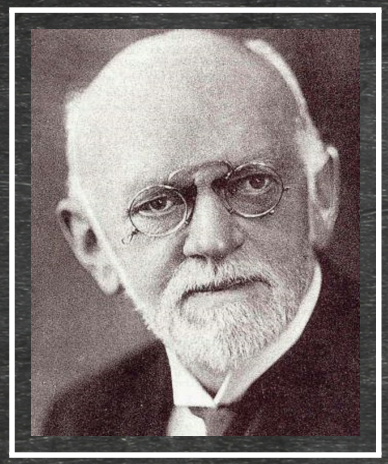
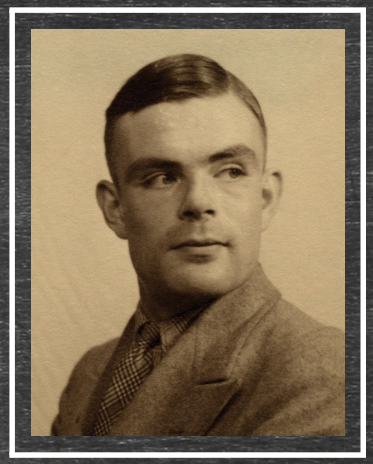
Logic, Algorithms, and Complexity

Yijia Chen
Fudan University

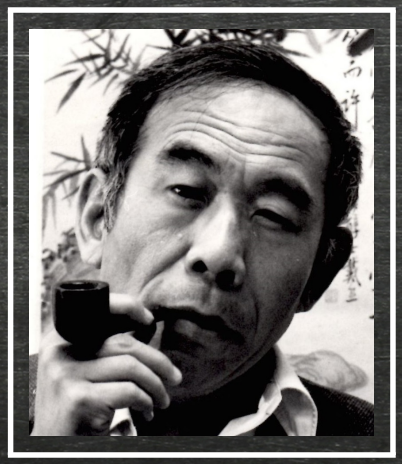


David Hilbert

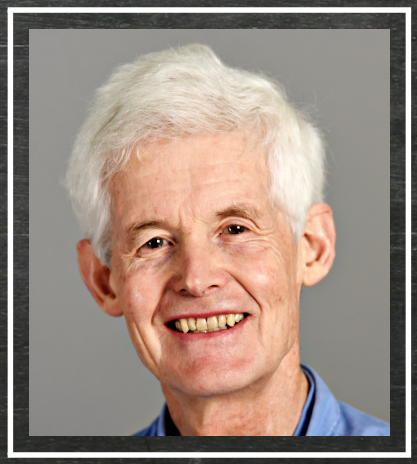


Alan Turing

Kurt Gödel

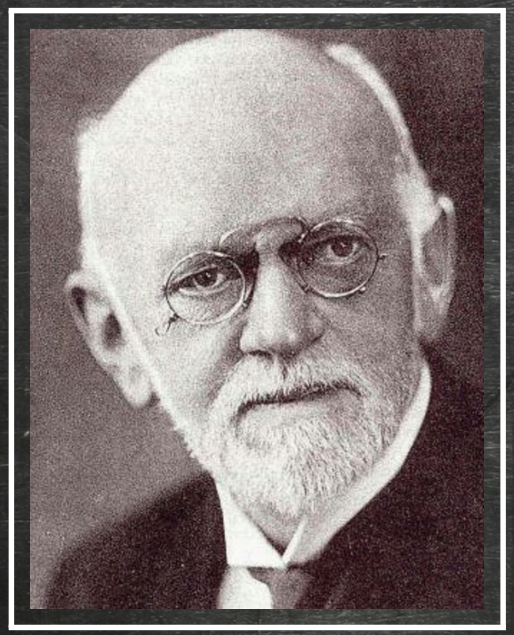


Hao Wang



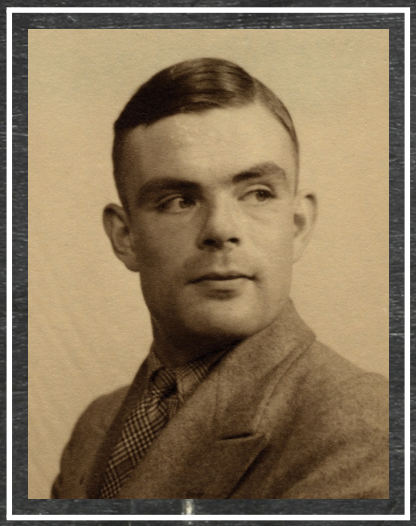
Stephen Cook

Entscheidungsproblem



The Entscheidungsproblem is solved when one knows a procedure by which one can decide in a finite number of operations whether a given logical expression is generally valid or is satisfiable. The solution of the Entscheidungsproblem is of fundamental importance for the theory of all fields, the theorems of which are at all capable of logical development from finitely many axioms.

D. Hilbert and W. Ackermann
Grundzüge der theoretischen Logik, 1928



ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. Turing.

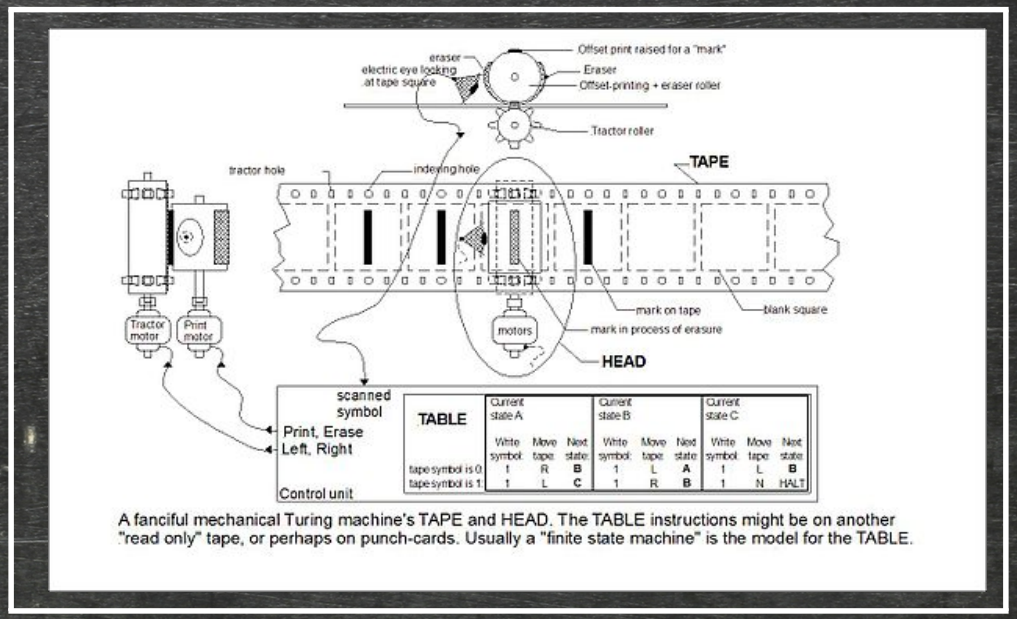
[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

