Computability and Intractability

Brief history:

1. Babylonian tablets.

2. Euclid, gcd of integers.

3. Babbage, difference engine and analytical engine.

4. Foundations of Mathematics and Logic.

Notation and conventions

- Express things textually.
- Strings over a finite alphabet A.
- All valid programs: $P_0, P_1, P_2, \ldots$
- All inputs/outputs: $I_0, I_1, I_2 \ldots$
- Encode inputs as natural numbers (convenience only).

Non-termination

Would like: general theory of computation in which all programs are guaranteed to terminate and produce an output.

Consider:

on input n run program P_n on n to obtain
    the output R;
if $R = I_0$ then return I_1 else return I_0;

A valid program P_m (say). Now look at output of P_m when run on m; get a contradiction!

Conclusion: Must drop requirement that all programs always terminate.

> on input n run program P_n on n;
> if this terminates let the output be R;
> if $R = I_0$ then return I_1 else return I_0;

A valid program P_m (say); previous argument shows that P_m does not halt on input m.

The Halting Problem

New goal: find a program H that takes arguments m, n and returns True if P_m halts on input n, otherwise it returns False.

Consider:

> if $H(n,n)$ then loop forever
> else halt (and return 0)

A valid program P_m (say). Now look at output of P_m when run on m; get a contradiction!

Conclusion: H does not exist; halting problem is unsolvable.

Diagonalization

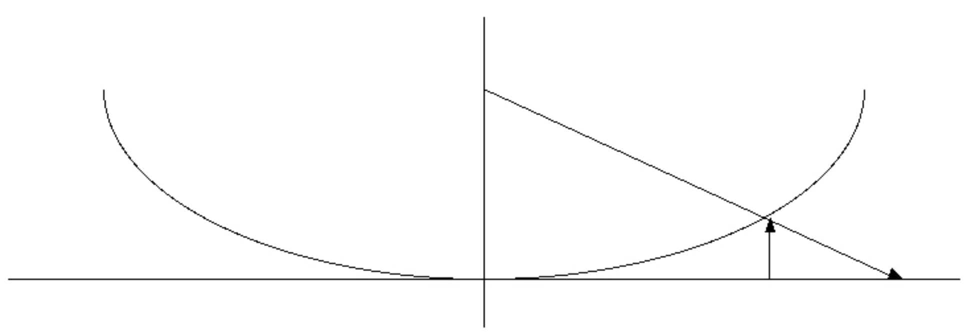
$$
\begin{array}{c|cccc}
 & 0 & 1 & 2 & \dots \\
\hline
P_0 & \boxed{P_0(0)} & P_0(1) & P_0(2) & \dots \\
P_1 & P_1(0) & \boxed{P_1(1)} & P_1(2) & \dots \\
P_2 & P_2(0) & P_2(1) & \boxed{P_2(2)} & \dots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{array}
$$

Cantor: cardinality and infinite sets

Integers and even integers:

$$\begin{array}{ccccccc} \ldots & -2 & -1 & 0 & 1 & 2 & \ldots \\ \ldots & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \ldots \\ \ldots & -4 & -2 & 0 & 2 & 4 & \ldots \end{array}$$

The real numbers and $(0,1)$**:**



$\mathbb{R}$ **versus** $\mathbb{N}$**:** Suppose there is a 1-1 correspondence between $(0,1)$ and $\mathbb{N}$, so can list $(0,1)$ as $\alpha_0, \alpha_1, \alpha_2, \ldots$ where

$$\alpha_i = 0.\alpha_{i0}\alpha_{i1}\alpha_{i2}\ldots$$

$$\begin{array}{c|cccc}
 & 0 & 1 & 2 & \ldots \\
\hline
\alpha_0 & \alpha_{00} & \alpha_{01} & \alpha_{02} & \cdots \\
\alpha_1 & \alpha_{10} & \alpha_{11} & \alpha_{12} & \cdots \\
\alpha_2 & \alpha_{20} & \alpha_{21} & \alpha_{22} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

Define

$$\delta_i = \begin{cases} 1, & \text{if } \alpha_{ii} \neq 1; \\ 2, & \text{if } \alpha_{ii} = 1. \end{cases}$$

Now $0.\delta_0\delta_1\delta_2\ldots$ is in $(0,1)$ but is different from each α_i!

X **versus** $\mathcal{P}(X)$**:** Suppose there is a function f from X *onto* $\mathcal{P}(X)$, i.e., for every $Y \in \mathcal{P}(X)$ there is a $y \in X$ such that $Y = f(y)$. Consider

$$A = \{\, x \in X \mid x \notin f(x) \,\}.$$

There must be an $a \in X$ such that $A = f(a)$. *But* by definition of A,

$$\begin{aligned} a \in A \text{ if and only if } & a \notin f(a) \\ \text{if and only if } & a \notin A! \end{aligned}$$

Paradise lost: Russell's paradox

$$R = \{\, x \mid x \text{ is a set and } x \notin x \,\}.$$

Now

$$R \in R \Leftrightarrow R \notin R.$$

In words: Consider catalogues; some list themselves and some do not. Try to build a catalogue of all catalogues that do not list themselves.

Truth and formal proof: Gödel

$S =$ 'This sentence is unprovable.'

System of deduction D,

$S_D =$ 'This sentence is unprovable in system D.'

$S_{D,n} =$ 'The statement in the system D whose number is n is unprovable in D.'

Formal models of computing

Requirements:

1. Computation within the model should proceed by a sequence of steps, each step being entirely mechanical. We want the model to be, at least in principle, physically realisable.

2. The model should support the computation of all things that we intuitively believe to be computable. This requirement rules out finite state machines.

3. The model should be simple, so that a 'theory of computation' can be developed without unnecessary complications.

Met by model proposed by Alan Turing in 1936