

Week 3 at a glance

Textbook reading: Chapter 1

No class on Week 3 Monday in observance of Martin Luther King Jr. Day.

Before Wednesday: read the definition of the union, concatenation, and star operations for languages, given as Definition 1.23 on page 44 and a useful example is Example 1.24.

Before Friday, read pages 45-46 (Theorem 1.25) that we'll refer to as a "closure proof".

For Week 4 Monday, read Introduction to Section 1.4 (page 77) which introduces nonregularity.

We will be learning and practicing to:

- Clearly and unambiguously communicate computational ideas using appropriate formalism. Translate across levels of abstraction.
 - Use precise notation to formally define the state diagram of finite automata.
 - Use clear English to describe computations of finite automata informally.
 - * **Motivate the use of nondeterminism**
 - * **State the formal definition of NFA**
 - * **Trace the computation(s) of a NFA on a given string using its state diagram**
 - * **Determine if a given string is in the language recognized by a NFA**
 - * **Translate between a state diagram and a formal definition of a NFA**
 - Give examples of sets that are regular (and prove that they are).
 - * **State the definition of the class of regular languages**
 - * **Give examples of regular languages, using each of the three equivalent models of computation for proving regularity.**
 - * **Choose between multiple models to prove that a language is regular**
 - * **Explain the limits of the class of regular languages**
 - Describe and use models of computation that don't involve state machines.
 - * **Given a DFA or NFA, find a regular expression that describes its language.**
 - * **Given a regular expression, find a DFA or NFA that recognizes its language.**
- Understand, guide, shape impact of computing on society/the world. Connect the role of Theory CS classes to other applications (in undergraduate CS curriculum and beyond). Model problems using appropriate mathematical concepts.
 - Explain nondeterminism and describe tools for simulating it with deterministic computation.
 - * **Given a NFA, find a DFA that recognizes its language.**
 - * **Convert between regular expressions and automata**

TODO:

Schedule your Test 1 Attempt 1, Test 2 Attempt 1, Test 1 Attempt 2, and Test 2 Attempt 2 times at PrairieTest (<http://us.prairietest.com>)

Review Quiz 3 on PrairieLearn (<http://us.prairielearn.com>), due 1/29/2025

Homework 2 submitted via Gradescope (<https://www.gradescope.com/>), due Tuesday 1/30/2025

In Computer Science, we operationalize “hardest” as “requires most resources”, where resources might be memory, time, parallelism, randomness, power, etc. To be able to compare “hardness” of problems, we use a consistent description of problems