By the correction application of one of theses arguments, conclusions are reached which are superficially similar to those of Gödel. These results have valuable applications. In particular, it is shown that the Hilbertian Entscheidungsproblem can have no solution.

On computable numbers, with an application to the Entscheidungsproblem
A. Turing, 1936.

Turing machines

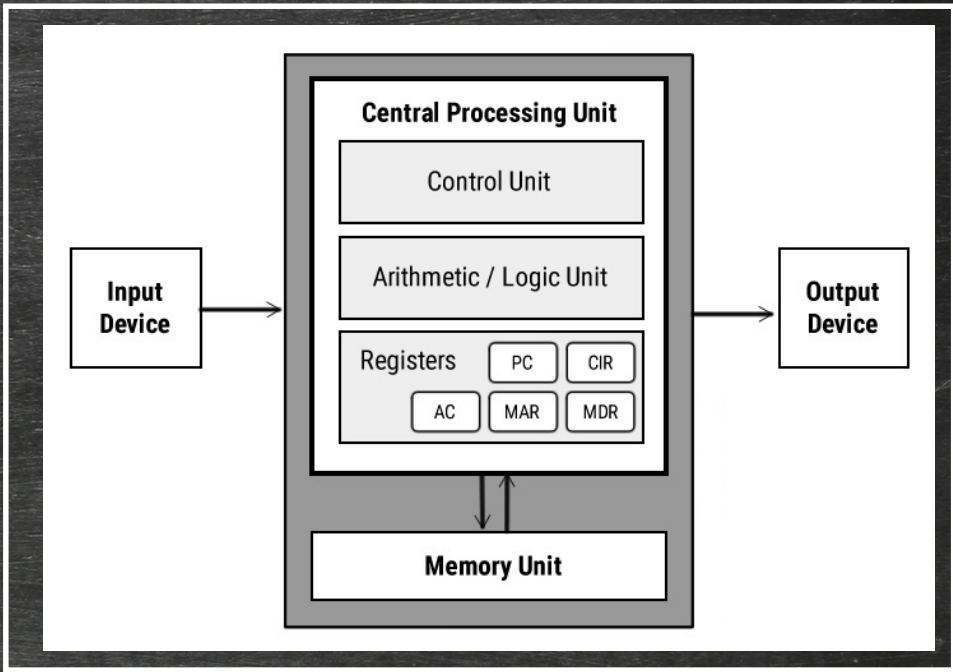


Definition

A Turing machine is a 7-tuple $M = (Q, \Gamma, b, \Sigma, \delta, q_0, F)$ where

1. Q is a finite set of states,
2. Γ is a finite set of tape symbols,
3. ...

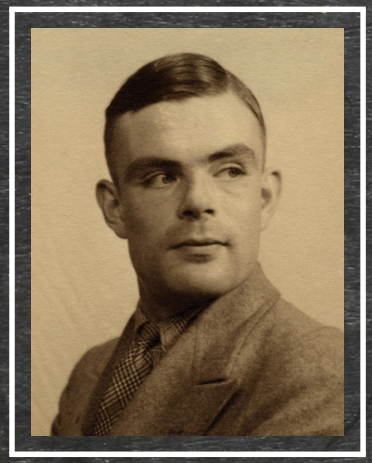
The real computers - von Neumann architecture



[B]y 1946, von Neumann would be assigning Turing's famous paper on computable numbers as required reading for his collaborators in the EDVAC project of constructing his computer.

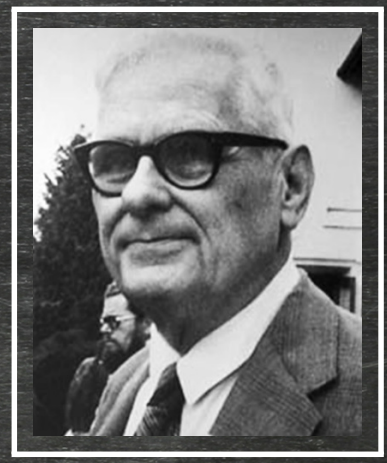
Turing and von Neumann's brains and their computers
S. Istrail and S. Marcus, 2012.

Computation models



Alan Turing

Kurt Gödel



Alonzo Church

- Turing machines.
- Recursive functions.
- Lambda Calculus.

[T]he correct definition of mechanical computability was established beyond any doubt by Turing.

Church-Turing Thesis

Any computable function is computable by a Turing machine.

So the earlier days of computer science were full of decidability results. That is, for a problem $Q \subseteq \Sigma^*$, we design a Turing machine, or an algorithm $\mathbb{A}$, such that for every $x \in \Sigma^*$,

- if $x \in Q$, i.e., a yes instance, then $\mathbb{A}$ will halt and output Yes,
- if $x \notin Q$, i.e., a no instance, then $\mathbb{A}$ will halt and output No.

A "simple" problem from logic

Given a formula of propositional logic, e.g.,

$$(X \vee \bar{Y} \vee Z) \wedge (\bar{X} \vee \bar{Z} \vee W) \wedge \cdots$$

can we assign truth values to all variables such that the formula is evaluated to true. Equivalently, for some unknown facts $X, Y, Z, W, \ldots$, can the following scenarios all be true?

