Prolog and Logic Programming

CS152
Chris Pollett
Outline

• Logic and Logic Programs
• Horn Clauses
• Resolution and Unification
• Prolog
Introduction

- So far this semester we have considered three language paradigms: procedural languages, object-oriented languages, and functional programming languages.
- We are now going to look at one more language paradigm: logic programming.
- Logic is closely related with computer programs. For instance, AND, OR, NOT logic gates can be used to build up computer circuits. We have briefly mentioned the formula-as-types interpretation. We have also talked about how ML served as the base language for several automated theorem provers.
- We haven't really said what logic was though…
- So we will talk a little bit about this before we talk about logic programming and then Prolog.
Logic and Logic Programs

• The kind of logic used in logic programming is the **first-order predicate calculus**.

• When we work in first-order logic, we usual work in a particular language.

• A **language** is specified by specifying its:
  – Constants -- things 0 or 1
  – Functions -- these may be of different arity: S(x) := x+1, Plus(x,y), Times(x,y), …
  – Predicates -- P(x), Q(x,y), … We imagine that predicates take inputs from some domain and return a true or false answer. For example equals(x,y) might take inputs which are natural numbers and returns true or false depending on whether x and y are equal.

• A **term** in the language is either a constant, a variable, or built from other terms using functions of the language.

• An **atomic formula** in the language is either a predicate whose parameters have been filled in with terms.
More First-Order Logic

- A **first-order formula** is either an atomic formula or built out of first-order formulas using AND (\(\land\)), OR (\(\lor\)), NOT (\(\neg\)), IMPLIES (\(\rightarrow\)), EXISTS (\(\exists x\)), FORALL (\(\forall x\)).

- For example, Even(x) := (\(\exists y\))( x = 2*y) is a first-order formula expressing x is an even number. Notice 2*y and x are terms, so x = 2*y is an atomic formula, so (\(\exists y\))( x = 2*y) is a formula.

- In the above, the variable x would be called a **free** variable and the variable y is called **bound**.

- Notice depending on the value of x, Even(x) may be either true or false.

- Typically in mathematics, we start with a set of formulas (axioms) which we think are always true and we see what others facts we are able to derive from this formulas.

- A formula derivable from our axioms is called a **theorem**.
Rules of Inference

• So given a set of true formulas, what are the legal inferences we can make?

• For example, if I know A is true, I can infer \( A \lor B \) is true. Similarly, if I know \( A(t) \) holds, I can infer \( (\exists y)A(t) \). Given \( A \rightarrow B \) and \( B \rightarrow C \), I can infer \( A \rightarrow C \).

• These are examples of valid rules of inference. There is a finite list of inferences I, such that given a list of axioms A and any statement T that follows from A, we can start from formulas in A and only apply inferences in I to get new formulas, and eventually reach the formula T.

• A derivation of T from A using I would be called a **proof**.

• A **logic programming language** is a notational system for writing logic statements together with specified algorithms for implementing inference rules.
Horn Clauses

- A **Horn clause** is a statement of the form:
  \[ a_1 \text{ AND } a_2 \text{ AND } \ldots \text{ AND } a_n \rightarrow b \]
- \( b \) is called the **head** of the clause, and the rest of the clause is called the **body**.
- Horn clauses are particularly simple formulas which are used for creating a computer language.
- A clause of the form \( \rightarrow b \) is called a **fact**, and might just be written as \( b \).
- As an example of how Horn clauses might be useful in terms of expressiveness, consider the following definition of the natural numbers:
  1. natural(0).
  2. natural(x) \( \rightarrow \) natural(successor(x)).
- **To prove** natural(successor(successor(0))). We can use axiom (1) together with axiom (2) twice and modus ponens.
Resolution and Unification

• There are two aspects to derivations involving Horn clauses and these will provide the basic algorithmic component of Prolog: resolution and unification.

• Given two Horn clauses:
  
  A <- A₁, …, Aₙ
  B <- B₁, …, Bₙ

  where Bᵢ = A, resolution is the rule of inference:
  B <- B₁, …, Bᵢ₋₁, A₁, …, Aₙ, Bᵢ₊₁, …, Bₙ.

• Unification was the process of pattern matching we used on the last slide to match natural(successor(x)) and natural(successor(successor(0))) by setting x = successor(x).
Prolog

• We started going over my Prolog tutorial: