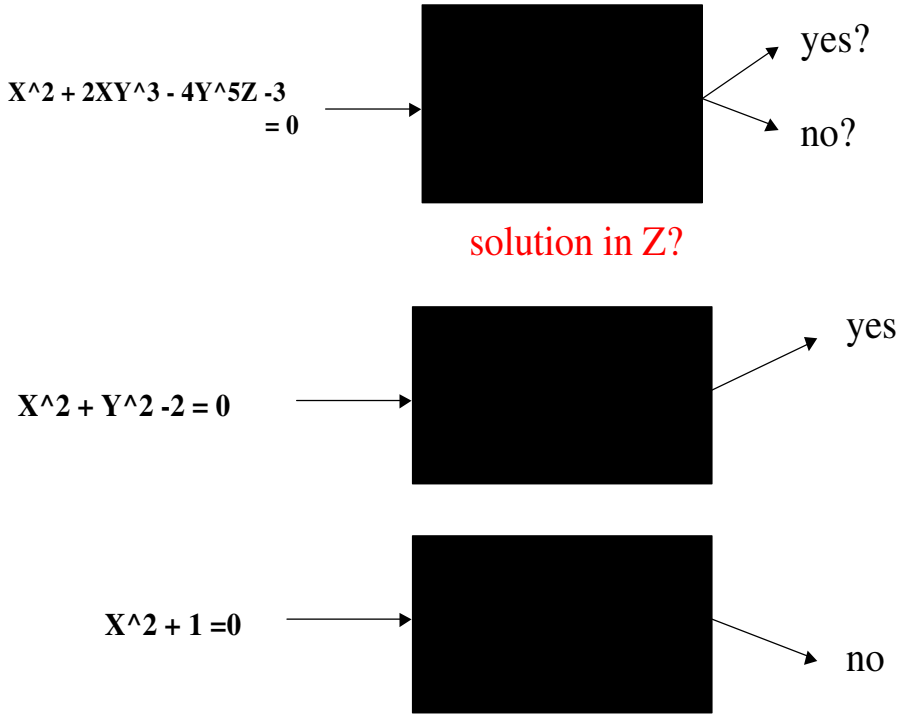
Carleton
UNIVERSITY

From Hilbert to Turing and beyond ...

Leopoldo Bertossi

Carleton University
School of Computer Science
Ottawa, Canada

An Old Problem Revisited



Example: Hilbert's 10th problem (1900): Is there a general mechanical procedure to decide if an arbitrary diophantine equation has integer roots?

David Hilbert (1862 - 1943)

It was an open problem to Hilbert's time

Mathematische Probleme.

Vortrag, gehalten auf dem internationalen Mathematiker-Kongreſs zu Paris 1900.

Von D. Hilbert in Göttingen.

Aus den Nachrichten der K. Gesellschaft der Wissenschaften zu Göttingen. Math.-phys. Klasse. 1900. Heft 3. Mit Zusätzen des Verfassers.

Wer von uns würde nicht gern den Schleier lüften, unter dem die Zukunft verborgen liegt, um einen Blick zu werfen auf die bevorstehenden Fortschritte unserer Wissenschaft und in die Geheimnisse ihrer Entwickelung während der künftigen Jahrhunderte! Welche besonderen Ziele werden es sein, denen die führenden mathematischen Geister der kommenden Geschlechter nachstreben? Welche neuen Methoden und neuen Thatsachen werden die neuen Jahrhunderte entdecken — auf dem weiten und reichen Felde mathematischen Denkens?

Die Geschichte lehrt die Stetigkeit der Entwickelung der Wissenschaft. Wir wissen, daſs jedes Zeitalter eigene Probleme hat, die das kommende Zeitalter löst oder als unfruchtbar zur Seite schiebt und durch neue Probleme ersetzt. Wollen wir eine Vorstellung gewinnen von der mutmaſslichen Entwickelung mathematischen Wissens in der nächsten Zukunft, so müssen wir die offenen Fragen vor unserem Geiste passieren lassen und die Probleme überschauen, welche die gegenwärtige Wissenschaft stellt, und deren Lösung wir von der Zukunft erwarten. Zu einer solchen Musterung der Probleme scheint mir der heutige Tag, der an der Jahrhundertwende liegt, wohl geeignet; denn die groſsen Zeitabschnitte fordern uns nicht bloſs auf zu Rückblicken in die Vergangenheit, sondern sie lenken unsere Gedanken auch auf das unbekannte Bevorstehende.

Die hohe Bedeutung bestimmter Probleme für den Fortschritt der mathematischen Wissenschaft im allgemeinen und die wichtige Rolle, die sie bei der Arbeit des einzelnen Forschers spielen, ist unleugbar. Solange ein Wissenszweig Überfluſs an Problemen bietet, ist er lebenskräftig; Mangel an Problemen bedeutet Absterben oder Aufhören der

9. Beweis des allgemeinsten Reziprozitätsgesetzes im beliebigen Zahlkörper.

Für einen beliebigen Zahlkörper soll das Reziprozitätsgesetz der $l$ten Potenzreste bewiesen werden, wenn l eine ungerade Primzahl bedeutet, und ferner, wenn l eine Potenz von 2 oder eine Potenz einer ungeraden Primzahl ist. Die Aufstellung des Gesetzes, sowie die wesentlichen Hilfsmittel zum Beweise desselben werden sich, wie ich glaube, ergeben, wenn man die von mir entwickelte Theorie des Körpers der $l$ten Einheitswurzeln[1]) und meine Theorie[2]) des relativ-quadratischen Körpers in gehöriger Weise verallgemeinert.

10. Entscheidung der Lösbarkeit einer diophantischen Gleichung.

Eine diophantische Gleichung mit irgend welchen Unbekannten und mit ganzen rationalen Zahlenkoeffizienten sei vorgelegt: *man soll ein Verfahren angeben, nach welchem sich mittelst einer endlichen Anzahl von Operationen entscheiden läßt, ob die Gleichung in ganzen rationalen Zahlen lösbar ist.*

11. Quadratische Formen mit beliebigen algebraischen Zahlenkoeffizienten.

Unsere jetzige Kenntnis der Theorie der quadratischen Zahlkörper[3]) setzt uns in den Stand, *die Theorie der quadratischen Formen mit beliebig vielen Variabeln und beliebigen algebraischen Zahlenkoeffizienten erfolgreich in Angriff zu nehmen.* Damit gelangen wir insbesondere zu der interessanten Aufgabe, eine vorgelegte quadratische Gleichung beliebig vieler Variabeln mit algebraischen Zahlenkoeffizienten in solchen ganzen oder gebrochenen Zahlen zu lösen, die in dem durch die Koeffizienten bestimmten algebraischen Rationalitätsbereiche gelegen sind.

---

1) Bericht der Deutschen Mathematiker-Vereinigung über die Theorie der algebraischen Zahlkörper **4**, 1897. Fünfter Teil.

2) Mathematische Annalen **51** und Nachrichten der K. Ges. d. Wiss. zu Göttingen 1898. Vgl. ferner die demnächst erscheinende Inauguraldissertation von G. Rückle Göttingen 1901.

3) Hilbert: Über den Dirichletschen biquadratischen Zahlenkörper, Mathematische Annalen **45**; Über die Theorie der relativ-quadratischen Zahlkörper, Berichte der Deutschen Mathematiker-Vereinigung 1897 und Mathematische Annalen **51**; Über die Theorie der relativ-Abel'schen Körper, Nachrichten der K. Ges. d. Wiss. zu Göttingen 1898; Grundlagen der Geometrie, Festschrift zur Enthüllung des Gauſs-Weber-Denkmals in Göttingen, Leipzig 1899, Kapitel VIII § 83.

Hilbert does not use the word "algorithm", but procedure (Verfahren) with the usual requirements

He does use explicitly "decision" (Entscheidung)

There had been "algorithms" since long before, e.g. for the gcd

Even decision algorithms, e.g. deciding if a number is prime

However, the term comes from the name of the Persian scientist and mathematician Abdullah Muhammad bin Musa al-Khwarizmi (9th century)

In the 12th century one of his books was translated into Latin, where his name was rendered in Latin as "Algorithmi"

His treatise "al-Kitab al-mukhtasar fi hisab al-jabr wal-muqabala" was translated into Latin in the 12th century as "Algebra et Almucabal" (the origin of the term algebra)

And Even More Hilbert ...

In the context of his far reaching program on foundations of mathematics, Hilbert identifies the following as crucial:

Das Entscheidungsproblem: (The Decision Problem)

Determine if the set of all the formulas of first-order logic that are universally valid (tautological) is decidable or not

$\forall x(P(x) \rightarrow P(c))$ and $(P(c) \rightarrow \exists x P(x))$ are valid (always true)

$\forall x \exists y(Q(x) \wedge P(x, y))$ is not (it is false in some structures)

Explicitly formulated in the 20's in the the context of the "engere Funktionenkalkül" (the narrower function calculus)

Hilbert, D. and Ackermann, W. "Grundzüge der Theoretischen Logik", Springer, 1928.

DIE GRUNDLEHREN DER MATHEMATISCHEN
WISSENSCHAFTEN IN EINZELDARSTELLUNGEN
BAND XXVII

D. HILBERT UND W. ACKERMANN

GRUNDZÜGE DER THEORETISCHEN LOGIK

VERLAG VON JULIUS SPRINGER IN BERLIN

Durch Umbenennung der Variablen erhält man aus den beiden Voraussetzungen:

1. $(Ew)\Phi(u, v, w)$.
2. $(\Phi(x, u, y) \,\&\, \Phi(u, v, w) \,\&\, \Phi(y, v, z)) \rightarrow \Phi(x, w, z)$.

