CDM

(Semi) Decidability

Klaus Sutner

Carnegie Mellon University

Fall 2023

Computability and Problems

In TCS and math, the main interest in the machinery of computability comes from the desire to solve certain discrete problems. Here are the most important types:

Decision Problems Return a Yes/No answer.

Counting Problems Count objects of a certain kind.

Function Problems Calculate a certain function.

Search Problems Select one particular solution.

Decision Problems

Definition

A decision problem consists of a set of instances and a subset of Yes-instances.

Problem:	**Primality**
Instance:	A natural number x.
Question:	Is x a prime number?

Here the set of all instances is $\mathbb{N}$ and the set of Yes-instances is $\{\, p \in \mathbb{N} \mid p \text{ prime} \,\}$.

The answer (solution) to any decision problem is just one bit (true or false).

Counting Problems

Definition

A counting problem consists of a set of instances, each instance I is associated with a set of "solutions" sol(I). We need to calculate the cardinality of sol(I).

Problem:	**Prime Counting**
Instance:	A natural number x.
Solution:	The number of primes $p \leq x$.

Here the set of all instances is $\mathbb{N}$ and the "solutions" for x are $\{\, p \leq x \mid p \text{ prime} \,\}$. In the literature, this function is denoted $\pi(x)$.

The answer to a counting problem is always a natural number.

Function Problems

Definition

A function problem consists of a set of instances and, for each instance I, a unique solution $\mathsf{sol}(I)$.

Problem:	**Next Prime**
Instance:	A natural number x.
Solution:	The least prime $p > x$.

Instances are again $\mathbb{N}$ and the solution for $x \in \mathbb{N}$ is $\mathsf{sol}(x) = p$ where p is the appropriate prime (uniquely determined).

We insist that there always is a solution (otherwise sol would be a partial function).

Search Problems

Definition

A search problem consists of a set of instances and, for each instance I, a set of solutions sol(I).

In this case, “the” solution is not required to by unique. And, we allow for sol(I) to be empty. This turns out to be very convenient in practice.

Problem:	**Factor**
Instance:	A natural number x.
Solution:	A natural number z, $1 < z < x$, dividing x.

Note that the set of solutions is empty if x is prime.

Solving a Problem

- To solve a decision problem, a computable function has to accept each instance of the problem as input, and return “Yes” or “No” depending on whether the instance is a Yes-instance.
- To solve a counting problem, a computable function has to accept each instance x of the problem as input, and return the appropriate count $|\mathsf{sol}(x)|$ as answer.
- To solve a function problem, a computable function has to accept each instance x of the problem as input, and return the unique $\mathsf{sol}(x)$.
- To solve a search problem, a computable function has to accept each instance x of the problem as input, and return either an element of $\mathsf{sol}(x)$ or “No” if $\mathsf{sol}(x)$ is empty (the instance has no solutions).

Connections

Note that **Primality**, **Prime Counting** and **Next Prime** are closely connected: an algorithm for one problem can be turned into an algorithm for the other, plus a modest bit of overhead (a white lie). This idea of a reduction is critical in computability theory and in complexity theory.

Factor is a bit different, though: there are primality tests that provide no insight into factors of a composite number.

In fact, we now know that Primality is easy in the sense that there is a polynomial time algorithm for it, but we fervently hope that Factor will turn out to be hard (certain cryptographic methods will fail otherwise).

Instances

Our examples have $\mathbb{N}$ as the set of instances. In general, for some general decision problem Π we may have

- a set I_Π of instances, and
- a set $Y_\Pi \subseteq I_\Pi$ of Yes-instances.

For example, I_Π might be the collection of all ugraphs, and Y_Π could be the collection of all connected ugraphs.

If we code everything as naturals, then I_Π is always trivial (certainly p.r.). The same holds for the other types of problems.

More Instances

One might wonder what an algorithm is supposed to do with input that is not an instance of the problem in question.

There are two choices:

- We don't care: the behavior of the algorithm can be arbitrary.
- More realistic: the algorithm returns some default output. This is perfectly reasonable since the collection of all inputs is always trivially decidable, certainly polynomial time.

  Something very fishy is going on if it takes a lot of effort even just to recognize a valid instance.

Simplifying

Since the collection of instances I_Π is always trivial, one usually ignores it altogether.

Instead one focuses on the set of Yes-instances Y_Π and identifies its complexity with the complexity of Π.

In particular, the problem Π is undecidable iff Y_Π is so undecidable.

Canonical Representations

Another issue is the choice of input data structure. Fortunately, in all practical cases there seems to be a canonical, natural choice. Essentially, all that is needed is:

- Natural numbers are given in binary.
- Nested lists of objects (hereditarily finite lists).

Note that numbers could actually be expressed as lists, but that seems a bit excessive (we might as well descend all the way into misery and use pure set-theory).

