More Turing Machines

CS254
Chris Pollett
Sep 6, 2006.

Outline

- Diagrams, examples, languages
- Recursive, RE, Functions
- Multi-Tape Turing Machines
- Time and Space classes
- Simulations

A Simple Turing Machine

- The transition function is the most important part of a TM's description.
- We will sometimes use a graphical notation to describe TM's and in particular this function.
- Given a in $\Sigma \cup \{L, R\} - \{\#\}$, define a machine $M_a = \{\{s,h\}, \Sigma, \partial, s\}$, where for each b in $\Sigma - \{\#\}$, $\partial(s,b) = (h, a)$. $\partial(s, \#) = R$.
- That is, if a is a symbol, the only thing M_a does is writes that symbol; if a is L or R then the only thing M_a does is either move left or right.

Building Bigger TMs

- Given three TMs with a common alphabet: M, N, P, we can build a new machine M´ which operates as follows:
  - Start in the initial state of M; operate as M until M would halt, then
  - if the currently scanned symbol is an *a*, start N
  - if the currently scanned symbol is an *b*, start P.
  - halt otherwise.
- Diagrammatically we write:

$$\begin{tikzcd} M \arrow[r, "b"'] \arrow[d, "a"] & N \\ P \end{tikzcd}$$

- As an exercise you should work out what M´'s transition function would look like.

More on Diagrams

- Similar to the if-else type diagram of the last slide we can have diagrams like:

  M ----> N

  Notice there is no label on the arrow. This means that if machine M is about to transition to its halt state h we instead have it transition to the start state of N.

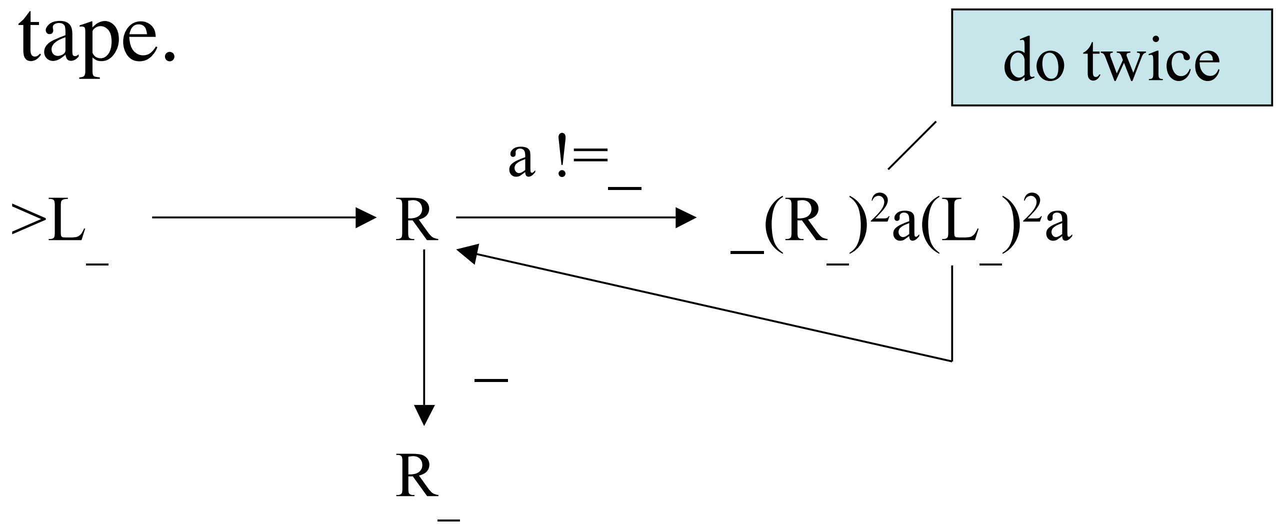
- We can also generalize the two branch construction of the previous slide toany fixed finite number of branches.

Examples

- We sometimes abbreviate M_R as R and M_a as a. We might also make abbreviations like Ra for the machine which does M_R then reading any symbol write an a. Similarly, we might have RR or La.
- Let !*a* denote all the symbols in Σ except *a*.
- Here is a machine $R_{\_}$ that scans right to the first space >R !_
- Here is a machine $L_{\_}$ that scans left to the first space >L !_

More Examples

- Here is a machine which when started with a string _w on the tape halts with _w_w on the tape.



