Logic Programming

Georg Moser

Institute of Computer Science @ UIBK

Summer 2015

Summary of Last Lecture

Observation

given a list [1,2,3] it can be represented as the difference of two lists

1. `[1,2,3] = [1,2,3] \ []`
2. `[1,2,3] = [1,2,3,4,5] \ [4,5]`
3. `[1,2,3,8] = [1,2,3,8] \ [8]`
4. `[1,2,3] = [1,2,3|Xs] \ Xs`

Definition

the difference of two lists is denotes as $As \setminus Bs$ and called difference list

Example

```
append_dl(Xs \ Ys, Ys \ Zs, Xs \ Zs).
```


Definite Clause Grammars (DCGs for short)

Example

```
sentence(sentence(NP,VP),Num) →
    noun_phrase(N,Num), verb_phrase(V,Num).
⋮
determiner(det(the),Num) → [the].
determiner(det(a),singular) → [a].
noun(noun(pie-plate),singular) → [pie-plate].
noun(noun(pie-plates),plural) → [pie-plates].
noun(noun(surprise),singular) → [surprise].
noun(noun(surprises),plural) → [surprises].
adjective(adj(decorated)) → [decorated].
verb(verb(contains),singular) → [contains].
verb(verb(contain),plural) → [contain].
sentence(PT) ⇒* ‘‘the decorated pie-plates contain a surprise’’
```


Outline of the Lecture

Logic Programs

introduction, basic constructs, database and recursive programming, theory of logic programs

The Prolog Language

programming in pure prolog, arithmetic, structure inspection, meta-logical predicates, cuts, extra-logical predicates, how to program efficiently

Advanced Prolog Programming Techniques

nondeterministic programming, incomplete data structures, definite clause grammars, meta-programming, constraint logic programming

Example (Facts)

```
father(andreas,boris).        female(doris).        male(andreas).
father(andreas,christian).    female(eva).          male(boris).
father(andreas,doris).                              male(christian).
father(boris,eva).            mother(doris,franz).  male(franz).
father(franz,georg).          mother(eva,georg).    male(georg).
```

Example

```
children(X,Cs) :- children(X,[],Cs).
children(X,A,Cs) :-
    father(X,C), \+ member(C,A), !, children(X,[C|A],Cs).
children(X,Cs,Cs).
```

Example

```
children(X,Kids) :- setof(C,father(X,C),Kids).
```


Second-Order Programming

Definitions

- the predicate *bagof*(*Template*, *Goal*, *Bag*) unifies *Bag* with the alternatives of *Template* that meet *Goal*
- if *Goal* has free variables besides the one sharing with *Template* *bagof* will backtrack
- fails if *Goal* has no solutions
- construct *Var* ↑ *Goal* tells *bagof* to existentially quantify *Var*
- the predicate *setof*(*Template*, *Goal*, *Bag*) is similar to *bagof* but sorts the obtained multi-set (bag) and removed duplicates

Example

```
kids(Kids) :- setof(Y, X ↑ (father(X,Y)),Kids).
```


Simple Application of Set Predicates

Example

```
no_doubles(Xs,Ys) :- setof(X,member(X,Xs),Ys).
```

Definition

- the predicate *findall*(*Template*, *Goal*, *Bag*) works as *bagof* if all excessive variables are existentially quantified
- SWI-Prolog notation for ↑: ^

Meta-Programming and Meta-Interpreters

Definition

- a meta-program treats other programs as data; it analyses, transforms, and simulates other programs
- a meta-interpreter for a language is an interpreter for the language written in the language itself
- relation *solve*(*Goal*) is true, if *Goal* is true with respect to the program interpreted

Example (simple meta-interpreter)

```
solve(true).
solve((A,B)) :- solve(A), solve(B).
solve(A) :- clause(A,B), solve(B).
```

Example (Simple Meta-Program)

```
eval(X,Y) :-
    number(X), X = Y.
eval(X,Y) :-
    nonvar(X), functor(X,F,N),
    built_in(F,N),
    functor(Z,F,N),
    eval_args(N,X,Z),
    Y is Z.
eval(X,Y) :-
    nonvar(X), functor(X,F,N),
    user_def(F,N),
    N1 is N + 1, functor(Z,F,N1),
    eval_args(N,X,Z),
    Z,
    arg(N1,Z,Y).
```