But recall that we ignore efficiency entirely at this point; we are only concerned with abstract computability. Things get trickier in the realm of low complexity classes.

Honest Instances (or: Don't Cheat)

In all our models the input must be given in explicit, unobfuscated form.

Not allowed are tricks like the following:

> Encode natural number n as $\langle 1, n\rangle$ whenever n is prime; and as $\langle 0, n\rangle$ when n is compound.

With this "input convention" primality testing would be trivial, but the coding procedure requires a lot of computational effort.

Worse, a similar trick could be used to trivialize any computational problem: just code the solution as part of the input.

Fuggedaboudit.

Canonical Representations

One might worry that dealing with specific data structures is messy, but in all practical cases there seems to be a canonical, natural choice. Essentially, all that is needed is:

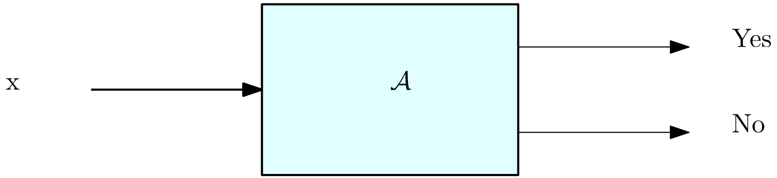
- Natural numbers are given in binary.
- Nested lists of objects (hereditarily finite lists).

Using standard coding machinery, all these representations can be translated into a natural number; in an intuitive, effective manner.

Of course, efficiency considerations fall by the wayside in this approach. If you want to get mileage out of existing physical realizations of computation you need something like C.

Decidability

Informally, a problem is decidable if there is a decision algorithm $\mathcal{A}$ that returns Yes or No depending on whether the input has the property in question.



Formally . . .

We can easily model this in terms of computable functions:

Definition

A set $R \subseteq \mathbb{N}^k$ is decidable if the characteristic function char_R is computable.

A decision problem is decidable if the set of Yes-instances is decidable.

Note that the characteristic function is always total: no matter what the input is, the computation will be finite and will produce a result.

Terminology

In keeping with the old recursive/partial recursive nomenclature, a decideable set is also called recursive.

This is arguably not exactly great terminology, but it is firmly entrenched.

Gottfried Leibniz (1646–1716)

In a way, the idea of decidability can be traced back to Leibniz's ars magna.

Ars Magna

- ars inveniendi: generate all true scientific statements.
- ars iudicandi: decide whether a given scientific statement is true.

It is obvious that if we could find characters or signs suited for expressing all our thoughts as clearly and as exactly as arithmetic expresses numbers or geometry expresses lines, we could do in all matters insofar as they are subject to reasoning all that we can do in arithmetic and geometry. For all investigations which depend on reasoning would be carried out by transposing these characters and by a species of calculus.

Decision Algorithms

Decision problems have been around since the day of the flood: one is interested in checking whether a number is prime, whether a polynomial is irreducible, whether a polygon is convex, and so on. Gauss certainly understood the computational difficulty of primality checking.

Leibniz notwithstanding, the formal study of decision problems from a computational perspective is relatively new.

The first big splash came in 1900, when Hilbert presented his famous list of 23 open problems at the International Congress of Mathematicians in Paris.

Hilbert's List

Hilbert's list was enormously influential throughout the 20th century.

Some of Hilbert's Problems

1. Prove the Continuum Hypothesis. Well-order the reals.
2. Prove that the axioms of arithmetic are consistent.

   . . .

8. Prove the Riemann Hypothesis.

   . . .

10. Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: to devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.

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, W. Ackermann
Grundzüge der theoretischen Logik, 1928

A Hedge

Note that Hilbert and Ackermann hedge a bit: the decision procedure for the Entscheidungsproblem needs to work only for areas where the fundamental assumptions can be finitely axiomatized.

That leaves open the possibility that some part of math might not be finitely axiomatizable, and thus outside of the reach of the Entscheidungsproblem.

Still, J. Herbrand pointed out that

> In a sense, the Entscheidungsproblem is the most general problem of mathematics.

Aside: Hardness

Hilbert's dream failed, there is no algorithm to solve the Entscheidungsproblem.

But we can scale back a bit: try to solve the Entscheidungsproblem only for a small domain. For example, only worry about propositional logic (rather than full first-order). In this capacity, the Entscheidungsproblem is an excellent source of difficult problems in complexity theory.

For suitable versions of the Entscheidungsproblem, instead of undecidability, we get computational hardness. For example, for $\mathbb{NP}$, we only need Boolean formulae of type

$$\exists\, x_1, \ldots, x_n \; \varphi(x_1, \ldots, x_n)$$

Closure Properties

Lemma

The decidable sets are closed under intersection, union and complement. In other words, the decidable sets form a Boolean algebra.

Proof.

Consider two decidable sets $A, B \subseteq \mathbb{N}^k$. We have two register machines M_A and M_B that decide membership.

