Computing Science (CMPUT) 325

Nonprocedural Programming

Martin Müller

Department of Computing Science
University of Alberta
mmueller@ualberta.ca

Winter 2016

Introduction to Prolog

- Prolog syntax is based on predicate calculus
- Examples of predicates:
  - `likes(mary, john)`
  - `greater(3, 2)`
  - `isPrime(19)`

Atoms in Predicate Calculus

- Atom, or atomic formula:
- `p(t1, ..., tn)`
- `p` is a **predicate symbol** (e.g. `likes, isPrime`)
- `ti` are called **terms** (a bit similar to symbolic expressions in Lisp)
- Terms can **only** be used as arguments in predicates

Terms

- A constant is a term (in Prolog: starts with lowercase letter)
  - Numbers are represented in Prolog as usual
- A variable is a term (in Prolog: starts with uppercase letter)
- Functions are terms, defined recursively:
- if `s1, ..., sk` are terms, and `f` is an k-ary function symbol, then `f(s1,...,sk)` is a term

More Details

- Functions and predicates in Prolog also start with lowercase letter
- Only variables start with uppercase
- Examples of predicates with variables:
  - `append(X, Y, Z)`
  - `p(X, f(Y), g(Z))`
- Note that in Prolog, the role of functions is **very** different
- Function symbols are mostly used to structure data, not used for computation
- We compute by doing inference with predicates

Binding Variables

- As in Lisp, variables can be *bound* to values
- The mechanisms for doing the binding are different
- The simplest one uses `=` , as in `X = 2`
- There are other ways to bind variables using unification, see later

Prolog Program

- A Prolog program is a collection of **clauses**
- A clause is either a **fact** or a **rule**
- Example of a fact:
  - `father(ken, mary).`
- Example of a rule:
  - `parent(X,Y) :- father(X,Y).`
- A Prolog program usually contains both rules and facts.
- Both rules and facts always end with a dot.
- If you forget it, Prolog will sit and wait for it (or give a syntax error).

More About Facts

- A fact is also called an unconditional clause.
- A fact like `father(ken, mary).` means that this predicate is unconditionally true within the current program.
- Facts and rules can also contain variables. Those are always **universally quantified**.
- Example: `awesome(X).` in Prolog means:
  - In logic: $\forall$ X awesome(X)
  - In words: “everything is awesome”

More About Rules

- `parent(X,Y) :- father(X,Y).`
- The `:-` symbol can be read as “if”
- Rules are **conditional** clauses
- The **head** of the clause is `parent(X,Y)`
- It is true under the condition that the **body** `father(X,Y)` is true
- It may be true in other cases as well, if there are more clauses with `parent` as head

Rules with Multiple Predicates in the Body; Multiple Rules

- The body can have a list of atoms
- All of them must be true to imply that the head is true
- `banana(X) :- yellow(X), fruit(X), bendy(X).`
- This rule means that all yellow bendy fruits are bananas.
- It does **NOT** mean that all bananas must be yellow bendy fruits.
- We can add more rules such as
- `banana(X) :- green(X), vegetable(X), notBent(X).`

Rules and Logical Consequence

- `parent(X,Y) :- father(X,Y).`
- X is a parent of Y **if** X is a father of Y
- parent(X,Y) is true whenever father(X,Y) is true
- father(X,Y) implies parent(X,Y)
- In logic: $\forall$ X $\forall$ Y father(X,Y) $\rightarrow$ parent(X,Y)
- In words: for all X and all Y: if X is a father of Y, then X is a parent of Y

General Form of Prolog Rules

- `A :- B1, B2, ..., Bn.`
- `A` and all `Bi` are atoms
- `A` is the head of the clause
- The list of `Bi`'s is the body
- In logic: $(B1 \wedge B2 \wedge \ldots \wedge Bn) \rightarrow A$
- In words: whenever all `Bi` are true, `A` is also true
- As before, universally quantified over all variables in the clause (see next slide)

Universal Quantifiers for Prolog Rules

- Given a rule `A :- B1, B2, ..., Bn.`
- Assume `X1, ..., Xm` are all the variables appearing anywhere within the rule
- In logic, this is same as having universal quantifiers for all the `Xi`:
- $\forall\ X1\ \forall\ X2\ \ldots\ \forall\ Xm$
  $(B1 \wedge B2 \wedge \ldots \wedge Bn) \rightarrow A$

Facts as Special Cases of Rules

- Fact `A.`
- Rule `A :- B1, B2, ..., Bn.`
- We can view a fact as a special case of a rule with zero atoms in the body
- We could also write the fact as `A :- true.` but it just adds clutter.
- “If true then A” is just a cluttered way of saying “A”.

