CST Part IB

Computation Theory

Andrew Pitts

Corrections to the notes and extra material available from the course web page:
www.cl.cam.ac.uk/teaching/1112/CompTheory/

Alan Turing
1912-54

Alonzo Church
1903-95

Introduction

Algorithmically undecidable problems

Computers cannot solve all mathematical problems, even if they are given unlimited time and working space.

Three famous examples of computationally unsolvable problems are sketched in this lecture.

- Hilbert's *Entscheidungsproblem*
- The Halting Problem
- Hilbert's 10th Problem.

Hilbert's *Entscheidungsproblem*

Is there an algorithm which when fed any statement in the formal language of first-order arithmetic, determines in a finite number of steps whether or not the statement is provable from Peano's axioms for arithmetic, using the usual rules of first-order logic?

Such an algorithm would be useful! For example, by running it on

$$\forall k > 1 \, \exists p, q \, (2k = p + q \wedge \mathit{prime}(p) \wedge \mathit{prime}(q))$$

(where $\mathit{prime}(p)$ is a suitable arithmetic statement that p is a prime number) we could solve *Goldbach's Conjecture* ("every strictly positive even number is the sum of two primes"), a famous open problem in number theory.

Hilbert's *Entscheidungsproblem*

Is there an algorithm which when fed any statement in the formal language of first-order arithmetic, determines in a finite number of steps whether or not the statement is provable from Peano's axioms for arithmetic, using the usual rules of first-order logic?

Posed by Hilbert at the 1928 International Congress of Mathematicians. The problem was actually stated in a more ambitious form, with a more powerful formal system in place of first-order logic.

In 1928, Hilbert believed that such an algorithm could be found. A few years later he was proved wrong by the work of Church and Turing in 1935/36, as we will see.

Decision problems

Entscheidungsproblem means "decision problem". Given

- a set S whose elements are finite data structures of some kind
  (e.g. formulas of first-order arithmetic)
- a property P of elements of S
  (e.g. property of a formula that it has a proof)

the associated decision problem is:

find an algorithm which
terminates with result 0 or 1 when fed an element $s \in S$
and
yields result 1 when fed s if and only if s has property P.

Algorithms, informally

No precise definition of "algorithm" at the time Hilbert posed the *Entscheidungsproblem*, just examples, such as:

- Procedure for multiplying numbers in decimal place notation.
- Procedure for extracting square roots to any desired accuracy.
- Euclid's algorithm for finding highest common factors.

Algorithms, informally

No precise definition of "algorithm" at the time Hilbert posed the *Entscheidungsproblem*, just examples.

Common features of the examples:

- finite description of the procedure in terms of elementary operations
- deterministic (next step uniquely determined if there is one)
- procedure may not terminate on some input data, but we can recognize when it does terminate and what the result is.

Algorithms, informally

No precise definition of "algorithm" at the time Hilbert posed the *Entscheidungsproblem*, just examples.

Common features of the examples:

- finite description of the procedure in terms of elementary operations
- deterministic (next step uniquely determined if there is one)
- procedure may not terminate on some input data, but we can recognize when it does terminate and what the result is.

e.g. multiply two decimal digits by looking up their product in a table

Algorithms, informally

No precise definition of "algorithm" at the time Hilbert posed the *Entscheidungsproblem*, just examples.

In 1935/36 Turing in Cambridge and Church in Princeton independently gave negative solutions to Hilbert's *Entscheidungsproblem*.

- First step: give a precise, mathematical definition of "algorithm".

  (Turing: Turing Machines; Church: lambda-calculus.)
- Then one can regard algorithms as data on which algorithms can act and reduce the problem to...

The Halting Problem

is the decision problem with

- set S consists of all pairs (A, D), where A is an algorithm and D is a datum on which it is designed to operate;
- property P holds for (A, D) if algorithm A when applied to datum D eventually produces a result (that is, eventually halts—we write $A(D)\downarrow$ to indicate this).

The Halting Problem

is the decision problem with

- set S consists of all pairs (A, D), where A is an algorithm and D is a datum on which it is designed to operate;
- property P holds for (A, D) if algorithm A when applied to datum D eventually produces a result (that is, eventually halts—we write $A(D)\downarrow$ to indicate this).

