GTI

Undecidability

A. Ada, K. Sutner
Carnegie Mellon University

Spring 2018

Total Recall

A set A is

finite	bijection $\{0, 1, \ldots, n-1\} \leftrightarrow A$
countably infinite	bijection $\mathbb{N} \leftrightarrow A$
uncountable	otherwise

So uncountable means that there is no surjection $\mathbb{N} \to A$, so we cannot associate an element from a countable collection to each element in A.

Consequences

- Almost all decision problems are undecidable.

  Reason: the collection of TMs is countable, but $\mathfrak{P}(\Sigma^\star)$ is uncountable.

- Almost all real numbers are non-computable.

  Reason: the collection of TMs is countable, but $\mathbb{R}$ is uncountable.

Here a real r is said to be computable if for some TM M (which outputs rational numbers):

$$\forall n\ \left(|M(n) - r| < 2^{-n}\right)$$

All you favorite reals like π, e, $\ln 2$ and so on are computable.

Cardinality Arguments

Cardinality arguments are very compact and elegant

. . . but they leave a bad aftertaste: they provide no concrete examples.

Here is another typical example: $\mathbb{N}^\star$ is countable.

The map $f : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$, $f(n) = (a, b)$ when $n = 2^a 3^b$, $(0, 0)$ otherwise, is surjective. Hence $\mathbb{N}^2$ is countable, by induction $\mathbb{N}^k$ is countable. Thus $\mathbb{N}^\star$ is a countable union of countable sets, and also countable.

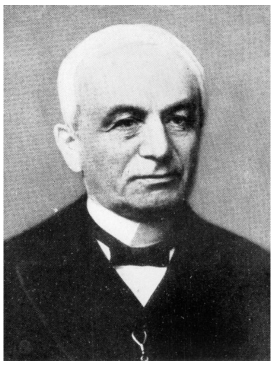
Yes, some bijection must exist, but we really have no idea what it looks like.

Leopold Kronecker, Semi-Constructivist

Die ganzen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk.

“Dear god” made the integers, everything else is the work of men.

Kronecker sabotaged Cantor every step of the way.



Coding

We would like a coding function, a polyadic map of the form

$$\langle\,.\,\rangle : \mathbb{N}^* \to \mathbb{N}$$

that allows us to decode: from $b = \langle\, a_0, a_1, \ldots, a_{n-1} \,\rangle$ we can recover n as well as all the a_i.

So the function must be injective.

Moreover, both the coding and decoding operations should be computationally cheap.

Computable Coding

In practice, we need three functions: the actual coding function, an decoding function and a length function:

$$\begin{aligned} \langle\,.\,\rangle &: \mathbb{N}^\star \to \mathbb{N} \\ \text{dec} &: \mathbb{N} \times \mathbb{N} \to \mathbb{N} \\ \text{len} &: \mathbb{N} \to \mathbb{N} \end{aligned}$$

If $b = \langle\, a_0, a_1, a_2, \ldots, a_{n-1} \,\rangle$ then $\text{len}(b) = n$ and $\text{dec}(b, i) = a_i$.

Again: in the set-theoretic universe, the existence of these functions is entirely trivial: $\mathbb{N}^\star$ is countable.

Pairs

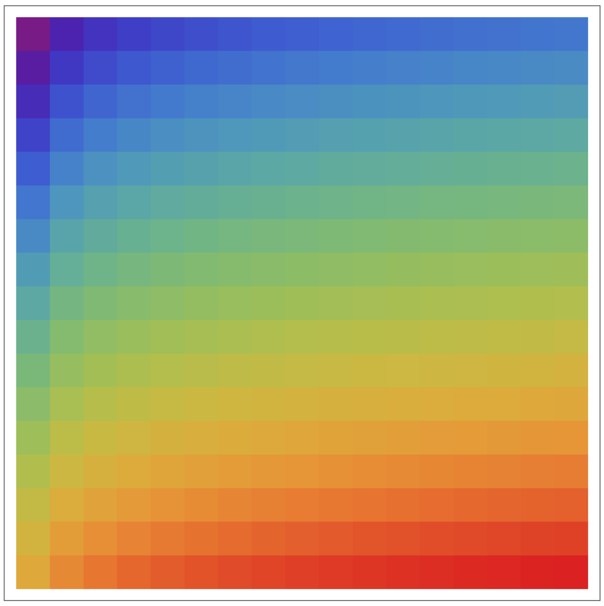
The first step is to select a pairing function, an injective map $\pi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.

