CSE491/596: Lecture Fri. 9/8/23

One HW problem will be: Consider the "spears and dragons" DFA from the demo, which had alphabet $\Sigma = \{\,, D, 0\}$ where '0' means an empty room. Build deterministic finite automata that model the following alterations to the game:

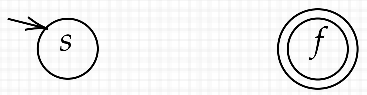
- (a) You may hold up to 2 spears, but not 3. If you have 2 spears and kill a dragon, you are down to one spear.
- (b) You may hold up to 2 spears, but may only hold 2 spears for a limited time. If you get two empty rooms after picking up the second spear, you have to drop down to carrying one spear.

In each case, the language of your machine should be the set of strings $x \in \Sigma^*$ that represent "dungeons" where the Player survives. It is fine for the Player to exit holding zero spears, one spear, or two spears.

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

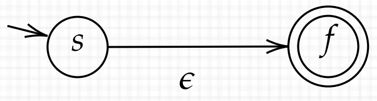
We give an inductive definition with three base cases B1,B2,B3 and three inductive cases I1,I2,I3. At the same time, we give an inductive procedure for building an equivalent NFA with ϵ-transitions.

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



$(\delta = \emptyset)$

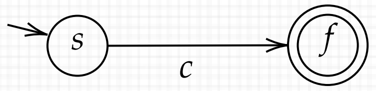
(B2) ϵ is a regexp; $L(\epsilon) = \{\epsilon\}$; $N_\epsilon =$



δ has (s, ϵ, f)

For all chars $c \in \Sigma$:

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

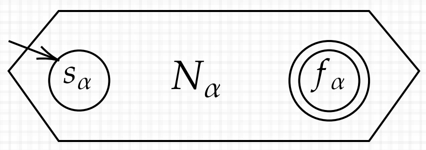


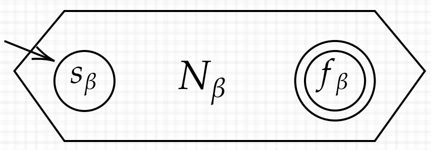
δ has (s, c, f)

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; $L(\gamma) = L(\alpha) \cup L(\beta)$.

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":





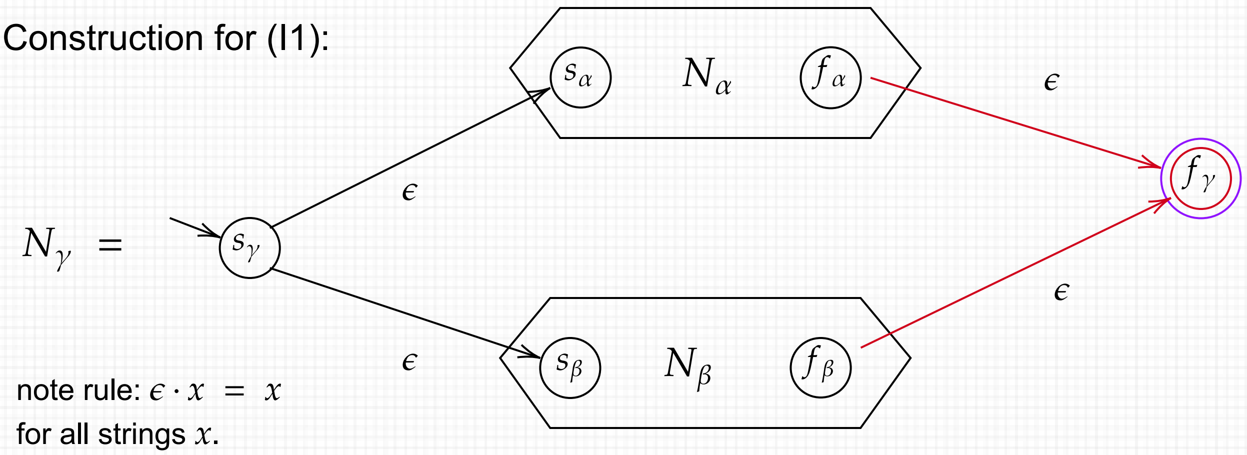
The goal is to connect them together to make N_γ with needed properties, also for the cases:

(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.)

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

Construction for (I1):

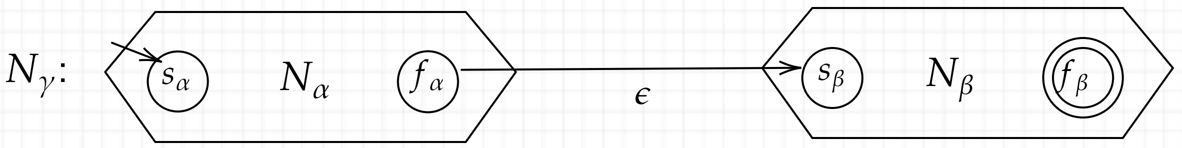


note rule: $\epsilon \cdot x = x$
for all strings x.