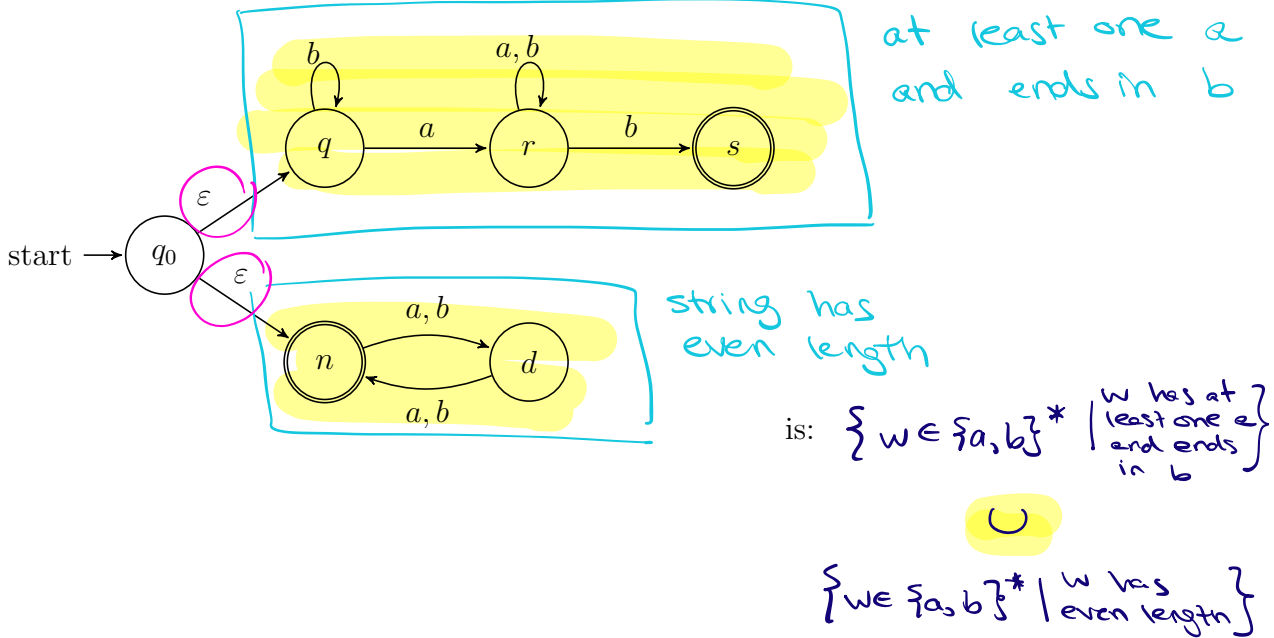
Input: String

Output: Yes/ No, where Yes means that the input string matches the pattern or property described by the problem.

So far: we saw that regular expressions are convenient ways of describing patterns in strings. **Finite automata** give a model of computation for processing strings and classifying them into Yes (accepted) or No (rejected). We will see that each set of strings is described by a regular expression if and only if there is a FA that recognizes it. Another way of thinking about it: properties described by regular expressions require exactly the computational power of these finite automata.

Wednesday: Automata constructions

Review: The language recognized by the NFA over $\{a, b\}$ with state diagram

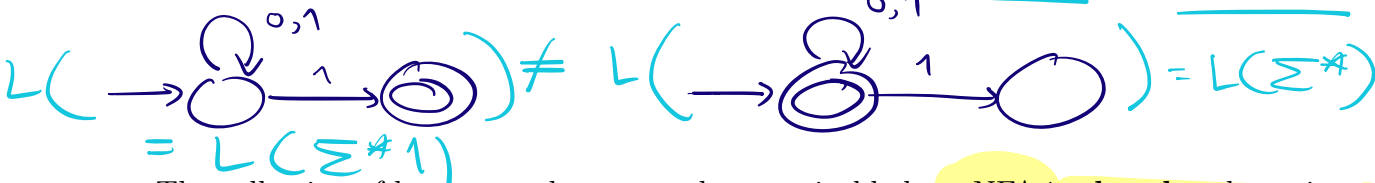
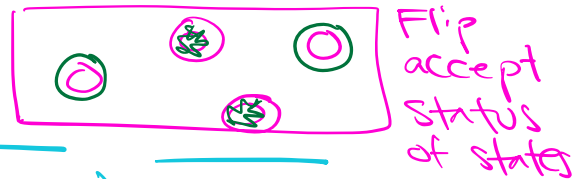


So far, we know:

- The collection of languages that are each recognizable by a DFA is **closed** under complementation.

Could we do the same construction with NFA?

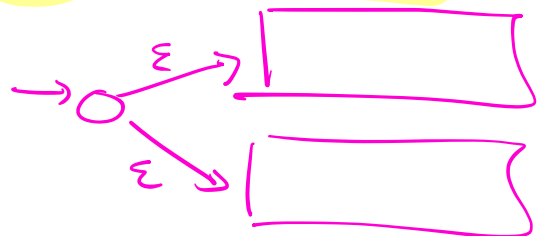
No! counterexample



- The collection of languages that are each recognizable by a NFA is **closed** under union.

Could we do the same construction with DFA?

No! Spontaneous move is not possible in DFA.



Happily, though, an analogous claim is true!

Suppose A_1, A_2 are languages over an alphabet Σ . **Claim:** if there is a DFA M_1 such that $L(M_1) = A_1$ and DFA M_2 such that $L(M_2) = A_2$, then there is another DFA, let's call it M , such that $L(M) = A_1 \cup A_2$.
Theorem 1.25 in Sipser, page 45

Proof idea: Keep track of both computations of M_1 and M_2 at once.

