Programming in Logic: Prolog

Prolog Execution &

Data Object Matching

Readings: Read Sections 2.1 & 2.2

of Bratko

Review

- Prolog knowledge base = relation collection.
- Relation identified by name/arity.
- Relation defined by clauses whose heads agree with that id (i.e., name & number of arguments)

Review cont'd

- Clauses have following forms:
  - **head :- body .**
  - **head .**
  - **:- body .**
- Queries are entered by the user (i.e., not in knowledge base) and have form of clause body.

Execution Examples

- Knowledge Base
  - *female(mary).*
  - *siblingOf(mary, peter).*
  - *sisterOf(S,P) :- siblingOf(S,P), female(S).*
- Queries:
  - *female(X).*
  - **X = mary ?**

Execution Examples cont’d

- Query: *sisterOf(X, peter)*.
- *sisterOf(X, peter)* matches *sisterOf(S,P)*
  with *X* binding with *S* and *peter* binding with *P*
- Now its body with bindings becomes the query:
  siblingOf(X,peter), female(X)
- *siblingOf(X,peter)* matches *siblingOf(mary, peter)*
  with *X* binding with *mary*
- Now *female(mary)* becomes query, and done.
- Returns success and **X = mary**

Prolog Program Execution

- Given query, Q, & knowledge base, B, Prolog:
  - For each top-level term, T, in Q Prolog:
    - Tries to match T against the head of a clause in KB.
    - If it fails to find one it returns failure.
    - If it finds one then the body of the clause becomes the current query and this process recurses.
    - If that process succeeds then Prolog returns success along with any bindings used to succeed.
    - If it fails then Prolog tries this loop again (i.e., tries to match T against the head of a different clause in KB).

Prolog Program Execution cont’d

- This process bottoms out either when a term matches a fact or when a term matches certain system relations that are guaranteed to succeed (e.g., *write/1*).
- Arguments to a relation are never “evaluated”, they are simply patterns. E.g., in *b :- a, c(a).* The first *a* is a relation, the second *a* is a pattern.

Types of Data Objects

- Simple Data Objects
  - Atoms
  - Numbers
  - Variables
- Structured Data Objects

Simple Data Objects: Atoms

- Atoms can be formed from:
  - Letters, digits, and underscore - must begin with lower case letter (e.g., *aB3_5C*)
  - Some sequences of special characters (e.g., <--->), some are already defined (e.g., ":-", "+", ...).
  - Strings of characters enclosed in single quotes (e.g., *'TomJones'*)

Simple Data Objects: Numbers

- Integers (e.g., 3, -15)
- Reals (e.g., -0.0035)

Simple Data Objects: Variables

- Syntax: Strings of letters, digits, underscores
  - must begin with either upper case letter or an underscore (e.g., *X, _I,* _).
- Variables:
  - can have a value (i.e., *bound*) or be *unbound*.
  - are not declared - created by use.
  - are not “typed”.

More About Prolog Variables

- Prolog variables start out unbound and get bound via matching, e.g., *X =point(3,4)*.
- Once a variable is instantiated, it can never become unbound nor can its value ever change.
- Assume *X* is currently unbound, consider the code :   "*X = 3, X = 5*",   what happens?

Anonymous Variables

- Variables used only once in a clause do not need a name. Unnamed variables are called anonymous variables.
- Anonymous variable written as underscore (**_**)
- Values are not reported for anonymous variables appearing in queries.
- *?- motherOf(Mother, _).*

  Mother = mary ?

Scope of Variables

- The lexical scope of non-anonymous variables is one clause.
  - *fatherOf(F,C) :- parentOf(F,C), male(F).*
- Each anonymous variable occurrence represents a new variable:
  - *parentOf(_, _)* matches *parentOf(ann, mary)* but *parentOf(X,X)* would not match.

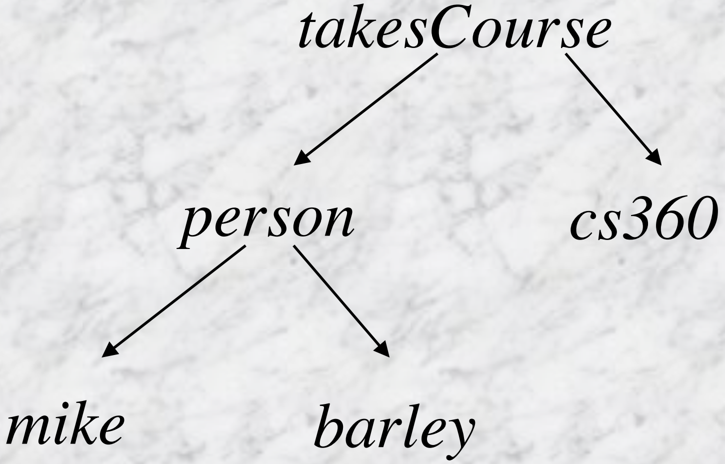
Structured Data Objects

- Structured data objects have a functor name and arguments:
  - *loves(john, mary)*
  - *single(john)*
- The functor name must be an atom.
- Arguments can be structures

  takesCourse(person(mike, barley), cs360)

Structures as Trees

takesCourse(person(mike, barley), cs360)



takesCourse

person

cs360

mike

barley

Matching

- The most important operation on data objects is *matching*.
- Matching is invoked two different ways.
  - Explicitly via the “=“ operator, e.g., *X = 5*
  - Implicitly when Prolog tries to match a goal against its knowledge base.