- X is true, or Y is false, or Z is true.
- X is false, Z is false, or W is true.
- $\ldots$

This is known as the satisfiability problem for propositional logic, or SAT.

A "simple" algorithm

```
1. for every possible assignment
2.    check whether the assignment satisfies the formula.
```

If the formula has n propositional variables, then we need to go through

$$2^n$$

many assignments, i.e., exponential many assignments.

The exponential running time

In real-life applications, $n = 100$ is minuscule. Even if a computer can go through 1000000 assignments per second, to finish all 2^{100} assignments, we need

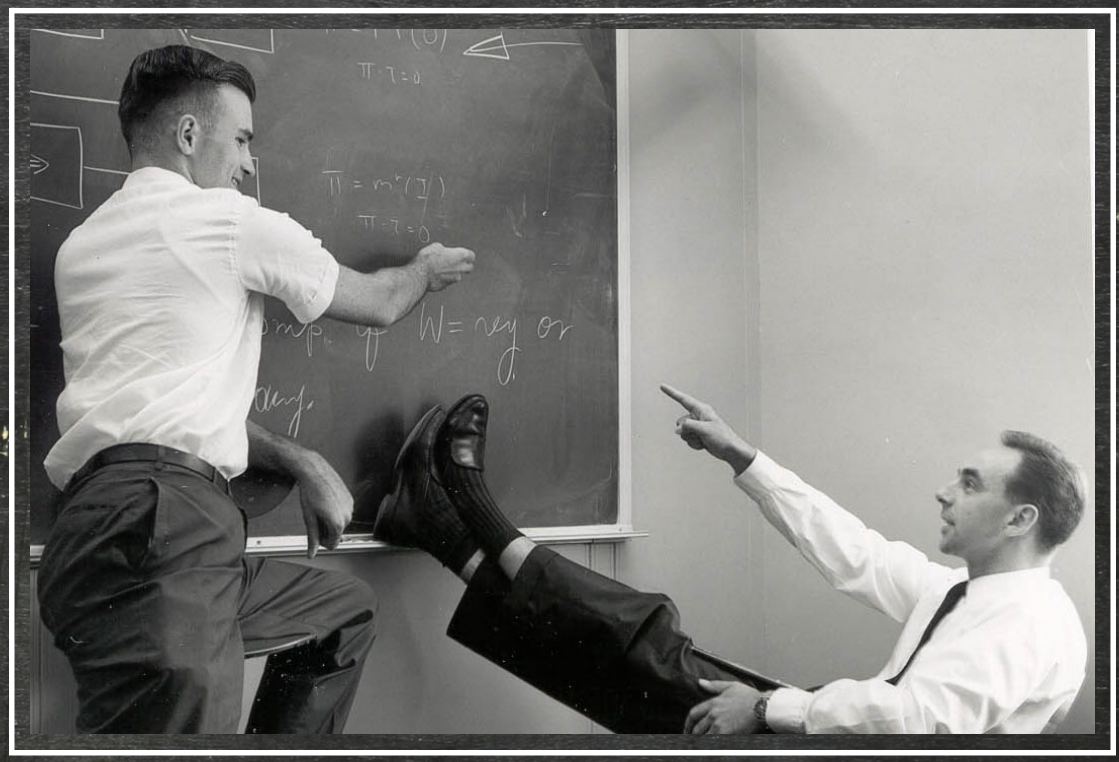
$$\begin{aligned} 2^{100}/1000000 &\approx 10^{24} \text{ seconds} \\ &\approx \frac{10^{24}}{31536000} \text{ years} \\ &\approx 3.17 \times 10^{16} \text{ years.} \end{aligned}$$

The Big Bang happened roughly

$$1.4 \times 10^{10}$$

years ago.

The birth of complexity theory



Richard E. Stearns Juris Hartmanis

On the computational complexity of algorithms
J. Hartmanis and R. E. Stearns, 1965.

We measure the computational complexity of a problem by the minimum resources needed by a Turing machine/algorithm to solve it. Typical resources include

1. time
2. space
3. parallelism
4. ...

Can we solve SAT in time

$$1000 \cdot n^2?$$

For $n = 100$

$$1000 \cdot 100^2/1000000 = 10 \text{ seconds.}$$

The computational complexity class P

Definition

A problem $Q \subseteq \Sigma^*$ is in polynomial time if there is an algorithm $\mathbb{A}$ and a polynomial $p(n)$ such that for every $x \in \Sigma^*$,

- if $x \in Q$, then $\mathbb{A}$ will halt in at most $p(|x|)$ steps and output Yes,
- if $x \notin Q$, then $\mathbb{A}$ will halt in at most $p(|x|)$ steps and output No.

P denotes the class of all polynomial time solvable problems.

After more than half a century, we still do not know whether SAT is in polynomial time, i.e., can be solved in time, e.g.,

$$1000 \cdot n^2 \text{ or even } 10^{10} \cdot n^{10^{10}}.$$

The core of computer science is not developed as fast as you might have heard otherwise.

What makes SAT special?

Given an assignment, it is easy to check (i.e., there is an efficient algorithm) whether the assignment satisfies a formula.

The motto:

the solution is hard to find but easy to verify.

