Chapter 11: Logic Languages

November 8, 2017

Declarative languages go for logic

- So far, based on (mostly) predicate calculus rather than lambda calculus

Declarative languages go for logic

- And vice versa: concepts common with the Prolog community influenced the type inferencing in languages like ML and Haskell.

```
flowery(X) <- rainy(x)

rainy(Portland)
------------
flowery(Portland)
```

Prolog

- While the logic programming languages are not successful in the real world (even Curry seems to have withered on the vine; the official mailing list doesn't have much recent traffic), they do have some mildly interesting features.

Prolog

- For instance, it is possible to solve the recently popular "Cheryl's Birthday" problem using Prolog: One interesting solution

Prolog

- Uses “Horn” clauses

Prolog

- Dense terminology, based on its logical roots. Terms can be of any of these types
  - Atom: identifier beginning with lowercase letter, or quoted string
  - Number: usual base 10 representations for integer and floating point numbers
  - Variable: identifier beginning with an upper case letter

(terminology continued)

- (terminology continued)
  - Structure (aka a "compound term"): an atom (called a "functor" in this case) and a list of arguments (which are themselves terms); the number of arguments is called "arity"

Prolog

- A “predicate” is a functor and its arity.

Prolog

- The Horn clauses are either “facts” or “rules”. “Facts” have no explicit righthand side (implicitly, they are “fact :- true.”), and “rules” do (e.g., “rule :- something.”)
- So how does this all work? You have to give a goal to be reached; the Prolog engine tries to use the declarations that you have made to deduce that goal (actually, it does the opposite: it tries to prove the negative of the goal false).

Prolog

- Supports lists.
- Supports “is” arithmetic (but try just “X.” with SWI-Prolog).

Prolog

```
?- [user].
rainy(seattle).
|: rainy(rochester).
|: cold(rochester).
|: snowy(X) :- rainy(X), cold(X).
|: % user://1 compiled 0.01 sec, 5 clauses
true.

?- snowy(X).
X = rochester.
```

Prolog (previous example continued)

```
?- [user].
cold(seattle).
Warning: user://2:38:
     Redefined static procedure cold/1
     Previously defined at user://1:26
|: cold(rochester).
|: % user://2 compiled 0.00 sec, 2 clauses
true.

?- snowy(X).
X = seattle ;
X = rochester.

?-
```

Imperative Prolog

- Supports “cuts”, allowing you to commit to a part of the search tree
- Cut allows not only efficiency gains by stopping re-reconsideration ad infinitum, but it allows us to create selection:

```
statement :- condition, !, then.
statement :- else.
```

Imperative Prolog

- How about a loop?

```
?- [user].
natural(1).
|: natural(N) :- natural(M), N is M+1.
|: looping(N) :- natural(I), write(I), nl, I = N, !.
|: % user://3 compiled 0.01 sec, 4 clauses
true.

?- looping(5).
1
2
3
4
5
true.

?-
```

Where logic languages are limited

- Prology is limited to Horn clauses, not all of first-order predicate calculus
- Execution order exists, and it's not clear on how to optimize this without programmer intuition
- Negation and the "closed world" assumption

The future?

- So far, nobody has done a lot with these. Maybe we should try to build these languages around satisfiability formulations?
- Lots of open questions, so lots of room for research!