COMP30026 Models of Computation

Undecidable Languages

Harald Søndergaard

Lecture 22

Semester 2, 2018

An Undecidable Language

Now let us study undecidable problems/languages.

We start by showing that it is undecidable whether a Turing machine accepts a given input string. That is,

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$$

is undecidable.

The main difference from the case of A_{CFG}, for example, is that a Turing machine may fail to halt.

TM Acceptance Is Undecidable

Theorem:

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$$

is undecidable.

Proof: Assume (for contradiction) that A_{TM} is decided by a TM H:

$$H\langle M, w\rangle = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

Using H we can construct a Turing machine D which decides whether a given machine M fails to accept its own encoding $\langle M\rangle$:

1. Input is $\langle M\rangle$, where M is some Turing machine.
2. Run H on $\langle M, \langle M\rangle\rangle$.
3. If H accepts, reject. If H rejects, accept.

TM Acceptance

In summary:

$$D(\langle M \rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ does not accept } \langle M \rangle \\ \textit{reject} & \text{if } M \text{ accepts } \langle M \rangle \end{cases}$$

But no machine can satisfy that specification!

Why? Because we obtain an absurdity when we investigate D's behaviour when we run it on its own encoding:

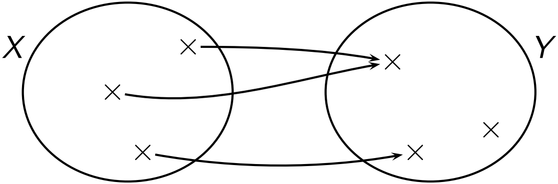
$$D(\langle D \rangle) = \begin{cases} \textit{accept} & \text{if } D \text{ does not accept } \langle D \rangle \\ \textit{reject} & \text{if } D \text{ accepts } \langle D \rangle \end{cases}$$

Hence neither D nor H can exist.

Injections, Surjections and Bijections

Recall that a function $f : X \to Y$ is

- surjective (or onto) iff $f[X] = Y$.
- injective (or one-to-one) iff $f(x) = f(y) \Rightarrow x = y$.
- bijective iff it is both surjective and injective.



Inverse Function

Given $f : X \to Y$, a function $g : Y \to X$ is its inverse iff $g \circ f = 1_X$ and $f \circ g = 1_Y$.

An inverse function, if it exists, is unique.

A function has an inverse iff it is bijective.

If $f : X \to Y$ is a bijection, we denote its inverse by $f^{-1} : Y \to X$.

Bijections and Enumerations

In Lecture 12 we looked at the bijection $d : \mathbb{Z} \to \mathbb{N}$ defined by

$$d(n) = \begin{cases} 2n - 1 & \text{if } n > 0 \\ -2n & \text{if } n \leq 0 \end{cases}$$

Its inverse function $e : \mathbb{N} \to \mathbb{Z}$ is

$$e(n) = \begin{cases} (n+1)/2 & \text{if } n \text{ is odd} \\ -n/2 & \text{if } n \text{ is even} \end{cases}$$

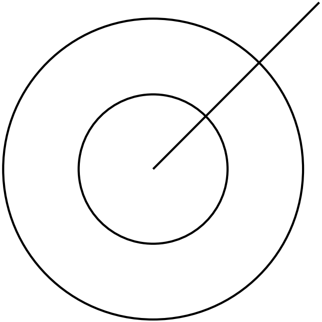
A bijection in $\mathbb{N} \to X$ gives us an enumeration of the set X.

e gives an enumeration of $\mathbb{Z}$, namely $0, 1, -1, 2, -2, 3, -3, 4, \ldots$

Galileo's Paradox

$\mathbb{N}$ and the set of perfect squares are in a one-to-one relation: f defined by $f(n) = n^2$ is a bijection.

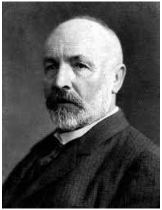
$$\begin{array}{cccccccc} \{ & 0 & 1 & 2 & 3 & 4 & 5 & \ldots\} \\ & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \\ \{ & 0 & 1 & 4 & 9 & 16 & 25 & \ldots\} \end{array}$$



Galileo: The outer circle has twice as many points as the inner circle, as the ratio between the circumferences is 2. Yet considering radial lines shows a one-to-one relation.

Comparing Sets: Cantor's Criterion

So what does 'equals' and 'less' mean for infinite cardinality?



How do we compare the "sizes" of infinite sets?

Cantor's criterion:

- $card(X) \leq card(Y)$ iff there is a total, injective $f : X \to Y$.
- $card(X) = card(Y)$ iff

  $card(X) \leq card(Y)$ and $card(Y) \leq card(X)$.

As a consequence, there are (infinitely) many degrees of infinity.

To Infinity and Beyond

X is countable iff $card(X) \leq card(\mathbb{N})$.

X is countably infinite iff $card(X) = card(\mathbb{N})$.

Examples: $\mathbb{Z}, \mathbb{N}^k$, and $\mathbb{N}^*$ (the set of all finite sequences of natural numbers) are all countably infinite.

Importantly, Σ^* is countable for all finite alphabets Σ, including the alphabet of characters on my keyboard.

$\mathcal{P}(\mathbb{N})$, $\mathbb{N} \to \mathbb{N}$, and $\mathbb{Z} \to \mathbb{Z}$ are uncountable, as can be shown by diagonalisation.

