Computing Science (CMPUT) 325

Nonprocedural Programming

Department of Computing Science
University of Alberta

Winter 2018

Logic Programming vs Other Programming Styles

- Review - major types of programming styles
- Imperative - use destructive assignments to change state
  - `int x = 5;`
  - `x = 15;`
- Functional programming - first part of this course
  - Computation as function application
- From now on: logic programming, declarative programming
  - Computation as logical inference
  - From facts and rules, derive answers to a given query

What is logic programming (LP)?

- Programs are written in the language of some logic
- Execution of a logic program is a theorem proving process
- Prolog, PROgramming in LOGic, is a LP language
- Prolog is based on a subset of first order predicate logic
- There are many other LP languages
- Prolog is (by far) the most popular LP language for general purpose programming

Prolog System, Tutorial and Labs

- We use SWI Prolog
- Open source Prolog system, see Resource page on eClass for download
- Installed on the undergrad machines
- Type `swipl` on the command line to start
- Prolog Tutorial **next week** in labs

Why logic programming (LP) and Prolog?

- Declarative style of programming
- Write a specification of a solution to a problem
- The specification is **executable** -
  Prolog can use it to find a solution
- The core of Prolog is a **search process**
  to find such a solution
- Prolog is further away from classical programming
  than functional programming

How is LP different from Imperative or Functional Languages?

- So far: specify all computations step by step
- Prolog: specify what properties a solution should have, then let Prolog search for it
- Understanding this search and using it to your advantage is the key for mastering Prolog
- Several other features of Prolog help with this - see next slide

Some Important Features of Prolog

- Procedure is defined recursively
- Invocation of a procedure is by pattern matching
- Prolog has a single but very powerful logic inference rule called unification
- Combining these features, you can write some amazingly short and powerful programs
- You can also get unexpected behavior and infinite loops very easily... we will learn best practices to stay out of trouble

Logic

- There are many different kinds of logic
  - Predicate logic, first order logic, higher order logic, modal logic, lambda calculus,...
- A logic is a language
- Like any language it has syntax and semantics
- Beyond ordinary language, a logic also has inference rules

Syntax

- Syntax: the rules about what are **well-formed formulas** in a logic
- Example: predicate logic: $a \vee b$
- Example: first-order logic: $\forall x\, p(x)$
- Syntax is usually the easy part of a logic
- Semantics is the hard part

Semantics

- Semantics in written or spoken language:
  the meaning of a text
- Semantics in logic: the meaning of a well-formed formula
- How to express the meaning in a formal way?
- The classic approach is called “model-theoretic” semantics
- We will discuss it a little bit now, in depth later in this course
- Other formal semantics exist - see `https://en.wikipedia.org/wiki/Formal_semantics_(logic)`

Logical Consequences

- Roughly, a semantics describes all the logical consequences of a formula
- Example: predicate logic formula F: $a \wedge b$
- $\wedge$ is logical `and`
- a and b are atoms
- Assume F is true. Then we can infer:
  - a is true
  - b is true
- Whenever F is true, both a and b have to be true as well

Expressing Logical Consequence In Natural Language

- In English, there are many different but equivalent ways of stating logic consequences. Some examples:
  - b is a logic consequence of F;
  - b follows (logically) from F
  - F implies b
  - F entails b
  - whenever F is true, b is also true
  - b is true because of F
  - ...

Inference Rules

- Given a collection of one or more formulas:
- An **inference rule** can be used to derive new formulas
- Examples:
  - Given the formula $a \wedge b$,
    we can derive the new formula a
  - Given the formula $a \wedge b$,
    we can derive the new formula b

Logical Consequence vs Inference

- There is a strong link between consequence and inference
- Semantics: a is a logical consequence of $a \wedge b$
- Inference: an inference rule allows us to create a new formula a from a given formula $a \wedge b$
- Ideally, we want inference rules that are:
  - **Sound** - they respect the semantics
  - **Complete** - they allow us to derive everything that is true according to the semantics

Sidebar: Our University’s Motto

- The UofA’s motto is `Quaecumque Vera`.
- It is Latin and means “Whatsoever things are true”
- It asks us to go out and find the truth about the world through research
- Similarly, sound and complete logical inference rules can be used to find the true formulas in a given “world”, which is given as a set of formulas

Example - Unsound Inference Rule

- Example of a bad inference rule:
- From $a \vee b$, derive $a \wedge b$
- $\vee$ means logical `or`
- In English: if *a* or *b* is true, then both *a* and *b* are true
- We know this is wrong in predicate logic
- A system that used such a rule would “prove” formulas that do not correspond to our intended semantics
- Example: “the ball is red or green” does not imply “the ball is red and green”

Example - Unsound Inference Rule

- We can show by using a counterexample that the implication:
  - if $a \vee b$, then $a \wedge b$
- ...is invalid in predicate logic.
- Proof: set a to `true`, b to `false`
- Then $a \vee b$ is `true`, and $a \wedge b$ is `false`
- The implication `true` $\rightarrow$ `false` is false according to the rules of logic.

More on Sound and Complete; Efficiency

- All inference rules must be sound with respect to the semantics
- It is good if they are also complete - if any logical consequence can be derived. But that is not a must for a useful system.
- Inference rules should also be **efficient** to be useful in practice

Natural Deduction is Inefficient

- There is a proof system called “natural deduction” which is close to normal logic rules, such as the one above
- From given set of formulas, derive new ones
- Efficiency problems:
  - We need many inference rules
  - Applying them makes the number of formulas we can derive explode
  - It is too hard to figure out which new formulas will be useful later, and which are just junk
  - Example: from a we can derive $a \vee a$, $a \wedge a$, $a \vee (a \wedge a)$, $a \wedge a \wedge a \wedge a \wedge a \wedge a$, ...
- Prolog has only a single inference rule called **resolution**. It is more efficient in general.

Summary so Far

- Discussed logic in terms of syntax, semantics, and inference rules
- Discussed problems with “human-like” inference rules
- Hinted that Prolog uses a different style of inference based on resolution