Sudoku

1		6			2	3		
	5				6		9	1
		9	5		1	4	6	2
	3	7	9		5			
5	8	1		2	7	9		
			4		8	1	5	7
			2	6		5	4	
		4	1	5		6		9
9			8	7	4	2	1	

1	4	6	7	9	2	3	8	5
2	5	8	3	4	6	7	9	1
3	7	9	5	8	1	4	6	2
4	3	7	9	1	5	8	2	6
5	8	1	6	2	7	9	3	4
6	9	2	4	3	8	1	5	7
7	1	3	2	6	9	5	4	8
8	2	4	1	5	3	6	7	9
9	6	5	8	7	4	2	1	3

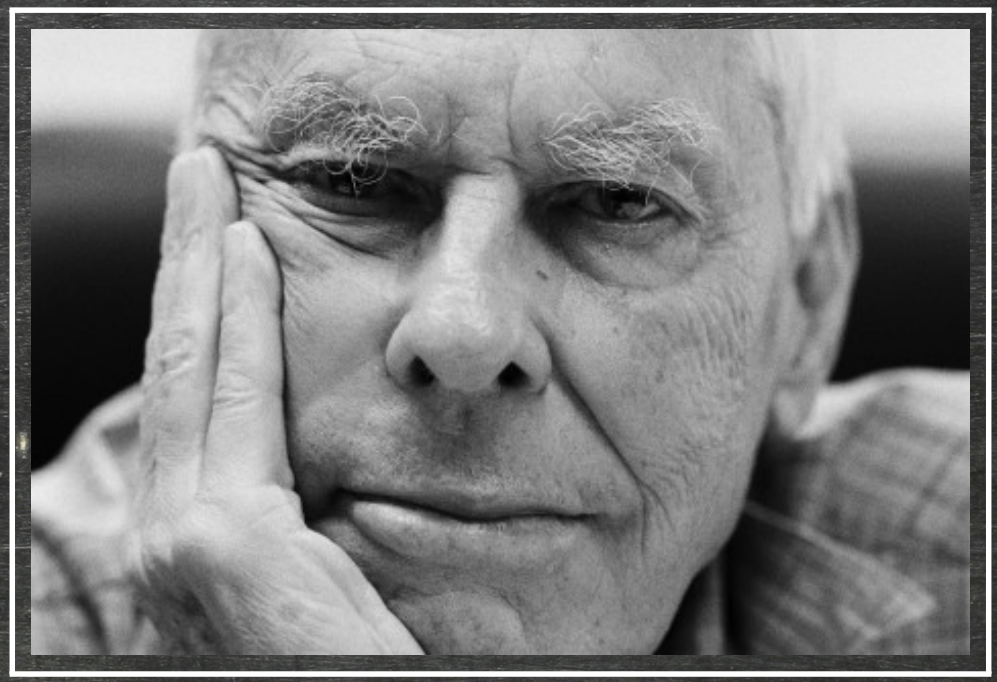
The computational complexity class NP

Definition

A problem $Q \subseteq \Sigma^*$ is in nondeterministic polynomial time if there is an algorithm $\mathbb{A}$, and two polynomials $p(n)$, $q(n)$ such that for every $x \in \Sigma^*$,

- if $x \in Q$, then there is a $y \in \Sigma^*$ of length $q(|x|)$ such that $\mathbb{A}$ will halt on (x, y) in at most $p(|x|)$ steps and output Yes,
- if $x \notin Q$, then for all $y \in \Sigma^*$ of length $q(|x|)$ the algorithm $\mathbb{A}$ will halt in at most $p(|x|)$ steps and output No.

NP denotes the class of problems in nondeterministic polynomial time.



Stephen Smale

P versus NP, a gift to mathematics from computer science.

The ubiquity of NP

- Mathematics: primality and factoring, solving systems of equations over $\mathbb{N}$, ...
- Physics: many problems in statistical physics, e.g., Ising models, ...
- Chemistry: autocatalysis, ...
- Biology: many problems associated with DNA sequencing, ...
- Economics: Nash Equilibrium, game theory, ...

It is one of the seven millennium problems in mathematics.

Cook-Levin Theorem

Stephen Cook

Lenoid Levin

Theorem (S. Cook, 1972; L. Levin, 1973)

SAT *problem is solvable in polynomial time if and only if every problem in* NP *can be solved in polynomial time.*

SAT *is* NP*-complete.*

Computer Science, or more precisely theoretical computer science has much of its root in logic.

Starting from 1980's, Algorithms and Complexity, the US synonym of theoretical computer science, are almost dominated by results, tools, and methods from algebra, combinatorics, and probability.

It is amazing, however, how different computer science is, especially theoretical computer science, in Europe and the US.

Logic activities in Europe
Y. Gurevich, 1994.

I suppose the "amicable separation" is now official.

On establishing the special interest group for logic of ACM
S. Arora, 2015.

There are always exceptions . . .

1. Understanding complexity classes by logic – finite model theory.
2. Designing algorithms by logic – algorithmic meta-theorems.
3. Understanding logic problems by complexity – Gödel's proof predicate complexity and Incompleteness by complexity.

Finite Model Theory

In Model Theory, one dominant type of questions is

which problems can be defined in a given logic.

Theorem

*The class of all 3-colorable graphs cannot be defined in first-order logic (*FO*).*

Nevertheless a graph $\mathcal{G}$ can be3-colored if and only if

$$\mathcal{G} \models \exists X_1 \exists X_2 \exists X_3 \left(\forall u \bigvee_{1 \leq i \leq 3} X_i u \wedge \forall u \bigwedge_{1 \leq i < j \leq 3} \neg(X_i u \wedge X_j u) \right.$$
$$\left. \wedge \forall u \forall v \Big(Euv \rightarrow \bigwedge_{1 \leq i \leq 3} \neg(X_i u \wedge X_i v) \Big) \right).$$

Fagin's Theorem

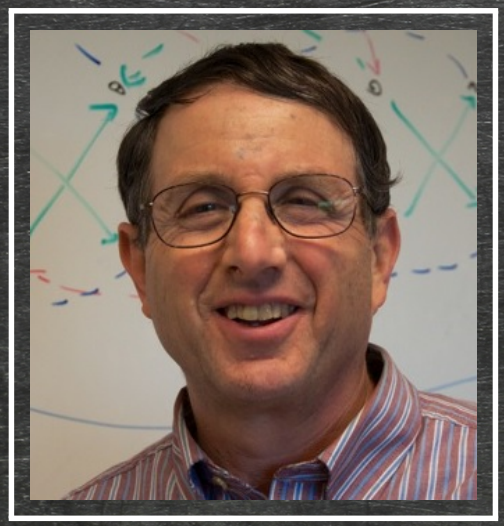
Theorem (Fagin, 1974)

A problem is in NP *if and only if it can be defined in existential second-order logic.*

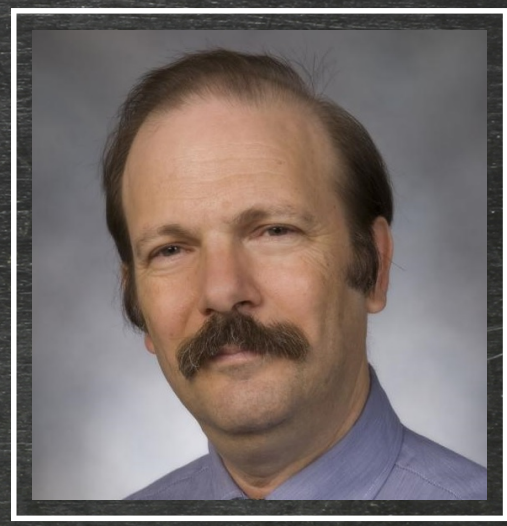
Ronald Fagin

This provides a super-clean machine-independent characterization of NP.

