COMS 381, Summer 2005
Supplementary Handout 5
Nondeterministic Turing Machines
Thursday, June 23rd

Just like finite automata and pushdown automata, Turing machines exist in nondeterministic as well as deterministic flavors. However, nondeterminism does not add any computational power to a Turing machine. That is, any nondeterministic Turing machine can be simulated by a deterministic one, as we will show.

Formally, a nondeterministic Turing machine can be defined as a 9-tuple

$$M = (Q, \Sigma, \Gamma, \Delta, \vdash, \sqcup, s, t, r)$$

Where everything is the same as for a deterministic machine, except that Δ now has the form:

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

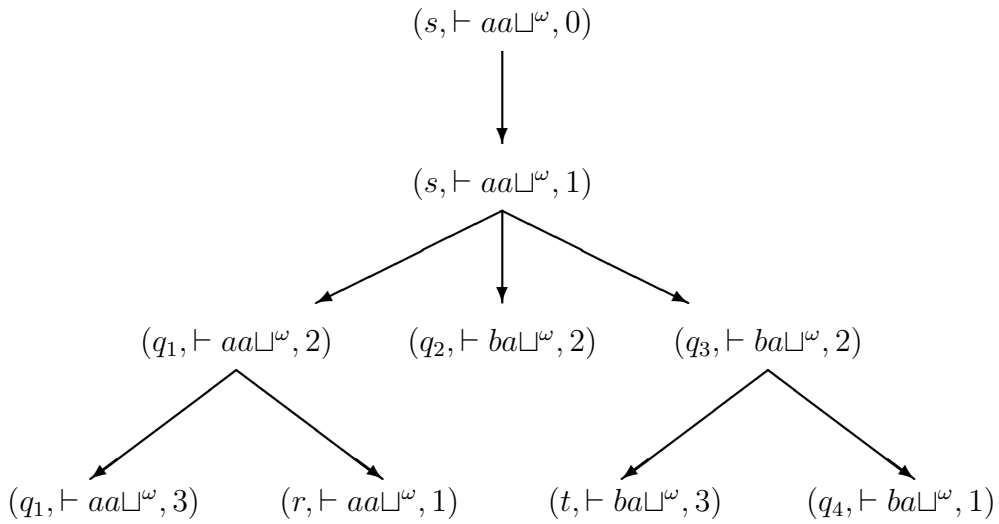
where $\mathcal{P}(X)$ denotes the power set of the set X. That is, for any given state q and tape symbol a, the TM may have a choice of which state to move to, which symbol to write on the tape, and whether to move the head left or right.

As with NFAs and NPDAs, the computation of a nondeterministic TM M on an input x can be conceptualized as a *computation tree*. The nodes in the tree will be configurations of the TM, and the root will be the starting configuration.

For example, suppose we have a machine M with the following transitions:

$\Delta(s, \vdash) = (s, \vdash, R)$
$\Delta(s, a) = \{(q_1, a, R), (q_2, b, R), (q_3, b, R)\}$
$\Delta(q_1, a) = \{(q_1, a, R), (r, a, L)\}$
$\Delta(q_3, a) = \{(t, a, R), (q_4, a, L)\}$

We can draw a computation tree for M on input $x = aa$ as follows:



A nondeterministic TM will accept iff *there exists* some path through that tree which ends in an accept state (call this type of path an *accepting path*). Note that this is exactly analogous to the situation for NFAs. Intuitively, the TM can 'guess' the correct path to follow, and then 'verify' the guess by actually carrying out the computation according to that path.

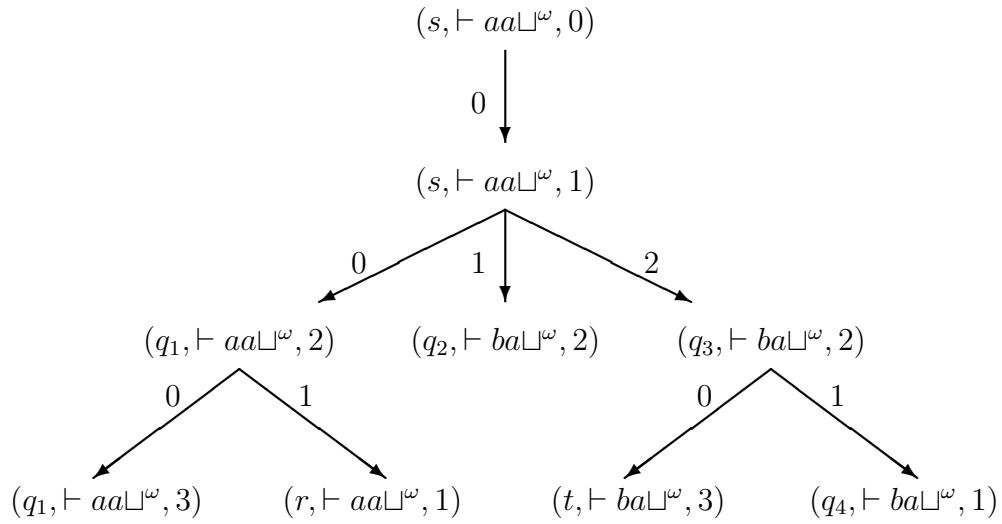
The above tree has an accepting path, leading to the configuration $(t, \vdash ba\sqcup^\omega, 3)$. It also happens to have a rejecting path, leading to $(r, \vdash aa\sqcup^\omega, 1)$, and two paths that don't terminate at this point. However, this doesn't matter; as long as there is an accepting path, the machine is defined to accept.

Equivalence of deterministic and nondeterministic TMs

Clearly, every deterministic TM is a nondeterministic TM also. To show equivalence, we only need to show how to simulate any nondeterministic TM N using a deterministic TM M.

Intuitively, our machine M is trying to find an accepting path in the computation tree of N operating on input x; M will accept if and only if it finds such a path. It will then follow immediately that M accepts $x \iff N$ accepts x.

In practice, this is easiest to implement by making M carry out a search on the computation tree of N on x. Call this tree T_x. Note that T_x must be a finitely-branching tree (why? look again at the definition of Δ if this is not obvious). Let k be the maximum branching factor of T_x. Then we can encode every finite-length path p in T_x by a number n_p in $k-$ary. The first digit of n_p will encode the first branch, the second the second branch, and so on. For instance, we can encode all paths in our previous example tree in this manner:



The maximum branching factor k is 3 in this case. The accepting path is encoded in ternary as 020, the rejecting path as 001, and so on.

Our machine M can therefore compute as follows:

M takes as input a description of N and the input string x. From the description of N, M can easily obtain the branching factor k for T_x, and as many initial levels of T_x as it needs at any given point in time. M will repeat the following loop until it accepts (if it never accepts, it loops forever)

- Let l be the current length of the computation path being simulated by M. Obtain from the description of N all the numbers in $k-$ary that encode valid computation paths of length l (that is, obtain encodings of all paths in T_x of length exactly l). Call the set of all these encodings P.
- Simulate N on each of the computation paths corresponding to an encoding in P (that is, for each encoding, run N with no nondeterminism permitted, *forcing* it to take the path represented by the encoding). Each of these paths is finite, so the simulation will stop after finitely many steps.
- If the simulation above ended in the accepting state of N for any path, accept. Otherwise, *even if the simulation ended in the reject state of N for some path*, increment l by one and continue in the loop.

Note that M must continue even if it finds a rejecting path in the computation tree of N. The existence of a rejecting path does not imply the nonexistence of an accepting path, as we saw in our example.