CSE491/596 Lecture Mon. 9/19/22 Regular and Nonregular Languages

Given a DFA $M = (Q, \Sigma, \delta, s, F)$, let us use the notation $\delta^*(p, x) =$ the state q that M is in after processing x from state p. We saw this as Δ^* for the DFA in the NFA-to-DFA proof. Note that

$$x \in L \iff \delta^*(s, x) \in F,$$

where $L = L(M)$, so

$$x \notin L \iff \delta^*(s, x) \notin F,$$

which is the same as writing

$$x \in \widetilde{L} \iff \delta^*(s, x) \in \widetilde{F}.$$

The upshot is that the DFA $M' = \left(Q, \Sigma, \delta, s, \widetilde{F}\right)$ gives $L(M') = \widetilde{L}$. This trick of complementing accepting and nonaccepting states does not, however, work for a general NFA. For example, if you try this on the NFAs N_k given for the languages L_k of binary strings whose kth bit from the end is a 1, then the new machine has an accepting loop at the start state on both 0 and 1 and so accepts every string, not just those in the complement of L_k. [I spent some time showing this from the picture of N_k in the previous lecture.] But thanks to Kleene's Theorem, being able to do it for DFAs is enough to prove:

Theorem 1: The complement of a regular language is always regular. ☒

Now suppose we have two DFAs $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ (note that Σ is the same). Let $L_1 = L(M_1)$ and $L_2 = L(M_2)$. Then let $\boldsymbol{op}$ be any binary operation on sets, such as $\cup$ or $\cap$ but note also difference $L_1 \setminus L_2$ and *symmetric difference*

$$L_1 \triangle L_2 = (L_1 \setminus L_2) \cup (L_2 \setminus L_1) = (L_1 \cup L_2) \setminus (L_1 \cap L_2) \; ,$$

whose corresponding Boolean operation $\boldsymbol{op'}$ is XOR, which is sometimes written $\oplus$. Then we have:

$$x \in L_1 \,\boldsymbol{op}\, L_2 \iff (x \in L_1 \,\boldsymbol{op'}\, x \in L_2) \iff \left(\delta_1^*(s_1, x) \in F_1\right) \boldsymbol{op'} \left(\delta_2^*(s_2, x) \in F_2\right)$$

When $\boldsymbol{op'}$ = AND, this is $\iff \; (\delta_1^*(s_1, x), \delta_2^*(s_2, x)) \in F_1 \times F_2$.
This means that **if** we define

$M_3 = (Q_3, \Sigma, \delta_3, s_3, F_3)$ with $Q_3 = Q_1 \times Q_2$ and $s_3 = (s_1, s_2)$,

and define $\delta_3((q_1, q_2), c) = (\delta_1(q_1, c), \delta_2(q_2, c))$, and use $F_3 = F_1 \times F_2$,

then $L(M_3) = L(M_1) \cap L(M_2)$.

We can use this **Cartesian product construction** for the other Boolean operations $\boldsymbol{op'}$. We just have to be more careful about how we define the final states. The general definition is

$$F_3 = \{(r_1, r_2) : r_1 \in F_1 \; \boldsymbol{op'} \; r_2 \in F_2\}.$$

Then $L(M_3) = L(M_1) \; op \; L(M_2)$. Thus we have shown the following theorem.

Theorem 2: The class of regular languages is closed under all Boolean operations.

Actually, we already could have said this right after Theorem 1 about complements. This is because OR is a native regular expression operation. OR and negation ($\neg$) form a complete set of logic operations. For instance, a AND $b \equiv \neg((\neg a)$ OR $(\neg b))$ by DeMorgan's laws.

Philosophical Interlude: Suppose L_1 and L_2 are the two regular languages you want to combine. If what you're given are DFAs M_1 and M_2 for them ("for them" means $L(M_1) = L_1$ and $L(M_2) = L_2$), then the combination M_3 can be quickly put together as above, and it doesn't matter what the operation is. But if you are originally given NFAs N_1 and N_2, it is not so easy. Well, it is easy for union/OR if you only need an NFA N_3: just join N_1 and N_2 in parallel with ϵ-arcs from a new start state as we saw in the NFA-to-regexp proof. For intersection/AND, hmmm....[Is there a way to make the Cartesian product construction idea work directly on two NFAs? That might be a good small-group discussion topic.] And if the operation is difference or symmetric difference there seems to be no way *in general* without first converting the NFAs to DFAs so you can apply the Cartesian product idea. Maybe in particular cases there are shortcuts, but in this course we emphasize general cases.

This all highlights a curious asymmetry between OR and AND. The former is a native regular expression operation. We at least a philosophical analogy to parallel circuits in Kirchoff's Laws. There does not, however, seem to be an electrical counterpart to AND. Even if you can do Cartesian product on two NFAs to handle AND, the new NFA is quadratically bigger than the two given NFAs. This may be more than analogy---it may be responsible for differences in cases where our brains find disjunctions easier to think about than conjunctions. [The difference actually really comes out when we contrast *disjunctive normal form,* which means OR-of-ANDs, with *conjunctive normal form*, which is AND-of-ORs. We will hit this when doing NP-completeness.]

Myhill-Nerode Theorem [John Myhill, UB †1987, Anil Nerode, Cornell]

Given a DFA $M = (Q, \Sigma, \delta, s, F)$ and two strings $x, y \in \Sigma^*$, suppose $\delta^*(s, x)$ and $\delta^*(s, y)$ both give the same state q. Then for any further string $z \in \Sigma^*$, the computations on the strings xz and yz go through the same states after q. In particular, they end at the same state r.

