CSE396 Lecture Tue. 4/6: Turing Machines (plus some CFL review and etc.)

We pick up with a recap from Thursday:

Definition: A *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

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

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 $M = (Q, \Sigma, \Gamma, \delta, \_, s, q_{acc}, q_{rej})$.

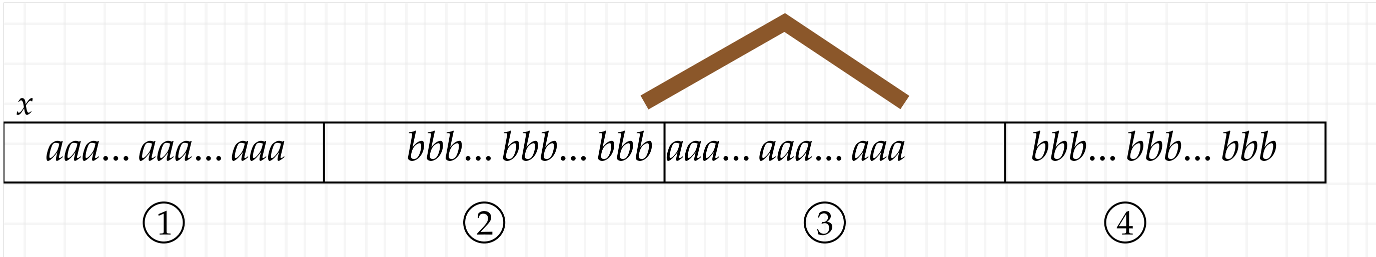
To define the language $L(M)$ formally, especially when M is properly nondeterministic (an NTM), requires defining *configurations* (also called *ID*s for *instantaneous descriptions*) and *computations,* but especially with DTMs we can use the informal understanding that $L(M)$ is the set of input strings that cause M to end up in q_{acc}, while seeing some examples first.

1. $L_1 = \left\{a^m b^n : n = m\right\}$, by default $\epsilon \in L_1$ since $n = m = 0$ is allowed.
2. $L_2 = \left\{a^m b^n : n > m\right\}$. [Show this example on the Turing Kit, as "MarEx94a.tmt".]
3. $L_3 = \left\{a^m b^n a^m b^n : m, n \geq 0\right\}$. [Not a CFL, but conceptually not much more difficult for a Turing machine than L_1.]
4. $L_4 = \left\{ww : w \in \{a, b\}^*\right\}$.

First, the CFL PL review: We can do a proof that works for L_3 and L_4 at the same time. In the "adversary argument" proof layout, the first two lines can be skipped:

Adv: "I have a CFG G such that $L(G) = L_3$"
You: "What is your $N = 2^{|V'|}$ when your G is converted to a Chomsky NF $G' = (V', \Sigma, \mathcal{R}', S')$ such that $L(G') = L_3 \setminus \{\epsilon\}$?" (Note that this automatically makes $N \geq 1$. You may suppose N is as large as **you** want it to be in order to avoid any "edge effects" near empty (sub)strings.)
Adv: "N"
You: "The string $x = a^N b^N a^N b^N$ belongs to L_3 (and also to L_4). Give me a breakdown $x =: yuvwz$ such that $|uvw| \leq N$ and at least one of u and w is not ϵ."
Also good would be for you to say: "The string $w = a^N b^{2N} a^N b^{2N}$ belongs to L_3 (and also to L_4). Give me a breakdown $w =: uvxyz$ such that $|vxy| \leq N$ and at least one of v and y is not ϵ."