Wendet man auf die zweite Voraussetzung die Regel VII, S. 28 an, so kann man sie umformen zu:

$$\Phi(u, v, w) \rightarrow [(\Phi(x, u, y) \,\&\, \Phi(y, v, z)) \rightarrow \Phi(x, w, z)].$$

Unter Benutzung der zu Formel (34) gehörigen Regel kann man daraus ableiten:

$$(Ew)\Phi(u, v, w) \rightarrow (Ew)[(\Phi(x, u, y) \,\&\, \Phi(y, v, z)) \rightarrow \Phi(x, w, z)].$$

Da nun $(Ew)\Phi(u, v, w)$ als richtig angenommen wurde, so ergibt sich weiter

$$(Ew)(\Phi(x, u, y) \,\&\, \Phi(y, v, z) \rightarrow \Phi(x, w, z)).$$

Daraus ergibt sich die Behauptung, indem man nach Regel γ' die Allzeichen (u) und (v) vorsetzt.

§ 11. Das Entscheidungsproblem im Funktionenkalkül und seine Bedeutung.

Nach der durch die letzten Beispiele gekennzeichneten Methode kann man den Funktionenkalkül insbesondere zur axiomatischen Behandlung von Theorien verwenden. Für diesen Zweck ist der Kalkül sehr geeignet. Infolge der streng formalen Behandlungsweise wird nämlich verhütet, daß bei der Ableitung aus den Axiomen versteckte Voraussetzungen mitbenutzt werden.

Die mathematische Logik leistet aber noch mehr als eine Verschärfung der Sprache durch die symbolische Darstellung der Schlußweisen. Nachdem einmal der logische Formalismus feststeht, kann man erwarten, daß eine systematische, sozusagen rechnerische Behandlung der logischen Formeln möglich ist, die etwa der Theorie der Gleichungen in der Algebra entsprechen würde.

Eine ausgebildete „Algebra der Logik" begegnete uns im Aussagenkalkül (man vgl. insbesondere § 4 — 9 des I. Kapitels). Die wichtigsten der dort erwähnten und gelösten Probleme waren das der *Allgemeingültigkeit* und der *Erfüllbarkeit* eines logischen Ausdrucks. Beide Probleme zusammen pflegt man auch kurz als das *Entscheidungsproblem* zu bezeichnen.

Bei dem Problem der *Allgemeingültigkeit* handelt es sich um die folgende Frage: *Wie kann man bei einem beliebigen vorgelegten logischen Ausdruck, der keine individuellen Zeichen enthält, feststellen, ob der Aus-*

druck bei beliebigen Einsetzungen für die vorkommenden Variablen eine richtige Behauptung darstellt oder nicht?

Bei dem Problem der *Erfüllbarkeit* handelt es sich um die Frage, *ob es überhaupt eine Einsetzung für die Variablen gibt, so daß durch den betreffenden Ausdruck eine richtige Behauptung dargestellt wird.*

Beide Probleme sind zueinander dual. Ist ein Ausdruck nicht allgemeingültig, so ist das Gegenteil erfüllbar und umgekehrt.

Das Entscheidungsproblem läßt sich nun auch für den Funktionenkalkül aufwerfen. Zu den Aussagenvariablen treten hier die Funktionsvariablen hinzu. Die Individuenvariablen wollen wir uns hier immer durch vorgesetzte Klammerzeichen gebunden denken. Man kann sich übrigens auf den Fall beschränken, daß die Aussagenvariablen fehlen. Diese lassen sich nämlich nach den in § 8 des I. Kapitels gemachten Bemerkungen immer eliminieren.

Ein Beispiel für eine allgemeingültige Formel des Funktionenkalküls ist die folgende Formel:

$$(x)(F(x) \vee F(x)).$$

Diese ist richtig, welches Prädikat auch für F eingesetzt wird. Die Formel

$$(Ex)F(x)$$

ist zwar nicht allgemeingültig, aber erfüllbar. Wir brauchen hier ja nur das Prädikat „mit sich selber identisch sein" zu nehmen. Dieses trifft nicht nur auf einen, sondern sogar auf alle Gegenstände zu. Daraus ergibt sich, daß auch

$$(x)F(x)$$

einen erfüllbaren Ausdruck darstellt. Weitere Beispiele für allgemeingültige Formeln sind

$$(x)(Ey)(R(x, x) \vee \bar{R}(x, y)),$$

$$(Ex)(y)(R_1(x, x) \vee R_1(y, y) \vee R_2(x, y))$$

sowie alle Formeln, die sich aus den logischen Axiomen ableiten lassen, z. B. unsere früheren Formeln (21)—(36). Nicht allgemeingültig ist dagegen

$$(x)F(x, x),$$

nicht erfüllbar

$$(Ex)(y)(F(x, x) \& F(x, y)).$$

Das Entscheidungsproblem ist gelöst, wenn man ein Verfahren kennt, das bei einem vorgelegten logischen Ausdruck durch endlich viele Operationen die Entscheidung über die Allgemeingültigkeit bzw. Erfüllbarkeit erlaubt.

Die Lösung des Entscheidungsproblems ist für die Theorie aller Gebiete, deren Sätze überhaupt einer logischen Entwickelbarkeit aus endlich

keine allgemeingültige logische Formel ist. Ähnliche Überlegungen gelten natürlich für jedes beliebige Axiomensystem. Die fundamentale Bedeutung, die das Entscheidungsproblem besitzt, dürfte damit genügend illustriert sein; *das Entscheidungsproblem muß als das Hauptproblem der mathematischen Logik bezeichnet werden.*

§ 12. Lösungen des Entscheidungsproblems für besondere Spezialfälle.

Während im Aussagenkalkül das Entscheidungsproblem unschwer zu lösen war, bildet im Funktionenkalkül die Auffindung eines allgemeinen Entscheidungsverfahrens ein noch ungelöstes schwieriges Problem. Für gewisse einfache Fälle ist es jedoch gelungen, ein derartiges Verfahren anzugeben. Der einfachste Fall, der sich hier darbietet, ist der folgende:

Es sollen in den logischen Ausdrücken nur Funktionsvariable mit einem Argument, also Prädikate im engeren Sinne, vorkommen; die mehrgliedrigen Prädikate werden also ausgeschlossen. Die grundsätzliche Möglichkeit der Entscheidung in diesem Bereich ist zuerst von *L. Löwenheim* erkannt worden[1]. Ein übersichtliches Entscheidungsverfahren ist von *H. Behmann* gegeben worden[2]. Löwenheim und Behmann nehmen übrigens zu den variablen Prädikaten noch die Identität als individuelle Relation hinzu.

Der Bereich der zugelassenen Formeln entspricht dem Bereiche, der durch Kombination des Aussagen- und Klassenkalküls entsteht, also dem Bereiche der Kantisch-Aristotelischen Logik.

Wir können uns auf elementarem Wege von der Entscheidbarkeit in dem angegebenen Bereiche überzeugen.

Es liege eine Formel aus diesem Bereich vor. Es sei k die Anzahl der verschiedenen in der betrachteten Formel vorkommenden Funktionszeichen $A, B \ldots K$. Wir behaupten nun: *Wenn die Formel in allen Fällen, wo der Individuenbereich aus höchstens 2^k Gegenständen besteht, immer richtige Aussagen (durch Einsetzung bestimmter Prädikate an Stelle von $A, B, \ldots K$) liefert, so liefert sie überhaupt immer richtige Aussagen.*

Zum Beweise nehmen wir an, daß die betrachtete Formel für ein gewisses System von mehr als 2^k Individuen bei Ersetzung der Funktionszeichen $A, B, \ldots K$ durch die bestimmten Prädikate A_0, $B_0, \ldots K_0$ eine falsche Aussage ergäbe. Wir wollen dann aus dieser Aussage eine andere falsche Aussage ableiten, die ebenfalls durch

---

[1] *L. Löwenheim:* Über Möglichkeiten im Relativkalkül. Math. Ann. Bd. 76.

[2] *H. Behmann:* Beiträge zur Algebra der Logik und zum Entscheidungsproblem. Math. Ann. Bd. 86.

sion concerning universal validity by the above methods. For all other prefixes we can actually find formulas which are universally valid in any finite domain of individuals, but not in infinite domains.[1]

Results by A. Church based on papers by K. Gödel show that the quest for a general solution of the decision problem must be regarded as hopeless.[2] We cannot report on these researches in detail within the limits of this book. We shall only remark that a general method of decision would consist of a certain recursive procedure for the individual formulas which would finally yield for each formula the value truth or the value falsehood. Church's work proves, however, the non-existence of such a recursive procedure; at least, the necessary recursions would not fall under the general type of recursion set up by Church, who has given to the somewhat vague intuitive concept of recursion a certain precise formalization.

To avoid misunderstanding, it should be noted that the impossibility of a general decision procedure does not mean that we can find definite formulas whose universal validity has been proved not to be decidable. To assume the existence of such a proof would in fact lead to an immediate contradiction. From such a proof it would follow that the formula would not be deducible from the axiom system of § 5. But by the completeness theorem of § 10, the satisfiability of the contradictory of the formula could then be proved. Thus the universal validity would be decided after all, viz. in the negative. In any case, therefore, the task of extending the class of formulas for which the decision is solved, remains rewarding and significant.

---

[1] Cf. the first paper by K. Schütte cited in footnote 2, p. 123.

[2] A. Church, *An unsolvable problem of elementary number theory*. Am. J. of Math. Vol. 58 (1936),—*A note on the Entscheidungsproblem; Correction to a note on the Entscheidungsproblem*, J. Symb. Logic. Vol. 1 (1936).

