Programming in Logic: Prolog

Introduction

Reading: Read Chapter 1 of Bratko

Overview

- Administrivia
- Knowledge-Based Programming
- Running Prolog Programs
- Prolog Knowledge Base Anatomy

Administrivia: Contact Info

- Instructor: Mike Barley
- Office: Rm 253 Math/Physics Bldg
- Phone: x6133
- Email: barley@cs.auckland.ac.nz
- Office Hours: Tues 1-3:00

Administrivia: Assessment

- 2 Homework assignments: 5% each
  - Assignment 1
    - Handed out: 5 March
    - Due back: 19 March
  - Assignment 2
    - Handed out: 19 March
    - Due back: 2 April
- Test: 10% - 2 May
- Exam: 30%

Administrivia: Which Version of Prolog?

- A copy of SICStus Prolog 3.3 is on cs26.
- Its path is: **/usr/local/bin/prolog**
- Assignments will be marked using this Prolog.
- **Warning:** Different versions of Prolog handle certain things slightly differently.
- Evaluation copies of SICStus Prolog 3.8 are freely available from **www.sics.se** but they are only "good" for a month.

Administrivia: Textbook

- Bratko, "Programming for Artificial Intelligence", 3rd edition. Required.

Administrivia: Prerequisites

- Understand syntax, semantics, and some basic results of first-order predicate calculus (PC).
- For example, DeMorgan's law applied to quantifiers: $\sim(\exists x \forall y\ loves(y,x) \vee hates(y,x))$
  - **A)** $\exists x \forall y \sim loves(y,x) \wedge \sim hates(y,x)$
  - **B)** $\forall x \exists y \sim loves(y,x) \wedge \sim hates(y,x)$

- Be able to translate English into PC and vice versa, e.g., no one loves everyone.
  - **A)** $\exists x \forall y \sim loves(x, y)$
  - **B)** $\sim \exists x \forall y\ loves(x, y)$
- Know the difference between:
  - $\exists x \forall y\ loves(x, y)$
  - $\forall x \exists y\ loves(x, y)$

Administrivia: Course Goals

- Give a glimpse of the beauty of logic programming.
- Give a taste of knowledge-based programming.
- Develop ability to write declarative specifications of program in Prolog.
- Develop ability to incrementally optimise a Prolog program.

Administrivia: Syllabus

- Pure Prolog: Chapters 1-3.3, 4
  - Purely Declarative Features
- Prolog: Chapters 3.4, 5, 6, 7
  - Introducing Control Features
- Advanced Features: Chapters 8.5, 9, 14
  - More of both

Declarative Programming

- Declarative programming describes *what to compute* rather than *how to compute it*.
- E.g., blueprints for a house are declarative, they describe what to build **not** how to build it.
- Describing "what" is often much easier than describing "how" (but not always).
- *Algorithm = Logic + Control* (R. Kowalski, 1979)
- Logic expressions are declarative.

Advantages of Declarative Style of Programming

- Simply encode your knowledge without worrying how the knowledge will be used.
- The underlying inference engine uses that knowledge to answer the user's queries.
- Knowledge can be used in many ways:
  - Is Mary Peter's sister?
  - Who is Peter's sister?
  - Who is whose sister?

Knowledge Bases

- A Knowledge Base has:
  - Knowledge in the form of:
    - Facts (e.g., Socrates is a man)
    - Rules (e.g., All men are mortals)
  - An inference engine
- A Knowledge Base uses its facts, rules, and inference engine to answer questions.
  - *Is Socrates mortal?*
  - **yes**

Logic Programming & Knowledge Bases

- Logic programming languages are one way to implement knowledge bases.
- Encode your knowledge base and your queries then the underlying inference engine will attempt to answer your queries.
- The inference engine answers your queries by building constructive proofs that your queries are entailed by the knowledge base.

Simple Example: Families

- Define the relevant relationships:
  - mother, father, brother, sister, aunt, uncle, cousin, ancestor, descendant
- Store the basic facts:
  - parents, siblings, and gender
- Ask your queries:
  - Who is whose sister?

