Logic Programming

Georg Moser

Institute of Computer Science @ UIBK

Summer 2015

Summary of Last Lecture

Example

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

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 (meta-interpreters for debugging)

```
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).
```


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

Expert Systems in Prolog

Expert Systems

expert systems typically consists of

- knowledge base
- inference engine

this separation is not suitable for a Prolog implementation

Employ Meta-Interpreters

we implement the following features of expert systems using meta-interpreters:

- interaction with the user
- explanation facility
- uncertainty reasoning

Toy Expert System

```
place_in_oven(Dish,top) :-
    pastry(Dish), size(Dish,small).
place_in_oven(Dish,middle) :-
    pastry(Dish), size(Dish,big).
place_in_oven(Dish,middle) :-
    main_meal(Dish).
place_in_oven(Dish,low) :-
    slow_cooker(Dish).

pastry(Dish) :- type(Dish,cake).
pastry(Dish) :- type(Dish,bread).

main_meal(Dish) :- type(Dish,meat).

slow_cooker(Dish) :- type(Dish,milk_pudding).
```


solve1/1

```
solve1(true) :-
        !.
solve1((A,B)) :-
        solve1(A), solve1(B).
solve1(A) :-
        A \= (_A1,_A2),
        clause(A,B), solve1(B).
solve1(A) :-
        askable(A), \+ known(A),
        ask(A,Answer),
        respond(Answer,A).

ask(A,Answer) :- display_query(A),read(Answer).

askable(type(_Dish,_Type)).
askable(size(_Dish,_Size)).

respond(yes,A) :- assert(A).
respond(no,A) :- assert(untrue(A)),fail.
```


Interaction (in the Naive)

```
interact(Goal) :-
         reset, solve1(Goal).

reset :- retractall(type(_Dish,_Type)),
         retractall(size(_Sish,_Size)),
         retractall(untrue(_Fact)).

?- interact(place_in_oven(dish,X)).
type(dish,cake)? yes.
size(dish,small)? no.
type(dish,bread)? no.
size(dish,big)? yes.
X = middle
```

Question

what about explanations for questions?

solve2/1

```
solve2(Goal) :- solve2(Goal,[]).

solve2(true,_Rules) :-
        !.
solve2((A,B),Rules) :-
        solve2(A, Rules), solve2(B, Rules).
solve2(A, Rules) :-
       A \= (_A1, _A2),
        clause(A,B),
        solve2(B,[rule(A,B)|Rules]).
solve2(A, Rules) :-
        askable(A), \+ known(A),
        ask(A, Answer), respond(Answer,A, Rules).

respond(why,A,[Rule|Rules]) :-
        display_rule(Rule),
        ask(A, Answer),
        respond(Answer,A, Rules).
```


Interaction with Explanations

```
interact_why(Goal) :- reset, solve2(Goal).
```

```
?- interact_why(place_in_oven(dish,X)).
type(dish,cake)? yes.
size(dish,small)? no.
type(dish,bread)? no.
size(dish,big)? why.
if pastry(dish) and size(dish,big)
then place_in_oven(dish,middle)
size(dish,big)? yes.
X = middle
```

Question

how to obtain general explanations

interpret/1

```
interpret((Proof1,Proof2)) :-
        interpret(Proof1), interpret(Proof2).
interpret(Proof) :-
        fact(Proof,Fact),
        nl, write(Fact),
        writeln(' is a fact in the database').
interpret(Proof) :-
        rule(Proof,Head,Body,Proof1),
        nl, write(Head),
        writeln(' is proved using the rule'),
        display_rule(rule(Head,Body)),
        interpret(Proof1).

extract_body((Proof1,Proof2),(Body1,Body2)) :-
        !, extract_body(Proof1,Body1),
        extract_body(Proof2,Body2).
extract_body((Goal <-- _Proof),Goal).
```


how/1

```
how(Goal) :- solve(Goal,Proof), interpret(Proof).

?- interact(place_in_oven(dish,X)).
% required for type and size of dish

?- how(place_in_oven(dish,top)).

place_in_oven(dish,top) is proved using the rule
if pastry(dish) and size(dish,small)
then place_in_oven(dish,top)

pastry(dish) is proved using the rule
if type(dish,bread)
then pastry(dish)

type(dish,bread) is a fact in the database

size(dish,small) is a fact in the database
```

Shortcomings with Explanation

- the explanation is exhaustive
  not intelligible for a knowledge base with 100 rules
- restrict explanation to one level:

```
pastry(dish) can be further explained
```

- Prolog computation is mirrored
- take expert knowledge into account:

```
interpret((Goal <-- Proof)) :-
    classification(Goal),
    write(Goal),
    writeln(' is a classification example').
```

- in general make use of filtered explanations

Remark

a more advanced example of expert system is in Chapter 21 in the book

Definition

the certainty of a goal is computed as follows

$$\mathrm{cert}(G) = \begin{cases} \min\{\mathrm{cert}(A), \mathrm{cert}(B)\} & G = (A, B) \\ \max\{\mathrm{cert}(B) \cdot \mathit{Factor} \mid \text{exists } \langle A : -B, \mathit{Factor}\rangle\} & G = A \end{cases}$$

Definition (clauses with certification factor)

```
clause_cf(place_in_oven(Dish,top),
                    (pastry(Dish), size(Dish,small)),0.7).
clause_cf(place_in_oven(Dish,middle),
                    (pastry(Dish), size(Dish,big)),1).
clause_cf(place_in_oven(Dish,middle),
                    main_meal(Dish),1).
clause_cf(place_in_oven(Dish,low),
                    slow_cooker(Dish),0.5).
% otherwise
clause_cf(Head,Body,1) :- clause(Head,Body).
```


solve3/1

```
solve3(true,1) :-
        !
solve3((A,B),C) :-
        !,
        solve3(A,C1),
        solve3(B,C2),
        minimum(C1,C2,C).
solve3(A,C) :-
        clause_cf(A,B,C1),
        solve3(B,C2),
        C is C1 * C2.

?- interact(place_in_oven(dish,X)).
% required for type and size of dish

?- solve3(place_in_oven(dish,top),C).
C = 0.7
```