From translation of 2nd German edition, Springer, 1938

Tackling the Problems

To give a positive answer, it was good enough to exhibit the decision algorithm (and prove that it works)

To give a negative answer, quite a different approach was necessary

How could it be possible to show that there was no such algorithm?

It's a universal quantification over the class of all algorithms: none of them should work ...

It was impossible to settle the problem in the negative without a mathematical characterization or formalization of the notion of algorithm (or the class of algorithms)

With that mathematical definition we would know what class of objects (the algorithms) we are talking about

To Hilbert's time there was no mathematical formalization of algorithm yet

To Hilbert's time there was only an intuitive notion:

- A specified discrete process that follows a finite and fixed set of rules
- Follows the same steps with the same input (determinism)
- ...

Hilbert himself talked about a "mechanical procedure"

So, what is an algorithm?

A first attempt could be as follows:

An algorithm is what can be programmed in (computed, calculated, ... with) a model X of computation ...

What model X of computation?

It was necessary to wait for the introduction and development of the first mathematical model (actually, models) of computation

This happened in the mid 30's

A model that could be accepted by the scientific community as a good formalization of the intuitive notion of a computing device (and indirectly of algorithm)

This acceptance criterion is not something that can be settled in mathematical terms

What is a good model (of anything)?

It can be judged by its (internal) mathematical correctness, the intuitions that it captures, its results, e.g. predictions, etc.

We will have to come back to this ...

In between, Hilbert's problems were still waiting ...

Turing Machines

A Turing machine (TM) is a computational model introduced by Alan Turing around 1936

The idea behind was to capture the so far only intuitive notions of: computational device, computational procedure, algorithm, computable function, etc.

One can safely say that computer science has it origins in the work by Turing (and by others around the same time and on the same subject)

The inception of the new discipline was started and motivated by the work and problems of mathematical logicians

In particular, Hilbert's Entscheidungsproblem was a crucial motivation

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

Reprinted with the kind permission of the London Mathematical Society from the Proceedings of the London Mathematical Society, ser. 2, vol. 42 (1936-7), pp. 230-265; corrections, Ibid, vol 43 (1937) pp. 544-546.

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.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers π, e, etc. The computable numbers do not, however, include all definable numbers, and an example is given of a definable number which is not computable.

Although the class of computable numbers is so great, and in many ways similar to the class of real numbers, it is nevertheless enumerable. In § 8 I examine certain arguments which would seem to prove the contrary. By the correct application of one of these arguments, conclusions are reached which are superficially similar to those of Gödel†. These results

† Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwant der Systeme, I", *Monatshefte Math. Phys.*, 38 (1931), 173–198.[a]

have valuable applications. In particular, it is shown (§ 11) that the Hilbertian Entscheidungsproblem can have no solution.

In a recent paper Alonzo Church† has introduced an idea of "effective calculability", which is equivalent to my "computability", but is very differently defined. Church also reaches similar conclusions about the Entscheidungsproblem‡. The proof of equivalence between "computability" and "effective calculability" is outlined in an appendix to the present paper.

1. *Computing machines.*

We have said that the computable numbers are those whose decimals are calculable by finite means. This requires rather more explicit definition. No real attempt will be made to justify the definitions given until we reach § 9. For the present I shall only say that the justification lies in the fact that the human memory is necessarily limited.

We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions $q_1, q_2, \ldots, q_R$ which will be called "m-configurations". The machine is supplied with a "tape" (the analogue of paper) running through it, and divided into sections (called "squares") each capable of bearing a "symbol". At any moment there is just one square, say the r-th, bearing the symbol $\mathfrak{S}(r)$ which is "in the machine". We may call this square the "scanned square". The symbol on the scanned square may be called the "scanned symbol". The "scanned symbol" is the only one of which the machine is, so to speak, "directly aware". However, by altering its m-configuration the machine can effectively remember some of the symbols which it has "seen" (scanned) previously. The possible behaviour of the machine at any moment is determined by the m-configuration q_n and the scanned symbol $\mathfrak{S}(r)$. This pair $q_n, \mathfrak{S}(r)$ will be called the "configuration": thus the configuration determines the possible behaviour of the machine. In some of the configurations in which the scanned square is blank (*i.e.* bears no symbol) the machine writes down a new symbol on the scanned square: in other configurations it erases the scanned symbol. The machine may also change the square which is being scanned, but only by shifting it one place to right or left. In addition to any of these operations the m-configuration may be changed. Some of the symbols written down

† Alonzo Church, "An unsolvable problem of elementary number theory", *American J. of Math.*, 58 (1936), 345–363.[a]

‡ Alonzo Church, "A note on the Entscheidungsproblem", *J. of Symbolic Logic*, 1 (1936), 40–41.[a]

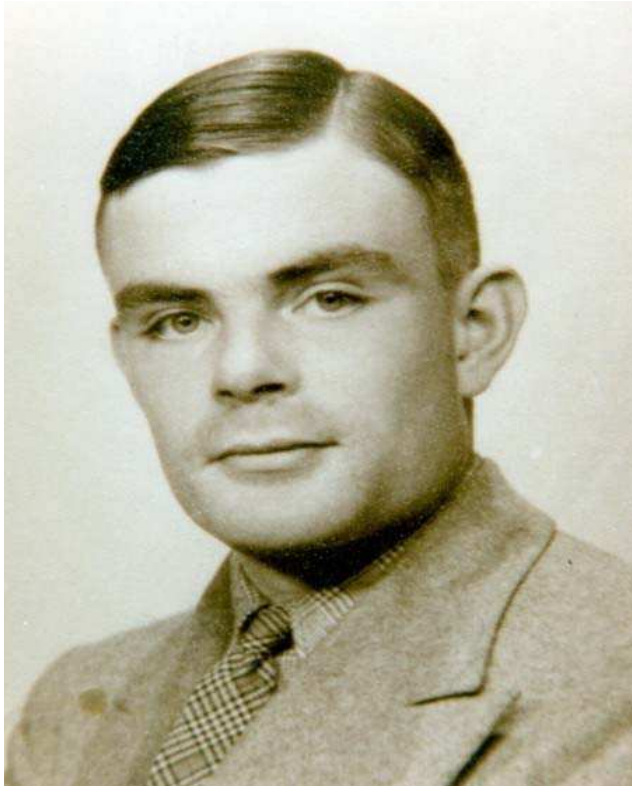
Turing is famous for (among other achievements):

- Having proposed a mathematical model of a computer, of algorithm, etc.
- Applying the model to the study of solvability of certain problems using computers (and algorithms)
- Establishing the first results about the unsolvability of certain problems by means of computers

  And then, establishing limits on what computers can do ...
- Introducing the idea of a universal machine that can simulate any other machine

  This idea was crucial for the later development of machines that could store programs (by John von Neumann and others)

- The Turing's Test in AI
- Breaking the cryptographic communication code of the German navy during the 2nd World War

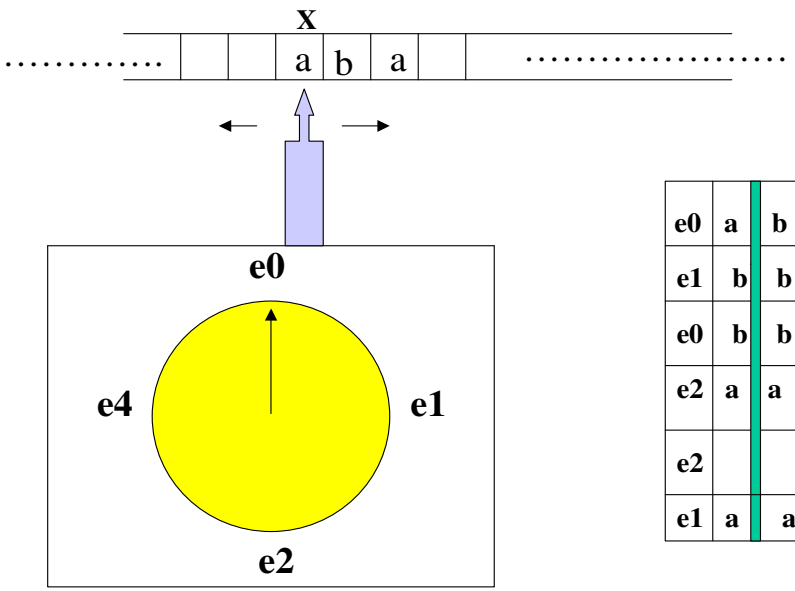


Alan Turing (1912-1954)

The most important award in computer science research is named after him: the ACM Turing Award

(ACM: Association for Computing Machinery)

A Turing Machine M:



e0	a	b	+1	e1
e1	b	b	+1	e2
e0	b	b	+1	e1
e2	a	a	+1	e2
e2			+1	e2
e1	a	a	+1	e4

software

It has:

- A control unit consisting of a finite set of states
- A finite alphabet Σ
- An infinite tape in two directions with cells

  We can think the cells as numbered: $\ldots, -2, -1, 0, 1, 2, \ldots$
- A reading/writing head that moves along the tape

- A transition function δ_M, the "software"

  It can be represented as a finite table

  The function may be partial, i.e. δ_M may not be always defined

  If M reaches a configuration (e, a) for which δ_M is undefined, M stops (halts)

Notice that a TM is a mathematical object: a finite number of finite set-theoretic objects

TM can be used to compute functions, e.g. the usual arithmetical functions

And many other functions on natural numbers ...

E.g. Is there a TM that computes the factorial function?

