3P17/4P17 COMPUTABILITY 2014-2015 Lecture Handout 3

A formalisation of tapes and computation

We have formally defined a Turing machine M as a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ but up until now, we have not formally defined what the tape is.

This is easy to do: Letting $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$, a "tape", along with symbols on it, is merely a function $T : \mathbb{N}_0 \to \Gamma$, where we always set $T(0) = \triangleright$. Then for integer $i > 0$, $T(i)$ can be thought of as the symbol "located i squares to the right" of $\triangleright$.

We can then define a configuration C as a 3-tuple $C = (q, T, i)$ where $q \in Q$ is a state, T is a tape function, and integer $i \geq 0$ is a head location.

Then in the Turing machine M, $C = (q, T, i)$ yields $C' = (q', T', i')$ if and only if $\delta(q, T(i)) = (q', x, d)$ where

$$T'(j) = \begin{cases} x & \text{if } j = i \\ T(j) & \text{if } j \neq i \end{cases} \quad \text{and} \quad i' = \begin{cases} i+1 & \text{if } d = R \\ i-1 & \text{if } d = L \end{cases}$$

.

Then computation with the Turing machine M on input $\sigma = \sigma_1\sigma_2\ldots\sigma_n$, defines a sequence of configurations $C_0, C_1, C_2, \ldots$, where the starting configuration is $C_0 = (q_0, T_0, 1)$ where $T_0(0) = \triangleright$, $T_0(i) = \sigma_i$ for $1 \leq i \leq n$, and $T_0(i) = \sqcup$ for $i > n$. The sequence will be finite if M halts on σ, and will be (countably) infinite if M does not halt on σ.

For a multitape Turing machine with k tapes, we can extend the above by having k simultaneous tape functions.

Non-deterministic Turing machines

A **non-deterministic** Turing machine M is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ with Q, Σ, Γ finite sets and $q_0, q_{\text{acc}}, q_{\text{rej}} \in Q$. These have the same meaning as in a single tape TM.

The transition function δ is different:

$$\delta : Q \setminus \{q_{\text{acc}}, q_{\text{rej}}\} \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\}),$$

where, for a set S, $\mathcal{P}(S)$ is the power set of S, i.e., the collection of all subsets of S.

Whilst in a deterministic TM computation is a sequence of configuration $C_0, C_1, \ldots$, in a non-deterministic TM it is a sequence of *sets* of configurations $\mathcal{S}_0, \mathcal{S}_1, \ldots$.

$\mathcal{S}_0 = \{C_0\}$ where C_0 is the starting configuration. Then $\mathcal{S}_{i+1}$ is derived from $\mathcal{S}_i$ in the following way: A configuration $C' = (q', T', i')$ is in $\mathcal{S}_{i+1}$ if and only if there is some configuration $C = (q, T, i) \in \mathcal{S}_i$ such that $(q', x, d) \in \delta(q, T(i))$ where

$$T'(j) = \begin{cases} x & \text{if } j = i \\ T(j) & \text{if } j \neq i \end{cases} \quad \text{and} \quad i' = \begin{cases} i+1 & \text{if } d = R \\ i-1 & \text{if } d = L \end{cases}$$

.

In other words, C' is in $\mathcal{S}_{i+1}$ if and only if there is some (non-halting) configuration C in $\mathcal{S}_i$ that yields C' under the action of the transition function (which may also yield many other configurations from C beside C'). If all configurations in $\mathcal{S}_i$ are halting configurations, then there are no further sets.

This creates a *computation tree*.

A non-deterministic Turing machine M **accepts** a string σ if there is some branch in the computation tree that accepts σ, that is, there is some $\mathcal{S}_i$ in the sequence with $C_i \in \mathcal{S}_i$ accepting.

A non-deterministic Turing machine M **rejects** a string σ if *every* branch in the computation tree results in a reject for σ, that is, there is some $\mathcal{S}_i$ in the sequence with every $C_i \in \mathcal{S}_i$ rejecting.

Searching a tree: Depth-first and breadth-first searches

Given a rooted, labelled tree $\mathcal{T}$, starting at the root, we can explore $\mathcal{T}$ in two ways. In a *depth-first search* (DFS), from the root, we go from child vertex to child vertex until we hit a leaf. Then we go up and back down to a sibling of the leaf. Once all siblings have been explored, go up twice then down and do the same for a sibling of the parent, and so on.

With a *breadth-first search* (BFS), we explore the tree one level at a time.

If $\mathcal{T}$ is infinite, a DFS can take us down an infinite path, meaning there could be vertices in $\mathcal{T}$ that we never explore. With a BFS, every vertex will be explored eventually.

A DFS can be implemented with a *stack*, and a BFS can be implemented with a *queue*.