Immerman-Vardi Theorem



Neil Immerman



Moshe Vardi

Theorem (N. Immerman, 1982; M. Vardi, 1982)

A problem is in P *if and only if it can be defined in least fixed-point logic.*

A model-theoretic formulation of P vs. NP

Corollary

P $\neq$ NP *if and only if the existential second-order logic and fixed-point logic have different expressive power.*

Now we can attack P vs. NP problem using model theory. So far it hasn't panned out, but has found numerous applications in database theory and formal verification.

Algorithmic Meta-Theorems

There is an efficient algorithm checking whether a graph with a certain structural property *satisfies a sentence in a certain* logic.

Many NP-hard problems can be solved in polynomial time on trees, i.e., Independent-Set, Dominating-Set, 3-Colorability, etc.

These phenomena can be mostly explained by Büchi's Theorem.

Julius Richard Büchi (1924 – 1984)

Monadic second-order logic

Monadic second-order logic (MSO) is the restriction of second-order logic in which every second-order variable is a set variable.

A graph $\mathcal{G}$ can be 3-colored if and only if

$$\mathcal{G} \models \exists X_1 \exists X_2 \exists X_3 \left(\forall u \bigvee_{1 \le i \le 3} X_i u \wedge \forall u \bigwedge_{1 \le i < j \le 3} \neg(X_i u \wedge X_j u) \right.$$
$$\left. \wedge \forall u \forall v \left(Euv \rightarrow \bigwedge_{1 \le i \le 3} \neg(X_i u \wedge X_i v) \right) \right).$$

MSO can also characterize Sat, Connectivity, etc. And by adding some weak form of counting into MSO we can do Independent-Set, Dominating-Set, etc.

Büchi's Theorem

By automata-theoretic approach:

Theorem (Büchi, 1960)

For every $\varphi \in$ MSO there is a linear time algorithm that decides whether a tree satisfies φ.

Trees are too restrictive ....

Neil Robertson

Paul Seymour

In their graph minor project, Robertson and Seymour introduced the notion of tree-width to measure how a given graph is similar to a tree.

Tree-width

graph	tree-width
tree	1
forest	1
cycle	2

Courcelle's Theorem

Bruno Courcelle

Courcelle's Theorem

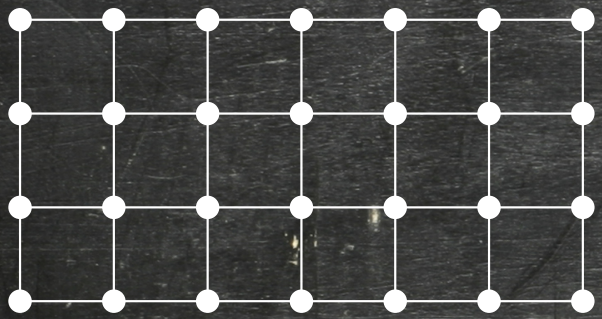
Theorem (Courcelle, 1990)

Let $w \in \mathbb{N}$ and $\varphi \in$ MSO. Then there is a linear time algorithm that decides whether a graph of tree-width $\leq w$ satisfies φ.

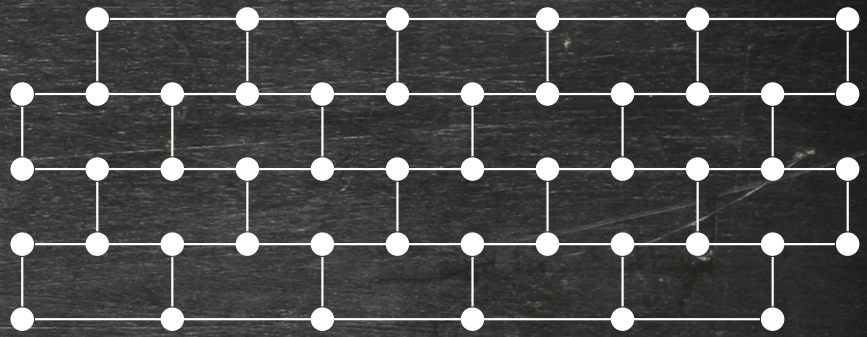
Corollary

Independent-Set, Dominating-Set, 3-Colorability, *etc, all can be solved in linear time on graphs of bounded tree-width.*

Graphs with unbounded tree-width



A (7×4)-grid



A (5×4)-wall

Planar graphs, graphs of bounded degree, ...

What else can we hope for?

First-order logic (FO) instead of MSO.

A graph $\mathcal{G}$ has a k-independent set if and only if

$$\mathcal{G} \models \exists x_1 \ldots \exists x_k \left(\bigwedge_{1 \leq i < j \leq k} (x_i \neq x_j \wedge \neg E x_i x_j) \right).$$

Frick and Grohe's Theorem

Theorem (Frick and Grohe, 2001)

Checking FO *properties on graphs of bounded local tree-width can be done in almost linear time.*

Local tree-width measures the tree-width of all the neighbourhoods of any vertex in a given graph. Both planar graphs and graphs of bounded degree have bounded local tree-width.

Corollary

For any fixed $k \in \mathbb{N}$, on planar graphs, detecting the existence of a k-independent set can be done in linear time.