$F : \mathbb{N} \to \mathbb{N}$, that is usually defined by recursion:

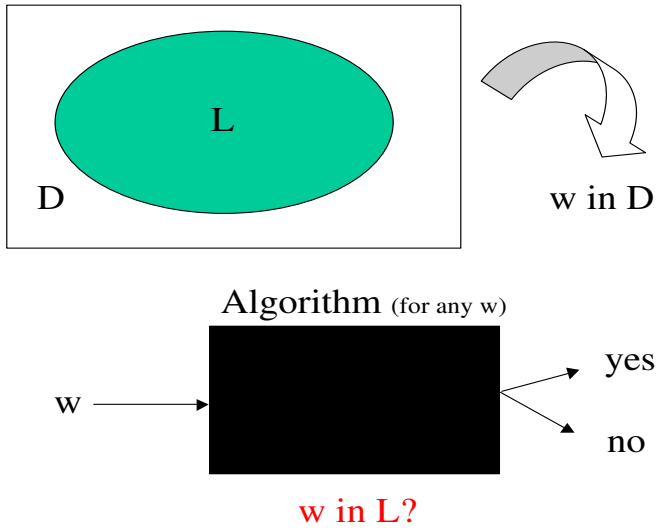
$$\begin{aligned} F(0) &:= 1 \\ F(n+1) &:= F(n) \times (n+1) \end{aligned}$$

TMs can be used to solve decision problems

For solving decision problems with a TM, one usually assumes that there are states q_Y, q_N for $Yes!, No!$, resp.

At them, the TM stops and the answer is read from the corresponding state

Decision Problems



Start from a domain of instances D and a subset L

Is there an algorithm to decide membership to L?

More precisely, we search for a general algorithm that takes inputs $w \in D$, and answers $Yes!$ or $No!$ depending on whether w belongs to L or not

Usually the problem is characterized in terms of finite words over a finite alphabet Σ: $L \subsetneq D \subseteq \Sigma^\star$

We are confronted to decision problems, and we look for algorithms to solve them

Given a decision problem, with $L \subseteq D$:

L is decidable (or solvable) if there is an algorithm AL that for any $w \in D$, AL answers $Yes!$ if $w \in L$ and $No!$ if $w \notin L$

There are many natural and common decision problems

Example: D is the set of all propositional formulas in "conjunctive normal form" (CNF) over a certain set of propositional variables, say $p_1, p_2, p_3, \ldots$

The formulas in D are conjunctions of disjunctions of "literals", i.e. propositional variables or negations thereof,

$$D = \{p_1, \neg p_2, p_3 \wedge \neg p_4, p_1 \wedge \neg p_1, (p_1 \vee \neg p_2) \wedge (p_2 \vee p_3 \vee \neg p_1), \ldots\}$$

$L := SAT$, the subset of D containing the formulas that are satisfiable

E.g. $p_1 \wedge (p_2 \vee \neg p_1)$ is satisfiable, but $p_1 \wedge (\neg p_1 \vee p_2) \wedge \neg p_2$ is not

All these formulas can be encoded (represented) as words over a finite alphabet

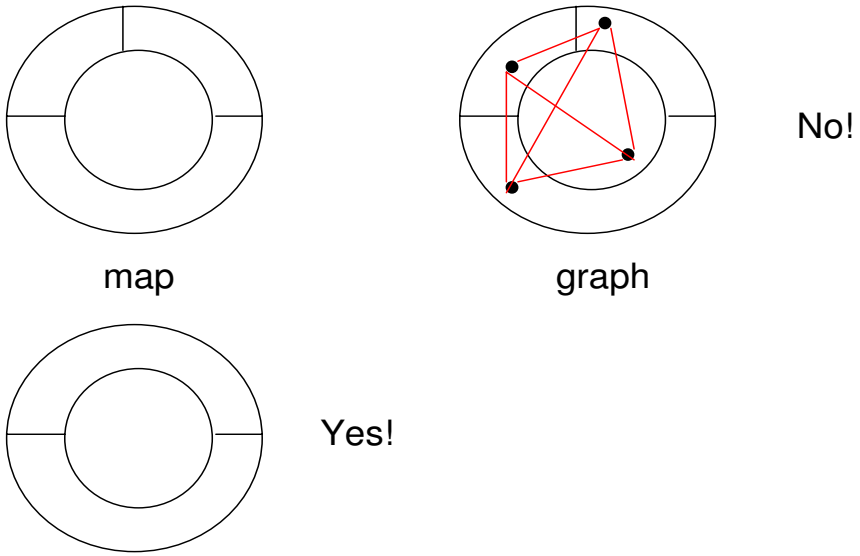
Is SAT decidable? I.e. is there a decision algorithm for it?

An algorithm to decide membership to SAT:

> Given an arbitrary formula $\varphi \in D$, construct and check the truth table of φ; if the final column has a 1 (T), answer $Yes!$, otherwise $No!$

We can say SAT is (computationally) decidable or solvable

Example: Is there an algorithm to decide if an arbitrary planar undirected (finite) graph (or map) G is colorable with 3 colors? (in such a way that vertices connected by an edge have different colors)

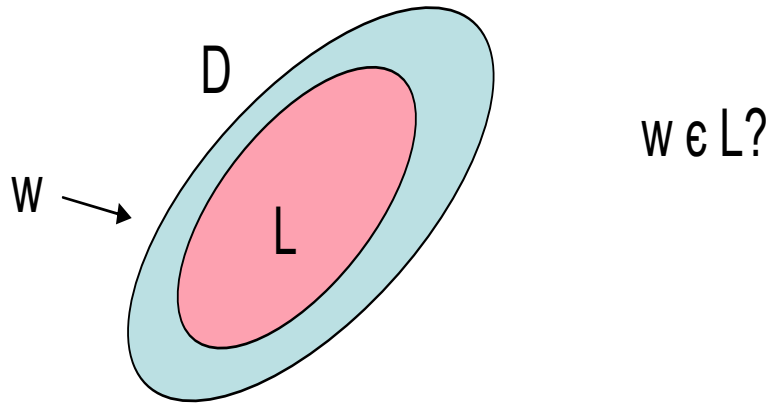


Here: D is the set of (encodings of) graphs

$3GC$ is the set of all graphs than can be colored with 3 colors

A solvable decision problem: Try the finitely many possible colorations with three colors and check if one works

Notice that a decision problem can be formulated as a problem of computing a function



Consider the characteristic function of L:

$ch_L : D \to \{0, 1\}$, defined by $ch_L(w) := \begin{cases} 1 & \text{if } w \in L \\ 0 & \text{if } w \in D \setminus L \end{cases}$

L is decidable iff the function ch_L is computable (i.e. there is an algorithm that computes it)

Algorithms

We proposed the TM as a model of computation

TMs seem to be powerful enough to compute what we consider to be computable

We haven't defined "algorithm" yet

We can make at this stage a commitment and give a precise definition

Definition: An algorithm is a procedure that can be programmed in (as) a TM (or computed with a TM)

This makes the notion of algorithm precise from the mathematical point of view

From a mathematical point of view it is O.K.

But is it a good definition?

It cannot be judged mathematically

We need a different kind of support for this definition

- It seems to capture the intuitions behind and practice of computation and use of algorithms
- People who have tried to program their favorite (intuitive) algorithms in (as) a TM have succeeded
- Alternative notions of algorithm that have been proposed have turned out to be equivalent in terms of computational power to TMs

If we accept this definition, we obtain right away the other characterizations missing:

Definition:

(a) A decision problem L ($\subseteq D$) is (computationally) solvable if there is TM that solves it

That is, the TM halts with every input from D by reaching either q_Y or q_N, but q_Y iff the input belongs to L

(b) A function $f : (\Sigma^\star)^n \rightarrow (\Sigma^\star)^m$ is computable iff it can be computed by means of a TM

We should say "Turing-decidable" and "Turing-computable"

In particular, this applies to functions of natural numbers

For example, $Sum : (\{0,1\}^\star)^2 \rightarrow \{0,1\}^\star$ is computable

Here the number are represented in, say binary notation

We accept that this is the right definition of computable function, i.e. we accept the so-called

Church's Thesis: *The class of computable functions coincides with the class of Turing-computable functions*

This thesis cannot be mathematically proved, we accept it or not (on other grounds)

It is impossible to mathematically prove a statement that says that two classes of objects coincide when one of them is mathematically defined, but not the other

("Turing-computable" has a mathematically precise meaning: computable by means of a TM)

Well, we are lying a bit ...

Church presented his own model of computation (or of computable function) a little before Turing, in a 1936 paper

Again, the Entscheidungsproblem at the very origin ...

Not everybody convinced that Church captured the essence of computable function, e.g. Goedel ...

Turing shows in his 1936/37 paper that the classes of Church-computable functions and Turing-computable functions coincide

After that Goedel was convinced!

The Turing model is such a natural model ...

Alonzo Church (1903-1995)

Today, Church's thesis is widely accepted

Church was one of those who at that time proposed a (superficially) different model of computation: the "lambda calculus" (λ-calculus), that is at the root of some modern (functional) programming languages like Lisp and Scheme

A NOTE ON THE ENTSCHEIDUNGSPROBLEM*,**

Alonzo Church

In a recent paper[1] the author has proposed a definition of the commonly used term "effectively calculable" and has shown on the basis of this definition that the general case of the Entscheidungsproblem is unsolvable in any system of symbolic logic which is adequate to a certain portion of arithmetic and is ω-consistent. The purpose of the present note is to outline an extension of this result to the engere Funktionenkalkül of Hilbert and Ackermann.[2]

In the author's cited paper it is pointed out that there can be associated recursively with every well-formed formula[3] a recursive enumeration of the formulas into which it is convertible.[3] This means the existence of a recursively defined function a of two positive integers such that, if y is the Gödel representation of a well-formed formula Y then $a(x,y)$ is the Gödel representation of the xth formula in the enumeration of the formulas into which Y is convertible.

