Models of Computation

Lectures by S Dantchev
Notes by Ben Napier

2019 Michaelmas Term

Contents

Chapter 1

Turing machines

Lecture 1
On 7/10

A **turing machine** is the first rigorous definition of computation. A turing machine has an infinite tape (acting as memory). There is a finite-state program that controls a tape head. The head can read, write, and move around (in both directions) on the tape.

Example. A typical program instruction: if the finite control is in state p and the head reads b, then write a, move the head to the left and go to state q.

Definition 1.1 (Turing machine)**.** A turing machine is a 7-tuple

$$(Q, \Sigma, \Pi, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}).$$

(i) Q is the set of states.

(ii) Σ is the input alphabet (not including the blank symbol).

(iii) Γ is the tape alphabet satisfying $\Sigma \subset \Gamma$ and $\sqcup \in \Gamma$.

(iv) $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function.

(v) $q_0 \in Q$ is the initial state.

(vi) q_{accept} is the accept state, and

(vii) q_{reject} is the reject state, note that $q_{\text{reject}} \neq q_{\text{accept}}$.

Turing machines solve decision problems, it will take an input and reply with yes (when in the accept state) or no (when in the reject state).

The tape content is unbounded but always finite. The first blank character marks the end of the tape content.

Remark. The distinction between the input and tape alphabet is that the input alphabet is the set of all symbols that can be found on the initial tape contents, where the tape alphabet is all the possible symbols that can be on the tape contents. So $\Gamma \setminus \Sigma$ is the set of symbols that can only be introduced to a tape by writing.

Definition 1.2 (Configuration)**.** A configuration consists of three items: the **current state**, the **tape content**, and the **head location**.

We say that a configuration C_1 **yields** a configuration C_2 is you can go from C_1 to C_2 is a single step.

The **start configuration** on an input $w \in \Sigma^\star$ consists of the start state q_0, the tape content w, and the head location being the first position of the tape.

An **accepting configuration** is a configuration whose state is q_{accept} and a similar definition exists for a **rejecting configuration**. Accepting and rejecting configurations are **halting configurations**.

We have a special way of denoting configurations. For a state q with strings u, v, we write uqv to denote the configuration with tape contents uv in state q where the head location is at the first symbol in v.

Definition 1.3 (Accepting input)**.** We say that a turing machine M **accepts** an input w if there exists a sqeuence of configuration $(C_1, C_2, \dots, C_k$ where $k \in \mathbb{N}$ such that

(i) C_1 is the start configuration of M on input w,

(ii) C_i yields C_{i+1} for all $0 \leq i < k$; and

(iii) C_k is an accepting configuration.

This is just a formal way of saying that an input is eventually accepted by a Turing machine.

Definition 1.4 (Language of a turing machine)**.** The set of strings accepted by a turing machine M is called the **language of** M and is denoted by $L(M)$.

Example. Here we describe a turing machine M that describes $A = \{0^{2^n} : n \geq 0\}$. That is, an input which is purely 0s that has a length of a non-negative integer power of 2.

(i) Sweep left to right accross the tape, crossing off every other 0.

(ii) If in stage (i) the tape contained a single 0, aceept.

(iii) If in stage 1 the tape contained more than a single 0 and the number of 0s was odd, reject.

(iv) Return the head to the left-hand end of the tape.

(v) Go to stage 1.

Lecture 2
On 10/10

A note on notation, we may use $S^\star$ to denote a finite length tuple containing only elements in S. This is not the same as the complex analysis definition where $\mathbb{C}^\star = \mathbb{C} \setminus \{0\}$. For example, $(0, 0, 1, 1, 0, 1) \in \{0, 1\}^\star$.

Definition 1.5. A language $\mathcal{L}$ is **Turing-Recognisable** if there exists a turing machine $\mathcal{M}$ that recognises it. That is, $\mathcal{L} = L(\mathcal{M})$.