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.

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



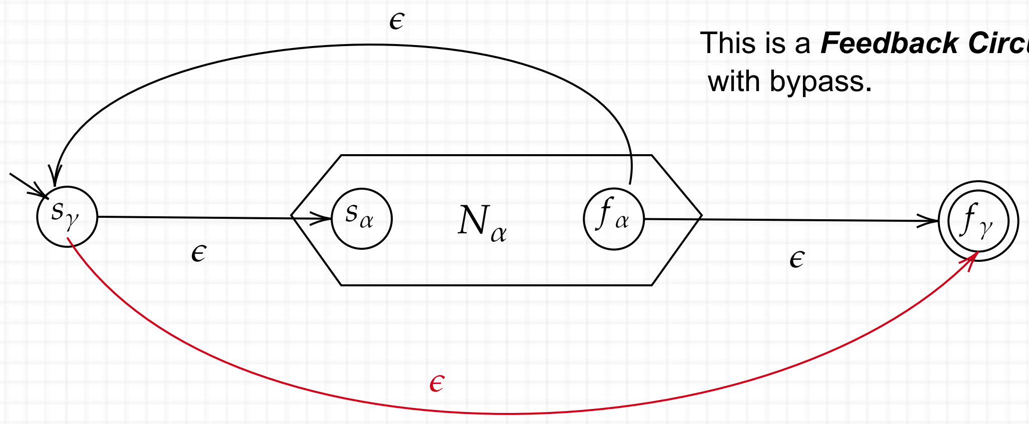
Then $L(N_\gamma) = L(N_\alpha) \cdot L(N_\beta)$ because....processing....

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 $L(N_\gamma) = L(\gamma)$.

Now we finish the proof with the Kleene Star case---named for Stephen Kleene:

(I3) Given any regexp α, $\gamma = \alpha^*$ is a regexp; $L(\gamma) = L(\alpha)^*$; and we can build:

N_γ =



This is a ***Feedback Circuit*** with bypass.

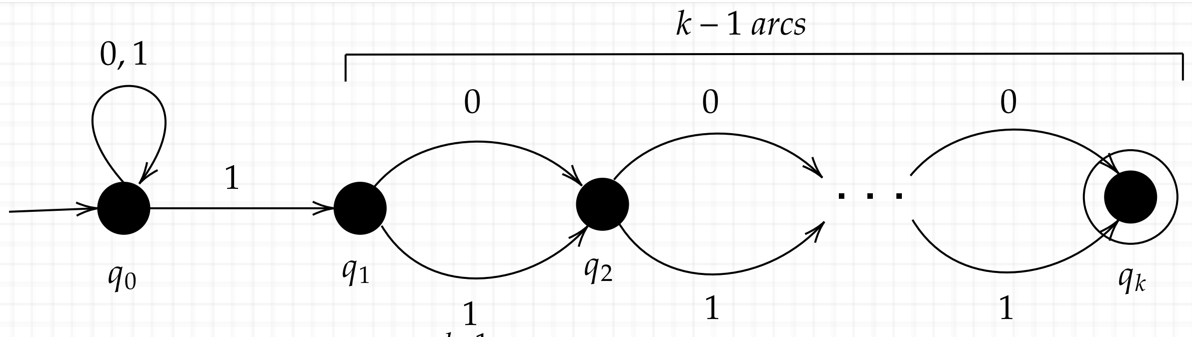
Is this good? We want to make $L(N_\gamma) = L(N_\alpha)^*$. Then the IH $L(N_\alpha) = L(\alpha)$ will give $L(N_\gamma) = L(\alpha)^* = L\left(\alpha^*\right) = L(\gamma)$ as needed---to finish the whole proof.

The Main Theorem About Regular Expressions and Finite Automata

Theorem: For any language A over an alphabet Σ, the following statements are equivalent:

1. There is a regular expression α such that $A = L(\alpha)$.
2. There is an NFA N such that $A = L(N)$.
3. There is a DFA M such that $A = L(M)$.

Example: The "Leap of Faith" NFAs N_k for any $k > 1$:



$$L(N_k) = (0+1)^*1(0+1)^{k-1}$$
$$= \left\{x \in \{0,1\}^* : \textit{the kth bit of } x \textit{ from the end is a } 1\right\}.$$

Fact (will be proved later): Whereas the NFA N_k has only $k+1$ states, the smallest DFA M_k such that $L(M_k) = L(N_k)$ requires 2^k states.
This is a case of **exponential blowup** in the NFA-to-DFA algorithm.
For now, we just care that an equivalent DFA **can** be built.