CSE491/596 Lecture Mon. 9/28 Single-Tape and Multi-Tape Turing Machines

[Lecture started by going over Problem Set 2, including the ideas for presentations staggered this week and next. Then it showed how definitions from Fri. 9/25 needed only a little modification to define multitape Turing machines:

Definition: A k-*tape Turing machine* is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, \_, s, F)$ where Q, s, F and Σ are as with a DFA, the *work alphabet* Γ includes Σ and the *blank* _, and

$$\delta \subseteq (Q \times \Gamma^k) \times (\Gamma^k \times \{\mathsf{L}, \mathsf{R}, \mathsf{S}\}^k \times Q) .$$

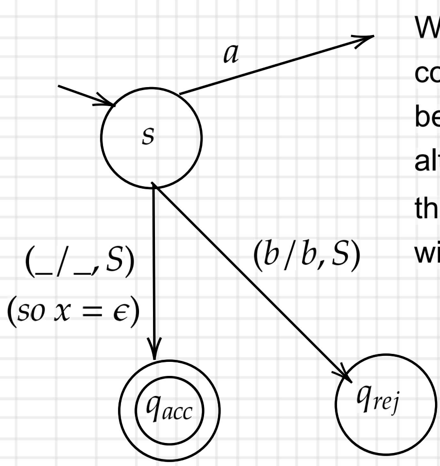
It is *deterministic* (a DTM) if no two instructions share the same first two components. A DTM is "in normal form" if F consists of one state q_{acc} and there is only one other state q_{rej} in which it can halt, so that δ is a function from $(Q \setminus \{q_{acc}, q_{rej}\}) \times \Gamma$ to $(\Gamma \times \{\mathsf{L}, \mathsf{R}, \mathsf{S}\} \times Q)$. The notation then becomes $M = (Q, \Sigma, \Gamma, \delta, \_, s, q_{acc}, q_{rej})$.

$(p, [c_1, c_2, \dots, c_k] / [d_1, \dots, d_k], [D_1, \dots, D_k], q)$ where p and q are states, c_j and d_j are chars, and $D_j \in \{\mathsf{L}, \mathsf{R}, \mathsf{S}\}$, $j = 1 \text{ to } k$

Then the lecture went into how to design a single-tape TM to recognize the language as follows.]

$$L = \left\{a^m\, b^n : n = m\right\}.$$

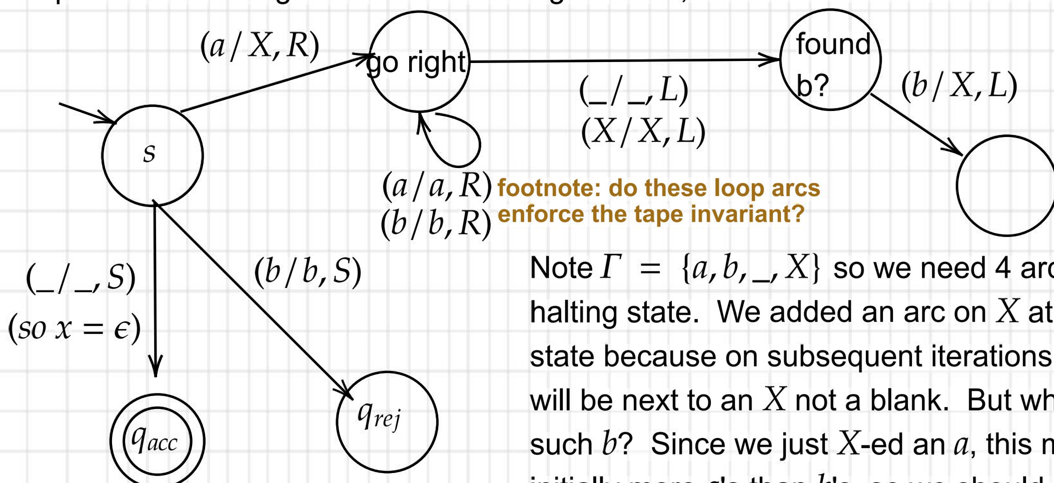
By default, n, m are natural numbers, so $n = m = 0$ is allowed, and so $\epsilon \in L$. Recall that when the input x is ϵ, the TM tape starts off completely blank. Otherwise, the TM starts in the configuration of scanning the first char of x, with the rest of the tape blank. So an initial scan of _ means that $x = \epsilon$ and we can make M accept right away. And if x starts with b then it cannot be in L, so we can make M reject right away. A Turing machine is not required to scan its entire input, though we can impose this requirement (and when we discuss time complexity classes, we will). This gives us a good beginning on how to build M to recognize L step-by-step with goal-oriented reasoning. [Lecture worked on the diagram "interactively"; here we show some stages.]