Graph minors

Theorem (Kuratowski, 1930; Wagner, 1937)

A graph is planar if and only if it excludes K_5 and $K_{3,3}$ as minors.

A crown jewel of graph theory, or mathematics at large:

Theorem (Robertson and Seymour, 1983 – 2004)

Any class of graphs closing under taking minors has a finite number of excluding minors.

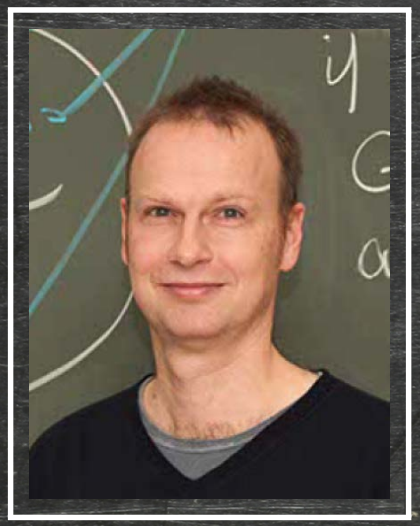
Flum and Grohe's Theorem

Theorem (Flum and Grohe, 2001)

Checking FO *properties on graphs excluding a minor can be done in almost linear time.*

[Frick and Grohe, 2001] and [Flum and Grohe, 2001] are not comparable, as witnessed by graphs of bounded degree (bounded local tree-width but not excluding a minor) and apex graphs (excluding a minor but not unbounded local tree-width).

After a long series of papers . . .



Martin Grohe

Stephan Kreutzer

Sebastian Siebertz

Theorem (Grohe, Kreutzer, and Siebertz, 2017)

Checking FO *properties on nowhere dense graphs can be done in almost linear time.*

Under some reasonable conditions, the result is optimal.

The computation model we use

A family of Boolean circuits $(C_n)_{n\in\mathbb{N}}$ are $\underline{\mathsf{AC}^0\text{-circuits}}$ if for every $n \in \mathbb{N}$

(i) C_n computes a Boolean function from $\{0,1\}^n$ to $\{0,1\}$;

(ii) the depth of C_n is bounded by a fixed constant;

(iii) the size of C_n is polynomially bounded in n.

AC^0 and parallel computation

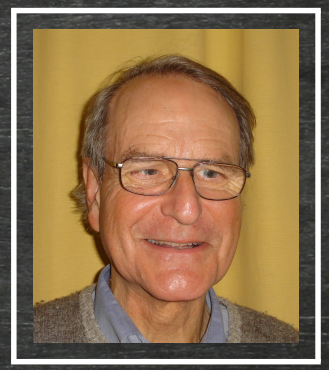
AC^0 circuits	parallel computation
# of input gates	length of input
depth	# of parallel computation steps
size	# of parallel processes

An algorithmic meta-theorem for AC^0

Theorem (*C.* and Flum, 2018)

Let $d \in \mathbb{N}$. Then checking FO *properties on graphs of tree-depth at most d can be done by AC^0-circuits of depth $O(d)$ and size $n^{O(d)}$.*

C.



Jörg Flum

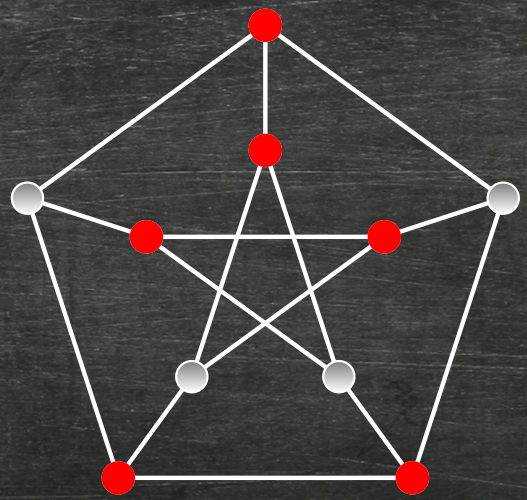
The k-vertex-cover problem

Definition

Let G be a graph and $k \in \mathbb{N}$. Then a subset $C \subseteq V(G)$ is a k-vertex-cover if

(i) for every edge $\{u, v\} \in V(G)$ either $u \in C$ or $v \in C$,

(ii) and $|C| = k$.



The peterson graph. The peterson graph with a 6-vertex-cover.

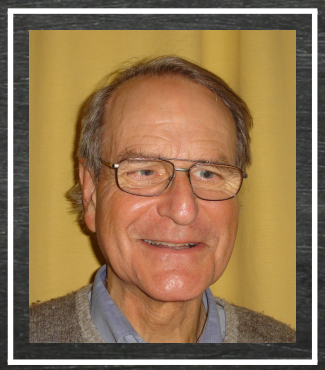
The k-vertex-cover problem by AC^0

Theorem (*C.* , Flum, and Huang, 2017)

For any $k \in \mathbb{N}$ the k-vertex cover problem can be solved by AC^0-circuits of depth 34.

That is, the k-vertex cover problem can be solved by a parallel computer in 34 steps.

C.



Jörg Flum

Xuangui Huang

Gödel's Proof Predicate

A letter from Gödel to von Neumann in 1956

One can obviously easily construct a Turing machine, which for every formula F in first order predicate logic and every natural number n, allows one to decide if there is a proof of F of length n ...

Let $\Psi(F, n)$ be the number of steps the machine requires for this and let

$$\varphi(n) = \max_F \Psi(F, n).$$

The question is how fast $\varphi(n)$ grows for an optimal machine ...

If there really were a machine with $\varphi(n) \sim k \cdot n$ (or even $\sim k \cdot n^2$), ..., it would obviously mean that in spite of the undecidability of the Entscheidungsproblem, the mental work of a mathematician concerning Yes-or-No questions could be completely replaced by a machine.

Gödel's Proof Predicate

GÖDEL	
Input:	An FO-sentence φ and $n \in \mathbb{N}$ in unary.
Problem:	Is there a proof of φ of length at most n?

