Lecture 1

History of mathematical logic in computer science

Print version of the lecture in *Logic and Proof*

presented on 29 April 2019

by Dr Christoph Haase

1 Practicalities

- General information:
  - Lectures in Lecture Theatre B:
    - Mon 11am-12pm, Wed 11am-12pm (Weeks 1-6)
    - Thu 11am-12pm (Weeks 1-4)
  - Five exercise sheets, new format this year
  - Departmental Classes
    - Tue 10-11am, Tue 11am-12pm, Tue 2-3pm, Weeks 2-6 in Room 013RHB
    - Submit by 5pm on preceding Friday
  - Lecture notes, slides, exercise sheets: online
  - Questions after lecture? Use Piazza in the first instance:

    `piazza.com/ox.ac.uk/spring2019/lptt19/`
  - Alternatively, discuss with your tutor or send me an email at

    christoph.haase@cs.ox.ac.uk
  - Office hours: Tue 1:30-2:30pm, Room 417
- Recommended textbooks:
  - 'Logic for Computer Scientists', U. Schöning
  - 'Mathematical Logic for Computer Science', M. Ben-Ari
  - 'Logic in computer science: modelling and reasoning about systems', M. Huth and M. Ryan
  - 'Handbook of Practical Logic and Automated Reasoning', J. Harrison
- Further literature:
  - 'Gödel, Escher, Bach: an Eternal Golden Braid', D. Hofstadter
  - 'Logicomix: An Epic Search for Truth', A. Doxiadis and C. Papadimitriou

2 About this course

> *When I was a student, even the topologists regarded mathematical logicians as living in outer space. Today the connections between logic and computers are a matter of engineering practice at every level of computer organization.*
>
> Martin Davis. *Influences of Mathematical Logic on Computer Science.*

Logic is fundamental to computer science. This is not surprising, given that computers are built from Boolean circuits. However, what has been called the *unusual effectiveness of logic in computer science* goes far beyond hardware design: it applies, among other things, to knowledge representation, programming-language theory, automated verification, complexity theory, databases, and constraint solving. The role of logic in computer science has been compared to that of calculus in physics and engineering, and some people even call logic the calculus of computer science.

This course focusses on the foundations of logic rather than its computer-science applications. We only briefly cover some applications for illustrative purposes, and mostly leave applications to subsequent courses in the areas mentioned above. However our emphasis is very much on the computational aspects of logic. In particular we study questions of decidability using notions from the *Models of Computation* course, including finite-state automata and Post's Correspondence Problem. We will also present the satisfiability problem in propositional logic as a prototypical search problem, thus making a connection with the first-year *Algorithms* course.

3 A historical perspective on logic

The study of logic arose from a desire to understand reasoning and argumentation. Aristotle (384–322 BC) compiled a list of *syllogisms*, which can be seen as arguments in which the conclusion follows from the hypotheses *merely by virtue of the meaning of the words if, then, and, or, is, all, are, some and none*. For example,

All *beings* are *mortal*	All B are M
All *humans* are *beings*	All H are B
All *humans* are *mortal*	All H are M

While Aristotle wrote down a compendium of valid arguments, Leibniz (1646–1716) was the first to envision a system of rules (or *calculus*) by which arguments could be systematically constructed and tested for validity. An important step toward this goal was worked out by George Boole (1815–1864) who proposed a set of equational rules for *propositional logic*. A particularly influential contribution of Boole was to give an algebraic formulation of logic. Boole's work was picked up by William Stanley Jevons (1835–1882) who built a mechanical computer, the *logic piano*, to carry out logical deductions.

A more expressive and powerful system than propositional logic, called *predicate logic*, was invented independently by Gottlob Frege (1848–1925) and Charles Sanders Pierce (1839–1914), partly motivated by problems in the foundations of mathematics. Propositional logic and predicate logic are the two main logical systems that we study in this course.

In the first half of the twentieth century logic played a central role in the study of the foundations of mathematics. Russell (1872–1970) and Whitehead (1861–1947) attempted to show in their *Principia Mathematica* how theorems in set theory, arithmetic, real analysis and geometry could be derived from well-defined axioms and rules of inference within a formal system of predicate logic. One of the

most celebrated outcomes of this research program is a negative result, by Kurt Gödel (1906–1978), who showed that no logical system of arithmetic could be both *consistent* (free of contradiction) and *complete* (capable of proving all true facts). Shortly thereafter Alonzo Church (1903–1995) and Alan Turing (1912–1954) independently showed that there is no algorithm for the *Entscheidungsproblem*, that is, the problem of deciding the validity of a given logic statement. The formulation and proof of this last result led directly to the notions of *Turing machine* and λ*-calculus*, thus laying the foundations of theoretical computer science. We will give a proof of this result in this course, building on concepts you have learned in Models of Computation.

4 Contemporary highlights of logic in computer science

More recent developments in logic have been heavily influenced by computer science. We highlight two among many important contributions. Claude Shannon (1916–2001) showed how to use electrical switches to compute Boolean functions, and is regarded as the founder of digital circuit design. Alan Robinson (1925–) discovered *resolution* and *unification*, thus contributing to the foundations of automated reasoning and logic programming. Resolution will be one of the main proof systems considered in this course.

A form of resolution underlies modern *SAT solvers*–computer programs for determining satisfiability of propositional formulas. The dramatic improvement in the performance of SAT solvers over the last 20 years has led to their successful application in areas such as automated verification, cryptography, and artificial intelligence planning.