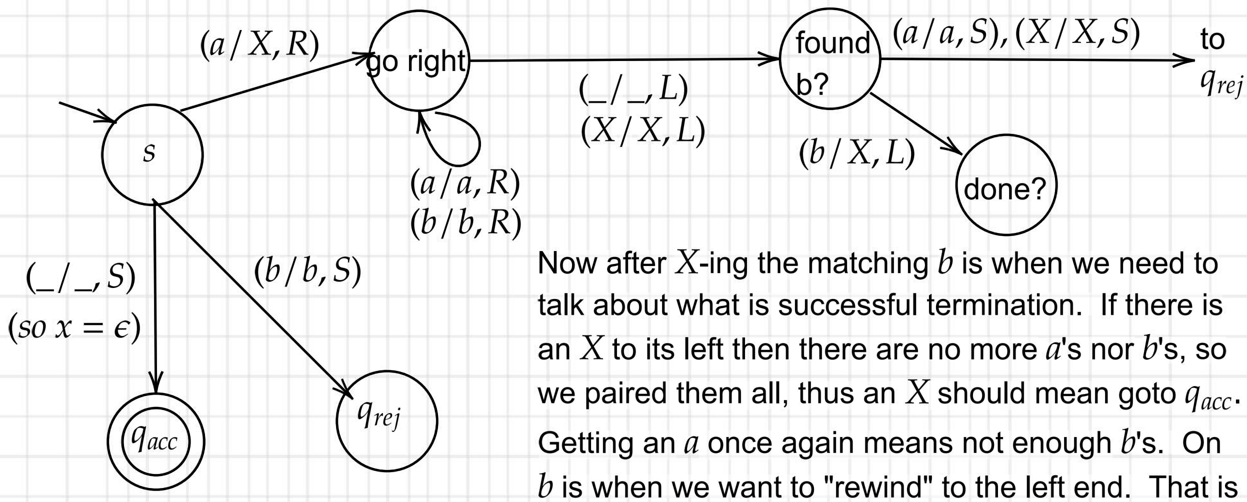
We've already been able to handle immediate accept and reject conditions in the start state. Now we decide strategy when x begins with a. The idea is to X-out a's and b's one-by-one in alternation. If we X-out always the leftmost a and the rightmost b then the string between (which after the first iteration is $a^{m-1}b^{n-1}$) will belong to L if and only if x does. So we can recurse and keep:

Tape Invariant: $X^*\, a^*\, b^*\, X^*$ and after X-ing a b the numbers of Xes on left and right are the same, so the string between them belongs to L if and only if the original x does.

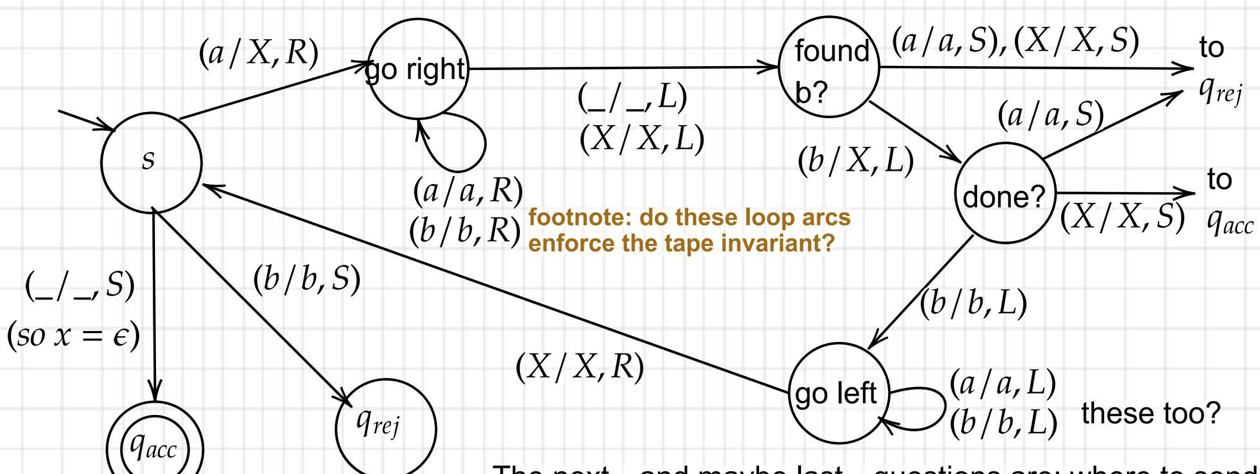
To perform the X-ing of one a then the rightmost b, add these states and instructions:



Note $\Gamma = \{a,b,\_,X\}$ so we need 4 arcs at each non-halting state. We added an arc on X at the "go right" state because on subsequent iterations the rightmost b will be next to an X not a blank. But what if there is no such b? Since we just X-ed an a, this means there were initially more a's than b's, so we should reject.