At this point *Adv.* takes the 5th Amendment and the hearing goes behind closed doors where you need to show how you can react to any valid breakdown that *Adv.* could give. This goes into a case-by-case analysis. Often some deft choice of prose can help you group the cases. It is also AOK to use the "compass limited to width N" visualization to convey the analysis:



Sometimes the prose can be helped by labeling "regions" of the string numerically (as above), especially when different regions have the same or similar string content. Then you can say:

By $uw \neq \epsilon$, at least one of u or w must have nonempty intersection with one of the regions. Cases:

1. If it is nonempty in region 1, then the "compass" (which is the uvw part---and note that $v = \epsilon$ is freely allowed: this is like closing the compass to use it as a brush) cannot reach region 3. Hence, if we "pump down" (i.e., take $i = 0$ to create the string $x^{(0)} = yvz$), we subtract at least one a from region 1---and possibly stuff from region 2 from either arm of the compass---but regions 3 and 4 stay the same. The resulting string hence has the form $x^{(0)} = a^i b^j a^N b^N$ where $i < N$ and $i + j < 2N$, so there is no possible way to parse it as a double-word, let alone as a member of L_3. So $x^{(0)} \notin L_3, L_4$ thus violating that either of them equals $L(G)$.
2. If it is nonempty in region 2, then the "compass" cannot touch region 4. Hence, if we "pump down", we subtract from region 2 and possibly 3 (not both 1 and 3, and indeed not 1 because that would have been already covered in case 1). The resulting word $x^{(0)}$ has the form $a^i b^j a^k b^N$ where $j < N$, so it cannot be a double-word, besides not being in L_3.
3. Similar to case 2 by symmetry.
4. Similar to case 1 by symmetry.

In all cases we have established that $x^{(0)} \notin L_3$ and also $x^{(0)} \notin L_4$. Thus neither can equal $L(G)$.
Since
Adv. was given the initial chance to produce a grammar, this means there cannot be one, so L_3 and L_4 are not CFLs. ☒

If we have, say $L_3' = \left\{a^m\ b^n\ a^m\ b^r : n \leq r\right\}$, then we have to change the pump-up vs. pump-down strategy depending on the case.

1. Case 1 can be the same since the a^m parts are required to be equal.
2. This time we need to make $x^{(2)} = yu^2vw^2z = yuuvwwz$. We can't necessarily say it creates a string of the form $a^i b^j a^k b^N$ anymore, because one or other of the strings u, w (figuratively, one "writing arm" of the compass) could include both some a's and some b's by straddling a boundary. But we can say it has the form $x^{(2)} = tab^N$ where t includes at least $N+1$ $b's$. Then $x^{(2)}$ cannot belong to L_3, nor to L_4 (that takes a little more argument).
3. This is no longer exactly similar to case 2 here, but you can write "like in case 2" before.
4. This is not exactly similar to case 2, because now you have to "pump down" so that the clinching conclusion is $x^{(0)} \notin L_3$.

Closure and Non-Closure Properties of CFLs.

We have seen that the union of two CFLs is a CFL: given G_1, G_2 with respective start symbols S_1, S_2, we make G_3 with extra start symbol S_3 and the rules $S_3 \rightarrow S_1 \mid S_2$ added to $\mathcal{R}_1 \cup \mathcal{R}_2$. Concatenation is done by adding $S_3 \rightarrow S_1 S_2$ instead, and Kleene star by $S_3 \rightarrow S_1 S_3 \mid \epsilon$ (which is the "list of G_1" design pattern). But note:
$L_3 = \left\{a^m\, b^n\, a^m\, b^r : m, n, r \geq 0\right\} \cap \left\{a^m\, b^n\, a^q\, b^n : m, n, q \geq 0\right\}$ which is an intersection of two CFLs. The left-hand one has grammar $S \rightarrow S_0 B,\ S_0 \rightarrow a S_0 a \mid B,\ B \rightarrow bB \mid \epsilon$. The right-hand one is similar. So:

Theorem: The class **CFL** of context-free languages is not closed under $\cap$, and hence is not closed under complements either. ☒

The complement of L_3 ***is*** a CFL. Recall $L_3 = \left\{a^m\, b^n\, a^m\, b^n : m, n \geq 0\right\}$. The reason is that a string x in the complement of L_3 either:

- does not have the aaaa-bbb-aaa-bbbb form, i.e., matches the complement of the regular expression $a^*b^*a^*b^*$, or
- it does match $a^*b^*a^*b^*$ and has the form $a^m b^n a^q b^r$ where not both $m = q$ and $n = r$. I.e., where $m \neq q$ OR $n \neq r$.