Computing with Turing Machines

- A **configuration of M** is a pair (q, $\#w\underline{a}v$) where q is a state of the TM, #w is the string to the left of the tape head, $\underline{a}$ is the current symbol being read, and v is the tape square sto the right of the head that are either in the input or have been seen so far during the computation.
- The initial configuration of M is (s, $\underline{\#}x$).
- A **computation** of M is a sequence of configurations of M

  $(s, \underline{\#}x) :- (q_1, \underline{w_1}) :- \ldots :- (q_m, \underline{w_m})$

  such that each configuration follows from the previous according to M’s ∂. Read :- as *yields*.
- A computation halts if either the state yes or no is reach.
- A machine M **accepts a languages** L if it stops with state yes when x is in the language and run forever otherwise.

Recursive and Recursive Enumerable

- A language L is said to be **recursively enumerable** if it is accepted by some Turing Machine
- A language L is said to be **recursive** if there is a Turing machine M which run on x that is in L, M halts in the yes state; and when run on an x not in L, M halts in the no state.

Proposition If L is recursive then it is recursively enumerable.

Proof. Suppose there is a M which decides L. We can make an M´which accept L as follows: M´ behaves the same as M except that whenever M is about to halt and enter a "no" state M´ moves right forever and never halts.

Computing Functions

- Turing machines will be used to model algorithms, so we'll often want to be able to compute functions.

Definition. Let f be a function from $(\Sigma - \{\_\})^*$ to Σ^*. Let M be a TM with alphabet Σ. We say M **computes** f if for any string x in $(\Sigma - \{\_\})^*$, M on input x (written as M(x)) halts with f(x) written on the tape.

- If f can be computed by some M we say f is a **recursive function** or f is **computable**.
- Our earlier example shows that the copying map is computable.
- We could code instances of networks as strings, and implement MAX FLOW on a TM using our algorithm from Chapter 1. This would show MAX FLOW is computable.

k tape machine

- One way you might try to improve the power of a TM is to allow multiple tapes.

Definition A k-string TM, where $k \geq 1$ is an integer, is a quadruple $M=(K,\Sigma,\partial,s)$ where K,Σ,s are as in the 1-tape case. Now, however, the transition functions is a map

$\partial: K \times (\Sigma \cup \{\#\})^k \to (K \cup \{h, \text{yes}, \text{no}\}) \times (\Sigma \cup \{L,R\})^k$

- Basically the heads on each tape can move independently of each other.
- For example, with a two tape machine an algorithm for palindrome testing is easy.
  - We set up the transition function so it first copies the first tape input to the second tape.
  - Then it rewinds the first tape and leaves the second tape at the end of the input.
  - Then the first tape moves right while the second tape moves left and we compare the two tape symbol by symbol. If they don’t match we hat in a no. If the second tape gets back to the # then we accept.

TIME and SPACE classes.

- We shall use the k-tape model of TM as our basic model to study time and space complexity.
- Let f:**N --> N**. We say that machine M operates within time f(n) if for any input string x, the time require by M on input x is at most f(|x|). Here |x| is the length of x as a string. We can make a similar definition for space.

Defn. We say that a language L is in **TIME**(f(n)) (resp. SPACE(f(n))) if it is decided by some k-tape TM in time f(n) (resp. space f(n)).

- For example, the algorithm for palindrome in time TIME(3(n+2)).
- You can show for a single tape machine for palindrome you need at least time $\Omega(n^2)$.
- How well can a 1-tape machine simulate a k-tape machine?

Simulating k-tape by 1-tape

Thm. Given any k-tape machine M that operates within time f(n), we can construct a 1-tape machine M´operating within time $O((f(n))^2)$.

Proof. Let $M=(K,\Sigma,\partial,s)$ be a k tape machine.

- The idea is M´ alphabet, Σ', is going to be expanded to include symbol #´ to denote the last used square of a tape. And we are going to add to Σ' a symbol $\underline{b}$ for each symbol b in Σ.
- A configuration of M can now be written as:

$(q, \#w_1\underline{a_1}v_1\#'w_2\underline{a_2}v_2\#'\ldots\#w_k\underline{a_k}v_k\#')$

- So except for the state which we can keep track of in K´ the rest of the state is a string over Σ'.
- We will use new states K´ to keep track of the state of M during a simulation step.
- To simulate M, we first convert the input into the initial configuration of M viewed as a string.
- Then to simulate a step we scan left to right the current configuration string, noting what symbol is being read by each tape in our finite control.
- Next we rewind the tape and we then do passes again to update each tapes configuration.
- In the worst case we need to expand the number of tape square of each tape by 1. So we could need $(k(f(|x|)+1)+1$, passes to simulate 1 step.
- So simulating $f(|x|)$ steps take at most $f(|x|)((k(f(|x|)+1)+1)$ times which is $O((f(n))^2)$.