Diagonalisation

The proof that A_{TM} is undecidable uses a technique (somewhat disguised) that is called diagonalisation.

Diagonalisation gives us a way of proving certain sets uncountable.

Diagonalisation Showing $\mathbb{Z} \to \mathbb{Z}$ Is Uncountable

Theorem: There is no bijection $h : \mathbb{N} \to (\mathbb{Z} \to \mathbb{Z})$.

Proof: Assume h exists. Then

$$h(1), h(2), \ldots, h(n), \ldots$$

contains every function in $\mathbb{Z} \to \mathbb{Z}$, without duplicates.

Now construct $f : \mathbb{Z} \to \mathbb{Z}$ as follows:

$$f(n) = h(n)(n) + 1$$

Then $f \neq h(n)$ for all n, so we have a contradiction.

Why This Is Called Diagonalisation

Here is some hypothetical listing of all the functions $h(1), h(2), \ldots$ that make up $\mathbb{Z} \to \mathbb{Z}$:

	1	2	3	4	5	6	...
h(1)	19	3	42	0	7	9	...
h(2)	42	42	42	42	42	42	...
h(3)	42	43	44	45	46	47	...
h(4)	6	93	17	84	6	93	...
h(5)	45	18	-8	-5	63	-9	...
⋮							

Why This Is Called Diagonalisation

Here is some hypothetical listing of all the functions $h(1), h(2), \ldots$ that make up $\mathbb{Z} \to \mathbb{Z}$:

	1	2	3	4	5	6	...
h(1)	19	3	42	0	7	9	...
h(2)	42	42	42	42	42	42	...
h(3)	42	43	44	45	46	47	...
h(4)	6	93	17	84	6	93	...
h(5)	45	18	-8	-5	63	-9	...
⋮							

f is defined in such a way that it cannot possibly be in the listing:

	1	2	3	4	5	6	...
f	20	43	45	85	64	...	...

Algorithms vs Functions

Consider the set of algorithms that realise functions $f : \mathbb{Z} \to \mathbb{Z}$.

How large is that set?

It is infinite, but we can enumerate it. It is contained in Σ^*, where Σ is the set of (printable) characters on my keyboard and as we have seen, that set is countable.

So there cannot be any more, say, Haskell functions, of type `Integer -> Integer` than there are integers. Namely, each Haskell function is represented finitely, as a finite sequence of symbols from a finite alphabet.

Algorithms vs Functions

However, we saw that $\mathbb{Z} \rightarrow \mathbb{Z}$ is not countable.

In other words, there are number-theoretic functions (in fact, lots of them) that do not have a corresponding algorithm.

So are there any "important" functions that are not computable?

As it turns out, yes, very much so!

Problems that Have No Algorithmic Solution

Some undecidable problems:

- Are two given CFGs equivalent?
- Are there strings that a given CFG cannot generate?
- Is a given CFG unambiguous?
- Will a given Python program halt for all input?
- Will it halt on input 42?
- Will a given Java program ever throw a certain exception?

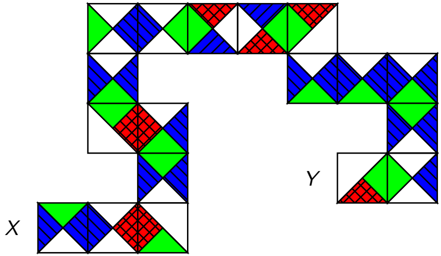
In the next lecture we explore more undecidable problems.

Domino Snakes

Consider a finite set of types of tiles ⊠.

There are infinitely many tiles of each type.

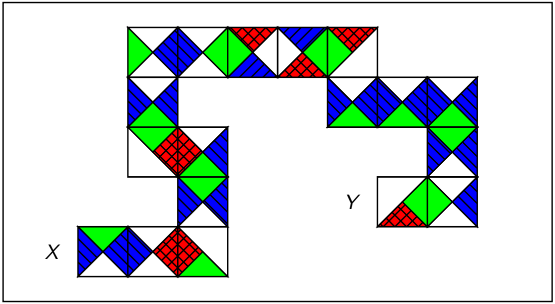
Can points X and Y in the plane be connected?



The (unconstrained) problem is decidable.

Domino Snakes

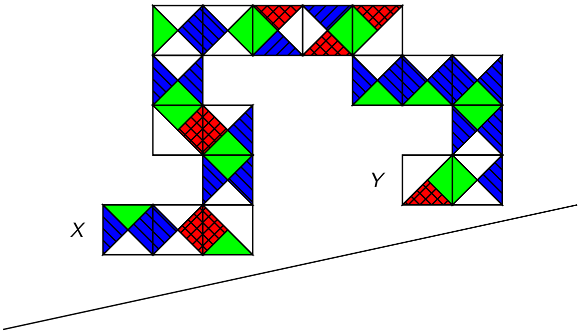
Can X and Y be connected?



In finite segment of plane: also decidable.

Domino Snakes

Can points X and Y be connected?



In half-plane: Undecidable!

Intuition is sometimes a poor guide to decidability.

Busy Beavers (Not Examinable)

I will make a video available (optional viewing) with an interesting example of an uncomputable function, and a proof of the undecidability of Turing machine halting-on-empty-input. You can find the slide set that I use in the video where all the other slides are.

In the next lecture we will use the technique of reduction to find a bunch of other interesting undecidable problems.