Implicit Matching Against Heads of Clauses in the Knowledge Base

- What happens when we try to match a goal against the head of a clause in the knowledge base?
- For example, matching the query *meeting(X, john)* against the clause head *meeting(Y, X)*, are the *X*'s in the same scope?
- No, in effect, the variables in the clause are made unique with respect to the goal. E.g., *meeting(Y', X')*

Matching in General

- Given two non-variables, if they aren’t the same type of non-variable then they won’t match.
  - *art* doesn’t match *5*
  - *art* doesn’t match *art(101)*
  - *5* doesn’t match *5.0*

Matching Atoms

- An atom only matches an atom if it is the same atom, e.g., *henry* matches *henry*, but *henry* doesn't match *hank*.
- What about *henry* and '*henry*'?
- It probably depends upon the version of Prolog being used. For SICStus Prolog, they match.
- Remember that '*Henry*' is an atom, while *Henry* is a variable!

Matching Numbers

- Numbers only match if they are the same number and type of number:
  - *56* matches *56*, but not *56.0*.

Matching Structures

- Structures only match if they have the same functor name, the same arity, and each of the arguments match.
- The expression *5 + 3* is really the structure +*(5, 3)*.
- If we asked Prolog the query *8 = 5 + 3* what do you think the answer would be?

Matching Bound Variable & Anything

- As far as matching is concerned, a bound variable is just its value:
  - so if it is bound to an atom, then matching proceeds according to the rules for matching atoms, etc.

Matching Unbound Variables with non-Structures

- Matching an unbound variable to an atom or a number causes that variable to become bound to that value.
- Matching an unbound variable with an unbound variable causes them to share their eventual binding.
  - Assume *X* and *Y* are unbound, then
  - *X = Y, X = 5* causes both *X* and *Y* to be bound to *5*

Matching Unbound Variables with Structures

- If the unbound variable does not appear inside the structure then the variable is bound to the structure.
  - *X = name(mike, Y)*
- If the unbound variable appears inside the structure then what happens depends upon which Prolog you're using. An example:
  - *X = name(mike, X)*

The Occurs Check

- Some Prologs implement the *occurs check*. In these Prologs if the unbound variable being matched occurs within the structure then the match fails and no instantiation occurs.
- Other Prologs don’t check! In these Prologs if the unbound variable being matched occurs within the structure then the match succeeds and the variable is instantiated to that structure.

Effect of No "Occurs Check"

- In these Prologs, what is the result of matching *X = name(mike, X)* ?
- *X* is instantiated to an infinite data structure:

  name(mike, name(mike, name(mike,...)))
- What happens if Prolog tries to print out the value of such a structure?
- Can such an infinite data structure ever be useful? Why/how?

Quick Quiz

- Assume all variables are initially unbound, what are the results of matching:
  - *a = ‘a’*
  - *a = 5*
  - *5 = 5.0*
  - *X = art(Y, bart(sam(Z, X)))*
  - *a(5, b(X, c(Y))) = a(X, b(Z,c(Z)))*
  - *a(Z, X) = a(X, b(Z))*

a(5, b(X, c(Y)))= a(X, b(Z,c(Z)))

- Name and arity agree, match arguments *X=5*
  X is unbound, so it matches *5* (& is bound to it)
  - Now must try to match *b(5, c(Y))* to *b(Z,c(Z))*
  - Name and arity match, so try matching arguments
    - *5 = Z*: *Z* is unbound, so they match and *Z* gets bound to *5*
    - *c(Y) = c(5)*: *Y* is unbound, so they match and *Y* gets bound to *5*
  - As a result of the matching, the structures have been instantiated to *a(5, b(5, c(5)))*.

Quick Quiz cont'd

- What the difference between a structured data object and a relation (remember they both have a name and an arity)?
- Given *a(X) :- b(5, c), c(b(5, c)).* What type of term is the first occurrence of *b(5, c)* ? What type of term is the second occurrence of *b(5, c)*?

Summary

- Prolog computes answer to query by matching query terms against heads of clauses in KB, when match occurs computation recurses on body of matched clause (with current bindings).
- If Prolog succeeds then successful bindings of query variables are returned as part of answer.
- Relation arguments are not evaluated.

Summary cont’d

- A data object is either a variable or a constant.
- Variables:
  - either bound or unbound.
  - either bound via explicit matching or implicitly via subgoal matching.
  - once bound, the value can never be unbound nor change.
- Constants can either be:
  - atoms, numbers, or structures.

Summary cont'd

- Structures have a functor name and an arity.
- Matching:
  - without unbound variables is trivial.
  - so is unbound variable against non-structure
  - so is unbound variable against structure not containing that unbound variable
- Result of matching unbound variable against structure containing that variable depends on whether that Prolog implemented occurs check.