Computation

ian.mcloughlin@gmit.ie

***Abstract*—The following document contains a set of notes about computation. We consider the efficiency of computations and also problems that are undecidable.**

I. Languages

Start with any set, call it an alphabet and denote it by A. For example:

$$A = \{0, 1\}$$

We define the strings of length i over A recursively:

$$A^0 = \{\epsilon\}$$
$$A^1 = A$$
$$A^i = \{as \mid a \in A, s \in A^{i-1}\}$$

The Kleene star A^* of the alphabet A is then union of all A^i:

$$\bigcup_{i \in \mathbb{N}_0} A^i$$

Here $\mathbb{N}_0$ is the set of natural numbers including zero:

$$A^* = \mathbb{N}_0 = \{0, 1, 2, 3, \ldots\}$$

A language L over an alphabet A is a subset of A^*:

$$L \subseteq A^*$$

II. Turing machines

Turing machines encapsulate the concept of a computer. They are simple machines, consisting of a single read/write head and an infinite tape of cells, all but a finite number of which are blank. The head is over a single cell of the tape and is in any one of a finite number of states at a given point in time. Turing machines perform one small step at a time, which involves reading the symbol in the current cell, overwriting it with a symbol, moving to the cell to the left or right, and changing state.

We define a (deterministic[1]) Turing machine M by a 7-tuple:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_a, q_f)$$

where: Q is a set of states; Σ is the input alphabet, not containing the blank symbol $\sqcup$; Γ is the tape alphabet, a superset of Σ containing $\sqcup$; δ is a map:

$$\delta : (Q \setminus \{q_a, q_f\}) \times \Gamma \to \Gamma \times \{L, R\} \times Q;$$

q_0 is the initial states; q_a is the accept state, which is a terminal state; q_f is the reject state, also a terminal state.

Before the Turing machines begins its operation, a finite number of consecutive non-blank cells contain symbols and the head is over the left-most of these. These non-empty cells form the Turing machine's input and can be viewed as a string.

When the Turing machine operates on an input, there are three possibilities. The first two are similar: the machine can accept the input by ending in the accept state or the machine can reject the input by ending in the reject state. In these cases the machine stops upon entering these states, which is usually called halting. The third possibility is that the machine might never stop operating.

The subset of Σ^* (the Kleene star of the input alphabet) containing all elements that are accepted by the Turing machine M is called the language of the Turing machine $L(M)$:

$$L(M) \subseteq \Sigma^*$$

A Turing machine that halts on all inputs is called a decider and is said to decide its language. Note it's possible for two distinct Turing machines M_1 and M_2 to accept or decide the same language: $L(M_1) = L(M_2)$.

III. State tables

A Turing machine can be summarised in five-columned table called a state table. Each row of the table describes how to perform the transition function δ for a given state and tape symbol. Table I gives an example of a state table for a Turing machine that accepts all strings over the alphabet $\{0, 1\}$ that contain an even number of 1's, including zero as an even number.

State	Input	Write	Move	Next
q_0	0	0	R	q_0
q_0	1	1	R	q_1
q_0	$\sqcup$	$\sqcup$	L	q_a
q_1	0	0	L	q_1
q_1	1	1	L	q_0
q_1	$\sqcup$	$\sqcup$	R	q_f

TABLE I: Turing machine state table

Provided we adopt some conventions, the state table can be used to describe the whole Turing machine. First, the initial state is the first one listed in the table. Second, the tape alphabet is given by the distinct symbols in the second column. Finally, the only difference between the input and tape alphabets is the blank symbol. Note that there is single row for each combination of state and tape symbol.

[1]All Turing machines will be assumed to be deterministic until we get to section VII.

IV. Decision problems

A decider makes a binary choice for a given input: accept or reject. The Turing machine M in this case performs a map:

$$f : \Sigma^* \to \{q_a, q_f\}$$

Sometimes this map is called an indicator function, as it indicates which elements of Σ^* are in M. In general, a map from a set to any set with two elements is called a decision problem. So, a decider performs a decision problem.

