CSC 2401 Introduction To Complexity - Fall 2001
Lecture #2, Friday Sept 14, 2001
Lecturer: Allan Borodin
Scribe: Michael McGuffin

2.1 Multi-tape Turing Machines

Although many models of computation are possible, we will be interested in Turing Machines, and in particular (Deterministic) *Multi-tape Turing Machines*. Turing Machines are very simple computers, however they can implement any algorithm and "efficiently" simulate the operation of any (classical) computer.

Informally, a Turing Machine is a device with a finite number of internal states, an input tape, possibly an output tape, and a finite set of work tapes. Each tape is a linear sequence of squares, each of which may be blank or contain a symbol. Each tape is unlimited in extent (in both directions), however we will require that, at any given time, the non-blank symbols on each tape cover a finite region beyond which all other squares are blank. Symbols on the tapes are read and written, one at a time, with heads that can be moved back and forth along each tape. For our purposes, the input tape is read-only and the output tape is write-only, however the work tapes are of course readable and writable.

A *Multi-tape Turing Machine* with k work tapes is defined by the tuple $\mathcal{M} = \langle Q, q_0, F, F', \Sigma, \Gamma, \delta \rangle$ where

- Q is a finite set of (internal) states
- $q_0 \in Q$ is the initial state
- $F \subseteq Q$ is a set of *accepting* states
- $F' \subseteq Q$ is a set of *rejecting* sates, such that $F \cap F' = \emptyset$
- Σ is a finite alphabet (or set) of symbols used for input (and output, if there is an output tape)
- $\Gamma \supseteq \Sigma \cup \{\flat\}$ is the set of work tape symbols, including the input alphabet Σ and a special blank symbol $\flat \notin \Sigma$
- δ is the *transition function*, a total function that maps the symbols currently readable on the tapes and the current state of the machine to the symbols to be written and the state of the machine after one time step. Specifically, if there is no output tape, then

  $$\delta : (Q - F - F') \times (\Sigma \cup \flat) \times \Gamma^k \to Q \times \Gamma^k \times \{L, S, R\}^{k+1}$$

  and, if there is an output tape, we have

  $$\delta : (Q - F - F') \times (\Sigma \cup \flat) \times \Gamma^k \to Q \times \Gamma^k \times \{L, S, R\}^{k+1} \times ((\Sigma \times \{R\}) \cup \{S\})$$

Note that often one chooses $\Sigma = \{0, 1\}$ since two symbols are enough to encode any finite input as efficiently (within a constant factor) as with any larger finite number of symbols.

Input Convention: Initially the input string $x \in \Sigma^*$ appears on contiguous squares of the input tape, with the read head of the input tape positioned on the leftmost symbol of the input string. All other squares of the input tape are blank, as are all the input squares of the work tapes and output tape. The Turing Machine's internal state is initially q_0.

Operation: Given a state $q_t \in Q$ and the symbols currently readable on the input tape and the k work tapes, the value of δ specifies (i) the subsequent state q_{t+1}, (ii) what symbols to write out on the work tapes (and possibly the output tape), and (iii) how to move each head: L for left, R for right, and S to stay still. The Turing Machine is said to *halt* when δ enters a state $q \in F \cup F'$, whereupon the Turing Machine *accepts* or *rejects* the input.

An input string $x \in \Sigma^*$ is said to be *accepted* by M if, after some finite number of time steps, the machine enters a state $q \in F$. Similarly, the string is said to be *rejected* if the machine enters a state $q \in F'$ after a finite number of time steps.

The language $\mathcal{L}(\mathcal{M})$ *accepted* by $\mathcal{M}$ is the set of input strings accepted by $\mathcal{M}$. Furthermore, if $\mathcal{M}$ halts on every input, we say that $\mathcal{L}(\mathcal{M})$ is *recognized* by $\mathcal{M}$, and $\mathcal{L}$ is a said to be a *computable*, *decidable*, or *recursive* language.

$\mathcal{M}$ is said to compute a (possibly partial) function $f : \Sigma^* \to \Sigma^*$ iff (i) $\mathcal{M}$ halts on input x and outputs $f(x)$ whenever $f(x)$ is defined, and (ii) $\mathcal{M}$ does not halt on input x whenever $f(x)$ is not defined. Clearly, every Turing Machine has a corresponding partial function f that it computes.

If f is the solution to a decision problem, i.e. f only takes on two values (e.g. **yes** and **no**), then we may choose to encode the "output" of a corresponding Turing Machine by whether the input is accepted or rejected. In such a case, an output tape is not neccesary.