Consider the system L of symbolic logic which arises from the engere Funktionenkalkül by adding to it: as additional undefined symbols, a symbol 1 for the number 1 (regarded as an individual), a symbol = for the propositional function = (equality of individuals), a symbol s for the arithmetic function $x+1$.

*Reprinted from the Journal of Symbolic Logic vol. 1 no. 1 (1936) and vol. 1 no. 3 (1936), by kind permission of the author, the Journal of Symbolic Logic and the Association for Symbolic Logic Inc.

**Received April 15, 1936. Correction received August 13, 1936.

1. "An unsolvable problem of elementary number theory." American Journal of Mathematics. vol. 58 (1936).[a]

2. Grundzüge der Theoretischen Logik. Berlin, 1928.

3. Definitions of the terms "well-formed formula" and "convertible" are given in the cited paper.

Now, with a precise mathematical definition of algorithm it becomes possible to prove that some (decision) problems are undecidable

That is unsolvable by algorithmic means, i.e. unsolvable by means of a TM

Are there any?

Undecidable Problems, Universal TMs

We will concentrate on $\Sigma = \{0, 1, \#\}$

With this alphabet we can represent, among other things, natural number, e.g. in binary notation

$\Sigma^\star$ is infinite, but countable: It can be put in one-to-one correspondence with the set of natural numbers

In consequence, $\mathcal{P}(\Sigma^\star)$ is uncountable

The set of languages over Σ is uncountable!

Now, let us consider TMs over alphabet Σ

How many TMs can we have?

Every machine is a finite device: it depends on a finite set of states, the finite alphabet Σ, the finite transition function, ...

So, we cannot generate (produce, describe, define) more than an infinite, but countable set of TMs

The set $\mathcal{T}$ of TMs over alphabet Σ is infinite and countable

That is, there is a bijective function $e : \mathcal{T} \rightarrow \mathbb{N}$

For a TM M, $e(M)$ is the natural number associated to M

Since $e(M)$ is a number, it has a representation in $\{0, 1\}$, e.g. the binary representation of number $e(M)$

$e(M)$ is the encoding of machine M as a binary string, i.e. $e(M) \in \{0,1\}^\star$

The existence of such a numerical encoding of TMs follows from a simple cardinality analysis

However, it is possible to give effective and explicit encodings of TMs as binary strings

We will not do this here, but just use the encoding

We have also obtained: The set of TMs is strictly smaller that the set of languages over Σ

Every decidable language over Σ requires the existence of a TM that decides it

Then, there cannot be more decidable languages than TMs:

The set of decidable languages over Σ is strictly smaller than the set of languages over Σ

Thus: There are languages over Σ that are undecidable!

A concrete undecidable problem? A first undecidable problem?

It will be related to the intimate nature of TMs

We recall that every TM M can be encoded as a binary string $e(M)$

Actually, this encoding is invertible, and it is possible (not done here) to effectively reobtain the components of M from $e(M)$

In particular, a TM could take the encoding of another TMs as input, "deconstruct" the encoding, reobtain the encoded machine, and simulate its behavior

We can go a bit further and consider also inputs for the machines that are being simulated

Consider the following algorithm U:

1. It takes inputs of the form $e(M)\#w$, with $e(M)$ the encoding of a machine M over Σ and $w \in \{0,1\}^\star$
   (at least these are the interesting inputs)
2. Obtain from $e(M)$ the components of M to ...
3. Simulate the run of machine M with input w
4. (If M stops with q_Y, return Yes!)
5. (If M stops with q_N -or whatever different from q_Y, return No!)
6. (If M does not stop, do not stop)

This is clearly an algorithm, an informal one

Appealing to Church's thesis, there is a TM T^U for this algorithm

So, T^U on inputs $e(M), w$ simulates the computation of machine M with input w

This is a universal Turing machine

The existence of such a universal machine was obtained from Church's thesis

However, it is possible to give an effective and explicit universal machine T^U (this was done by Turing himself)

This machine is obviously interesting *per se*

We will also use it later on

And the idea behind will be inspiring now ...

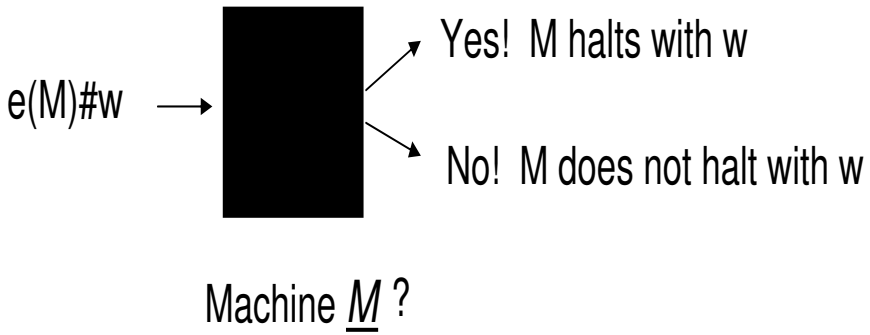
A First Undecidable Problem: The Halting Problem

Is it possible to decide if a TM with a certain input halts or not?

More precisely, is there a Turing machine $\mathcal{M}$ to decide if any TM machine M, encoded as $e(M)$, with an input $w \in \{0,1\}^\star$ halts or not?

We are considering the decision problem

$L_H = \{e(M)\#w \mid e(M)$ is the encoding of a TM M that, with input w, halts$\}$



No!: There is no such a machine $\mathcal{M}$

Theorem: L_H, the halting problem for TMs, is undecidable

The computational problem of deciding by means of computers if a Turing machine halts with a given input or not is unsolvable by means of computers

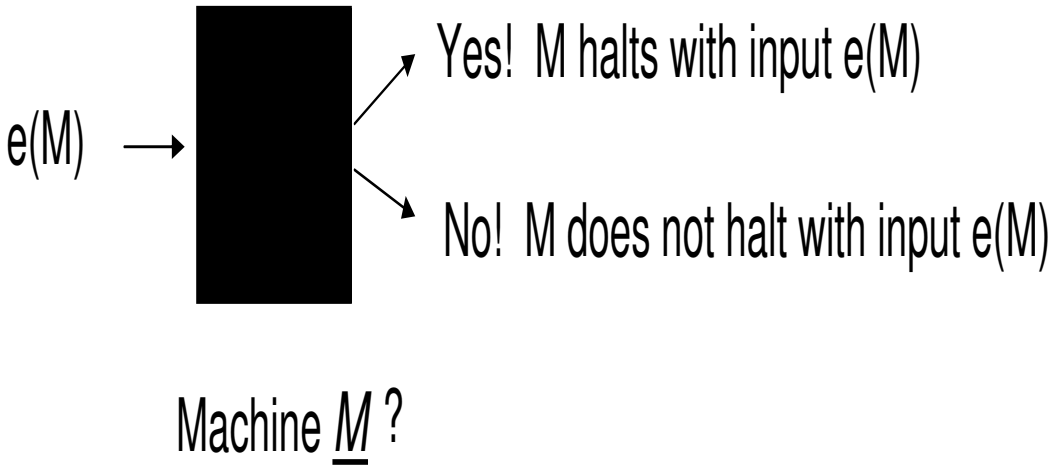
"Modern version": There is no program in Java that can be used to verify if an arbitrary program in Java stops or not)

Thus, there are provable limits on what computers can do ...

How can we prove the theorem?

A. Turing first proved that a special version of the halting problem is unsolvable:

Is there a TM $\mathcal{M}$ that decides if an arbitrary TM M fed with is own code $e(M)$ as input stops or not?



$L_{SH} := \{w \in \{0,1\}^\star \mid w$ is code $e(M)$ of a machine M and M with input $e(M)$ halts$\}$

Lemma: The "special halting problem" L_{SH} is unsolvable

This is a special case of the halting problem L_H, and is already unsolvable ...

Why this one?

It exploits the idea of self-reference or of a "diagonal construction" that is quite useful in mathematics

L_{SH} is our first unsolvable decision problem (or undecidable problem)

Proof of the lemma: Assume, by contradiction, that there is such a machine $\mathcal{M}$

For it it holds then: $\mathcal{M}(e(M))$ answers $Yes!$ iff $M(e(M))$ halts

($\mathcal{M}(w)$ denotes the computation of $\mathcal{M}$ with input w, etc.)

Now, build a new machine $\mathcal{M}'$ that uses $\mathcal{M}$ as a subroutine:

$$\mathcal{M}'(w) := \begin{cases} Yes! & \text{if } \mathcal{M}(w) \text{ says } No! \\ \infty \; loop & \text{if } \mathcal{M}(w) \text{ says } Yes! \end{cases}$$

Let us feed $\mathcal{M}'$ with its own code

$\mathcal{M}'(e(\mathcal{M}'))$ halts $\iff$ $\mathcal{M}'(e(\mathcal{M}'))$ answers $Yes!$

$\iff$ $\mathcal{M}(e(\mathcal{M}'))$ answers $No!$

$\iff$ $\mathcal{M}'(e(\mathcal{M}'))$ does not halt

A contradiction!

Now we can prove that the (general) halting problem is undecidable

It should be easy ...

Intuitively, L_H is more general than L_{SH} so it should be undecidable too

More precisely: Assume, by contradiction, that L_H is decidable

Then there is a TM $\mathcal{M}$ that decides it: $\mathcal{M}(e(M), w)$ answers $Yes!$ iff $M(w)$ halts

We can use $\mathcal{M}$ to decide L_{SH} as follows: For arbitrary TM M, just run $\mathcal{M}(e(M), e(M))$

This decides if $M(e(M))$ halts

A contradiction!