There are many possibilities, the following choice arguably yields one of the most intuitive coding functions.

$$\pi(x, y) = 2^x(2y + 1)$$

For example

$$\pi(5, 27) = 32 \cdot 55 = 1760 = 110111\,00000_2$$



In Binary

Note that the binary expansion of $\pi(x, y)$ looks like so:

$$y_k y_{k-1} \dots y_0\, 1\, \underbrace{00 \dots 0}_{x}$$

where $y_k y_{k-1} \dots y_0$ is the standard binary expansion of y (y_k is the most significant digit). Hence the range of π is $\mathbb{N}_+$ (but not $\mathbb{N}$).

This makes it easy to find the corresponding unpairing functions:

$$x = \pi_1(\pi(x, y)) \qquad y = \pi_2(\pi(x, y)).$$

Extending to Sequences

$$\langle\, \mathsf{nil} \,\rangle := 0$$
$$\langle\, a_0, \dots, a_{n-1} \,\rangle := \pi(a_0, \langle\, a_1, \dots, a_{n-1} \,\rangle\,)$$

Here are some sequence numbers for this particular coding function:

$$\langle\, 10 \,\rangle = 1024$$
$$\langle\, 0, 0, 0 \,\rangle = 7$$
$$\langle\, 1, 2, 3, 4, 5 \,\rangle = 532754$$

Informally ...

Here is a sequence number and its binary expansion:

$$\langle 2,3,5,1 \rangle = 20548$$
$$= 1\underbrace{0}_{1}1\underbrace{00000}_{5}1\underbrace{000}_{3}1\underbrace{00}_{2}$$

So the number of 1's (the digitsum) is just the length of the sequence, and the spacing between the 1's indicates the actual numerical values.

It follows that the coding function is injective and surjective, right?

It's a Bijection

Lemma

$\langle . \rangle : \mathbb{N}^* \to \mathbb{N}$ *is a bijection.*

Proof. Suppose

$$\langle a_0, \ldots, a_{n-1} \rangle = \langle b_0, \ldots, b_{m-1} \rangle$$

We may safely assume $0 < n \leq m$ (why?).

Since π is a pairing function, we get $a_0 = b_0$ and $\langle a_1, \ldots, a_{n-1} \rangle = \langle b_1, \ldots, b_{m-1} \rangle$.

By induction, $a_i = b_i$ for all $i = 1, \ldots, n-1$ and $0 = \langle \text{nil} \rangle = \langle b_n, \ldots, b_{m-1} \rangle$. Hence $n = m$ and our map is injective.

Exercise

Prove that the function is surjective.

Exercises

Exercise

Construct a register machine that computes the length function len

Exercise

Construct a register machine that computes the decoding function dec

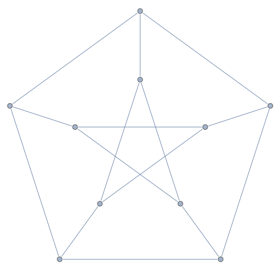
Exercise

Construct a register machine that uses coding to computer the prime decomposition of a number.

That's It!

We can now code any discrete structure as an integer by expressing it as a nested list of natural numbers, and then applying the coding function.

For example, the so-called Petersen graph on the left is given by the edge list, a nested list of integers, on the right.



$$((1,3),(1,4),(2,4),(2,5),(3,5),\\(6,7),(7,8),(8,9),(9,10),(6,10),\\(1,6),(2,7),(3,8),(4,9),(5,10))$$

Coding Petersen

The code numbers of the edges are

$$34,66,258,132,260,1028,520,4104,16400,\\65568,16448,131136,65664,262400,1049088$$

Unfortunately, coding up this sequence produces a number with 485598 decimal digits.

Coding vs Data Structures

To be clear: coding is a conceptual tool.

For efficient computation one has to deal with the discrete structures (words, lists, trees, graphs, hash tables, DFAs, TMs . . .) directly. That's the job of a programming language.