Definition 1.6. A language $\mathcal{L}$ is **Turing-Decideable** if there exists a turing machine $\mathcal{M}$ that accepts all $w \in \mathcal{L}$ and rejects all $w \notin \mathcal{L}$.

So we say that $\mathcal{M}$ recognises $\mathcal{L}$ if it accepts every word in $\mathcal{L}$, but $\mathcal{M}$ decides $\mathcal{L}$ if it accepts every word and rejects every word not in $\mathcal{L}$.

You may see the terminology 'r.e.' (stands for **recursively enumerable**) instead of Turing-Recognisable and **recursive** instead of Turing-Decidable.

Remark. The set of all turing machines is countable, we will return to this.

Definition 1.7 (Multitape turing machines)**.** A **multitape turing machine** is the same as a regular turing machine with several tapes, each of them having its own head. In the formal definition, we have the following transition function

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k.$$

Theorem 1.8. Every multiple tape turing machine has an equivalent single tape turing machine.

Proof. Say we have k tapes $w_1, w_2, \ldots, w_k$. We then consider the tape

$$w_1 + \# + w_2 + \# + \ldots + \# + w_k.$$

So we have each tape delimited by a special character $\#$. We then alter our alphabet slightly. Consider Γ as the alphabet of the single tapes. Then the alphabet for our multitape is Γ' where $\Gamma \subset \Gamma'$ and

$$\forall\, \gamma \in \Gamma\ \exists\, \gamma' \in \Gamma'$$

where γ' is a special copy of each symbol used to represent the head in the multitape for each individual tape. We must now modify our transistion function to handle the movement between tapes. But here, we are at the point where a single tape turing machine now replicates a multitape turing machine. $\square$

This proof shows how we talk about Turing machines. We do not need to be rigorous (although we can) in these proofs, and we build upon previous theorems a lot in our proofs. So we know now that if we can find a multitape Turing machine that can execute something, we can find a single tape Turing machine to do the same thing.

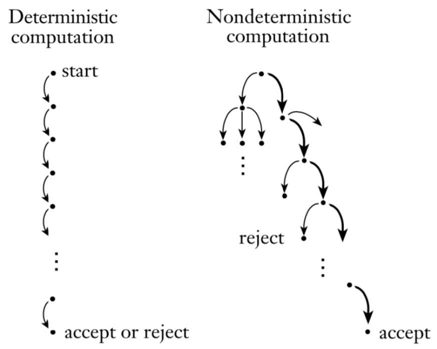

Figure 1.1: Deterministic and non-deterministic Turing machine tree diagram.

Definition 1.9. A **non-deterministic turing machine** has a transition function of the form

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

where $\mathcal{P}$ represents the power set function.

You can see a graphical representation of a non-deterministic Turing machine compared to a deterministic one. Informally, in a non-deterministic Turing machine there may exist several choices for the next state (and movement / writing of the head).

Theorem 1.10. Every non-deterministic turing machine has an equivalent deterministic turing machine.

Proof. The general idea for this proof is to consider the tree of all possible computations of a non-deterministic turing machine. We start from the root and do a breadth-first search and accept only if an accepting configuration is found. This can be done on a multitape turing machine, and hence there exists an equivalent deterministic turing machine. □

Lecture 3 On 24/10

The following is less of a theorem in mathematical theorem and more of an important concept.

Theorem 1.11 (Church-Turing)**.** The intuitive notion of an algorithm is equivalent to the notion of computation defined by a Turing machine.

Proposition 1.12. Every Turing machine M can be encoded as a word over a finite alphabet. We use $\langle M \rangle$ to denote the **encoding** of a Turing machine M.

Theorem 1.13. There is a Turing machine U that takes input $\langle M \rangle$ and a word w and can simulate M on w. U is called a **universal Turing machine**.

Example (The halting problem)**.** The **halting problem** is the problem of determining, given a Turing machine M and a word w, does M terminate on w.

Proposition 1.14. The halting problem is Turing-recognisable.

Proof. This is quite clear; we construct a universal Turing machine on (M, w) which accepts if M terminates. □

Proposition 1.15. The halting problemn is not Turing-decidable.

