Mathematics for Computer Science

Logic and Discrete Structures

Daniel Paulusma

Department of Computing Science

Introduction to Logic

(module based on slides by Iain Stewart)

Durham University

So what is logic?

- Every logic (for there are many!) comprises
  - formal language for making statements about certain objects
  - formal system for reasoning about properties of these objects.

- "*Why do we need a special language*? *Why not English*?"
  - English is so rich that it cannot be formally described.
  - The meaning of an English sentence can be ambiguous
    - "*Fruit flies like a banana.*"
    - "*I once shot an elephant in my pyjamas.*
      How he got in my pyjamas I'll never know."
- Objective
  - to carry out precise and rigorous arguments about assertions and proofs, and to implement these arguments and proofs
  - we need a language whose structure (syntax) can be precisely described and whose meaning (semantics) can be unambiguously defined.

So what is logic?

- Once the formulae of a logic have been defined, there are two fundamental aspects
  - a system of deduction by which proofs can be constructed
    - a proof system
  - a notion of meaning by which the truth (or falsity) of some property of some object can be determined
    - its semantics.
- Logic has history
  - Aristotle (384 BC - 322 BC) is generally regarded as the founder of formal logic.
- The logics we will study are
  - propositional logic
  - first-order (or predicate) logic
    - though there are many other logics
      - e.g., modal logic, temporal logic, infinitary logic, second-order logic, …
      - all of which have applications in Computer Science.

So what is logic?

- A logic generally has three components
  - syntax
    - the definition of the (well-formed) formulae of the logic
  - semantics
    - the association of meaning and truth to the formulae of the logic
  - proof system
    - the manipulation of formulae according to a system of rules.
- What we would like from the syntax, semantics and a proof system is that
  - all the “true” (semantics) formulae should be “provable” (syntax, proof system)
    - completeness
  - a formula that is “provable” (syntax, proof system) should be “true” (semantics)
    - soundness.
- We also want to check whether a formula is “true” or “provable” using a computer
  - even better, we want to do this “quickly”.
- As we shall see, sometimes (most of the time?) our world is less than ideal …

Logic in action: programming languages

- The syntax of a *programming language* determines
  - exactly which combinations of symbols constitute a legitimate program.

- Intuitively, assignments are of the form
  - `s := x + y` or `t := z - y` or `x := 4`

  and programs are sequences of assignments such as
  - `s := x + y; x := x - 3; t := z + z; z := z + t.`

- We can give semantics to our programs as follows
  - initially, all variables have some given non-negative integer values
  - we execute the program, with the usual definitions of $+$ and $-$.

- Thus, an input for a given program ρ is
  - a specification ν of a non-negative integer value for every variable of ρ.

- An input ν might be said to *satisfy* ρ if
  - throughout the execution of ρ with the variables initially valued by ν
    - no variable ever takes a negative value.

Logic in action: circuits

- A *logic gate* performs a Boolean operation on digital inputs
  - it provides the result of this operation as output.
- Logic gates are composed to form *logic circuits*
  - fundamental parts of computers.
- The (intended) behaviour of a logic circuit is modelled by a *truth table*
  - details the output of the circuit for every possible combination of inputs
- Propositional logic can be used to automatically realise a (possibly incomplete) specification of a logic circuit as a truth table.

  Moreover, this can be done so that the use of components is minimized.

Logic in action: databases

- A *database* (a.k.a. *table*, *relation*) is a structured collection of logical records.
- A *database query language*
  - a language for asking (and answering) questions of this structured data.
- Almost all database query languages are built on *SQL*
  - e.g., a database Music might contain the following *records* (a.k.a. *tuples*, *rows*)

Group	Number	Genre	Nationality	Active
The Sound	4	post-punk	UK	no
Stereolab	8	mixed	mixed	no
Nick Cave	7	rock	Australia	yes
The Associates	4	new wave	UK	no

  - here, each record has 5 *attributes* (a.k.a. *columns*, *properties*), each taking a value from a specific *domain*.
- SQL allows us query a database
  - e.g., `SELECT Group, Genre FROM Music`
- The expressive power of SQL is very closely related to that of predicate logic.

Logic in action: formal methods

- *Formal methods*
  - use of mathematically-based techniques for the specification and verification of computer systems

  prove that programs have certain properties
  don't just rely on testing.
- *Model checking* is a branch of formal methods where
  - a computer system is first modelled as some mathematical structure
  - then a specific property that this system *might* have is expressed by a formula of some logic.
- We then computationally verify as to whether the particular formula is satisfied by the mathematical model
  - that is, whether the actual computer system has the specific property.

Logic in action: formal methods

- Microprocessor design
  - all major microprocessor manufacturers use model checking methods as a part of their design process.
- Design of data-communications protocol software
  - model checkers have been used as rapid prototyping systems for validating new data-communications/security protocols.
- Critical software
  - NASA uses model checking to look for bugs in code developed by the space program.
- Operating systems
  - Microsoft is using model checking to verify the correct functioning of new Windows device drivers.