Theorem (Folklore)

GÖDEL *is* NP*-hard.*

Hard sentences

By the undecidability of the Entscheidungsproblem, there are true FO-sentences φ whose shortest proofs have length at least

$$2^{|\varphi|}, 2^{2^{|\varphi|}}, \ldots$$

E.g., the Four Color Theorem, Fermat's Last Theorem, and possibly the Riemann hypothesis, P $\neq$ NP, ...

Hard-Gödel	
Input:	An FO-sentence φ and $n \geq 2^{2^{\lvert\varphi\rvert}}$ in unary.
Problem:	Is there a proof of φ of length at most n?

Theorem (Buhrman and Hitchcock, 2008)

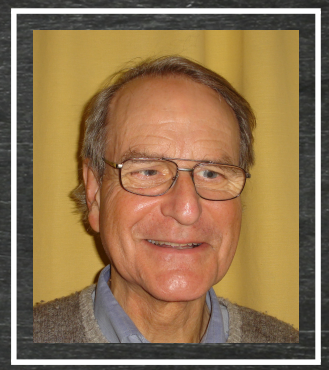
Hard-Gödel *is not* NP*-hard under some plausible complexity assumption.*

The complexity of Hard-Gödel

Theorem (*C.* and Flum, 2010)

Under some plausible complexity assumption, Hard-Gödel *is not solvable in polynomial time.*

C.



Jörg Flum

The construction problem

Constr-Gödel	
Input:	An FO-sentence φ and $n \in \mathbb{N}$ in unary.
Problem:	Construct a proof of φ of length at most n, if it exists.

Theorem (Folklore)

If Gödel *can be solved in polynomial time, then* Constr-Gödel *can be solved in polynomial time as well.*

Lemma (Folklore)

There is a polynomial time algorithm $\mathbb{A}$ *solving* Constr-Gödel *using* Gödel *as an oracle.*

The caveat:

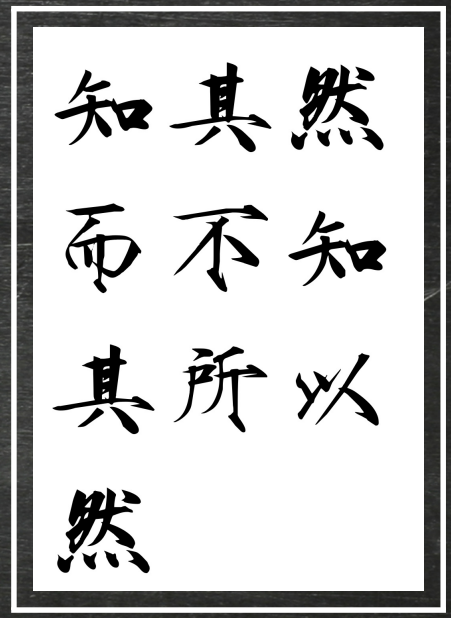
On input (φ, n) the algorithm $\mathbb{A}$ will ask n many oracle queries.

But in reality:

when proving a hard φ, we only prove a small number of key lemmas.

Theorem (*C.* and Flum, 2010)

There is a no polynomial time algorithm $\mathbb{A}$ *solving* Constr-Gödel *which only asks* Gödel *a small number of times.*



Incompleteness by Complexity

Goedel's Incompleteness Theorems

Theorem (First Incompleteness, 1931)

For any reasonable formal system Γ *including basic arithmetic, there is an* FO*-sentence* φ *such that neither* φ *nor its negation can be proved in* Γ*.*

Intuitively, we can never design an algorithm which can automatically proves all true arithmetic sentences.

Goedel's Incompleteness Theorems

Theorem (Second Incompleteness, 1931)

For any reasonable formal system Γ *including basic arithmetic, we can write down a sentence* $\mathrm{cons}(\Gamma)$ *to express that* Γ *is consistent (i.e., not self-contradictory), however* Γ *cannot prove this sentence* $\mathrm{cons}(\Gamma)$.

Intuitively, mathematics, once formalized, cannot guarantee its own correctness.

Levin's almost forgotten algorithm

КРАТКИЕ СООБЩЕНИЯ

УДК 519.14

УНИВЕРСАЛЬНЫЕ ЗАДАЧИ ПЕРЕБОРА

Л. А. Левин

В статье рассматривается несколько известных массовых задач «переборного типа» и доказывается, что эти задачи можно решать лишь за такое время, за которое можно решать вообще любые задачи указанного типа.

Universal sequential search problems
L. Levin, 1973.

1. Cook-Levin Theorem: the NP-completeness of SAT.
2. Every algorithm computing a function has an optimal inverter.

Levin's optimal inverters

Let $F : \Sigma^* \to \Sigma^*$ be a function computed by an algorithm $\mathbb{F}$.

Definition

An inverter of F is an algorithm that for every y in the image of F computes an x with $F(x) = y$.

Theorem (Levin, 1973)

There is an optimal inverter $\mathbb{I}_{\mathrm{opt}}$ for F (with respect to $\mathbb{F}$). That is, for every inverter $\mathbb{I}$ and every y in the image of F, we have

$$t_{\mathbb{I}_{\mathrm{opt}}}(y) \leq O\big(t_{\mathbb{I}}(y) + t_{\mathbb{F}}(\mathbb{I}(y))\big)^2,$$

Intuitively, $\mathbb{I}$ is as fast as any other optimal inverter.

A preordering on algorithms

Let $Q \subseteq \Sigma^*$ be a decidable problem.

Definition

Let $\mathbb{A}$ and $\mathbb{B}$ be two algorithms for Q. We say $\mathbb{A}$ is as fast as $\mathbb{B}$ on yes instances, written $\mathbb{A} \leq \mathbb{B}$, if for every $x \in Q$

$$t_{\mathbb{A}}(x) \leq (t_{\mathbb{B}}(x) + |x|)^{O(1)}.$$

Almost optimal algorithms and Stockmeyer's Theorem

Definition

An algorithm $\mathbb{A}$ for Q is almost optimal if for every algorithm $\mathbb{B}$ for Q we have $\mathbb{A} \leq \mathbb{B}$.