Now after X-ing the matching b is when we need to talk about what is successful termination. If there is an X to its left then there are no more a's nor b's, so we paired them all, thus an X should mean goto q_{acc}. Getting an a once again means not enough b's. On b is when we want to "rewind" to the left end. That is when we need X to stop a leftward loop. So we cannot loop at the "done?" state itself but need another state:



The next---and maybe last---questions are: where to send the arc on X, and what actions to do? Most in particular:

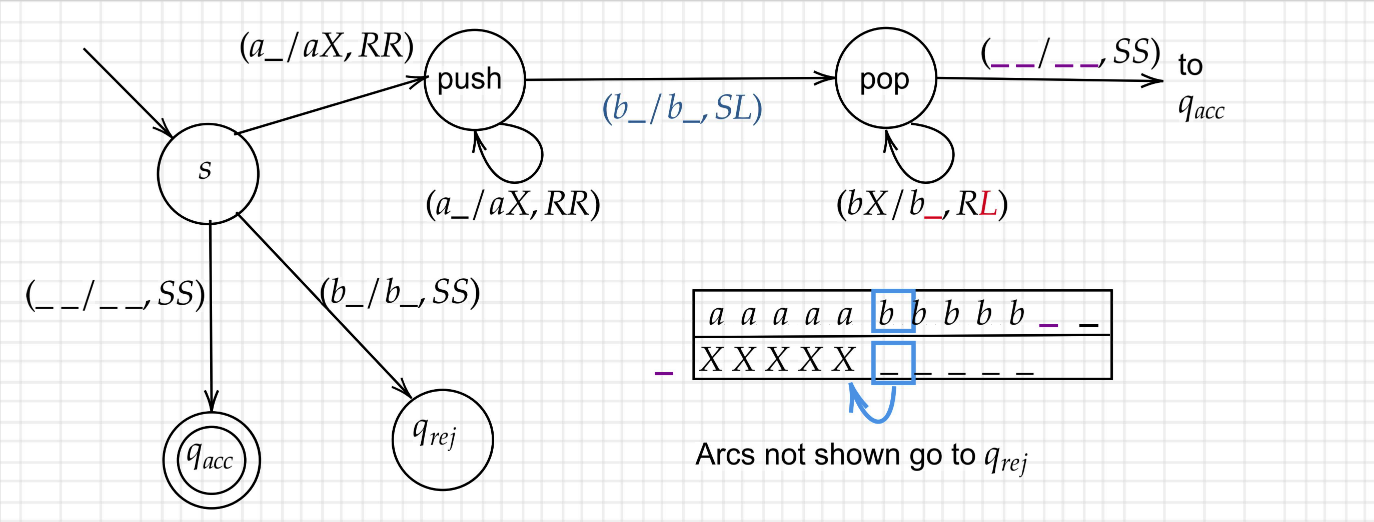
Can we complete the loop and the machine by making it be $(X/X,R)$ going back to start?

One thing to note is that if the char seen after executing $(X/X, R)$ is a b, then by the tape invariant it means there are no more a's but still at least one b since we went from "done" to "go left", so this is the case $m < n$. Well, in that case we should reject, and the arc on b going to q_{rej} is already there from the initial design. So: *this is OK and M is complete.*

[This is the question where my Mon. 9/28 lecture left off. I will pick up here.]

Note that the input x can belong to $a^* b^*$ without belonging to L. Those strings abide by the tape invariant initially, and we can already see that M works correctly on those strings. But what if x is something like $aababb$? Will our M accept when it shouldn't? **That's what the footnote is about.**

Assuming M is correct---or quickly fixable if not---we can ask, how long does it take to accept a good $x = a^n b^n$ in terms of n? The answer is, it takes $\Theta(n^2)$ steps, owing to lots of backing-and-forthing. Can we make it run faster? There is a way to make it run much faster on one tape, in $O(n \log n)$ time, but we can get an optimal $O(n)$ running time by using a second tape:



Note the straightforwardness of the design as well as the efficiency. Also note the usefulness of having the second tape be two-way infinite with a blank to the left of the "column" initially holding the first a in x (if any). An alternative convention is to make both tapes one-way infinite but with a special char $\wedge$ in cell 0 at the left end on tape 1---so that the *initial configuration* I_0 has $\wedge x_1 \cdots x_n$ on tape 1 and just $\wedge$ on tape 2 "underneath" the $\wedge$ on tape 1. We can still start with the tape heads scanning the cells in "column 1" even if both are blank (so $x = \epsilon$). Then the final accepting instruction in the "pop" state becomes $(\_\wedge / \_\wedge, SS)$.

This two-tape DTM has the properties that:

- the input tape head never moves L and never changes a character;

- whenever the second tape moves L, it writes a blank in the cell it just left.

The second condition forces the second tape to behave like a ***stack*** (except for some "flex" in how top-of-stack is treated). A TM obeying these condiitons is formally equivalent to a **pushdown automaton** (PDA). A language is *context-free* (and belongs to the class CFL) if it is recognized by some PDA that may be nondeterministic (an NPDA); if the machine is deterministic (hence a DPDA) then it belongs to the class DCFL. Every regular language is a DCFL, and $\left\{a^n\ b^n\right\}$ is an example of a DCFL that is not regular. We will not say much more about CFLs and DCFLs.