Some Rules

- *motherOf(M,O) :- parentOf(M,O), female(M).*
- *sisterOf(S,P) :- siblingOf(S,P), female(S).*
- *auntOf(A,N) :- sisterOf(A,X), parentOf(X,N).*
- *grandmotherOf(G,P) :-*
  motherOf(G,X), parentOf(P).
- *ancestor(A,P) :- parentOf(A,P).*
- *ancestor(A,P) :- parentOf(A,X), ancestor(X,P).*
- ...

Some Facts

- *male(john).*
- *female(mary).*
- *male(peter).*
- *parentOf(john, mary).*
- *siblingOf(mary, peter).*
- *parentOf(ann, john).*
- *parentOf(mark, ann).*
- ...

Some Queries

- **?-** *sisterOf(mary, peter).*
- **?-** *sisterOf(mary, Who).*
- **?-** *sisterOf(Sis, peter).*
- **?-** *sisterOf(Sister, Sibling).*
- **?-** *ancestorOf(A,P).*

Their Answers

- **?-** *sisterOf(mary, peter).*
  - **yes**
- **?-** *sisterOf(mary, Who).*
  - **Who = peter ?** *;*
  - **no**
- **?-** *sisterOf(Sis, peter).*
  - **Sis = mary ?** *;*
  - **no**

More Answers

- **?-** *sisterOf(Sister, Sibling).*
  - **Sibling = peter,**
  - **Sister = mary ?** *;*
  - **no**

Last Answer

- **?-** *ancestorOf(A,P).*
  - **A = john,**
  - **P = mary ?** *;*
  - **A = ann,**
  - **P = john ?** *;*
  - **A = mark,**
  - **P = ann ?** *;*
  - **A = ann,**
  - **P = mary ?** *;*
  - **...**

Running Prolog

- Create knowledge base using favorite editor.
- Type **/usr/local/bin/prolog** on cs26.
- Load that knowledge base into Prolog:
  - *['myKnowledgeBase.pl'].*
- Ask queries:
  - *sisterOf(X,Y).*
- Exit Prolog:
  - *halt.*

Prolog Knowledge Base Anatomy

- Knowledge Base
  - Relations
    - Clauses
      - Terms

Terms

- Terms are things like atoms, numbers, variables, and structures:
  - *tom, 25.3, X, name(mike, barley)*
- In *happy(P) :- paid(P, X), spend(P,Y), X>Y*
  happy(P), :-, paid(P, X),
  spend(P,Y), X>Y, P, X, and *Y* are all terms.

Anatomy of a Clause

- All clauses terminated by full-stop("**.**").
- Clauses have the form:
- *head :- body* .

Head of a Clause

- The head may be the relation name with arguments or may be missing, Examples:
  - *likes(X,Z) :- likes(X,Y), likes(Y,Z).*
  - *likes(mike,X) :- true.*
  - *:- write(***).*
- *likes(mike,X) :- true.* ⇔ *likes(mike,X)* .
- Clauses with missing bodies are called *facts*.
- Facts with variables are called *universal* facts.

Body of a Clause

- Body is an expression composed of terms.
- When the clause head is missing then the body is executed at load-time.

Anatomy of a Relation

- A relation is identified by its name and its arity (# of arguments) - name /arity
  - likes/2 is a different relation from likes/3
- A relation is defined by the clauses whose heads match the relation id, e.g., the clause
  - *ancestor(A, P) :- parentOf(A, P).* is part of the definition of ancestor/2

Anatomy of a Query

- Queries are input by the user (rather than part of the knowledge base).
- Queries have clause body syntax & semantics, notably variables are existentially quantified.
- When query has variables, & Prolog succeeds in proving it follows from KB, Prolog displays variable bindings used in proof.

Quick Quiz

- What do you ignore (at least initially) in declarative-style programming?
- What are the two main components of a knowledge-based system?
- What is the type of knowledge encoded in a Prolog knowledge base?

Quick Quiz cont'd

- By what two things are relations identified?
- In a Prolog knowledge base, what constitutes the definition of a relation?
- What forms can a clause take?
- What are the two parts of a clause?
- What terminates a clause?
- Give examples of different types of terms.

Summary

- Declarative programming focuses on specifying **what** you want, not on **how** to get it.
- Knowledge based systems provide an underlying inference engine, the user provides (in declarative form) the knowledge and the queries.
- Prolog can be viewed as a type of knowledge based programming system.

Summary cont'd

- 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)

Summary 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.