CSE491/596 Lecture Monday Sept. 7, 2020

The formal definition of a *finite automaton* is a 5-tuple (i.e., an object) $N = (Q, \Sigma, \delta, s, F)$ where:

- Q is a finite set of *states* `set<State> Q;`
- Σ is the *input alphabet* `set<char> Sigma;`
- s, a member of Q, is the *start state* (also called q_0) `State s;`
- F, a subset of Q, is the set of *accepting* states (also called *final* states) `set<State> F;`
- δ is a finite set of *instructions* (also called *transitions*) of the form (p, c, q) where $p, q \in Q$ and $c \in \Sigma$; an NFA with ϵ-transitions (NFA$_\epsilon$) also allows (p, ϵ, q). `set<Triple<State,char,State> > delta;`

The machine is *deterministic* (a DFA) if $(\forall p \in Q)(\forall c \in \Sigma)(\exists! q \in Q) : (p, c, q) \in \delta$. Else it is "properly" *nondeterministic* (an NFA).

So DFA is a special case of an NFA. When we have a DFA M, we can regard δ as a function from $Q \times \Sigma$ to Q. With an NFA, we could regard δ as a function from $Q \times \Sigma$ to 2^Q, which is the set of all subsets of Q and called the *power set* of Q. But in most cases I prefer to think of δ as a set of instructions---the same as "trominoes" in my previous lecture.

Say that N *can process* a string x ***from*** state p ***to*** state q if there is a sequence of instructions

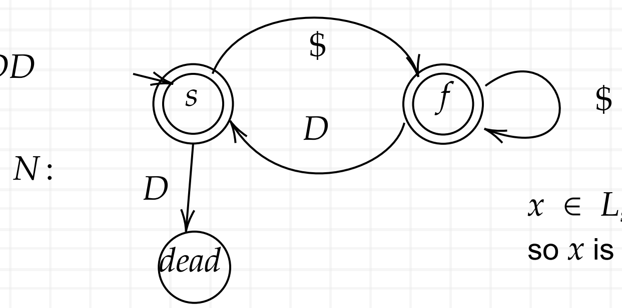
$$(p, c_1, q_1)(q_1, c_2, q_2)(q_2, c_3, q_3) \cdots (q_{m-2}, c_{m-1}, q_{m-1})(q_{m-1}, c_m, q)$$

such that $c_1 c_2 \cdots c_m = x$. Then we write $x \in L_{pq}$ (with N understood). Now formally define:

$$L(N) = \cup_{f \in F} L_{sf}.$$

If N has only one accepting state (a design goal we can meet for NFAs but often not for DFAs) then the language is just L_{sf}. We will find the L_{pq} concept especially handy with "GNFAs" on Fri.

$x = \$DD$

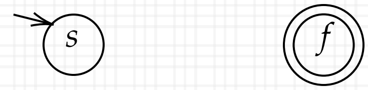


$x \in L_{s,dead}$ but $dead$ is not accepting so x is not in the language.

Without the dead state and arc to it, the NFA N on input $x = \$DD$ would "crash" in state s. Even though s is an accepting state (and even though this would count as legal termination by a Turing machine), not all of x would be processed, so it does not count in the FA's language. With the dead state present, x gets processed to $dead$, but $dead \notin F$ so $x \notin L(N)$ still.

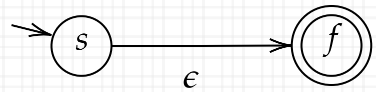
Regular Expressions and Their Corresponding NFAs (with ϵ-transitions):

(B1) $\emptyset$ is a regexp; $L(\emptyset) = \emptyset$; $N_{\emptyset} =$



$(\delta = \emptyset)$

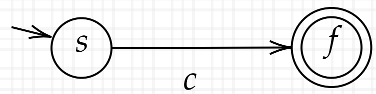
(B2) ϵ is a regexp; $L(\epsilon) = \{\epsilon\}$; $N_{\epsilon} =$



δ has (s, ϵ, f)

For all chars $c \in \Sigma$:

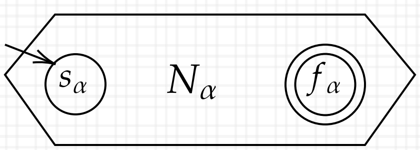
(B3) $\mathbf{c}$ is a regexp; $L(\mathbf{c}) = \{c\}$; $N_{\mathbf{c}} =$

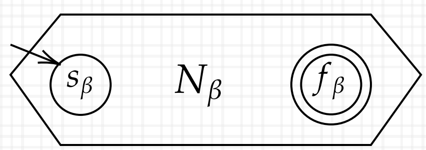


This completes the *basis* of an *inductive definition* of regular expressions. Now let α and β be any two regular expressions, with languages $A = L(\alpha)$ and $B = L(\beta)$. By *inductive hypothesis* (**IH**) we have NFAs N_α and N_β such that $L(N_\alpha) = A$ and $L(N_\beta) = B$. Then:

(I1) $\gamma = \alpha + \beta$ is a regexp; $L(\gamma) = A \cup B$.