Turing and Church's work shows that the Halting Problem is undecidable, that is, there is no algorithm H such that for all $(A, D) \in S$

$$H(A, D) = \begin{cases} 1 & \text{if } A(D)\downarrow \\ 0 & \text{otherwise.} \end{cases}$$

There's no H such that $H(A,D) = \begin{cases} 1 & \text{if } A(D)\downarrow \\ 0 & \text{otherwise.} \end{cases}$ for all (A,D).

Informal proof, by contradiction. If there were such an H, let C be the algorithm:

> "input A; compute $H(A,A)$; if $H(A,A) = 0$ then return 1, else loop forever."

There's no H such that $H(A,D) = \begin{cases} 1 & \text{if } A(D)\downarrow \\ 0 & \text{otherwise.} \end{cases}$ for all (A,D).

Informal proof, by contradiction. If there were such an H, let C be the algorithm:

> "input A; compute $H(A,A)$; if $H(A,A) = 0$ then return 1, else loop forever."

So $\forall A\,(C(A)\downarrow \leftrightarrow H(A,A) = 0)$ (since H is total)

and $\forall A\,(H(A,A) = 0 \leftrightarrow \neg A(A)\downarrow)$ (definition of H).

There's no H such that $H(A,D) = \begin{cases} 1 & \text{if } A(D)\downarrow \\ 0 & \text{otherwise.} \end{cases}$ for all (A,D).

Informal proof, by contradiction. If there were such an H, let C be the algorithm:

> "input A; compute $H(A,A)$; if $H(A,A) = 0$ then return 1, else loop forever."

So $\forall A\,(C(A)\downarrow \leftrightarrow H(A,A) = 0)$ (since H is total)

and $\forall A\,(H(A,A) = 0 \leftrightarrow \neg A(A)\downarrow)$ (definition of H).

So $\forall A\,(C(A)\downarrow \leftrightarrow \neg A(A)\downarrow)$.

There's no H such that $H(A,D) = \begin{cases} 1 & \text{if } A(D)\downarrow \\ 0 & \text{otherwise.} \end{cases}$ for all (A,D).

Informal proof, by contradiction. If there were such an H, let C be the algorithm:

> "input A; compute $H(A,A)$; if $H(A,A) = 0$ then return 1, else loop forever."

So $\forall A\,(C(A)\downarrow \leftrightarrow H(A,A) = 0)$ (since H is total)

and $\forall A\,(H(A,A) = 0 \leftrightarrow \neg A(A)\downarrow)$ (definition of H).

So $\forall A\,(C(A)\downarrow \leftrightarrow \neg A(A)\downarrow)$.

Taking A to be C, we get $C(C)\downarrow \leftrightarrow \neg C(C)\downarrow$, contradiction!

There's no H such that $H(A,D) = \begin{cases} 1 & \text{if } A(D)\downarrow \\ 0 & \text{otherwise.} \end{cases}$ for all (A,D).

Informal proof, by contradiction. If there were such an H, let C be the algorithm:

> "input A; compute $H(A,A)$; if $H(A,A) = 0$ then return 1, else loop forever."

So $\forall A\,(C(A)\downarrow \leftrightarrow H(A,A) = 0)$ (since H is total)

and $\forall A\,(H(A,A) = 0 \leftrightarrow \neg A(A)\downarrow)$ (definition of H).

So $\forall A\,(C(A)\downarrow \leftrightarrow \neg A(A)\downarrow)$.

Taking A to be C, we get $C(C)\downarrow \leftrightarrow \neg C(C)\downarrow$,

contradiction! $p \leftrightarrow \neg p = (p \wedge \neg p) \vee (\neg p \wedge \neg\neg p) = F \vee F = F$

There's no H such that $H(A,D) = \begin{cases} 1 & \text{if } A(D)\downarrow \\ 0 & \text{otherwise.} \end{cases}$ for all (A,D).

Informal proof, by contradiction. If there were such an H, let C be the algorithm:

why is A "a datum on which A is designed to operate"?

> "input A; compute $H(A,A)$; if $H(A,A) = 0$ then return 1, else loop forever."

So $\forall A\,(C(A)\downarrow \leftrightarrow H(A,A) = 0)$ (since H is total)

and $\forall A\,(H(A,A) = 0 \leftrightarrow \neg A(A)\downarrow)$ (definition of H).

So $\forall A\,(C(A)\downarrow \leftrightarrow \neg A(A)\downarrow)$.

Taking A to be C, we get $C(C)\downarrow \leftrightarrow \neg C(C)\downarrow$, contradiction!