Theorem (Stockmeyer, 1974)

Every EXP*-hard problem Q has no almost optimal algorithm. Moreover, for every algorithm $\mathbb{A}$ for Q we can compute another algorithm $\mathbb{B}$ for Q with $\mathbb{A} \not\leq \mathbb{B}$.*

Provable algorithms

We fix:

(1) a true and effective theory T, e.g., ZFC;

(2) an effective enumeration of all algorithms $\mathbb{A}_1, \mathbb{A}_2, \ldots$.

Therefore we can formalize by first-order logic

$$\mathbb{A}_i \text{ decides } Q,$$

and talk about that a string π is a proof for

$$\pi : T \vdash \mathbb{A}_i \text{ decides } Q.$$

The function F_T

Consider the function F_T defined by

$$F_T(i, \pi, x, b) = x$$

if

1. $\pi : T \vdash \mathbb{A}_i$ decides Q;
2. if $\mathbb{A}_i$ accepts x, then $b = 1$;
3. if $\mathbb{A}_i$ rejects x, then $b = 0$.

Otherwise, let $F_T(y) = \perp$.

$F_T(i, \pi, x, b)$ can be computed by an algorithm $\mathbb{F}_T$ in time

$$f(i) \cdot |\pi| \cdot t_{\mathbb{A}_i}(x)$$

for a computable function $f : \mathbb{N} \to \mathbb{N}$.

Inverters for F_T and T-provable algorithms

Let $\mathbb{I}$ be an inverter for F_T. Then consider the algorithm $\mathbb{A}_{\mathbb{I}}$:

1. `simulate` $\mathbb{I}$ `on input` x, `say the output is` (i, π, x, b)
2. `if` $b = 1$ `then accept else reject.`

Then $\mathbb{A}_{\mathbb{I}}$ decides Q, since T is a true theory. And, $t_{\mathbb{A}_{\mathbb{I}}}(x) = O\big(t_{\mathbb{I}}(x)\big)$.

Let $\mathbb{A}_i$ be an algorithm with $\pi : T \vdash \mathbb{A}_i$ decides Q. Then the straightforward algorithm $\mathbb{I}_i$ computing

$$x \mapsto \begin{cases} (i, \pi, x, 1), & \text{if } \mathbb{A}_i \text{ accepts } x, \\ (i, \pi, x, 0), & \text{if } \mathbb{A}_i \text{ rejects } x. \end{cases}$$

inverts F_T. Moreover, $t_{\mathbb{I}_i}(x) = O(t_{\mathbb{A}_i}(x))$.

Recall Levin's optimality ...

There is an optimal inverter $\mathbb{I}_{\mathrm{opt}}$ for F_T such that for every inverter $\mathbb{I}$ and every x in the image of F_T we have

$$t_{\mathbb{I}_{\mathrm{opt}}}(x) \leq O\big(t_{\mathbb{I}}(x) + t_{\mathbb{F}_T}(\mathbb{I}(x))\big)^2.$$

Now assume $\pi : T \vdash \mathbb{A}_i$ decides Q. We put all the pieces together:

1. $\mathbb{A}_{\mathbb{I}_{\mathrm{opt}}}$ decides Q, and $t_{\mathbb{A}_{\mathbb{I}_{\mathrm{opt}}}}(x) = O(t_{\mathbb{I}_{\mathrm{opt}}}(x))$.
2. $\mathbb{I}_i(x) = (i, \pi, x, b)$ with $t_{\mathbb{I}_i}(x) = O(t_{\mathbb{A}_i}(x))$.
3. $t_{\mathbb{F}_T}(\mathbb{I}_i(x)) = t_{\mathbb{F}_T}(i, \pi, x, b) \leq f(i) \cdot |\pi| \cdot t_{\mathbb{A}_i}(x) = O(t_{\mathbb{A}_i}(x))$.

Thus

$$t_{\mathbb{A}_{\mathbb{I}_{\mathrm{opt}}}}(x) \leq \big(|x| + t_{\mathbb{A}_i}(x)\big)^{O(1)}.$$

That is, $\mathbb{A}_{\mathbb{I}_{\mathrm{opt}}} \leq \mathbb{A}_i$.

A complexity-theoretic proof of the First Incompleteness

Theorem (Gödel, 1931)

For every sufficiently strong, effective and consistent theory T there exists a true sentence φ such that $T \nvdash \varphi$.

Proof. (*C.* , Flum, and Müller, 2011).

Recall Stockmeyer's Theorem:

Every EXP-hard problem Q has no almost optimal algorithm. Moreover, for every algorithm $\mathbb{A}$ for Q we can compute another algorithm $\mathbb{B}$ for Q with $\mathbb{A} \not\leq \mathbb{B}$.

Let Q be EXP-hard, and $\mathbb{A}_{\mathbb{I}_{\text{opt}}}$ decides Q. By the previous discussion, we have an algorithm $\mathbb{A}_i$ for Q with $\mathbb{A}_{\mathbb{I}_{\text{opt}}} \not\leq \mathbb{A}_i$.

Then

$$T \nvdash \mathbb{A}_i \text{ decides } Q.$$

□

Compared to the classical proof . . .

Our diagonalization is "outside logic" and hidden in Levin's Theorem and Stockmeyer's Theorem.

A complexity-theoretic proof of the Second Incompleteness

Theorem (Gödel, 1931)

Every sufficiently strong, effective and consistent theory T cannot prove its own consistency, i.e., $T \nvdash \mathrm{cons}(T)$.

The key step of a complexity-theoretic proof:

Theorem (*C.* , Flum, and Müller, 2011)

Let T be a sufficiently strong and effective true theory. Then for every theory $T^* \supseteq T$

$$T^* \textit{ proves that } \mathbb{A}_{\mathbb{I}_{\mathrm{opt}}} \textit{ decides } Q \iff T^* \vdash \mathrm{cons}(T).$$

Conclusions

- Logic is the root of modern computer science, although its role has been diminishing in the last few decades.
- Without any doubt, complexity theory is a central piece for our understanding of computation, particularly the P vs. NP problem. Its importance is far beyond computer science.
- Combined with other mathematical tools and methods, I believe that logic will continue to contribute to our understanding of computation.

Thank You!