In the proof of the undecidability of L_H we used the idea of reducing a problem to another

We reduced the special halting problem to the general halting problem

We can exploit this technique to show that:

- Certain problems are decidable given that others are
- Certain problems are undecidable given that others are

Since we already have our first two undecidable problems, we can use this technique to establish that other problems are also undecidable

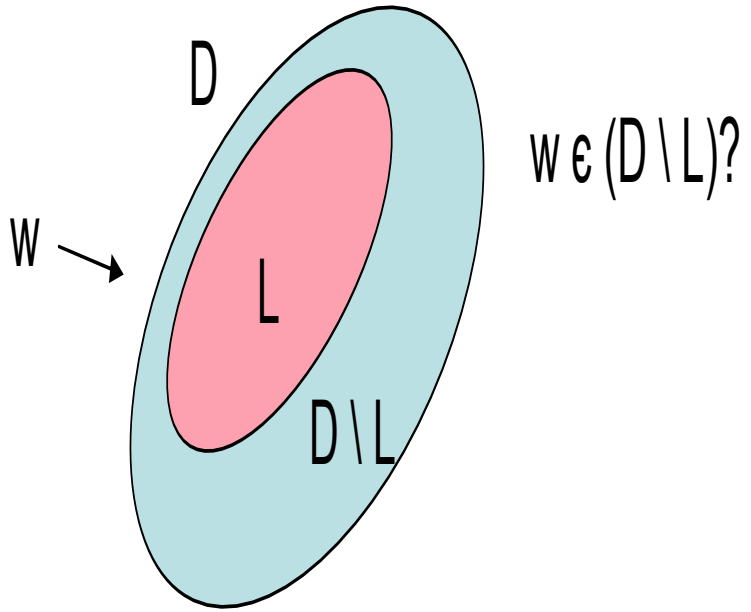
Two simple results are useful in this direction

Proposition: Let L, D be a decision problem, $L \subseteq D$. L is decidable iff its complement $D \setminus L$ is decidable

$$L \text{ is decidable} \iff (D \setminus L) \text{ is decidable}$$

By the contrapositive implication:

$$L \text{ is undecidable} \iff (D \setminus L) \text{ is undecidable}$$



Proof: ($\Rightarrow$) Use a TM M for deciding L as a subroutine for a machine M^c for deciding $D \setminus L$:

When M answers $Yes!$, M^c answer $No!$

When M answers $No!$, M^c answer $Yes!$

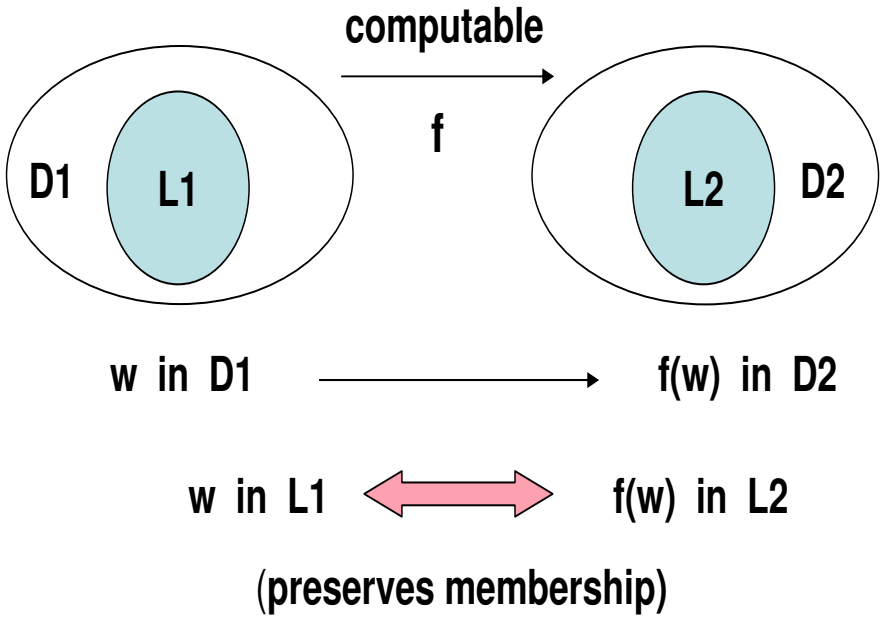
Again the idea of reduction (of $D \setminus L$ to L)

Now we make the notion of reduction precise

Definition: Let D_1, L_1 and D_2, L_2 decision problems

L_1 is reducible to L_2 if there is a function $f : D_1 \longrightarrow D_2$, such that

- f is computable (by a TM)
- For every $w \in D_1,\ w \in L_1 \Longleftrightarrow f(w) \in L_2$



Proposition: If L_1 is reducible to L_2, then

- L_2 decidable $\implies$ L_1 decidable
- L_1 undecidable $\implies$ L_2 undecidable

The second implication is just the contrapositive of the first one

Proof: If L_2 is decidable, then it has a decision algorithm AL_2

If L_1 is reducible to L_2, there is a reduction function f that can be computed by an algorithm AL_f

We use both as subroutines for an algorithm for deciding L_1:

1. Given arbitrary $w \in D_1$ $\quad$ $(w \in L_1?)$
2. Using AL_f compute $f(w)$, that belongs to D_2
3. Use AL_2 to decide if $f(w) \in L_2$
4. From the answer returned by AL_2 read the answer for w

Notation: $L_1 \leq L_2$ denotes that L_1 is reducible to L_2

Example: As expected, $L_{SH} \leq L_H$

In fact, the following function works $h: \ e(M) \mapsto e(M)\#e(M)$

- When $e(M)$ is an input for the special halting problem, $h(e(M))$ is an input for the (general) halting problem
- h is clearly computable
- $e(M) \in L_{SH} \iff h(e(M)) \in L_H$

Notice: $L_1 \leq L_2 \quad \not\Longleftrightarrow \quad L_2 \leq L_1$

We have a powerful technique to prove many undecidability results

Proposition: Deciding if two TMs (over the same alphabet) compute the same (possibly partial) function is unsolvable

$L_{sf} := \{e(M_1)\#e(M_2) \mid$ ... and for every $w, M_1(w) = M_2(w)\}$

($M(w)$: result at end of computation or ∞ if no stop)

Proof: We can try $L_H \leq L_{sf}$

That is, we want to answer if a machine stops using an answer about machines computing the same function

$f : \quad e(M)\#w \mapsto ??$

We have to come up with the right question on the RHS

It has to be of the form of an input for L_{sf}

Which M_1, M_2 above? Two machines, depending upon M, w

Machine M_1:

$$M_w(u) := \begin{cases} 0 & \text{if } M(w) \text{ halts} \\ \infty \ loop & \text{if } M(w) \text{ does not halt} \end{cases}$$

This is a constant function, with value always 0 or always undefined

The other machine, M_2, is the one that computes: $\mathbb{O}: \quad u \mapsto 0$

This is constant and always takes value 0

M_w and $\mathbb{O}$ compute the same function iff M halts with w

$f: \quad e(M)\#w \ \mapsto \ e(M_w)\#e(\mathbb{O})$

- On the RHS we have an input for L_{sf}
- The pair $e(M_w)\#e(\mathbb{O})$ can be computed from M, w
- M halts with w iff M_w and $\mathbb{O}$ compute the same function, i.e. $e(M)\#w \in L_H$ iff $e(M_w)\#e(\mathbb{O}) \in L_{sf}$

Proposition: The problem of deciding if a machine with the empty tape (as input) halts is undecidable

Proof: It holds $L_H \leq L_\epsilon := \{e(M) \mid M(\epsilon) \text{ halts}\}$

In fact, given M and w, build a new machine M_w:

1. Erases whatever is on the tape as input (if any)
2. Writes w on the tape
3. Behaves as M (with input w)

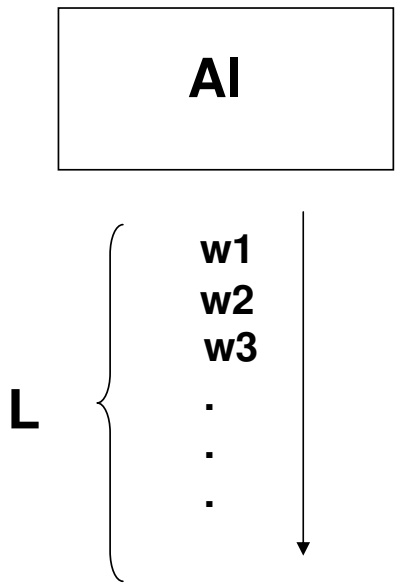
Clearly M_w (and $e(M_w)$) is computable

It holds: M halts with input w iff M_w halts with empty input

Recursively Enumerable Languages

We introduce a natural notion that is a weaker version of decidability

Intuitively, $L \subseteq \Sigma^\star$ is recursively enumerable (r.e.) if there is an algorithm (TM) that, once started, lists one-by-one all and only the elements of L, with repetitions allowed, and in no prespecified order



The condition about the order is important

If we impose an order, we could decide about membership just by looking at the order in which elements are displayed

This is a useful characterization

However, it is possible to give a more precise definition and equivalent characterization

Definition: $L \subseteq \Sigma^\star$ is r.e. iff it is the language $L(M)$ accepted by some TM M, where

$$L(M) := \{w \in \Sigma^\star \mid M \text{ with input } w \text{ halts at state } q_Y\}$$

The machine M, for inputs $w \in (\Sigma^\star \setminus L)$, may reach q_N or not halt

So, M may not be (implement) a decision algorithm for L

(Hence the difference between a language being "decided" and "accepted" by a TM: in the former case, the machine must always halt at q_Y or q_N)

However, it is obvious from the definition that

Proposition: Every decidable language is also r.e.