Idea: Run both M_A and M_B on input x, returning output b_A and b_B where $b_A, b_B \in \{0, 1\}$.

For intersection return $\min(b_A, b_B)$,

for union return $\max(b_A, b_B)$,

for the complement of A return $1 - b_A$.

$\square$

Even Better: Effective Boolean Algebra

As usual, we have used elementary arithmetic to express logical operations.

conjunction	$\min(b_A, b_B)$	$b_A \wedge b_B$
disjunction	$\max(b_A, b_B)$	$b_A \vee b_B$
negation	$1 - b_A$	$\neg b_A$

But this means that the Boolean algebra of decidable sets is effective: we can represent the elements by natural numbers (recall the index machinery), and the corresponding algebraic operations are then computable (even primitive recursive).

We have a computable structure, not just an abstract algebra.

How Bad Can It Be?

Here is an innocent question:

Is every decision problem $A \subseteq \mathbb{N}$ decidable?

If this were true we could construct, for any $A \subseteq \mathbb{N}$, a register machine M_A that, on input any number $x \in \mathbb{N}$, halts with output $\text{char}_A(x)$.

Unfortunately, the answer is **NO**.

Here is a cardinality argument due to Cantor that shows that there are lots of undecidable problems.

Counting

Theorem

There are undecidable decision problems.

Proof.

There are uncountably many subsets of $\mathbb{N}$ (to be precise: $2^{\aleph_0}$).

But there are only countably many register machines (recall our coding machinery).

Hence uncountably many problems $A \subseteq \mathbb{N}$ are not decidable.

$\square$

Note that this argument, like others in set theory, leaves a bitter after-taste: it does not produce any concrete undecidable problem. As usual, there is a more constructive approach.

Going Concrete

Fortunately, our study of clones of computable functions points us in the right direction when it comes to finding a concrete undecidable problem.

Recall our observation that the eval operator for computable functions is necessarily partial: sometimes eval(e, x) must be undefined, otherwise we are faced with a contradiction.

This opens the door for a decision problem.

Halting

Problem:	**Halting**
Instance:	Index $e \in \mathbb{N}$.
Question:	Does register machine M_e halt on input e?

A slightly more precise way to express this would be to fix a universal register machine $\mathcal{U}$ and to ask whether $\mathcal{U}$ on input program e and data e halts:

$$\mathcal{U}(e, e) \downarrow$$

The choice of the URM is entirely arbitrary, but will not affect the decidability status of Halting.

Other Halting

It may seem more natural to consider the following version:

Problem:	**Full Halting**
Instance:	Index $e \in \mathbb{N}$, an argument $x \in \mathbb{N}$.
Question:	Does register machine M_e halt on input x?

As we will see, there is essentially no difference between the two versions, and the first one is slightly more elegant.

Yet Other Halting

We can also get rid of the input altogether: the machine can start with a precomputation that produces the input for the main computation.

Problem:	**Pure Halting**
Instance:	Index $e \in \mathbb{N}$.
Question:	Does register machine M_e halt?

In this case, all registers are zeroed out before the computation starts. Again, this is essentially the same as the other two versions. Arguably, this is the most elegant variant.

Halting is Undecidable

A standard diagonalization argument shows that Halting cannot be solved by a register machine.

Theorem

The Halting Problem is undecidable.

And, of course, we could do this in any model of computation, there always is a universal “machine.”

Proof, Informal

Assume Halting is decidable. Then the following program

```
// impossible function

    if( halt(z,z) )
        return  eval(z,z) + 42;
    else
        return 0;
```

produces a contradiction.

Here `halt(z,z)` is the halting tester that exists by assumption, and `eval(z,z)` is the universal RM: run M_z on input z.

Proof, Formal

Suppose Halting (version 1) is decidable.

Then we can define a function g by cases:

$$g(z) \simeq \begin{cases} \{z\}(z) + 1 & \text{if } \{z\}(z) \downarrow, \\ 0 & \text{otherwise.} \end{cases}$$

This function g is computable and even total. Hence, g has some index e, which promptly produces a contradiction: $g(e) = g(e) + 1$.

Diagonalization

Diagonalization was invented by Cantor in the context of set-theory to establish the uncountability of the reals.

It is rather surprising that this tool also turns out to be of central importance in the computational universe.

Alas . . .

Diagonalization always leaves a bad aftertaste—one has the desired object, but the construction is oddly elusive, and feels insufficiently "concrete."

Can one make the problem with Halting a little more tangible?

The Busy Beaver problem is directly connected to Halting: if we could filter out all machines of a certain size that fail to halt, then we could easily determine a champion among the remaining machines (in principle, as we have seen, even for Turing machines with 6 states this is entirely impossible in any physical sense).

A Real Math Problem

Arguably the most important, longstanding open problem in math is the Riemann Hypothesis.