In the first case, we send $S \rightarrow S_4$ where S_4 is the start symbol of a grammar for the regular language $\sim a^*b^*a^*b^*$. In the second case, we get the union of two CFLs with start symbols S_5 and S_6 that handle the two inequality cases, each of which has a *single* dependency. (Note that $\left\{a^m b^n : n \neq m\right\} = \left\{a^m b^n : n > m\right\} \cup \left\{a^m b^n : n < m\right\}$ so it is the union of two CFLs.)

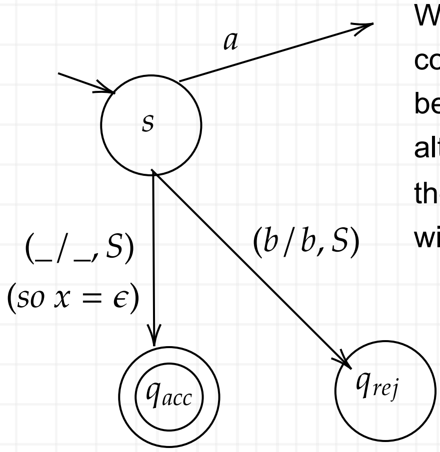
The complement of the double-word language L_4 is also a CFL. We have seen a tricky CFG that generates $\{yz : |y| = |z|\ but\ y \neq z\}$. It started $S \rightarrow AB \mid BA$ where $P_A \equiv$ "I derive exactly the strings of odd length with an a in the middle" and $P_B \equiv$ "I derive exactly the strings of odd length with a b in the middle". The final useful closure property is:

Theorem: If L is a CFL and R is a regular set, then $L \cap R$ is a CFL.

The proof will come after we introduce Pushdown Automata as a special kind of 2-tape Turing machines. But we can apply it to make it easier to show that L_4 is not a CFL: Suppose L_4 were a CFL. Then since $a^*b^*a^*b^*$ is regular, $L_4 \cap a^*b^*a^*b^*$ would be a CFL. But $L_4 \cap a^*b^*a^*b^* = L_3$. And we had the easier proof that L_3 is not a CFL by the CFL PL. So L_4 is not a CFL. This helps us avoid the uglier direct case analysis we encountered for L_4. Similarly, if
$L = \left\{x \in \{a, b, c\}^* : \#a(x) = \#b(x) = \#c(x)\right\}$, then we can simplify showing that L is not a CFL by taking $L' = L \cap a^+b^+c^+ = \left\{a^n b^n c^n : n \geq 1\right\}$. We proved that L' is not a CFL already.

Back to Turing Machines now.

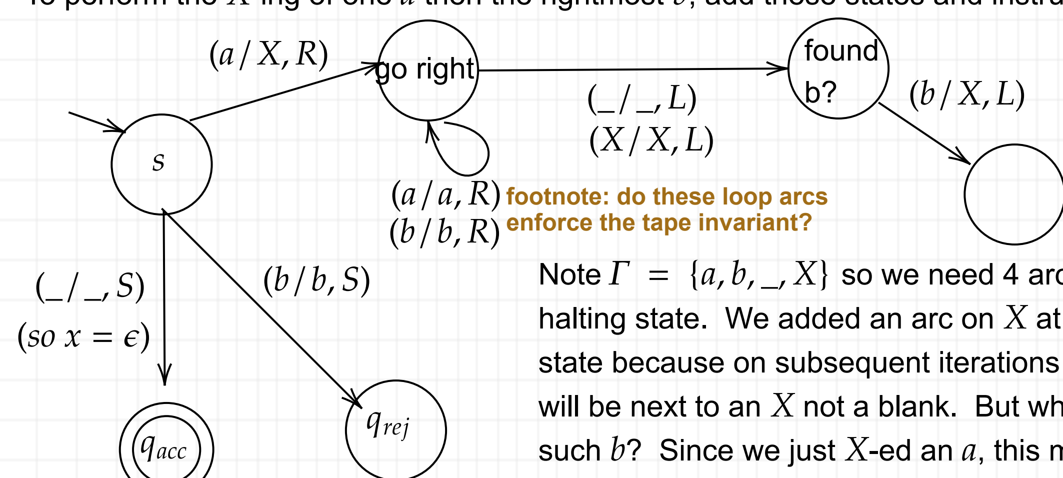
Going back to L_1, note $n = m = 0$ is allowed, and so $\epsilon \in L_1$. When the input x is ϵ, the TM tape starts off completely blank. Otherwise, the TM starts in the **configuration** of scanning the first char of x, with the rest of the tape blank. So an initial scan of _ means that $x = \epsilon$ 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_1 step-by-step with goal-oriented reasoning. [Lecture might work on the diagram "interactively"; here we show some stages.]