The definition can be made relative to a decidable domain D, with $L \subseteq D \subseteq \Sigma^\star$

Requiring that D is decidable is natural: we should be able to decide if an input is legitimate

Recursively enumerable languages are also called semi-decidable, partially decidable, Turing-recognizable, etc.

What is the connection with the initial intuition?

- If L is recursively enumerated by an algorithm

  AL: $w_1, w_2, w_3, ...,$

  the following algorithm M is a semi-decision algorithm, i.e. M accepts L:

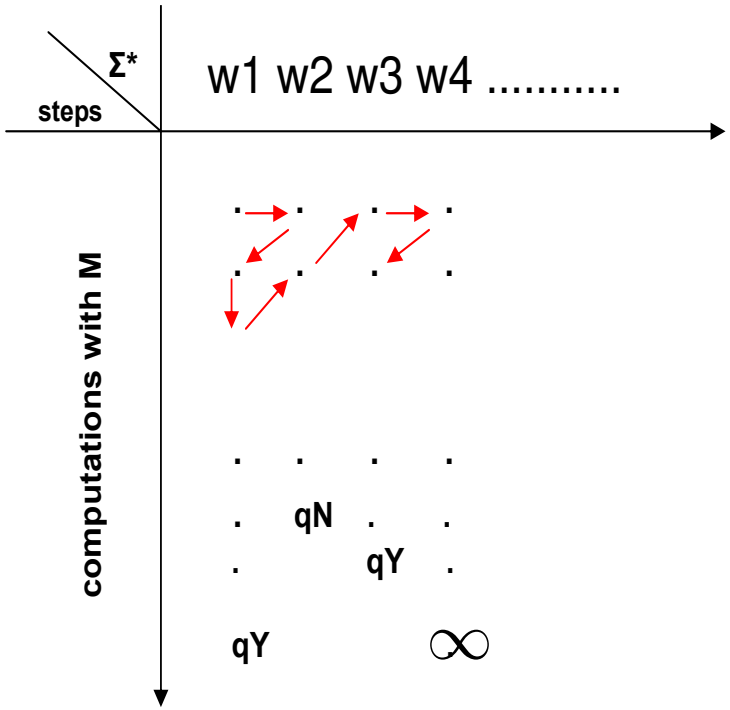
  For an arbitrary input $w \in \Sigma^\star$, turn on AL and wait to see if w appears

  If yes, answer Yes! (or go to a state q_Y)

  Every $w \in L$ eventually appears and gets answer Yes!

  No element of $(\Sigma^\star \setminus L)$ will get answer Yes!

- If $L = L(M)$ according to the definition of r.e., the following is an intuitive enumeration algorithm for L



Systematically simulate the computations with M for all the elements of $\Sigma^\star$, but not the whole computations (we might not stop with a particular one), but in parallel, step by step as in the figure

Output w whenever q_Y is reached

Some of the vertical computations may be infinite, but we will not get stuck at any of them

Only and all elements of L will be displayed

Just to become familiar with the use of the informal characterization of r.e. languages, we prove again

Proposition: Every decidable language is r.e.

Proof: An enumeration algorithm for a decidable L

Start systematically generating elements of $\Sigma^\star$, one after the other, each time one, say w, is generated, internally decide if $w \in L$

If $Yes!$, return w; otherwise (i.e. $No!$), hide w and continue with the following element ...

So, we have: L decidable $\implies$ L r.e.

However: L r.e. $\not\implies$ L decidable

That is, r.e. is a provably a weaker notion than decidability

Any counterexample?

Proposition: L_H is r.e.

Proof: Use the universal TM T^U, that, on input $e(M)\#w$ simulates the computation of machine M with input w (steps 1.-3. on page 45)

Extend steps 1.-3. with

4. T^U halts with $Yes!$ iff M halts with input w (at q_Y^M or q_N^M)
5. T^U runs forever if $M(w)$ does

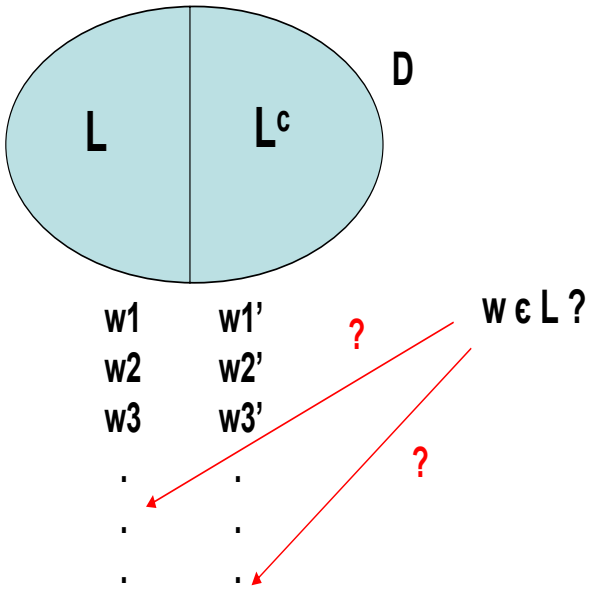
Thus, $L(T^U) = \{e(M)\#w \mid M \text{ with input } w \text{ halts}\}$

That is, L_H is the language accepted by a TM

In the following, we denote with L^c the complement of L, i.e. it is $D \setminus L$ (for a decidable $D \subseteq \Sigma^\star$)

Proposition: (S. Kleene)

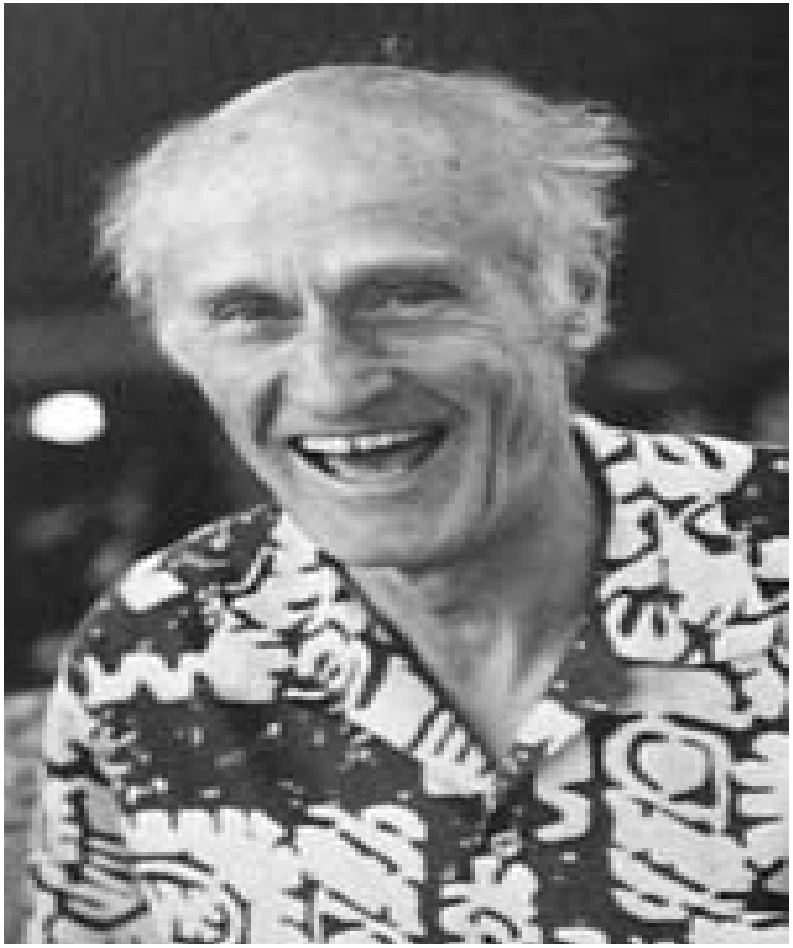
L and L^c both r.e. $\Longrightarrow$ L (and L^c) decidable



In order to decide if $w \in L$, turn on algorithms for L and L^c, and wait to see here w appears

It has to appear in one and only one of the two lists

Corollary: $L_H^c = \{e(M)\#w \mid \ldots M \text{ does not halt with } w\}$ is not r.e. (and, of course, L_H^c is also undecidable)



Stephen Kleene (1909-1994)

Kleene introduced another mathematical model of computable function; the "general recursive functions" (1936)

Recursively enumerability naturally appears in different situations

It is implied by decidability, but usually r.e. appears on its own

Example: Consider the language of propositional logic based on the set of propositional variables $P = \{p_1, p_2, \ldots\}$

For example, $(p_3 \wedge \neg p_2), \neg p_5, (p_2 \rightarrow p_6), \ldots$ are elements of this language

Let D be the domain containing all these legitimate propositional formulas

Consider the language

$TAUT := \{\varphi \in D \mid \varphi$ is a tautology, i.e. true for every truth valuation$\}$

E.g. $(p_6 \vee \neg p_6),\ (p_8 \rightarrow p_8) \in TAUT$

$TAUT$ is r.e.