To be more concrete, subsequently we will briefly present some examples of recent research outcomes that have resulted from research into computational logic. The topics we cover here are of an academic nature in order to ease understanding. It should, however, be noted that concepts, methods and algorithms from computational logic are applied on a daily basis in industry, for instance in the design process of hard- and software.

4.1 Finding a needle amongst $1,566 \times 10^{349}$ needles

Our first example is an old conjecture posed by Paul Erdős in the 1930s.

Erdős discrepancy conjecture
For any $C > 0$ and any infinite sequence $x_1 x_2 x_3 \cdots$ of $+1$'s and -1's there exist $d, k \in \mathbb{N}$ such that

$$|\sum_{1 \leq i \leq k} x_{id}| > C.$$

This conjecture was widely believed to be true, however only the case $C = 1$ had been resolved until the 1990s. In 2009, the conjecture was taken on by the Polymath project [3]. The attempt to solve the problem failed, even proving the case $C = 2$ seemed impossible:

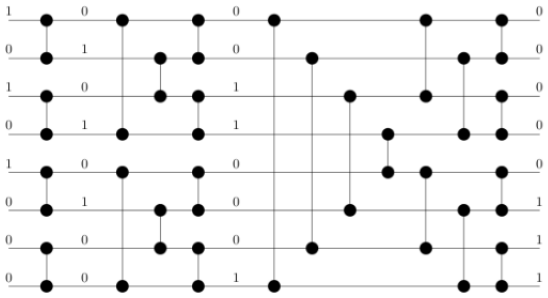
> "Given how long a finite sequence can be, it seems unlikely that we could answer this question [for $C = 2$] just by a clever search of all possibilities on a computer."

In 2014, B. Konev and A. Lisitsa from the University of Liverpool developed a clever encoding of the Erdős discrepancy conjecture into a propositional satisfiability problem, and showed that any sequence of length at least 1161 has discrepancy at least 3 [4]. The computer-generated proof showing that no sequence of length 1161 has discrepancy 2 took 13Gb to store. The achievement of B. Konev and A.

Lisitsa demonstrated the role SAT solvers can nowadays play, even in pure scientific areas such as extremal combinatorics. However, the success of the machines was only of short duration. In September 2015, T. Tao announced a proof of the Erdős discrepancy conjecture, which has since turned into a theorem[5].

4.2 Optimal sorting networks

Sorting is one of the most fundamental algorithmic tasks computers perform on an every day basis. When sorting small amounts of data, using efficient algorithms such as QuickSort can create too much of an overhead. In this case, sorting networks can do a better job for inputs of a fixed length. A sorting network consists of wires and comparators, where inputs flow from the left to the right. Whenever inputs reach a comparator, they are either swapped or left untouched, depending on their relative order. The principle is easily understood from the following illustration:



n	5	6	7	8	9	10	11	12	13	14	15	16
d	5	5	6	6	7	7	8	8	9	9	9	9

Figure 1: Optimal depth d for sorting networks with n inputs.

While optimal sorting networks up to 8 inputs had been found by D. Knuth and R. Floyd in the 1960s, getting the bounds tight for n up to 16 required much longer. Using a sophisticated SAT encoding, the Oxford students D. Bundala and J. Závodný showed in 2014 that the previously known upper bounds for d were indeed tight [2]. Note that this entails showing, for instance, that no sorting network with 16 inputs and depth strictly less than 9 can exist, a challenging combinatorial problem.

4.3 Leibniz' ontological proof

While so far we have focussed on achievements made by encoding problems into propositional satisfiability problem, interactive theorem provers are another area of lively research. Roughly speaking, *interactive theorem provers* provide environments in which axioms and theorems can be formally expressed and reasoned about. Many interactive theorem provers have tactics which can automatically discharge some proof obligations. Whenever they get stuck, they ask the user to provide hints or proofs.

A notable recent application of interactive theorem provers in philosophical logic has been a proof of Leibniz's ontological argument for the existence of God within the axioms of his algebra of concepts. Bentert et al. formalised Leibniz's algebra of concepts in the theorem prover Isabelle/HOL, and showed that depending on a certain interpretation of Leibniz's words, his proof of the existence of God is

valid [1]. While in this course we will neither attempt to prove nor disprove the absence of god, their work provides an interesting example of how research into interactive theorem provers can provide new perspectives on topics from other fields such as philosophy.

References

[1] Matthias Bentert, Christoph Benzmüller, David Streit, and Bruno Woltzenlogel Paleo. Analysis of an ontological proof proposed by Leibniz. In Charles Tandy, editor, *Death and Anti-Death, Volume 14: Four Decades after Michael Polanyi, Three Centuries after G.W. Leibniz*. Ria University Press, 2016. Preprint: `http://christoph-benzmueller.de/papers/B16.pdf`.

[2] Daniel Bundala and Jakub Zavodny. Optimal sorting networks. In Adrian Horia Dediu, Carlos Martín-Vide, José Luis Sierra-Rodríguez, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings*, volume 8370 of *Lecture Notes in Computer Science*, pages 236–247. Springer, 2014.

[3] Justin Cranshaw and Aniket Kittur. The polymath project: Lessons from a successful online collaboration in mathematics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1865–1874, New York, NY, USA, 2011. ACM.

[4] Boris Konev and Alexei Lisitsa. A SAT attack on the Erdős discrepancy conjecture. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 219–226. Springer, 2014.

[5] Terence Tao. The Erdos discrepancy problem. *arXiv preprint arXiv:1509.05363*, 2015.