Now to complete the *induction case* (I1) we need to show how to build an NFA$_\epsilon$ N_γ such that $L(N_\gamma) = L(\gamma)$. What we have to work with is (are) N_α and N_β. We know they have start states we can call s_α and s_β. Taking a cue from the base case NFAs, and mainly for convenience, we may suppose they have unique accepting states f_α and f_β. Besides that, we make no assumptions about their internal structure, so we draw them as "blobs":





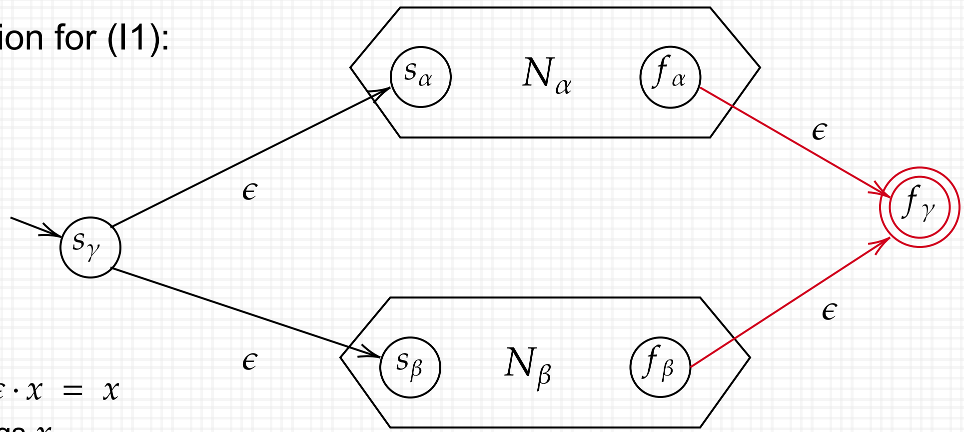
The goal is to connect them together to make N_γ with needed properties, also for the cases:

(I2) $\gamma = \alpha \cdot \beta$ is a regexp; $L(\gamma) = A \cdot B$.

(I3) $\gamma = \alpha^*$ is a regexp; $L(\gamma) = A^*$. (In I3 we have only N_α given.)

Construction for (I1):

$N_\gamma \ =$



note rule: $\epsilon \cdot x \ = \ x$
for all strings x.

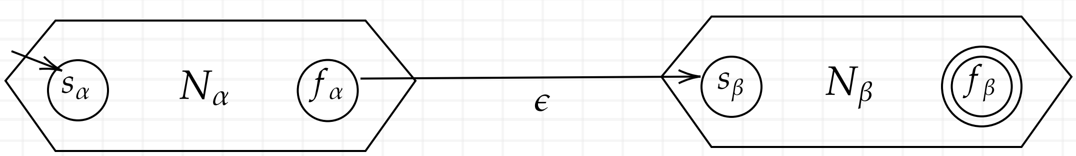
This builds N_γ, but we still need to prove it is correct, i.e., $L(N_\gamma) \ = \ L(\gamma)$. *Note the rhythm:*

1. $L(N_\gamma) \ = \ L(N_\alpha) \ \cup \ L(N_\beta)$ — by machine construction;
2. $L(N_\alpha) \ = \ L(\alpha)$ and $L(N_\beta) \ = \ L(\beta)$ — by inductive hypothesis;
3. Thus $L(N_\gamma) \ = \ L(\alpha) \ \cup \ L(\beta) \ = \ L(\alpha + \beta) \ = \ L(\gamma)$ — by definition of γ.

[I will continue as time permits by copy-and-paste and moving things around to do the other two inductive cases to complete the proof. But first, are you completely happy with N_γ as it stands?]
[Answer was *no*: adding the state f_γ and ϵ-arcs shown in red "preserves the invariant" of the NFAs all having a single accepting state.]

(I2) $\gamma \ = \ \alpha \ \cdot \ \beta$ is a regexp; $L(\gamma) \ = \ A \cdot B \ = \ \{xy : \ x \ \in A \ \wedge \ y \ \in \ B\}$.

N_γ:



Then $L(N_\gamma) \ = \ L(N_\alpha) \ \cdot \ L(N_\beta)$ because....processing....

To write the reasoning out: N_γ can process a string z from its start state $s_\gamma \ = \ s_\alpha$ to its (unique) final state $f_\gamma \ = \ f_\beta$ if and only if z has a first part x that gets processed from s_α to f_α and a second part y that gets processed from s_β to f_β (with the ϵ from f_α to s_β silently in-between). I.e.: $z \ \in \ L(N_\gamma) \iff z \ \in \ \{x \cdot y : \ x \ \in \ L(N_\alpha) \ \wedge \ y \ \in \ L(N_\beta)\} \iff z \ \in \ L(N_\alpha) \cdot L(N_\beta)$. Thus $L(N_\gamma) \ = L(N_\alpha) \cdot L(N_\beta)$. By **IH**, this equals $L(\alpha) \cdot L(\beta)$, which by how the semantics of $\gamma \ = \ \alpha \cdot \beta$ is defined via $L(\gamma) \ = \ L(\alpha) \cdot L(\beta)$ finally gives us the needed conclusion $L(N_\gamma) \ = \ L(\gamma)$.

The proof will be finished with the star case (I3) on Wednesday.