A history lesson

non-examinable

- Looking forward
  - logic underpins much of modern Computer Science
    - e.g., AI, information retrieval, security, …

  … but the reason is because
  - logic has its foundations firmly rooted in the past.
- During the late 1800's and early 1900's there was a concerted effort to completely formalize Mathematics and the notion of a mathematical proof.
- *Gottlob Frege*, a German mathematician, had attempted to show that all of Mathematics grew out of logic
  - in doing so, in 1879 he invented first-order predicate logic.
- His intention was to show that there was
  - a set of axioms (basic and obvious facts)
  - a set of logical rules (unambiguous)

  so that
  - all true mathematical statements expressible in Frege's logic could be inferred from these axioms and using these rules.

Russell's Paradox

non-examinable

Durham University

- However, in 1901 *Bertrand Russell*, a British philosopher, devised what is now known as *Russell's Paradox*
  - dealt a killing blow to Frege's hopes.

- In Frege's logical system, one could define certain sets of objects
  - Russell showed that the set R can be so defined

$$R = \{A \text{ is a set} : A \notin A\}.$$

- The paradox arises when one asks whether $R \notin R$

  - if $R \notin R$ then R satisfies the premise to be in R
    - so $R \in R$

  - if $R \in R$ then R satisfies the premise to be in R
    - so $R \notin R$.

Hilbert's Programme

non-examinable

Durham University

- The general quest to formalize Mathematics culminated in *Hilbert's Programme*, named after the German mathematician *David Hilbert* who proposed it in 1921.
- Essentially, Hilbert believed that
  - all mathematical statements could be written in a formal language and manipulated according to formal rules
  - all true mathematical statements could be proved in the formalism
  - there would be an "algorithm" to decide whether any mathematical statement is true or not.
- Hilbert thought that there might be some sort of "computational" logic "defining" all of Mathematics.

Gödel's Incompleteness Theorems

non-examinable

Durham University

- However, Hilbert's Programme was dealt a devastating blow in 1931 by the Austrian mathematician *Kurt Gödel* who proved what are now known as *Gödel's Incompleteness Theorems*.
- Essentially, *Gödel's First Incompleteness Theorem* shows that Hilbert's Programme is doomed to failure
  - in any acceptable logical system powerful enough to describe the arithmetic of the natural numbers
    - there are *true* things about the natural numbers that cannot be *proven* in the system.
- An aside
  - from 1940, Gödel and Einstein were colleagues at the Institute of Advanced Study at Princeton, USA
  - in later life, Einstein confided that
    - "*his own work no longer meant much; he came to the Institute merely ... to have the privilege of walking home with Gödel*".

Logic and computation

non-examinable

Durham University

- Part of Hilbert's Programme was to solve the *Entscheidungsproblem* which asks for
  - an "algorithm" that will take as input
    - a description of a formal language and
    - a mathematical statement in the language
  - produce as output
    - either "true" or "false", according to whether the statement is true or false.

- In 1936 *Alonzo Church* and in 1937 *Alan Turing* published independent papers showing that a general solution to the Entscheidungsproblem is impossible
  - Turing proved his result by reformulating Kurt Gödel's proofs but in the context of computation and using what are now known as *Turing machines*.
  - Church's notion of a computer was the *lambda calculus* which led to *functional programming languages* such as Haskell and LISP.

The Church-Turing Thesis

non-examinable

Durham University

- Prompted by Hilbert's Programme and Gödel's Incompleteness Theorems
  - there was much activity in the 1930's around the notion of computation.

- As well as Gödel, Turing, and Church, other researchers had proposed different notions of computation (such as Rosser, Kleene, and Post)
  - all these notions were proven to be equivalent.

- The *Church-Turing Thesis* is the *thesis* that all of these (equivalent) notions of computation embody what it means to be computable
  - so, we not only have a definition of "computable" …

    … but *proofs* that certain problems are "uncomputable" (or "unsolvable").

Computational complexity

non-examinable

Durham University

- With the advent of the digital computer in the 1950's, the notion of *efficiently computable* rose to the fore, but it was not until the 1970's that notions such as
  - **P**, polynomial-time
    - "efficiently solvable"
  - **NP**, non-deterministic polynomial-time
    - "efficiently checkable"

  were precisely formulated and studied.
- Remarkably, in a letter from the 1950's from Kurt Gödel to John von Neumann, Gödel actually almost precisely defines the **P** versus **NP** problem
  - a computer science problem that had not at that time been formulated and that we still have not solved!
- An award of $1 million has been offered by the *Clay Mathematics Institute* for a solution to the **P** versus **NP** problem.
- Logic lies at the heart of the **P** versus **NP** problem.