But as a matter of principle, it suffices to understand computable arithmetic functions

$$f : \mathbb{N}^k \to \mathbb{N}$$

Register machines are very good at this, Turing machines not so much.

Aside: Gödel's Incompleteness Theorem

Gödel's theorem (1931) is arguably the most important result in proof theory: for any formal system Γ that captures just a bit of arithmetic, there is a formula φ that is true, but not provable in Γ.

The details are very technical, but the proof uses two central ideas that we are familiar with:

- Coding: one can express logical formulae and proofs as numbers, and one can manipulate these numbers just the way one can manipulate formulae and proofs.
- Diagonalization: using this coding schema, one can set up a formula that says, in essence: "I am not provable."

Exercises

Exercise

Give a proof that our coding function is surjective.

Exercise

Find unpairing functions for our pairing function, and for Cantor's pairing function.

Exercise

Come up with another pairing function and corresponding unpairing functions.

Exercise

Show how to check if a number is a code number given dec *and* len*. What properties do you need?*

Turing Machines are Discrete

Claim

A Turing machine is a finite, discrete structure.

But that means we can associate every TM M with a code number $\widehat{M}$, a so-called index for M.

So we obtain an enumeration M_0, M_1, M_2, ... of all TMs. If e is not an index, let M_e be some default machine.

It's Effective

The point is that we can compute with an index instead of the actual machine.

For example, there is an easily computable function

$$\mathsf{sc} : \mathbb{N} \longrightarrow \mathbb{N}$$

so that sc$(\widehat{M})$ is the number of states of TM M. Similarly, there is a simple function

$$\mathsf{comp} : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$$

such that comp$(\widehat{M_1}, \widehat{M_2})$ is the index of the machine obtained by running machine M_2 after M_1 (sequential composition).

An Interpreter

How about a function

$$\mathsf{eval} : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$$

such that eval$(\widehat{M}, x) = M(x)$?

That's a bit more complicated to figure out in detail, but not really all that hard.

Note that eval is computable, otherwise there would be no CS departments.

A Conundrum

Since eval is computable, so is the function

$$f(x) := \mathsf{eval}(x, x) + 1$$

f may not make much sense as an arithmetic function, but it is perfectly well-defined, and certainly computable. Needless to say, this is another example of diagonalization.

But then there must be a TM that computes it, say, a machine with index $\widehat{f}$. Alas, now we're in trouble:

$$\mathsf{eval}(\widehat{f}, \widehat{f}) = f(\widehat{f}) = \mathsf{eval}(\widehat{f}, \widehat{f}) + 1$$

☠

The Fix

The only way to solve this problem is to give up on total functions: we must consider partial functions that can be undefined on some inputs.

When dealing with partial functions it is better to write things like

$$f(x) \simeq g(x)$$

with the intended interpretation:

- Both sides are defined and equal, or
- both sides are undefined.

So $\mathsf{eval}(\widehat{f}, \widehat{f}) \simeq \mathsf{eval}(\widehat{f}, \widehat{f}) + 1$ and everything is fine.

Halting

Dealing with general computable functions automatically leads to partial functions, there is no way around this: otherwise we would have to give up on eval.

There are important subclasses of computable functions that are all total, but in the general case we are stuck with partial functions. So we naturally run into the following decision problem:

Problem:	**Halting**
Instance:	An index e, an argument x.
Question:	Does M_e halt on input x?

Decidability

It would be nice to be able to check whether $M(x)$ converges before actually running the computation. Alas

Theorem

The Halting Problem is undecidable.

Proof.

Suppose that there is a Turing machine `halt(e,x)` that solves the HP.