In its original form, RH looks like a solid chunk of complex analysis. You start with the Riemann zeta function

$$\zeta(s) = \sum_{n \geq 1} \frac{1}{n^s}$$

where $s \in \mathbb{C}$, $\mathrm{Re}(s) > 1$.

The condition $\mathrm{Re}(s) > 1$ is needed to ensure convergence of the series (think about $s = 1$).

Better Riemann

Using a technique known as analytic continuation, one can then extend ζ to a function of type $\mathbb{C} \to \mathbb{C}$ with a pole at $s = 1$.

The reason ζ is so hugely important is that it is closely connected to prime numbers, a fact already known to Euler.

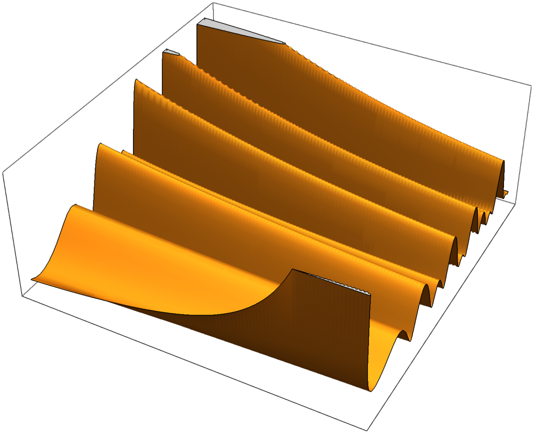
$$\zeta(s) = \prod_p \frac{1}{1 - p^{-s}}$$

where the product is over all primes.

A better understanding of ζ would have implications for the distribution of primes. A more general form of RH implies that a particular primality test runs in polynomial time (G. Miller).

Visualization

Since ζ is a complex function, it is rather hard to draw.



The modulus of $\zeta(s)$ for $0 \leq \mathrm{Re}(s) \leq 1$ and $0 \leq \mathrm{Im}(s) \leq 50$.

Roots

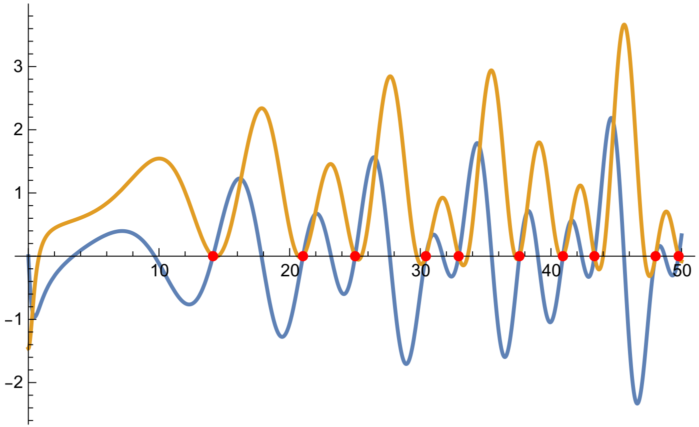
One can show that $\zeta(-2k) = 0$ for all positive integers k (trivial zeros).

Alas, there are other, non-real roots that are much harder to understand. Here is the big conjecture, proposed by Bernhard Riemann in 1859:

All non-real roots s have the property $\text{Re}(s) = 1/2$.

Note that this statement seems to require considerable horse-power: first we need to define the analytic extension of a power series, then we want to argue about some of roots of the resulting function.

Complex Roots



Complex and real parts of $\zeta(1/2 + i\,t)$ for $0 \leq t \leq 50$.

The Computability Sledgehammer

So far, we are dealing with complex analysis. The next step is to reduce all this complicated material to computation.

A formula of arithmetic is said to be Π_1 if it can be written down by using just one universal quantifier over $\mathbb{N}$, plus bounded quantifiers, propositional logic and standard arithmetic.

$$\Theta \equiv \forall\, n\, \phi(n)$$

where ϕ is a formula of basic arithmetic. The relation $\phi(n)$ is in particular primitive recursive and easy to check.

But checking Θ itself is more problematic: on the face of it, we have to perform an infinite computation, we have try out all possible values of n.

It's Π_1

Here is the big surprise: the Riemann Hypothesis is Π_1.

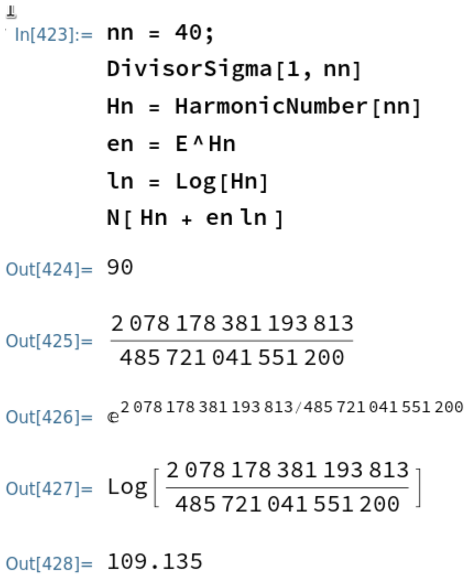
This may sound patently wrong, but can be shown with enough effort: let $H_n = \sum_{k \leq n} 1/k$ be the nth harmonic number, and $\sigma(n)$ the divisor function (total sum of all divisors of n). These are all primitive recursive.