From HP to *Entscheidungsproblem*

Final step in Turing/Church proof of undecidability of the *Entscheidungsproblem*: they constructed an algorithm encoding instances (A, D) of the Halting Problem as arithmetic statements $\Phi_{A,D}$ with the property

$$\Phi_{A,D} \textit{ is provable } \leftrightarrow \; A(D)\downarrow$$

Thus any algorithm deciding provability of arithmetic statements could be used to decide the Halting Problem—so no such exists.

Hilbert's *Entscheidungsproblem*

Is there an algorithm which when fed any statement in the formal language of first-order arithmetic, determines in a finite number of steps whether or not the statement is provable from Peano's axioms for arithmetic, using the usual rules of first-order logic?

With hindsight, a positive solution to the *Entscheidungsproblem* would be too good to be true. However, the algorithmic unsolvability of some decision problems is much more surprising. A famous example of this is...

Hilbert's 10th Problem

Give an algorithm which, when started with any Diophantine equation, determines in a finite number of operations whether or not there are natural numbers satisfying the equation.

One of a number of important open problems listed by Hilbert at the International Congress of Mathematicians in 1900.

Diophantine equations

$$p(x_1, \ldots, x_n) = q(x_1, \ldots, x_n)$$

where p and q are polynomials in unknowns $x_1, \ldots, x_n$ with coefficients from $\mathbb{N} = \{0, 1, 2, \ldots\}$.

Named after Diophantus of Alexandria (c. 250AD).

Example: "find three whole numbers x_1, x_2 and x_3 such that the product of any two added to the third is a square"
[Diophantus' *Arithmetica*, Book III, Problem 7].

In modern notation: find $x_1, x_2, x_3 \in \mathbb{N}$ for which there exists $x, y, z \in \mathbb{N}$ with

$$(x_1x_2 + x_3 - x^2)^2 + (x_2x_3 + x_1 - y^2)^2 + (x_3x_1 + x_2 - z^2)^2 = 0$$

Diophantine equations

$$p(x_1, \ldots, x_n) = q(x_1, \ldots, x_n)$$

where p and q are polynomials in unknowns $x_1, \ldots, x_n$ with coefficients from $\mathbb{N} = \{0, 1, 2, \ldots\}$.

Named after Diophantus of Alexandria (c. 250AD).

Example: "find three whole numbers x_1, x_2 and x_3 such that the product of any two added to the third is a square"
[Diophantus' *Arithmetica*, Book III, Problem 7].

In modern notation: find $x_1, x_2, x_3 \in \mathbb{N}$ for which there exists $x, y, z \in \mathbb{N}$ with

$$x_1^2 x_2^2 + x_2^2 x_3^2 + x_3^2 x_1^2 + \cdots = x^2 x_1 x_2 + y^2 x_2 x_3 + z^2 x_3 x_1 + \cdots$$

[One solution: $(x_1, x_2, x_3) = (1, 4, 12)$, with $(x, y, z) = (4, 7, 4)$.]

Hilbert's 10th Problem

Give an algorithm which, when started with any Diophantine equation, determines in a finite number of operations whether or not there are natural numbers satisfying the equation.

- Posed in 1900, but only solved in 1970: Y Matijasevič, J Robinson, M Davis and H Putnam show it undecidable by reduction to the Halting Problem.

Hilbert's 10th Problem

Give an algorithm which, when started with any Diophantine equation, determines in a finite number of operations whether or not there are natural numbers satisfying the equation.

- Posed in 1900, but only solved in 1970: Y Matijasevič, J Robinson, M Davis and H Putnam show it undecidable by reduction to the Halting Problem.
- Original proof used Turing machines. Later, simpler proof [JP Jones & Y Matijasevič, J. Symb. Logic 49(1984)] used Minsky and Lambek's register machines

Hilbert's 10th Problem

Give an algorithm which, when started with any Diophantine equation, determines in a finite number of operations whether or not there are natural numbers satisfying the equation.

- Posed in 1900, but only solved in 1970: Y Matijasevič, J Robinson, M Davis and H Putnam show it undecidable by reduction to the Halting Problem.
- Original proof used Turing machines. Later, simpler proof [JP Jones & Y Matijasevič, J. Symb. Logic 49(1984)] used Minsky and Lambek's register machines—we will use them in this course to begin with and return to Turing and Church's formulations of the notion of "algorithm" later.