Note that there is an alternative formulation of the Multi-tape Turing Machine where there are no rejecting states F'. In this case, δ is a partial function, and the Turing Machine is said to reject an input if it enters a state where δ is undefined.

Turing Machines are equivalent to all other classical computers in the sense that they are all capable of computing any mechanical algorithm or so-called *effective procudure* or *recursive operation.* Furthermore, Turing Machines can compute functions as quickly and in as little space as any other *"classical"* computer, within at most a polynomial factor.

2.2 Measuring Time and Space

The **running time** $time_{\mathcal{M}}(x)$ of $\mathcal{M}$ on $x \in \Sigma^*$ is the number of moves made by $\mathcal{M}$ processing x until it halts. If $\mathcal{M}$ does not halt on x, then $time_{\mathcal{M}}(x) = \infty$

by definition.

Similary, the **working space** $space_{\mathcal{M}}(x)$ of $\mathcal{M}$ on $x \in \Sigma^*$ is the sum of the number of non-blank positions (ever written) on each work tape (we exclude the input tape and output tape from consideration). If there is no maximum, then $space_{\mathcal{M}}(x) = \infty$.

Observe that $space_{\mathcal{M}}(x) = \infty$ implies $time_{\mathcal{M}}(x) = \infty$, however the converse is not generally true.

The **time complexity** $t_{\mathcal{M}}$ of $\mathcal{M}$ is $t_{\mathcal{M}}(n) = max\{time_{\mathcal{M}}(x) : |x| = n\}$. Similarly, the **space complexity** $s_{\mathcal{M}}$ of $\mathcal{M}$ is $s_{\mathcal{M}}(n) = max\{space_{\mathcal{M}}(x) : |x| = n\}$.

$\mathcal{M}$ is said to recognize language $\mathcal{L}(\mathcal{M})$ in time $T(n)$ iff

$$\overset{\infty}{\forall}\, x, time_{\mathcal{M}}(x) \leq T(|x|)$$

where $T(n)$ is a function from integers to integers. The symbol $\forall^{\infty}\, x$ should be read "for almost all x", and means for all but finitely many exceptions, or (in the case of natural numbers) for all inputs greater than some fixed lower bound.

Similarly, $\mathcal{M}$ recognizes $\mathcal{L}(\mathcal{M})$ in space $S(n)$ iff

$$\overset{\infty}{\forall}\, x, space_{\mathcal{M}}(x) \leq S(|x|)$$

where $S(n)$ is a function from integers to integers.

2.3 Complexity Classes

Roughly, a complexity class is a set of languages that can be recognized within some limits of time and/or space.

Let $T(n)$ and $S(n)$ be nondecreasing functions from integers to integers. We define $DTIME(T(n))$ as the set of languages L for which there exists a Turing Machine $\mathcal{M}$ that recognizes L and that runs in time $T(n)$, i.e.

$$DTIME(T(n)) = \{L \in \Sigma^* : \exists \mathcal{M} : L = \mathcal{L}(\mathcal{M}), \mathcal{M}\ recognizes\ L\ in\ time\ T(n)\}$$

(The "D" in DTIME stands for "deterministic".) Similarly,

$$DSPACE(S(n)) = \{L \in \Sigma^* : \exists \mathcal{M} : L = \mathcal{L}(\mathcal{M}), \mathcal{M}\ recognizes\ L\ in\ space\ S(n)\}$$

It can be shown that, for all $c > 0$,

$$DTIME(T(n)) = DTIME(\lceil c \cdot T(n) \rceil + n)$$

$$DSPACE(S(n)) = DSPACE(\lceil c \cdot S(n) \rceil)$$

The class of all languages recognized in **polynomial time** is

$$P = \bigcup_{k \geq 1} DTIME(n^k) = \bigcup_{p,\ p\ a\ polynomial} DTIME(p)$$

Note that, given a polynomial p with $\deg p \leq k$,

$$DTIME(p(n)) \subseteq DTIME(n^{k+1})$$

Similarly, the class of languages accepted in **polynomial space** is

$$PSPACE = \bigcup_{k \geq 1} DSPACE(n^k)$$

The class of languages accepted in **logarithmic space** is

$$DL = DLOG = LOGSPACE = \bigcup_{c \geq 0} DSPACE(\lceil c \cdot log_2 n \rceil) = DSPACE(log_2 n)$$