The RH is equivalent to: for all n,

$$\sigma(n) \leq H_n + e^{H_n} \log H_n$$

As written, this looks like real arithmetic involving e and $\log$. In the actual argument, everything has to be rephrased in terms of rational numbers, and one needs to be very careful with error estimates.

Example: $n = 40$



```
In[423]:= nn = 40;
          DivisorSigma[1, nn]
          Hn = HarmonicNumber[nn]
          en = E^Hn
          ln = Log[Hn]
          N[Hn + en ln]

Out[424]= 90

Out[425]= 2078178381193813
          ----------------
          485721041551200

Out[426]= e^(2078178381193813/485721041551200)

Out[427]= Log[2078178381193813
              ----------------]
              485721041551200

Out[428]= 109.135
```

The Riemann Register Machine

Based on this inequality, one can now construct a register machine $\mathcal{M}$ that runs through a (potentially infinite) loop and tries to check the inequality for all n.

If $\mathcal{M}$ finds a counterexample, it stops and returns 42. Otherwise $\mathcal{M}$ runs forever.

Claim: $\mathcal{M}$ halts iff the Riemann Hypothesis is false.

There even are estimates on how large $\mathcal{M}$ would need to be (though existing work uses Turing machines rather than register machines; less than $10,000$ states).

Pushing Things

Similarly we can design a register machine $\mathcal{Z}$ that systematically enumerates all first-order logic theorems provable in Zermelo-Fraenkel with Choice. This works since the axioms of ZFC are easily decidable.

Now suppose $\mathcal{Z}$ is set up to halt when it finds a proof of $\emptyset = \{\emptyset\}$, otherwise it runs forever.

By Gödel's incompleteness theorem, and assuming that ZFC is consistent, we cannot prove in ZFC that $\mathcal{Z}$ never halts, nor can we prove that it halts.

So ZFC is too weak to say anything about the Halting behavior of $\mathcal{Z}$.

Halting is seriously hard.

The Next Step

Decidability makes direct algorithmic sense, we are trying to test for certain properties in a computational manner.

Alas, it turns out that there is a weaker property, called semidecidability, that is very closely related to decidability, and that is arguably more fundamental and ultimately more important.

Semidecidability

Here is a generalization of decidability:

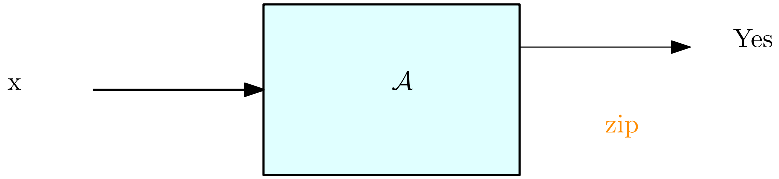
Definition

A set $A \subseteq \mathbb{N}^k$ is semidecidable if there is a register machine that, on input x, halts if $x \in A$, and fails to halt otherwise.

We will call this a semidecision procedure. As one might suspect, the term semi-recursive is also used.

You can think of this as a broken decision algorithm: if the answer is Yes, the algorithm works properly and stops after finitely many steps. But, if the answer is No, it just keeps running forever, it never produces a result.

In Pictures



Just to be clear: zip is not output, it just means “there is no output.”

Quoi?

Note that Halting is the critical natural example of a semidecidable problem: we can determine convergence in finitely many steps, but divergence takes forever. This is exactly the idea behind an unbounded search: we find a witness in finitely many steps if there is one, but otherwise we keep searching forever.

Here is the critical connection between decidability and semidecidability:

Lemma

A set is decidable iff the set and its complement are both semidecidable.

Proof

Clearly, A decidable implies that both A and $\overline{A}$ are semidecidable.

But the opposite direction is far from trivial: we have two semidecision procedures $\mathcal{A}_0$ and $\mathcal{A}_1$, but we don't know which one is going to halt. So we cannot simply run, say, $\mathcal{A}_0$ first.

The way around this problem is to run both procedures in parallel on the given input: we combine two computations into one, alternating steps. We stop as soon as one of the sub-computations terminates.

Intuitive, this no surprise; every operating system does this. But it actually is another fundamental property of computation: we can interleave two computations into a single one.

Closure Properties

Lemma

The semidecidable sets are closed under intersection and union.

Proof.

For intersection we can simply run the two semidecision-procedures sequentially.

But for union we again need to interleave two computations.