Example (Yet Another Simple Meta-Program)

```
edit :- edit(file([],[])).
edit(File) :- read(Command), edit(File,Command).

edit(_File,exit) :- !.
edit(File,Command) :-
        apply(Command,File,File1),
        !,
        edit(File1).
edit(File,Command) :-
        write(Command),
        write(' is not applicable'),
        !,
        edit(File).

apply(up,file([X|Xs],Ys),file(Xs,[X|Ys])).
apply(up(N),file(Xs,Ys),file(Xs1,Ys1)) :-
    N > 0,
    up(N,Xs,Ys,Xs1,Ys1).
```


Example (And the Last Example)

```
accept(S) :-
    initial(Q),
    accept(Q,S).
accept(Q,[X|Xs]) :-
    delta(Q,X,Q1),
    accept(Q1,Xs).
accept(Q,[]) :-
    final(Q).
initial(q0).
final(q2).
delta(q0,0,q0).
delta(q0,0,q1).
delta(q0,1,q0).
delta(q1,1,q2).
```


Example (meta-interpreter with proofs)

```
solve(true,true).
solve((A,B),(ProofA,ProofB)) :-
    solve(A,ProofA),
    solve(B,ProofB).
solve(A,(A :- Proof)) :-
    clause(A,B),
    solve(B,Proof).
```

Example

```
father(andreas,boris).       female(doris).        male(andreas).
father(andreas,christian).   female(eva).          male(boris).
father(andreas,doris).                             male(christian).
father(boris,eva).           mother(doris,franz).  male(franz).
father(franz,georg).         mother(eva,georg).    male(georg).
son(X,Y) :- father(Y,X), male(X).
```

Example (A Meta-Interpreter with Proofs (cont'd))

```
:- solve(son(christian,andreas),Proof).
```

```
Proof ↦ (son(christian, andreas) <--
    (father(andreas, christian)<--true,
    male(christian)<--true))
```

Example (Tracing Pure Prolog)

```
trace(Goal) :- trace(Goal,0).

trace(true,Depth).
trace((A,B),Depth) :-
    trace(A,Depth), trace(B,Depth).
trace(A,Depth) :-
    clause(A,B),
    display(A,Depth),
    Depth1 is Depth + 1,
    trace(B,Depth1).

display(A,Depth) :- tab(Depth), write(A), nl.
```


Example

```
system(A is B).         system(read(X)).          system(integer(X)).
system(clause(A,B)).    system(A < B).            system(A >= B).
system(write(X)).       system(functor(T,F,N)).   system(system(X)).
```

Example

```
trace(Goal) :- trace(Goal,0).

trace(true,Depth) :- !.
trace((A,B),Depth) :- !, trace(A,Depth), trace(B,Depth).
trace(A,Depth) :- system(A), A, !, display2(A,Depth), nl.
trace(A,Depth) :-
    clause(A,B), display2(A,Depth), nl,
    Depth1 is Depth + 1, trace(B,Depth1).
trace(A,Depth) :-
    \+ clause(A,B), display2(A,Depth),
    tab(8), write(f),nl,fail.

display2(A,Depth) :- Spacing is 3*Depth, tab(Spacing), write(A).
```


Meta-Interpreters for Debugging

Example (Control Execution)

```
solve(true,_D,no_overflow) :-
        !.
solve(_A,0,overflow([])).
solve((A,B),D,Overflow) :-
        D > 0,
        solve(A,D,OverflowA),
        solve_conjunction(OverflowA,B,D,Overflow).
solve(A,D,no_overflow) :-
        D > 0,
        system(A), !, A.
solve(A,D,Overflow) :-
        D > 0,
        clause(A,B),
        D1 is D - 1,
        solve(B,D1,OverflowB),
        return_overflow(OverflowB,A,Overflow).
```


Example (Control Execution (cont'd))

```
solve_conjunction(overflow(S),_B,_D,overflow(S)).
solve_conjunction(no_overflow,B,D,Overflow) :-
solve(B,D,Overflow).

return_overflow(no_overflow,_A,no_overflow).
return_overflow(overflow(S),A,overflow([A|S])).

isort([X|Xs],Ys) :- isort(Xs,Zs), my_insert(X,Zs,Ys).
isort([],[]).

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