg(N1,X,ArgX), arg(N1,Y,ArgY),
        unify(ArgX,ArgY), N2 is N1-1,
        unify_args(N2,X,Y).
```

Cut (!)

Cut is a predicate that is used to reduce the number of (unnecessary) goal reductions. Assume we have a unary goal G and a clause G' :- G1, ..., Gi-1, !, Gi+1, ..., Gn, where G and G' have an mgu θ.

If cut has been reduced (after (G1, ..., Gi-1)θ) the following holds:

- No other clause will be used for reducing G.
- Alternative reductions for (G1, ..., Gi-1)θ will not be tested.
- The remaining goals (Gi+1, ..., Gn)θ are not affected by the cut.

Example (green cut)

```
merge([X|Xs],[Y|Ys],[X|Zs]):-
        X =< Y, !,
        merge(Xs,[Y|Ys],Zs).
merge([X|Xs],[Y|Ys],[Y|Zs]):-
        X > Y, !,
        merge([X|Xs],Ys,Zs).
merge([],Ys,Ys):-!.
merge(Xs,[],Xs).
```

Exempel (red cut)

```
delete(_,[],[]).
delete(X,[X|L1],L2):-
        delete(X,L1,L2).
delete(X,[Y|L1],[Y|L2]):-
        X \== Y,
        delete(X,L1,L2).
```

```
delete(_,[],[]).
delete(X,[X|L1],L2):- !, delete(X,L1,L2).
delete(X,[Y|L1],[Y|L2]):- delete(X,L1,L2).
```

Example

```
min(X,Y,X):- X =< Y.
min(X,Y,Y):- X > Y.
```

```
min(X,Y,X):- X =< Y, !.
min(X,Y,Y).
```

```
min(X,Y,Z):- X =< Y, !, X=Z.
min(X,Y,Y).
```

Example

```
if_then_else(P,Q,_):- P, Q.
if_then_else(P,_,R):- \+ P, R.
```

```
if_then_else(P,Q,_):- P, !, Q.
if_then_else(_,_,R):- R.
```

```
not(G):- G, !, fail.
not(_).
```

Extralogical predicates

read(X)	- read term from current stream
write(X)	- write term to current stream
see(File)	- open the file File for reading
seen	- close the current file
tell(File)	- open the file File for writing
told	- close the current file

?- write(‘Give answer: '), read(Answer).
Give answer: *yes.*
Answer = yes

Example

```
process_data(Input,Output):-
        see(Input),
        tell(Output),
        repeat,
        read(X),
        do_something(X),
        X == end_of_file,
        told,
        seen.

   do_something(end_of_file):-!.
   do_something(X):-
        transform(X,Y),
        write(Y), write('.'),nl.
```

Extra-logical predicates

assert(C)	- The clause C is added to memory
asserta(C)	- C is added first
assertz(C)	- C is added last
retract(H)*	- The first clause with a head unifying with H is removed.
retractall(H)	- All clauses having a head that unifies with H are removed
clause(H,B)*	- Unify H with the head and B with the body of a rule.

*More than one answer can be obtained (upon back-tracking).

Example

```
p(a). p(b). p(c).
q(1). q(2). q(3).

?- p(X), q(Y), assert(r(X,Y)), fail.

r(a,1). r(a,2). r(a,3).
r(b,1). r(b,2). r(b,3).
r(c,1). r(c,2). r(c,3).
```

Findall

findall(E,G,L)
- for every reduction of G, add element E to L

```
parent(a,b). parent(e,b). parent(b,c). parent(b,d).

?- findall(P,parent(P,C),L).
   L = [a,e,b,b]
?- findall((GP,GC), (parent(GP,P), parent(P,GC)),L).
   L = [(a,c),(a,d),(e,c),(e,d)]
```

Bagof

bagof(E,G,L)

- For each reduction of G, where non-quantified (i.e., they are not in E or specified with ^) are assumed to refer to the same instance, add E to L.

```
parent(a,b). parent(e,b). parent(b,c). parent(b,d).

?- bagof(P,parent(P,C),L).
   C = b L = [a,e];
   C = c L = [b];
   C = d L = [b]
?- bagof(P,C^parent(P,C),L).
   L = [a,e,b,b]
```

Setof

setof(E,G,L)

- as bagof(E,G,L), but where L is sorted and without duplicates.

```
parent(a,b). parent(e,b). parent(b,c). parent(b,d).

?- setof(P,parent(P,C),L).
   C = b L = [a,e];
   C = c L = [b];
   C = d L = [b]
?- setof(P,C^parent(P,C),L).
   L = [a,b,e]
```