Logic Programming

Manipulating Programs

Temur Kutsia

Research Institute for Symbolic Computation
Johannes Kepler University of Linz, Austria
kutsia@risc.uni-linz.ac.at

Contents

Introduction

- Programs as data.
- Manipulating Prolog programs with other Prolog programs.
- Meta-Programming

`clause` Predicate

`clause(X,Y)`

- Built-in binary predicate, very important if one wishes to construct programs that examine or execute other programs.
- Satisfying `clause(X,Y)` causes `X` and `Y` to be matched with the head and body of an existing clause in the database.
- `X` must be instantiated so that the main predicate of the clause is known.

`clause` Predicate

Satisfying `clause(X,Y)`

- If there are no clauses that match `X`, the goal fails.
- If there is more than one clause that matches, Prolog returns the first one. The other matches will be chosen, one at a time, when Prolog backtracks.

`clause` Predicate. Examples

```
append([],X,X).
append([A|B],C,[A|D]):-append(B,C,D).

?- clause(append(L1,L2,L3),Y).

L1 = []
L2 = L3
Y = true ;

L1 = [_G463|_G464]
L3 = [_G463|_G467]
Y = append(_G464, L2, _G467) ;

No
```

A Version of `listing` Predicate

`list1(X)`

- Satisfying the goal `list1(X)` will print out the clauses in the database whose head matches `X`.
- The definition of `list1(X)` will involve `clause` with `X` as the first argument.
- Therefore, `X` has to be sufficiently instantiated.

Definition of `list1`

```
list1(X):-
    clause(X,Y),
    output_clause(X,Y), write('.'),nl,fail.
list1(X).

output_clause(X,true):-!,write(X).
output_clause(X,Y):-write((X:-Y)).
```

How Does `list1` Work?

- The first clause causes a search for a clause whose head matches `X`.
- If one found, it is printed and a failure is generated.
- Backtracking will reach the `clause` goal and find another clause, if there is one, and so on.
- When there is no more clause to be found, the `clause` goal will fail.
- At this point, the second clause for `list1` will be chosen, so the goal will succeed.
- As a “side effect", all the appropriate clauses will have been printed out.

How Does `output_clause` Work?

- Specifies how the clauses will be printed.
- It looks for a special case of the body `true`. In this case it just prints the head.
- Otherwise, it writes out the head and the body, constructed with the functor `:-`.
- The “cut” in the first rule for `output_clause` says that the first rule is the only valid possibility if the body is `true`.
- The “cut” is essential because the example relies on backtracking.

Writing Prolog Interpreter in Prolog

Idea:

- Define what it is to run a Prolog program by something which is itself a Prolog program.

The `interpret` Predicate

Idea:

- `interpret(X)` succeeds as a goal exactly when `X` succeeds as a goal.
- `interpret` is similar to built-in predicate `call`, but is more restricted: It does not deal with cuts or built-in predicates

The `interpret` Predicate

```
interpret(true):-!.
interpret((G1,G2)):-!,
    interpret(G1),
    interpret(G2).
interpret(Goal):-
    clause(Goal,MoreGoals),
    interpret(MoreGoals).
```

The `interpret` Predicate

- The first clause of `interpret` deals with the special case when the goal is true.
- The second clause deals with the case when a goal is a conjunction.
- The third clause covers a simple goal: The procedure is the following:
  1. Find a clause whose head matches the goal
  2. `interpret` the goals in the body of that clause.
- Limitations: The program will not cope with programs using built-in predicates, because such predicates do not have clauses in the usual sense.

The `consult` Predicate

- `consult` is provided as a built-in predicate in most systems.
- Interesting to see how it can be defined in Prolog.
- A simplified definition.

Program for `consult`

```
consult(File):-
    seeing(Input),
    see(File),
    repeat,
    read(Term),
    process(Term),
    seen,
    see(Input),
    !.
```

Program for `consult`, Cont.

```
process(Term):-end_of_file_mark(Term),!.
process(?- Q):-!,call(Q),!,fail.
process(Clause):-assertz(Clause),fail.
```

Program for `consult`. Explanations.

- `seeing(X)` succeeds iff the name of the current stream matches `X`.
- `see(X)` opens the file `X`, if it is not already open, and defines the current input stream to originate from `X`. Error if `X` is not instantiated, or if `X` names a file that does not exist.
- `seen` closes the current input stream, and defines the current input stream to be the keyboard.

Program for `consult`. Explanations.

- `repeat` provides an extra way to generate multiple solutions through backtracking.
- Can be defined as:

```
repeat.
repeat:-repeat.
```

- `read` reads the next term from the current input stream and matches it with `X`. The term must be followed by a dot ".", which does not become a part of the term, and at least one non-printing character (e.g. a character that causes a new line to appear)

Program for `consult`. Explanations.

- `end_of_file_mark` should be defined by the user.
- It is supposed to succeed when its argument is instantiated to the term used to represent the end of file
- This term is implementation dependent.

Program for `consult`. Explanations.

- `process` should cause an appropriate action to be taken for each term from the input file.
- A `process` goal only succeeds when when its argument is the end of file mark.
- Otherwise, a failure occurs after the appropriate action, and backtracking goes back to the `repeat` goal.
- If a term read for the file represents a question (second clause for `process`), an attempt is made to satisfy the appropriate goal immediately using the `call` predicate.