□

Note: We do not have closure under complement in general: Halting and the last lemma prohibit that.

No Way Around It

A similar problem arises in complexity theory. For example, there seems to be no reasonable way to express finite graphs as data structures, so that a graph is Hamiltonian iff its data structure has some simple property (the first byte is $\mathbf{0}$).

If we were to represent graphs by such magic data structures, a substantial amount of computation would have to go into just creating the data structures.

This is cheating, data structures are supposed to be canonical and require no special care and feeding.

Universality

Pick any model of computation: Herbrand-Gödel, μ-recursive, λ-definable, Turing-computable, register machine computable.

As Turing has shown, there is a universal "machine" $\mathcal{U}$ which can be used to produce an effective enumeration $(\{e\})_e$ of all computable functions:

$$\{e\}(x) \simeq \mathcal{U}(e, x)$$

Exercise

Figure out what the universal "machine" would look like in the other models.

Kleene Normal Form

Here is a closer look at Halting and semidecidability. Suppose we have a universal register machine $\mathcal{U}$ producing an enumeration $\{e\}$ of all computable functions.

We claim that there is a primitive recursive relation $T(e,x,t)$ and a primitive recursive function D (in fact, both T and D are quite straightforward) such that

$$\{e\}(x)\downarrow \iff \exists t\, T(e,x,t)$$

$$\{e\}(x) \simeq D(\min\big(t \mid T(e,x,t)\big)$$

$T(e,x,t)$ essentially means: the computation of $\mathcal{U}$ on e and x terminates after at most t steps.

Kleene's T Predicate

$T(e, x, t)$

- e the index of a register machine M
- x an input argument for M
- t a witness for a halting computation of M on input x.

The witness is usually the (sequence number that codes) a sequence of configurations $C_0, C_1, \ldots, C_n$ of M.

Since we have an alleged witness, T is easily decidable and primitive recursive.

Moreover, given the right t, it is easy to read off the output of the computation (that's D's job).

Σ_1

Recall the concept of a Π_1 formula from above. Analogously, we can define a Σ_1 formula to have the form

$$\Theta \equiv \exists\, n\, \phi(n)$$

Then $\{e\}(x) \downarrow$ is Σ_1.

Similarly, for any computable function f, the statement $f(x) \simeq y$ is Σ_1: we only need one unbounded search over a primitive recursive property.

Undecidability

The reason $\{e\}(x) \simeq y$ is not decidable is that we cannot bound the existential quantifier in $\{e\}(x)\downarrow \iff \exists t\, T(e,x,t)$ in a computable manner.

Essentially, the only way we can find the right t is by running the computation, there is no clever computational shortcut in general. This is a perfect example of computational incompressibility.

Note that this is not a problem in any real algorithm: given some particular input we can always compute ahead of time an upper bound on the running time of the algorithm. Ditto for memory requirement.

Proof: check all the algorithms in 451.

Still Unhappy?

Halting may well seem like a somewhat unsatisfactory example of an undecidable problem: it's a perfect case of navel gazing. Certainly it does not deal with a question at least superficially unrelated to computability.

Actually, anyone who has ever written a complicated program in a language like C would have to admit that Halting is really quite natural.

But how about undecidable problems that are of independent interest? Perhaps something that was studied even before the concept of an algorithm was invented?

Warning: Treacherous Arithmetic

Basic arithmetic on the natural numbers may seem fairly straightforward; tedious on occasion, but not truly complicated.

Wrong, wrong, wrong. Consider the numbers

$$n^{17} + 9 \qquad \text{and} \qquad (n{+}1)^{17} + 9$$

They turn out to be coprime up until we hit

$$8424432925592889329288197322308900672459420460792433$$

the first counterexample, about 8.42×10^{51}.

Where the hell does this huge number come from?

Hilbert’s 10th Problem

Perhaps the most famous example of an undecidability result in mathematics is Hilbert’s 10th problem (HTP), the insolubility of Diophantine equations.

A Diophantine equation is a polynomial equation with integer coefficients:

$$P(x_1, x_2, \ldots, x_n) = 0$$

The problem is to determine whether such an equation has an integral solution.

Theorem (Y. Matiyasevic, 1970)

It is undecidable whether a Diophantine equation has a solution in the integers.

Examples

Pythagorean triples: it is not hard to classify all the solutions of

$$x^2 + y^2 - z^2 = 0$$

But tackling

$$x^k + y^k - z^k = 0$$

for $k > 2$ was one of the central problems of number theory and it took more than 350 years to show that solutions only exist for $k = 1, 2$.

The Proof

The proof of undecidability of Diophantine equations is way too complicated to be presented here, but the main idea is a reduction using very clever coding tricks:

> *Show that decidability of Hilbert's 10th problem implies decidability of the Halting problem.*

More precisely, call a set $A \subseteq \mathbb{Z}$ Diophantine if there is a polynomial P with coefficients over $\mathbb{Z}$ such that

$$a \in A \iff \exists\, x_1, \ldots, x_n \in \mathbb{Z} \left(P(a, x_1, \ldots, x_n) = 0 \right).$$

This condition is rather unwieldy, it is usually fairly difficult to show that a particular set is in fact Diophantine.