V. Complexity

A decider takes a finite number of steps before halting on a given input. We are typically interested in knowing the maximum number of steps $f(n)$ the decider takes on all inputs of length n.

We can often express $f(n)$ using a simple formula, such as $f(n) = 2n + 2$. We are interested in knowing how $f(n)$ grows in relation to n. For this, we use big-O notation:

Definition. *A function $f(n)$ is said to be $O(g(n))$ if there exist two positive integers c and n_0 such that $cg(n) \geq f(n)$ for all $n \geq n_0$.*

In the previous example, $f(n) = 2n+2$ is $O(n)$ which can be seen by setting $g(n) = n$, $c = 3$ and $n_0 = 2$ as per Fig. 1.

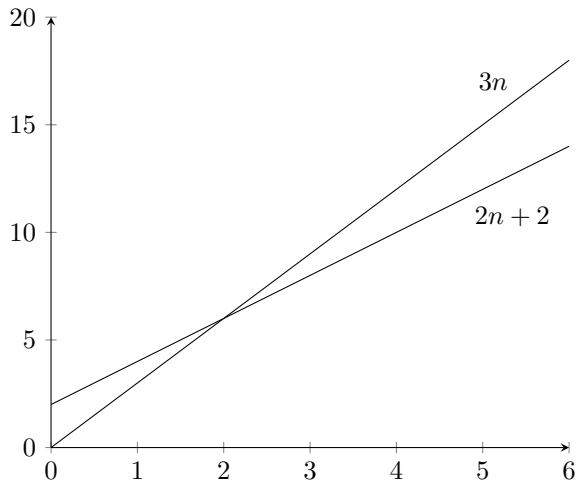

Fig. 1: $2n + 2$ is $O(n)$

VI. P

We denote by P the set of languages that are each decidable by some Turing machine in $O(n^k)$ steps for some $n \in \mathbb{N}$ where $\mathbb{N}$ is the set of natural numbers without zero, $\{1, 2, 3, \ldots\}$. The P here is short for polynomial.

Here, we define a polynomial is an expression in a variable n made up of constants (elements on $\mathbb{N}$), powers of n to a constant, addition, subtraction and multiplication. For example, $n^5 + 2n + 4$ is a polynomial whereas $2^n + n^2 + 5$ is not because it is made up of something other than those elements listed above, namely 2^n.

VII. Non-determinism

We consider now a modification to Turing machines: we allow the Turing machine to transition to zero or more states simultaneously from a given state when reading a given symbol. This requires a modification to the transition function as follows:

$$\delta : Q \times \Gamma \to \Gamma \times \{L, R\} \times \mathcal{P}(Q)$$

where $\mathcal{P}(Q)$ is the powerset of Q — the set of all subsets of Q. From the state table point of view, the effect is that there may be zero, one, or more rows for each combination of state and tape symbol. In practice, we can view this as the Turing machine being able to fork, at will, into two or more branches of computation. Note that every deterministic Turing machine is basically a non-deterministic Turing machine[2].

We say an input is accepted by a non-deterministic Turing machine if any of its branches end in the accept state. It may be that other branches end in the fail state or compute forever. If all branches halt for all inputs, we say the non-deterministic Turing machine decides its language.

How does non-determinism affect the computations Turing machines can perform? The bad news is that this doesn't give a Turing machine any extra power. Turing proved in his 1936 paper *On computable numbers with an application to the encheidungsproblem* that some languages are not the language of any Turing machine, let alone decidable by a Turing machine. Allowing non-determinism does not change the languages that can be accepted or decided.

It's not known whether computations can be done more efficiently on non-deterministic Turing machine than on deterministic ones. Examples abound of efficient non-deterministic Turing machines that accept or decide languages for which no known efficient deterministic Turing machine is known. However, it is not known whether such deterministic Turing machines exist. A special case of this question is the P versus NP problem.

VIII. NP

The NP complexity class is the non-deterministic equivalent of the deterministic P complexity class.

IX. Church-Turing thesis

X. Undecidable problems

XI.

[2]You do have to change the transition function to map to $\{q\}$ rather than q, but that's a trivial change.