ECE251: Programming Languages & Translators Fall 2010

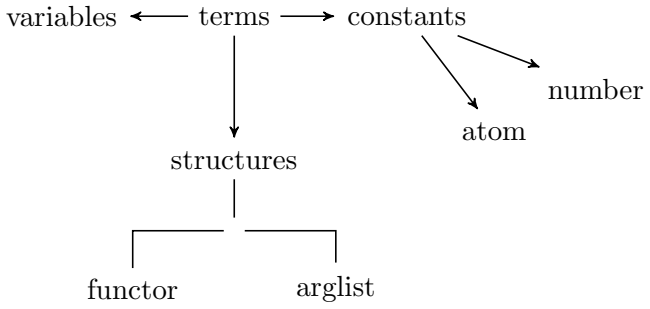
Lecture 28 — November 24, 2010

Patrick Lam *version 2*

Prolog, a Logic Programming Language

Prolog is one example of a logic programming language. Let's see how the concepts I introduced last time show up in Prolog. By the way, I've installed `swi-prolog` on the `ece251` server, and you can try out the code examples below on that server.

Prolog operates on a database of clauses. Each Prolog clause contains a set of terms. I describe the constituents of terms below.



A *predicate* is a functor name, plus the number of arguments (arity) that you're supposed to give to that functor (e.g. `rainy/1` or `partner/2`).

Each clause can be either a fact or a rule. Facts have no right-hand sides; the left-hand side is unconditionally true.

```
sun_sets_too_early(December).
```

Rules have both left-hand sides and right-hand sides; in keeping with the interpretation of Horn clauses, the LHS is true when the RHS is true.

```
stupid(X) :- not_enough_sleep(X).
```

The `X` is universally quantified: the rule is saying that all `X` who do not get enough sleep are stupid[1]

Defining clauses. Note that the database of clauses has to be loaded from disk, using the `consult` command. You can then carry out queries on the database.

[1] Note that this clause does not state causality. Hint: the causality is that not getting enough sleep will make you stupid.

Using clauses

Recall that the programmer defines a set of clauses, and then the user executes queries on this database. After you've told Prolog to consult your database, you can thus enter queries.

Let's say that your database contains the clauses:

```
rainy(vancouver).
rainy(victoria).
```

You can then consult the database and query the interpreter:

```
?- consult('rainy.prolog').
% rainy.prolog compiled 0.00 sec, 556 bytes
true.
?- rainy(C).
C = vancouver
```

The interpreter finds the first fact that satisfies the query. It then waits for you to type something. You can type a semicolon, at which point it tells you the next fact that satisfies the query:

```
?- rainy(C).
C = vancouver ;
C = victoria.
```

Or you could have typed a period, at which point the interpreter stops the query.

Unification

We've only seen very straightforward queries so far. Prolog's a bit smarter than that. In particular, it can substitute the head of a clause C_1 into the body of a clause C_2, as in the following example:

```
graduated_from(mit, derek).
graduated_from(mit, patrick).
graduated_from(mcgill, patrick).
graduated_from(waterloo, derek).
same_school(X, Y) :- graduated_from(Z, X), graduated_from(Z, Y).
```

Given the query `same_school(patrick, derek)`, Prolog can instantiate `X` with `patrick` and `Y` with `derek`, thus finding that there *exists* some `Z`, namely `mit`, which it can unify the relevant `graduated_from` structure into the body of the `same_school` clause.

Here are the rules for unification:

- Constant `c` unifies only with itself.

- Structures `s(a1)` unifies with `t(a2)` iff `s` and `t` have the same arity, and the corresponding arguments in `a1` and `a2` all unify.
- A variable `X` unifies with anything. Unifying `X` with an uninstantiated variable `Y` means that if either `X` or `Y` gets a value later, then the other variable does as well. Unifying `X` with anything else gives `X` the value of that anything else.

In Prolog, two terms are equal if they unify. The textbook contains this extended example:

```
?- a=a.
true.
?- a = b.
false.
?- foo(a,b) = foo(a,b).
true.
?- X = a
X = a.
?- foo(a, b) = foo(X, b).
A = a.
?- A = B.
A = B.
```

Note that the last query unifies two variables without instantiating them. Prolog will, however, instantiate variables if it has to.

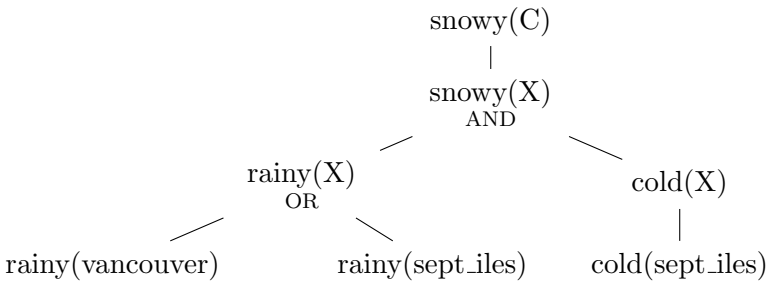
Search/Execution Order

Let's see how Prolog carries out queries. Basically, it starts with the goal and works backwards, attempting to unify clauses until it proves the goal or can't find a way to reach it. Since it doesn't know about which clauses it should unify, it might have to *backtrack*.

The best way to proceed is with an example. Consider this Prolog program from the textbook:

```
rainy(vancouver).
rainy(sept_iles).
cold(sept_iles).
snowy(X) :- rainy(X), cold(X).
```

The query `snowy(C).` gives the following search tree:



The search begins at `snowy(C)` and finds the rule `snowy(X)`, which then becomes the goal, with `C` unified to `X`. Prolog must then establish both `rainy(X)` and `cold(X)`. It establishes `rainy(X)` by finding `rainy(vancouver)`, thus (temporarily) unifying `X` with `vancouver`. But then it must also find `cold(vancouver)`. Because this fails, Prolog backtracks and so it unifies `rainy` with `sept_iles`. Going back up the tree, it finds `cold(sept_iles)` and returns `snowy(sept_iles)`.