Diophantine Sets

We want to describe $a \in \mathbb{Z}$ with some particular property:

Even: $a = 2x$.

Non-zero: $a\,x = (2y + 1)(3z - 1)$.

Non-negative: $a = x_1^2 + x_2^2 + x_3^2 + x_4^2$ (Lagrange's theorem)

Similarly we can show closure under intersection (sum of squares) and union (product). But note that complements don't work in general.

Again: Semidecidability

It turns out that exactly the semidecidable sets are Diophantine. In particular, the Halting Set is Diophantine, and so it must be undecidable whether an integer polynomial has an integral solution.

It is clear that every Diophantine set is semidecidable: given a, we can simply enumerate all possible $\boldsymbol{x} \in \mathbb{Z}^n$ in a systematic way, and compute $P(a, \boldsymbol{x})$.

If we ever hit 0, we stop; otherwise we run forever.

Surprisingly the opposite direction also holds, but this is much, much harder to show.

Details

First M. Davis was able to show that every semidecidable set A has a Davis normal form: there is a polynomial such that

$$a \in A \iff \exists\, z\, \forall\, y < z\, \exists\, x_1, \ldots, x_n \big(P(a, x_1, \ldots, x_n, y, z) = 0 \big).$$

Davis, Putnam and Robinson then managed to remove the offending bounded universal quantifier at the cost of changing P to an exponential polynomial (containing terms x^y).

Lastly, Matiyasevic showed how to convert the exponential polynomial into an ordinary one.

Bounding Roots

Note that if we could produce a computable bound on the size of a possible root we could use brute-force search to determine whether one exists (in principle, in reality we die an exponential death).

Consider a univariate polynomial $p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots a_0$ of degree n (so $a_n \neq 0$).

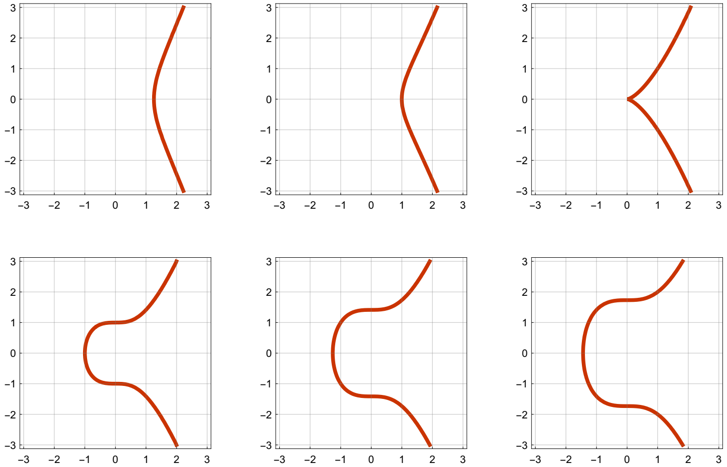
By rearranging terms a bit and using the standard properties of inequalities over the reals we find that for any root x

$$|x| \leq n \cdot \frac{a_{\max}}{|a_n|}$$

where $a_{\max} = \max(|a_{n-1}|, \ldots, |a_0|)$. Done by search.

Elliptic Curves

How about a simple elliptic curve, say, $y^2 = x^3 + c$? Here is $-2 \leq c \leq 3$.



Discrete Case

For real solutions, this scenario is trivial. But we are looking for integral solutions, and then things get incredibly messy.

It is known that the least integral solution, if it exists, is bounded by

$$\mathsf{exp}((10^{10}|c|)^{10000})$$

So, in principle, this is decidable by brute force search.

In practice, nothing moves; even if we were to allow nondeterminism. Elliptic curves turn out to be useful in cryptography.

Cubes

How about sums of three cubes?

$$x^3 + y^3 + z^3 = c$$

Over $\mathbb{Z}$, it is not known that this is decidable.

Worse, it may well be that c is the sum of 3 cubes iff $c \neq \pm 4 \pmod 9$. But that's an open problem.

Recent result, found by a big computer search:

$$33 = 8866128975287528^3 - 8778405442862239^3 - 2736111468807040^3$$

This just in . . .

Andrew Sutherland at MIT and Andrew Booker at U Bristol used over a million hours of compute time on the Charity Engine to find

$$42 = -80538738812075974^3 + 80435758145817515^3 + 12602123297335631^3$$

Bounding Roots, II

So it would not be unreasonable to think that integer roots of $P(x_1, \ldots, x_n)$ can be bounded by some rapidly growing but computable function of n, the degree d of P, and the largest coefficient c.

