Algorithms and Complexity, weeks 1-5 key points

Sam Barrett

March 8, 2021

Turing machine basics I

- Turing machines are **precise models of computation**
- First tape of a Turing machine is **always** a read-only **input tape**.
- The k^{th} tape is always the output tape
- The output tape is also considered a work tape
- The alphabet of a TM is denoted as Γ, it is **finite**.
- Each tape must always start with the *left-of-tape* symbol, $\triangleright$
- $\{0, 1\}^*$ is the set of bitstrings, with ε denoting the empty string.

Turing machine basics II

In a single stage of computation a TM may:

- reads the character at each tape head
- writes a character at each work tape head
- may move each tape head to the left or to the right. **note: our tapes are not recursive, if a head on the leftmost cell moves left, it stays put**

what does it mean to say that M **computes** f?

It means that for every bitstring $x \in \{0,1\}^*$, if we start in state $q_{\texttt{start}}$ with the initial configuration showing x (meaning x appears on the input tape and and the work tapes are blank), when we run M, we eventually reach $q_{\texttt{halt}}$ with the output tape showing $\triangleright$ on the leftmost cell and then the bitstring $f(x)$ followed by all blanks.

Computable functions

Basics

Definition

(Computable functions) We say a function $f : \{0,1\}^* \to \{0,1\}^*$ is **computable** if there exists some Turing machine that computes it and **non-computable** if there isn't.

No variation of Turing machine affects this fact, giving us:

Thesis

(Church's Thesis) any algorithm that computes a function from bitstrings to bitstrings can be converted into a Turing machine that computes the same function

Boolean functions, languages and decidability I

Definition

A *language* can be defined as any set of words

Definition

A **boolean function** is a function of the form:
$f : \{0,1\}^* \to \{0,1\}$. Noting that the output is a single bit rather than a bitstring.

There is a one-to-one correspondence between languages and boolean functions.

- For a given boolean function f the corresponding language is the set of bitstrings x s.t. $f(x) = 1$
- For a language L, the corresponding boolean function sends x to 1 if $x \in L$ and to 0 otherwise.

Boolean functions, languages and decidability II

This allows us to treat boolean functions, languages and decision problems as essentially the same thing.
A decision problem is said to be **decidable** when the corresponding boolean function is **computable**. I.e. given a language L, for L to be decidable there must exist some Turing machine that will start with a bitstring x and will run continuously until it halts and upon halting there will be a 1 on the output tape if $x \in L$ or 0 if it is not in the language.

Data Representation

- We can encode many real-life data types as bitstrings, but not all. (e.g. Real numbers cannot)
- We can encode multiple inputs as a single bitstring.

Code as Data

- We can not only encode many data types as bitstrings, we can encode program code or even other Turing machines as bitstring inputs to a TM as our TMs are essentially 6-tuples (see full notes for formal definition).
- We can therefore say that for **any** bitstring α we can construct a corresponding TM: M_α

The Universal Turing Machine, $\mathcal{U}$

$\mathcal{U}$ is a Turing machine interpreter written as a Turing machine. It takes 2 inputs (encoded as a single input): α and x, where α is the bitstring describing the machine to be interpreted and x is the bistring input.
We define $\mathcal{U}$ as having 4 tapes and the basic alphabet of $\{\triangleright, \square, 0, 1\}$. Intuitively, $\mathcal{U}$ works by simulating the M_α by *providing* it with the 3 non-input tapes to M_α as its input, work and output tape respectively.

Diagonalisation & the Halting problem I

Problem

(The halting problem) the set of pairs $\langle \alpha, x \rangle$ *(encoded as a single bitstring) such that the machine* M_α *executed on input* x *halts.*

Diagonalisation & the Halting problem II

Turing's proof is as follows:

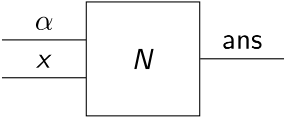
Proof.

Suppose that N is a machine that solves the halting problem.
We can convert it into a machine N' that, given x, runs forever if $\langle x, x \rangle \in \texttt{HALT}$ (i.e. the machine M_x executed on x halts), and halts otherwise.
We know $N' = M_\alpha$ for some α, i.e. there exists some bitstring α that represents our new machine, as we know every machine can be represented as a bitstring.
Running N' on α halts if it runs forever and runs forever if it halts.
We have derived a contradiction.

$\square$

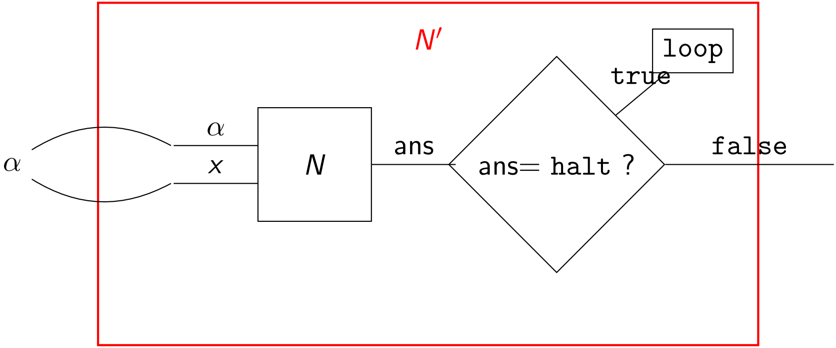
Diagonalisation & the Halting problem III

More intuitively, we can consider the described machine N as:



Where `ans` is whether N halts.
We can then construct the wrapper N' as:

Diagonalisation & the Halting problem IV



Where the outermost α is the bitstring of N'
You can clearly see that if N were to halt then N' would hang. We can therefore derive a contradiction as if we pass N' into N' it would have to half if it hangs and vice versa!

Upper bound notation

- If we have two function $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$
- We say that $f(n)$ is $O(g(n))$ is f is **no bigger** than g up to **a** constant factor, i.e. given a constant c and a value n_0 s.t. $\forall n, n \geq n_0$ we have:

$$f(n) \leq c \cdot g(n)$$

- We say that $f(n)$ is $o(g(n))$ if f is **not as big as** g, even up to any constant factor. Or, if, **for any** $\varepsilon > 0$, there is a n_0 s.t. $\forall n, n \geq n_0$ we have:

$$f(n) \leq \varepsilon \cdot g(n)$$

  Whenever $f(n)$ is $o(g(n))$ is it also $O(g(n))$, this is the case if you take c to be 1

Lower bound notation

- If we have two function $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathbb{N} \to \mathbb{N}$
- We say that $f(n)$ is $\Omega(g(n))$ when $g(n)$ is $O(f(n))$
- We say that $f(n)$ is $\omega(g(n))$ when $g(n)$ is $o(f(n))$
- We say that $f(n)$ is $\Theta(g(n))$ when it is **both** $O(g(n))$ and $\Omega(g(n))$
  This informally means $f(n)$ *and* $g(n)$ *are the same, up to a constant factor*