Proof. Let M be a Turing machine. Assume for a contradiction that there exists a Turing machine

$$H(M, w) = \begin{cases} \text{accept} & M \text{ terminates on } w \\ \text{reject} & M \text{ does not terminate on } w. \end{cases}.$$

Now let us construct another Turing machine D such that

$$D(M) = \begin{cases} \text{accept} & H(M, M) \text{ rejects} \\ \text{loop} & H(M, M) \text{ accepts.} \end{cases}$$

Now let us consider what happens when we run D on itself, $D(D)$. There are two scenarios:

(i) $D(D)$ accepts (thus terminates), and therefore $H(D, D)$ rejects and hence D does not terminate on D; a contradiction;

(ii) $D(D)$ does not terminate, and therefore $H(D, D)$ accepts and hence D terminates on D; a contradiction.

Therefore, H can not exist. □

Theorem 1.16. A language L is Turing-decidable if and only if both L and $\bar{L}$ are Turing-recognisable.

Proof. $\Rightarrow$ This is clear.

$\Leftarrow$ Here, we run the machines that L and $\bar{L}$ is parallel. If L's machine accepts then we accept, if $\bar{L}$ machine accepts then we reject.

□

Lecture 7
On 21/11

Definition 1.17 (Composition)**.** Let $f : R^k \to S$ and $g_1, g_2, \dots g_k : R^n \to S$. The function $h : R^n \to S$ is obtained from f and $g_1, g_2, \dots, g_k$ by **composition** if

$$h(\boldsymbol{x}) = f(g_1(\boldsymbol{x}), g_2(\boldsymbol{x}), \dots, g_k(\boldsymbol{x})).$$

Definition 1.18. Let $f : R^n \to S$ and $g : R^{n+2} \to S$ be total functions. The function $h : R^{n+1}$ is obtained from f and g by primitive recursion if

$$\begin{aligned} h(x_1, \dots, x_n, 0) &= f(x_1, \dots, x_n) \\ h(x_1, \dots, x_n, t+1) &= g(t, h(x_1, \dots, x_n, t), x_1, \dots, x_n). \end{aligned}$$

Definition 1.19 (Primitive recursion)**.** A function is called **primitive recursive** if it can be obtained from the initial functions by a finite number of applications of composition and primitive recursion.

Definition 1.20 (Total function and partial functions)**.** A **total function** $f : R \to S$ is such that for all $x \in R$, $f(x)$ exists (that is, it is a function). A **partial function** from R to S (denoted maybe by $f : R \to /S$) is defined as the function $f : R' \to S$ such that $R' \subset R$.

Proposition 1.21. The following functions are primitive recursive:

(i) addition;

(ii) subtraction;

(iii) multiplication;

(iv) integer division;

(v) exponentiation;

(vi) integer logarithm; and

(vii) nth prime number.

Definition 1.22 (Gödel number)**.** Let $(x_1, x_2 \dots x_n)$ be a sequence. The **Gödel number** of the sequence is defined as

$$p_1^{x_1} \cdot p_2^{x_2} \cdot \dots \cdot x_{n-1}^{x_{n-1}} \cdot x_n^{x_n}$$

where p_i is the ith prime number.

Proposition 1.23. The Gödel number of any sequence is primitive recursive.

It is clear to see that a Gödel number can uniquely identify a sequence; hence, a string w over a finite alphabet Σ can be encoded by a single number $[w]$, and so can a Turing machine $[\langle M\rangle]$ but we shorten this to $[M]$ or even just M.

We can also see that a configuration of a Turing machine M can be uniquely encoded as a single number.

Moreover, (albeit less obvious) if a configuration C yields the configuration C', the step function S such that $S(C) = C'$ is primitive recursive.

Then, of course, our step-counter function f can be defined as

$$f(M, w, 0) = (q_{\text{initial}}, 0, w)$$
$$f(M, w, t+1) = S(f(M, w, t)).$$