- If $r \in F$, then $xz \in L$ and $yz \in L$, where $L = L(M)$.
- If $r \notin F$, then $xz \notin L$ and $yz \notin L$.
- Either way, $L(xz) = L(yz)$, for all z.

Suppose, on the other hand, we have strings x, y for which there exists a string z such that

$$L(xz) \neq L(yz).$$

Then M cannot process x and y to the same state. Moreover, this goes for *any* DFA M such that $L(M) = L$. In particular, every such DFA must at least *have* two states.

Now let us build some definitions around these ideas. Given any language L (not necessarily regular) and strings x, y "over" the alphabet Σ that L is "over", define:

- x and y are L-*equivalent*, written $x \sim_L y$, if for all $z \in \Sigma^*$, $L(xz) = L(yz)$.
- x and y are *distinctive for* L, written $x \nsim_L y$, if there exists $z \in \Sigma^*$ s.t. $L(xz) \neq L(yz)$.

Example: $L = \{w : \#1(w) \text{ is even}\}$. Then $x = 1101$ and $y = 1011$ give $x \sim_L y$, even though neither x nor y belong to L. E.g. $z = 01$ makes $xz = 110101$ and $yz = 101101$ both belong to L, so $L(xz) = L(yz) = true$. And with $z = 101$ instead, both would be false.

Lemma 1. The relation $\sim_L$ is an equivalence relation.

Proof: We need to show that it is
- Reflexive: $x \sim_L x$ is obvious.
- Symmetric: indeed, $y \sim_L x$ immediately means the same as $x \sim_L y$.
- Transitive: Suppose $w \sim_L x$ and $x \sim_L y$. This means:
  - for all $v \in \Sigma^*$, $L(wv) = L(xv)$ and
  - for all $z \in \Sigma^*$, $L(xz) = L(yz)$.
    Because v and z range over the same span of strings, it *follows* that
  - for all $z \in \Sigma^*$, $L(wz) = L(xz)$ and $L(xz) = L(yz)$.
    Hence we get:
  - for all $z \in \Sigma^*$, $L(wz) = L(yz)$.
    So $w \sim_L y$.

This ends the proof. ☒

Any equivalence relation on a set such as Σ^* partitions that set into disjoint *equivalence classes*. So $x \nsim_L y$ is the same as saying x and y belong to different equivalence classes. If you make E_3 be the language where the number of 1s is a multiple of 3, you get 3 equivalence classes. And so on...]
$S = \{101, 100, 1011\}$

Now say that a set S of strings is ***Pairwise Distinctive for*** L if all of its strings belong to separate equivalence classes under the relation $\sim_L$. Other names we will use are "distinctive set" and "**PD set**" for L. This is the same as saying:

- for all $x, y \in S$, $x \neq y$, there exists $z \in \Sigma^*$ such that $L(xz) \neq L(yz)$.

Thus we can re-state something we said above as:

Lemma 2. If L has a PD set S of size 2, then any DFA M such that $L(M) = L$ must process the two strings in S to different states, so M must have at least 2 states.

Note: "L has" does not mean S must be a subset of L, it just means "has by association." Now we can take this logic further:

Lemma k. If L has a PD set S of size k, then any DFA M such that $L(M) = L$ must process the k strings in S to different states, so M must have at least k states.

I've worded this to try to make it as "obvious" as possible, but actually it needs proof: Suppose we have a DFA M with $k-1$ or fewer states such that $L(M) = L$. Then there must be (at least) two strings in S that M processes to the same state. This follows by the **Pigeonhole Principle**.

[tell story]
[finish proof]
Then explain why we get the infinite case:

Lemma ∞. If L has a PD set S of size ∞, then any DFA M such that $L(M) = L$ must process the strings in S to different states, so M must have at least ∞ states...but then M is not a *finite* automaton. So L is not accepted by any finite automaton...which means **L is not a regular language.** ☒

Myhill-Nerode Theorem, first half: If L has an infinite PD set, then L is not regular.

Example: $L = \left\{a^n\, b^n : n \geq 0\right\}$. $\Sigma = \{a, b\}$. $S = \left\{a^n : n \geq 0\right\} = a^*$. Let any $x, y \in S$, $x \neq y$, be given. Then there are different numbers i and j such that $x = a^i$ and $y = a^j$. Take $z = b^i$. Then $xz = a^i b^i \in L$, but $yz = a^j\, b^i \notin L$, because $i \neq j$. Thus $L(xz) \neq L(yz)$. Thus for all $x, y \in S$ with $x \neq y$, there exists z such that $L(xz) \neq L(yz)$. Thus S is PD for L. Since S is infinite, L is not regular, by MNT. ☒

[Then I drew a connection from this to the idea of playing the spears-and-dragons game when you can save any number of spears. In the basic case where you can save at most 1 spear the DFA has 3 states, and these are mandated because $S = \{\epsilon, \$, D\}$ is a PD set of size 3. In particular, even though both $x = \epsilon$ and $y = \$$ are strings **in** the language L_1 of the 1-spear game, they are distinctive **for** L_1 because $z = D$ kills you in the former case (i.e., $xz = \epsilon D = D \notin L_1$) but you stay alive in the latter case (i.e., $yz = \$D \in L_1$). If you can save up to 2 spears, then $\epsilon, \$, \$$ are three distinctive strings (plus D to make a fourth). Well, if you can save unlimited spears, then $S_\infty = \{\epsilon, \$, \$, \$\$, \ldots\}$ becomes an infinite PD set by similar logic to the $\left\{a^n\, b^n\right\}$ example. So the most liberal form of the game gives no longer a regular language.]