Then we can cobble together the following machine:

In Pseudo-Code

```
// evil TM
// input: e

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

Here `halt(e,x)` is the halting tester that exists by assumption, and `eval(e,x)` is our interpreter: it runs M_e on input x.

But . . .

ETM has some index $\widehat{e}$.

- ETM halts on all inputs, including $\widehat{e}$.
- On input $\widehat{e}$, ETM returns eval$(\widehat{e}, \widehat{e}) + 42$, but it is supposed to return eval$(\widehat{e}, \widehat{e})$.

Thus ETM is contradictory, and our initial assumption about `halt(e,x)` is wrong.

Stronger Result

Our proof actually shows that the following version of Halting is already undecidable.

Problem:	**Halting′**
Instance:	An index e.
Question:	Does M_e halt on input e?

Note that Halting is at least as difficult as Halting′: any algorithm for Halting automatically also solves Halting′.

A Reduction

Perhaps surprisingly, we can also show that Halting′ is at least as difficult as Halting:

Given e and x, an instance of Halting.

Construct a TM M' with index e' that, on input z, runs M_e on x (thus M' ignores its input).

Then (e, x) is a Yes-instance of Halting iff e' is a Yes-instance of Halting′.

Clearly, e' is computable from e and x.

So if we know that Halting′ is undecidable, it follows that Halting is also undecidable.

Nothing New

Every CS person is familiar with this approach: we want to solve problem A, and we already have a powerful algorithm in a library that solves problem B.

Rather than trying to find an algorithm for A directly, we translate an instance of problem A into an equivalent instance of problem B, and then use the given algorithm.

For example, removing duplicates from a list easily reduces to sorting.

Too Self-Absorbed?

Some might object that Halting is a bit too much concerned about computability to count as a compelling example of a natural undecidable problem.

Fine, but there are lots of other decision problems that, on the face of it, have nothing to do with computation, but are similarly undecidable.

A good test is the following: the problem should have been invented by someone who is not a recursion theorist (computability expert).

Hilbert's 10th Problem

Perhaps the most famous example of an undecidability result in mathematics is Hilbert's 10th problem, the insolubility of Diophantine equations, announced by Hilbert in 1900.

A Diophantine equation is a polynomial equation with integer coefficients:

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

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.

Different Numbers

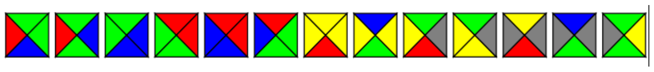
Note that the choice of $\mathbb{Z}$ as ground ring is important here. We can ask the same question for polynomial equations over other rings R (always assuming that the coefficients have simple descriptions).

- $\mathbb{Z}$: 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.

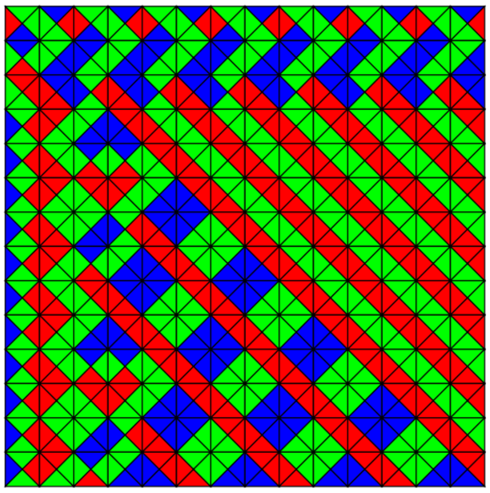
Wang Tiles

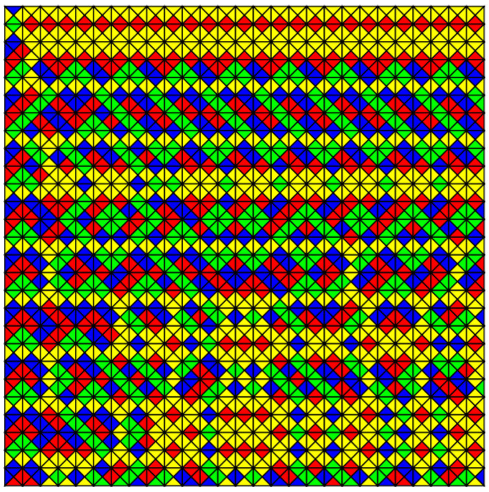