Example Prolog Program - Family Tree


```
grandparent(X,Z) :- parent(X,Y), parent(Y,Z).
parent(X,Y) :- father(X,Y).
parent(X,Y) :- mother(X,Y).

father(ken, mary).
mother(lily, mary).
mother(mary, john).
```

- Some facts about father and mother relations
- General rules about parents and grandparents
- First we load this program called `family.pl` into Prolog with `[family]`, or `consult(family)`.

Example Prolog Program - Family Tree continued

- After loading, we can ask Prolog about which family relations follow from the facts and rules above
- `?- grandparent(ken,john).`
- Prolog will say **yes**
- `?- grandparent(mary,ken).`
- Prolog will say **no**
- We can also consult user, as in `[user]`, to read Prolog code from the terminal. Stop with Ctrl-D to get back to the usual prompt.

General Form of Prolog Queries

- A query is also called a goal
- General form `?- C1, C2, ..., Ck.`
- `?-` is the Prolog prompt, waiting for your queries
- `Ci` are atoms, called subgoals
- Don't. forget. the. dot. at. the. end. Or. nothing. will. happen.
- Prolog will try to prove all of `C1, C2, ..., Ck`, using the facts and rules in your program
- Does C1 and C2 and ... and Ck follow logically from your program?
- Equivalently: if all the statements in your program are true, then is the conjunction C1 and C2 and ... and Ck also true?

Prolog Execution

- Rough outline of algorithm:
- to solve a goal `?- C1, C2, ..., Ck.`
- Try to solve each subgoal `Ci` from left to right
- To solve a subgoal, find a clause in the program whose head can be "matched" with the subgoal
- Replace the subgoal by the body of the clause
- Apply the variable bindings, if any
- This step is a little bit like function application in Lisp
- If all subgoals are eventually solved, then the original goal is solved.

Sample of Execution

- Query: `grandparent(ken,john).`
- Try to match with head of clause
  `grandparent(X,Z) :- parent(X,Y), parent(Y,Z).`
- The match works if we bind the variables as follows:
- `X = ken` and `Z = john`
- Now we replace he original query with the body, after substituting the variables:
- New query: `parent(ken,Y), parent(Y,john).`
- Note that Y is still an unbound variable.

Sample of Execution - Continued

- New query: `parent(ken,Y), parent(Y,john).`
- Important: in queries, free variables are **existentially quantified**
- We do **not** need to show that the query is true **for all Y**
- We only need to find at least one solution:
  There exists some Y such that `parent(ken,Y)` and `parent(Y,john)` are both true (for the same choice of Y)

Sample of Execution - Continued

- New query: `parent(ken,Y), parent(Y,john).`
- To show `parent(ken,Y),`
  we can use `parent(X,Y) :- father(X,Y).`
- This gives subgoal `father(ken,Y).`
- The overall goal at this step is
  `father(ken,Y), parent(Y,john).`
- `father(ken,Y)` matches the fact
  `father(ken, mary).`
- So `father(ken,Y)` can be solved
  by setting `Y = mary`.

Sample of Execution - Continued

- Query: `parent(ken,Y), parent(Y,john).`
- `father(ken,Y)` solved by setting `Y = mary`.
- Remaining goal now: `parent(mary,john).`
- To solve `parent(mary,john).` we can use `parent(X,Y) :- mother(X,Y).`
- So, must solve `mother(mary,john).`
- Easy! This is a fact in our program. Finally, all subgoals solved!
- The original query `?- grandparent(ken,john).` is answered with `yes.` Prolog found a proof for it!
- How exactly did it find this proof? More next time.

Summary of Prolog Intro

- Discussed basics of Prolog syntax
  and its foundation in logic
- Sample program for family tree
- Sample execution steps - how queries can be resolved by matching with head of rule, then replacing it by body
- The matching may involve binding variables
- How to select the rule to match,
  and what if the matching fails? More next time.