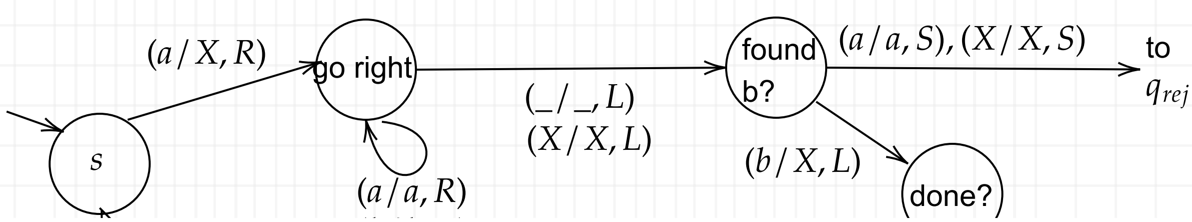
We've already been able to handle immediate accept and reject conditions in the start state. Now we decide strategy when x begins with a. The idea is to X-out a's and b's one-by-one in alternation. If we X-out always the leftmost a and the rightmost b then the string between (which after the first iteration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

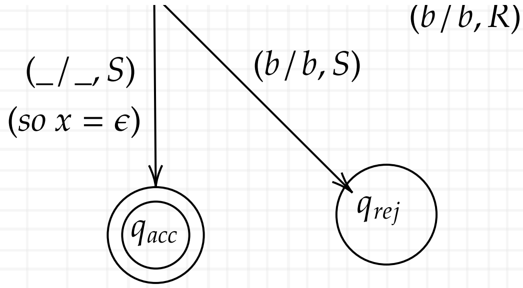
Tape Invariant: $X^*\ a^*\ b^*\ X^*$ and after X-ing a b the numbers of Xes on left and right are the same, so the string between them belongs to L_1 if and only if the original x does.

