LP Foundations, Prolog

In Text: Chapter 15

1

Logic Programming -- Basic Principles

- LP languages are **declarative**
  - Declarative => uses "declarations" instead of assignment statements + control flow
  - Declarative semantics: there is a simple way to determine the meaning of each statement; doesn't depend on how the statement might be used to solve a problem
  - much simpler than imperative semantics
- Logic programming languages are nonprocedural
  - Instead of specifying how a result is to be computed, we describe the desired result and let the computer system figure out how to compute it

Logic Programming Example

- To see declarative vs. procedural differences, consider this logic pseudocode for sorting a list:

```
sort(old_list, new_list) ⇐
    permute(old_list, new_list) and sorted(new_list)
sorted(list) ⇐
    ∀j such that 1 ≤ j < n: list(j) ≤ list(j+1)
```

- Prolog is an example of a logic programming language.

Prolog Name Value System

- Prolog is **case sensitive**
- Object names (atoms) starting with a lower case letter
- Literals include integers, reals, strings
- "Variable" identifiers start with an upper case letter
- Predicate names (functions) start with lower case letters (like objects, but distinguishable by context):

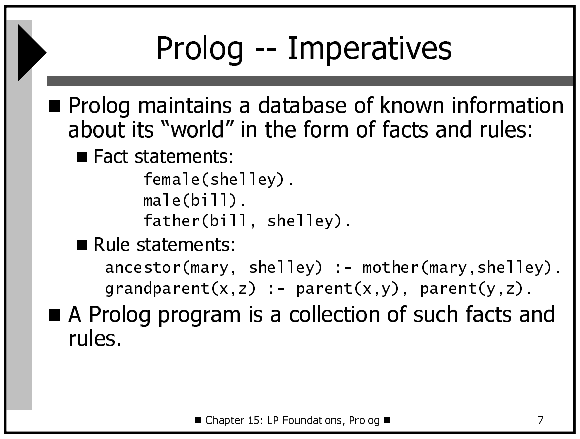
  <name> (<list of arguments>)

Prolog Name Value System (cont.)

- "Latent" typing, as in Scheme
- Types — atoms, integers, strings, reals
- Structures — lists, similar to LISP (see later)
- Scope
  - Atoms and predicate names are all **global**
  - Predicate parameters and "variables" are local to rule in which they are used
  - No global variables or state
- State of the program does **not** include value memory
- "Variables" in Prolog don't change value once they are bound (like mathematical variables)

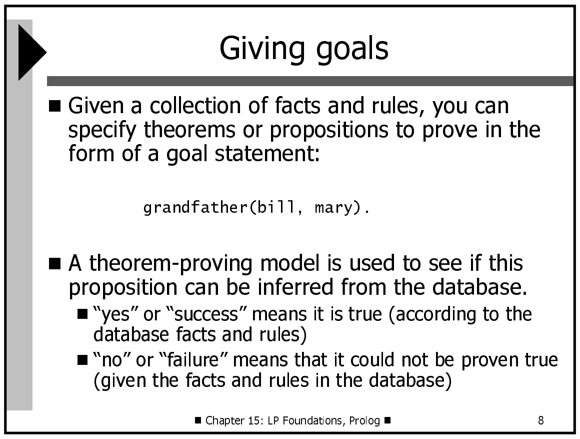
Prolog Statements

- Three kinds:
  - Fact statements
  - Rule statements
  - Goal statements
- Typically, facts + rules define a program
- Goal statements cause execution to begin
  - You give a goal to run your program



Prolog -- Imperatives

- Prolog maintains a database of known information about its "world" in the form of facts and rules:
  - Fact statements:
```
      female(shelley).
      male(bill).
      father(bill, shelley).
```
  - Rule statements:
```
  ancestor(mary, shelley) :- mother(mary,shelley).
  grandparent(x,z) :- parent(x,y), parent(y,z).
```
- A Prolog program is a collection of such facts and rules.



Giving goals

- Given a collection of facts and rules, you can specify theorems or propositions to prove in the form of a goal statement:
```
      grandfather(bill, mary).
```
- A theorem-proving model is used to see if this proposition can be inferred from the database.
  - "yes" or "success" means it is true (according to the database facts and rules)
  - "no" or "failure" means that it could not be proven true (given the facts and rules in the database)