Perhaps something insanely huge, using the Ackermann function:

$$A\left(n!\,|c|^d, n^{n^{d+42}}\right)$$

would work?

If not, try an even higher level of the Ackermann hierarchy or one of Friedman's monsters?

Bounding Roots is Hard

Alas, finding bounds even in very concrete cases turns out to be quite difficult. For example, consider again the three cubes problem

$$x^3 + y^3 + z^3 = 74.$$

The smallest solution is

$$\begin{aligned} x &= -284650292555885 \\ y &= -66229832190556 \\ z &= 283450105697727 \end{aligned}$$

Even Fermat is Hard

For Fermat type problems $x^n + y^n = z^n$ we have a complete answer thanks to Wiles. Note that the solutions (x, y, z) have the nice property that $(\alpha x, \alpha y, \alpha z)$ is also a solution.

So we can rephrase this as solving $x^n + y^n = 1$ over the rationals.

This naturally leads to the question "How many rational points are there on an algebraic curve?", another brutally hard problem (see Falting's Theorem).

More Problems

Or try to find a positive solution for the seemingly innocuous Pell equation:

$$x^2 - 991y^2 - 1 = 0.$$

Of course, $x = 1$, $y = 0$ is a trivial solution. The smallest positive solution here is

$$\begin{aligned} x &= 379516400906811930638014896080 \\ y &= 12055735790331359447442538767 \end{aligned}$$

The Sun God's Herd

An old puzzle, supposedly due to Archimedes, about the size of the herd of cattle owned by the sun god, comes down to solving a system of linear equations and then the Pell equation

$$x^2 - 410286423278424\,y^2 - 1 = 0.$$

In this case, the least positive solution has 103273 and 103265 decimal digits.

Exercise (in futility)

Try to find a positive solution to $313(x^3 + y^3) = z^3$.
Or try $x^3 + y^3 + z^3 = 33$.

Different Rings

Note that the choice of $\mathbb{Z}$ as ground ring is important here. We can ask the same question for polynomial equations over other rings R (always assuming that the coefficients have simple descriptions).

- $\mathbb{Z}$: undecidable
- $\mathbb{Q}$: major open problem
- $\mathbb{R}$: decidable
- $\mathbb{C}$: decidable

Decidability of Diophantine equations over the reals is a famous result by A. Tarski from 1951, later improved by P. Cohen.

One Implication

It is true that an algorithm for HTP over $\mathbb{Z}$ would produce an algorithm for HTP over $\mathbb{Q}$.

Consider $P(\boldsymbol{x}) = 0$ with $\boldsymbol{x} \in \mathbb{Q}$. This is equivalent to

$$\exists\, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{Z} \left(P(\boldsymbol{y}/\boldsymbol{z}) = 0 \wedge z_1 \ldots z_k \neq 0 \right)$$

Of course, this is exactly the wrong direction.

Exercise

Why does undecidability over $\mathbb{Z}$ not simply imply undecidability over $\mathbb{Q}$? What is the obstruction?

Hard Polynomials

Since we can encode arbitrary semidecidable sets as Diophantine equations, we can in particular encode universal sets.

That means that there is a single polynomial with parameter a for which the question

$$\exists\, x_1, \ldots, x_n\, P(a, x_1, \ldots, x_n) = 0$$

is already undecidable.

As one might suspect, there is a trade-off between the degree d of such a polynomial and its number of variables n. Here are some known (d, n) pairs that admit universal polynomials:

$$(4, 58), (8, 38), (12, 32), \ldots, (4.6\, 10^{44}, 11), (8.6\, 10^{44}, 10), (1.6\, 10^{45}, 9)$$

Jones, Sato, Wada, Wiens Polynomial

$$
\begin{gathered}
(k+2)\Big(1-[wz+h+j-q]^2-[(gk+2g+k+1)(h+j)+h-z]^2- \\
[16(k+1)^3(k+2)(n+1)^2+1-f^2]^2-[2n+p+q+z-e]^2- \\
[e^3(e+2)(a+1)^2+1-o^2]^2-[(a^2-1)y^2+1-x^2]^2- \\
[16r^2y^4(a^2-1)+1-u^2]^2-[n+l+v-y]^2-[(a^2-1)l^2+1-m^2]^2- \\
[ai+k+1-l-i]^2-[((a+u^2(u^2-a))^2-1)(n+4dy)^2+1-(x+cu)^2]^2- \\
[p+l(a-n-1)+b(2an+2a-n^2-2n-2)-m]^2-[q+y(a-p-1)+ \\
s(2ap+2a-p^2-2p-2)-x]^2-[z+pl(a-p)+t(2ap-p^2-1)-pm]^2\Big)
\end{gathered}
$$

This polynomial P has 26 variables and degree 25. $P(\mathbb{N}^{26}) \cap \mathbb{N}$ is the set of prime numbers. Note the factor $(k+2)$ up front.