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

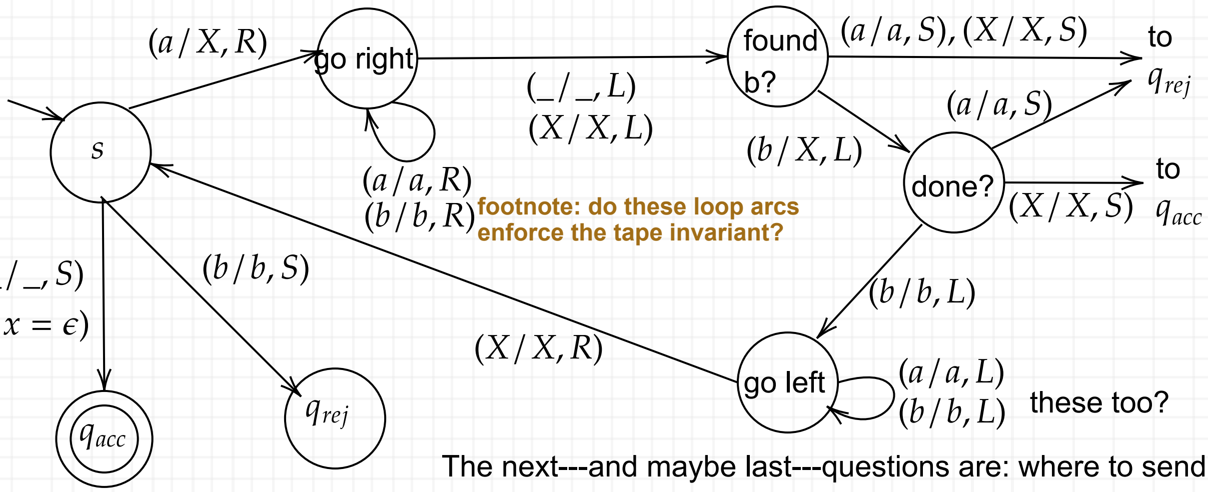


Note $\Gamma = \{a, b, \_, X\}$ so we need 4 arcs at each non-halting state. We added an arc on X at the "go right" state because on subsequent iterations the rightmost b will be next to an X not a blank. But what if there is no such b? Since we just X-ed an a, this means there were initially more a's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

Tape Invariant: $X^* \; a^* \; b^* \; X^*$

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.*

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.**

Two-Tape Turing Machines

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 $\Theta\left(n^2\right)$ 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,

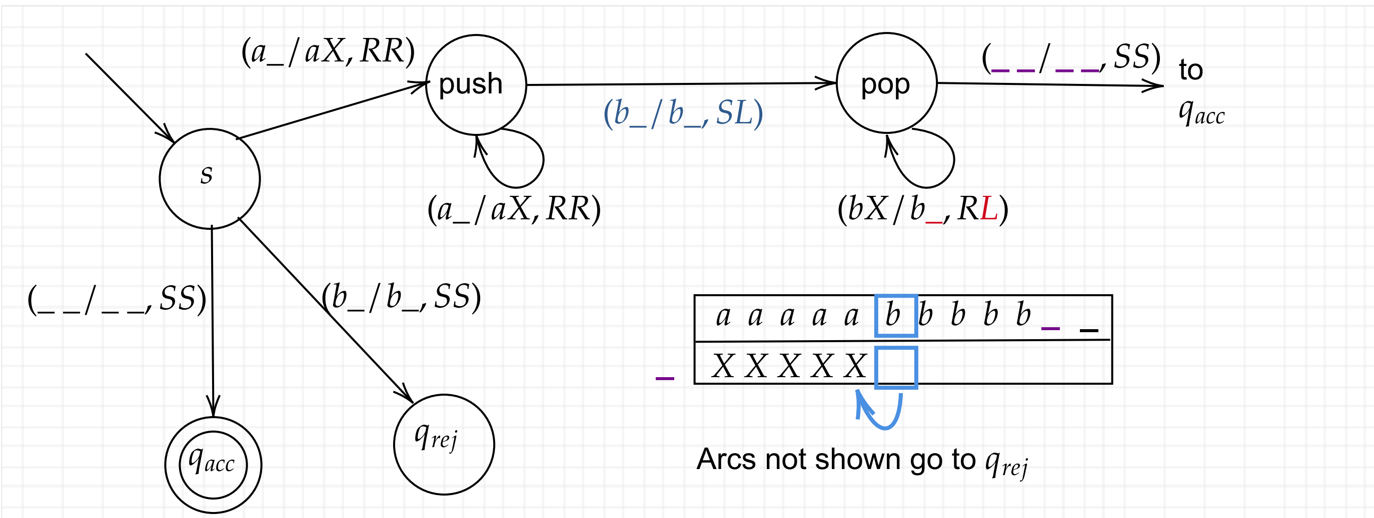
Definition: A k-*tape Turing machine* is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, \_, s, F)$ where Q, s, F and Σ are as with a DFA, the *work alphabet* Γ includes Σ and the *blank* _, and

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

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 $M = (Q, \Sigma, \Gamma, \delta, \_, s, q_{acc}, q_{rej})$. All **instructions** (still also called **5-tuples** or just **tuples**) have the form

$$(p, [c_1, c_2, \ldots, c_k] / [d_1, \ldots, d_k], [D_1, \ldots, D_k], q) \quad \text{with } p, q \in Q,\ c_j, d_j \in \Gamma, \text{ and } D_j \in \{\mathsf{L}, \mathsf{R}, \mathsf{S}\}\ (j = 1 \text{ to } k)$$

Example 2-tape TM "on paper" for $L_1 = \left\{a^m\, b^n : n = m\right\}$:



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 a DCFL that is not regular.