Here is a more geometric problem: we are given a collection of tiles, unit squares that are colored as follows:

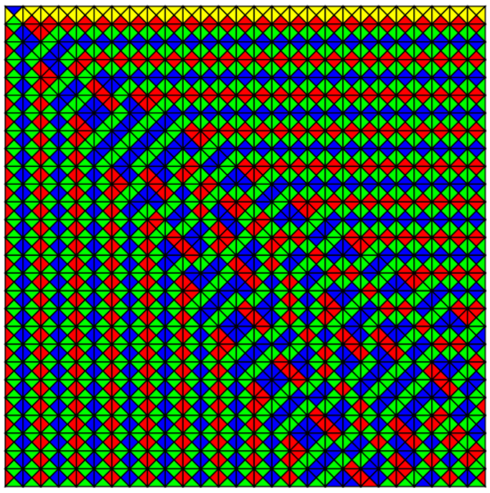


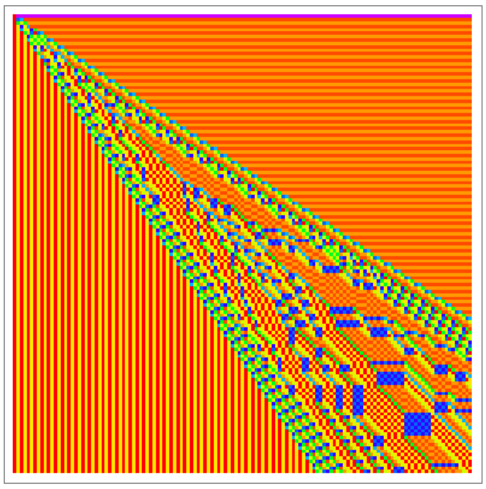
There is an infinite supply of each type of tile, rotations and reflections are not allowed.

The goal is to cover the whole plane with these tiles, in a way that all adjacent colors match.









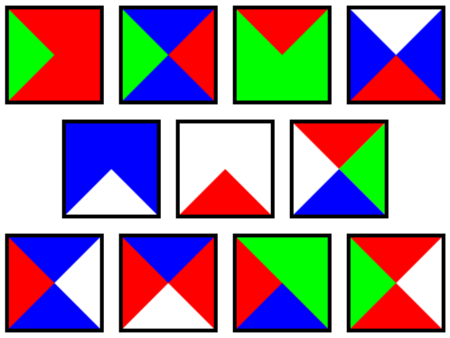
Aperiodic Tiling

Theorem (R. Berger 1966)

The Tiling Problem is undecidable.

This result is based on a surprising and somewhat counterintuitive fact: there are sets of Wang tiles that admit a tiling of the whole plane, but not a periodic one (Hao Wang originally conjectured in 1961 that such tiles to not exist).

Aperiodic Wang Tiles



Interestingly, small sets of Wang tiles can be constructed using finite state machines and old-fashioned hyperbolic geometry.

So What Is It?

We have a negative characterization for Halting: it's undecidable.

But how about a positive characterization?

Definition

A set A is semidecidable if there is a Turing machine M that, on input x, halts if $x \in A$, and diverges otherwise.

You can think of this machine as a broken decider: it only works properly on Yes-instances, but fails completely on all No-instances.

Note: Halting is semidecidable, as are the Diophantine equation and Tiling problems.

Decidable vs. Semidecidable

In many ways, semidecidable sets are more fundamental than decidable ones.

Lemma

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

Proof.

Left to right is trivial.

Right to left: Beware the perils of sequential computation.

□

Closure

Lemma

Semidecidable sets are closed under union and intersection.

Proof.

Argue about running the corresponding TMs (in parallel for union).

□

Dire Warning: But they are not closed under complement: otherwise the Halting set would be decidable.

Hilbert's 10th: The Proof

The full proof is too complicated to be presented here, but the main idea is a reduction:

> *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

Even numbers: $a = 2x$.

Non-zero: $ax = (2y-1)(3z-1)$.

Naturals (Lagrange): $a = x_1^2 + x_2^2 + x_3^2 + x_4^2$.

Closure under intersection (sum of squares) and union (product).

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

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

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.

Theorems

One of the reasons semidecidability is so important is that, given a formal axiomatic system Γ, the set of theorems provable in Γ is always semidecidable: essentially, a TM can just search over all possible proofs in the system.

In some cases, the set of theorems is actually decidable, but the decision algorithm can be quite complicated.

Example

- Theory of Abelian groups: decidable.
- Theory of groups: undecidable.

So computability provides a tool to classify arbitrary mathematical domains, even those that seem to have nothing to do with computation.

Definition: Semi-/Decidability