This follows from the decidability of $TAUT$ (to decide, just build the truth table of the formula and check)

However, there is a natural and common algorithm for r.e.:

It is known from mathematical logic that there is a deductive calculus that can be used to (syntactically) derive all the formulas that are logical consequences of a given set of premises (other formulas)

The elements of $TAUT$ are those formulas that can be derived from the empty set of premises; i.e. they are unconditionally true

A possible deductive calculus is the following; it consists of logical axioms and one deduction rule:

A1. $\varphi \to (\psi \to \varphi)$

A2. $(\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi))$

A3. $(\neg\varphi \to \neg\psi) \to (\psi \to \varphi)$

A4. $(\varphi \wedge \psi) \to \varphi$

A5. $(\varphi \wedge \psi) \to \psi$

A6. $(\varphi \to \psi) \to ((\varphi \to \chi) \to (\varphi \to (\psi \wedge \chi)))$

A7. $\varphi \to (\varphi \vee \psi)$

A8. $\psi \to (\varphi \vee \psi)$

A9. $(\varphi \to \chi) \to ((\psi \to \chi) \to ((\varphi \vee \psi) \to \chi))$

MP. The "modus ponens" rule:

$$\frac{(\varphi \quad \to \quad \psi) \qquad \varphi}{\psi}$$

Each axiom is a pattern for a certain class of tautologies, and the "metaformulas" in the axioms can be replaced by any concrete propositional formulas

E.g. in A1. we can make $\varphi = \neg\neg p_2$, and $\psi = (p_2 \wedge \neg p_8)$:

$\neg\neg p_2 \to ((p_2 \wedge \neg p_8) \to \neg\neg p_2)$ is a (concrete, specific) tautology

A derivation of a formula (from the empty set of premises) is a finite sequence of formulas

Each of them is an instance of one of the axioms (or (meta)tautologies) or can be obtained by MP from previous formulas in the sequence

A derivation of the (meta) tautology $\varphi \to \varphi$:

1. $(\varphi \to ((\varphi \to \varphi) \to \varphi)) \to ((\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi))$ (A2. with $\varphi, (\varphi \to \varphi), \varphi$)
2. $(\varphi \to ((\varphi \to \varphi) \to \varphi))$ (A1. with $\varphi, (\varphi \to \varphi)$)
3. $(\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi)$ (MP with 1. and 2.)
4. $\varphi \to ((\varphi \to \varphi)$ (A1.)
5. $\varphi \to \varphi$ (MP. with 3. and 4.)

With this deductive calculus all the tautologies can be derived

What about the r.e. of $TAUT$?

Use "systematically" the logical axioms and the MP rule to start deriving one after the other all and only tautologies

(A similar reasoning can be applied to languages generated by context-free grammars)

This procedure captures the positive. i.e. $Yes!$, cases, but not the negative, i.e. $No!$, cases

So, it is only a semi-decision algorithm for a decidable problem

The Entscheidungsproblem Revisited

Theorem: (A. Church) The set of all valid sentences of a language $L(S)$ of first-order predicate logic (FOL) is undecidable, i.e.

$$VAL := \{\varphi \in L(S) \mid \varphi \text{ is a sentence and } \models \varphi\}$$

is undecidable!

$\models \varphi$ means $\emptyset \models \varphi$, i.e. φ is consequence of the empty set of axioms, i.e. unconditionally true, i.e. always true

(S with at least one binary predicate plus equality)

Proof: (sketch) We know that deciding if an arbitrary Turing machine with an arbitrary input stops is unsolvable (undecidable problem)

FOL is expressive enough to describe how an arbitrary Turing machines operates

In particular, with the existential quantifier it is possible to express that "there is a halting state"

More precisely, to every machine M with input w, it is possible to effectively associate a sentence $\varphi^{M,w}$ such that

$$M \text{ with input } w \text{ halts} \iff \models \varphi^{M,w}$$

This reduction shows that the problem on the RHS cannot be solvable (otherwise, the one on the LHS would be solvable)

Hilbert's 10th Problem Revisited

With a precise, mathematical definition of algorithm and decidability, it became possible to attack Hilbert's problem (in its negative direction)

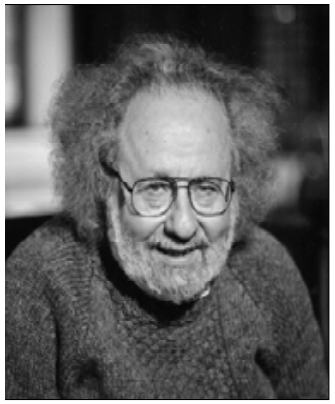
Theorem: (Y. Matiyasevich, 1970) Hilbert's 10 Problem is undecidable, i.e. there is no algorithm to decide if an arbitrary diophantine equation has roots in the set $\mathbb{Z}$ of the integer numbers

This result relies in the following main technical result

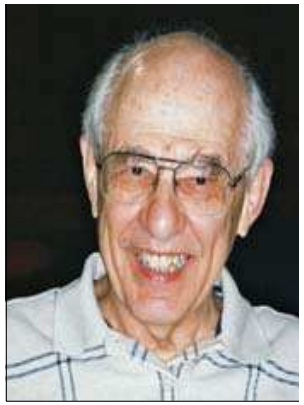
It was obtained by extending previous work and concluding the line of attack first proposed and followed by Julia Robinson, Hilary Putnam and Martin Davis

Theorem: (Matiyasevich) Every r.e. relation over natural numbers is diophantine

Julia Robinson



Martin Davis



Hilary Putnam

What is a diophantine relation over natural numbers?

Let $R \subseteq \mathbb{N}^n$, i.e. an n-ary relation over naturals

R is diophantine iff there is a polynomial $p(x_1, \ldots, x_n, y_1, \ldots, y_m)$ with coefficients in $\mathbb{Z}$, such that, for all $a_1, \ldots, a_n \in \mathbb{N}$:

$$(a_1, \ldots, a_n) \in R \quad \Leftrightarrow \quad \exists y_1 \cdots \exists y_m p(a_1, \ldots, a_n, y_1, \ldots, y_m) = 0$$

is true in $\mathbb{N}$ (also quantifiers over $\mathbb{N}$)

For example, the unary relation "being a perfect square" ($PSq \subseteq \mathbb{N}^1$) is diophantine: for all $a \in \mathbb{N}$

$$PSq(a) \Leftrightarrow \exists y(a - y^2) = 0$$

Here the polynomial is $p(x, y) : \; x - y^2$

Proving that every r.e. set (relation) of natural numbers is diophantine is the difficult part (we do not give the proof)

(Proving that every diophantine relation is r.e. is easy)

Robinson, Davis, Putnam proved this result but their "diophantine polynomials" had the exponential function (x^y), so they were not exactly polynomials, but "almost" ...

Something like $p(x, y, z) : \ x^2y - xy^z$ is not exactly a diophantine polynomial

So, it was left open how to get rid of the exponentiation, replacing it by polynomial expressions

This was achieved by Matiyasevich

The polynomial p can be effectively obtained from the r.e. relation R

That is, it can be computed (from the TM that provides the recursive enumeration algorithm for R)

That every r.e. relation can be expressed by a diophantine polynomial is initially surprising

However, the recognizing machine can be "arithmetized":

- We have seen already that the machine can be encoded as a number
- Its operation or dynamic can also be captured by numerical functions that turn out to be polynomial

Before proving that the last theorem implies the undecidability of Hilbert's problem, a useful remark:

If $HP(\mathbb{Z})$, i.e. Hilbert's problem asking for existence of roots in $\mathbb{Z}$, is solvable, then it is also solvable when asking for roots in $\mathbb{N}$

This can be obtained by reduction from $HP(\mathbb{N})$ to $HP(\mathbb{Z})$

Using the fact that every natural number is the sum of 4 squares, i.e. we use the reduction

$$p(x, \ldots) = 0 \quad \mapsto \quad p(x_1^2 + x_2^2 + x_3^2 + x_4^2, \ldots) = 0$$

In this way every negative root in $\mathbb{Z}$ for the RHS can be replaced by its positive version (the sign disappears anyway due to the square)

Thus, it is good enough to prove that $HP(\mathbb{N})$ is unsolvable

Let prove that

Every r.e. relation is diophantine $\implies$ $HP(\mathbb{N})$ unsolvable

In fact: Consider L_H, the halting problem, which is r.e.

L_H can be seen as a 2-ary relation over natural numbers, i.e. $L_H \subseteq \mathbb{N}^2$: Its inputs are binary strings, i.e. natural numbers in binary representation

Then, there is a polynomial $p^U(x, y, z_1, \ldots, z_m)$ with coefficients in $\mathbb{Z}$, such that

$$(e(M), w) \in L_H \;\Leftrightarrow\; \exists z_1 \cdots \exists z_m \, p^U(\underline{e(M)}, \underline{w}, z_1, \ldots, z_m) = 0$$

Here, $\underline{w}$ is the number represented in binary by the binary string w, ...

The quantifiers on the RHS are over $\mathbb{N}$

If it were possible to decide the RHS, it would be possible to decide membership to L_H

Then, $HP(\mathbb{N})$ is unsolvable

In this proof we used again a reduction: $L_H \leq HP(\mathbb{N})$

It was achieved via the polynomial p^U for L_H

This is a "universal diophantine equation", considering that L_H is recursively enumerated by the universal TM T^U

Conclusions

All this is at the very root of our discipline

At the rise of computer science

Computation is investigated in terms of mathematical models and mathematical "techniques"

Limits on what computation can do were established

Problems of mathematical logic and the work of mathematical logicians were the basis for further developments

Among many others:

- Non-determinism in computation and non-deterministic TMs

  Seminal work on non-deterministic automata by M. Rabin and D. Scott, logicians again ...

- Study the complexity of decision problems

  Setting quantitative limits on their solvability

  In terms of temporal, spatial, ... resources

- Most prominently, the work by S. Cook on the dichotomy between solvability in polynomial time vs. non-deterministic polynomial time

$$P \overset{?}{\subsetneq} NP$$

  A million dollar problem ... Open since 1971 ...

- In this case SAT, i.e. satisfiability of propositional logical formulas, was the protagonist

  It became the first candidate to be in $NP \setminus P$ if they are different ...