Formal construction:

Let A_1 be a language over Σ and A_2 be a language over Σ and let $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ be a DFA with $L(M_1) = A_1$ and let $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ be a DFA with $L(M_2) = A_2$

We want to build M with $L(M) = A_1 \cup A_2$.

Define $M = (Q, \Sigma, \delta, q_0, F)$ where

$$Q = \{ (q, q') \mid q \in Q_1 \text{ and } q' \in Q_2 \} = Q_1 \times Q_2$$

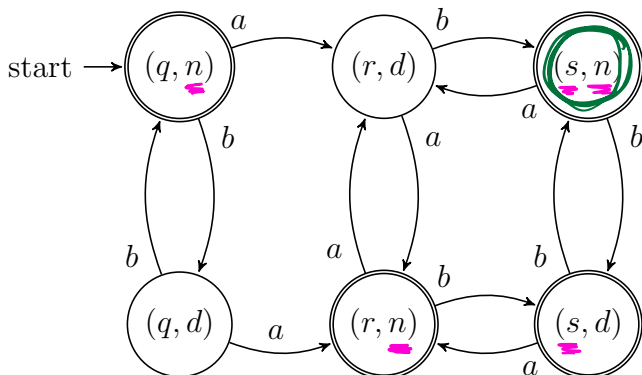
$\delta: Q \times \Sigma \rightarrow Q$ is given by

$$\delta((q, q'), x) = (\delta_1(q, x), \delta_2(q', x))$$

$$q_0 = (q_{01}, q_{02})$$

$$F = F_1 \times Q_2 \cup Q_1 \times F_2$$

Example: When $A_1 = \{w \mid w \text{ has an } a \text{ and ends in } b\}$ and $A_2 = \{w \mid w \text{ is of even length}\}$.



Note: $|Q_1 \times Q_2| = |Q_1| |Q_2|$

$$F_1 = \{s\}$$

$$F_2 = \{n\}$$

$F_1 \times F_2$

Suppose A_1, A_2 are languages over an alphabet Σ . **Claim:** if there is a DFA M_1 such that $L(M_1) = A_1$ and DFA M_2 such that $L(M_2) = A_2$, then there is another DFA, let's call it M , such that $L(M) = A_1 \cap A_2$.
Footnote to Sipser Theorem 1.25, page 46

Proof idea: Same construction as union, except require both computations to be accepting.

Formal construction:

Let A_1 be a language over Σ
and A_2 be a language over Σ
and let $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ be a
DFA with $L(M_1) = A_1$ and let

$M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ be a DFA with $L(M_2) = A_2$

We want to build M with $L(M) = A_1 \cap A_2$.

Define $M = (Q, \Sigma, \delta, q_0, F)$ where

$$Q = \{ (q, q') \mid q \in Q_1 \text{ and } q' \in Q_2 \} = Q_1 \times Q_2$$

$\delta: Q \times \Sigma \rightarrow Q$ is given by

$$\delta((q, q'), x) = (\delta_1(q, x), \delta_2(q', x))$$

$$q_0 = (q_{01}, q_{02})$$

$$F = F_1 \times F_2$$

Claim: $L(M) = L(M_1) \cap L(M_2)$

Pf: \subseteq
 \supseteq

So far:

- class of languages recognized by DFA is closed under complement, union, intersect
- class of languages recognized by NFA is closed under union
- class of languages recognized by regular expressions is closed under union, setwise concatenation, Kleene star

Friday: Regular languages

So far we have that:

- If there is a DFA recognizing a language, there is a DFA recognizing its complement.
- If there are NFA recognizing two languages, there is a NFA recognizing their union.
- If there are DFA recognizing two languages, there is a DFA recognizing their union.
- If there are DFA recognizing two languages, there is a DFA recognizing their intersection.

Our goals for today are (1) prove similar results about other set operations, (2) prove that NFA and DFA are equally expressive, and therefore (3) define an important class of languages.

A language is regular means

there is a regular expression that describes it

or, equivalently,

there is a DFA that recognizes it

or, equivalently,

there is a NFA that recognizes it

$$A_1 \circ A_2 = \{ \underline{uv} \mid u \in A_1 \text{ and } v \in A_2 \} = \{ w \in \Sigma^* \mid \exists u \in \Sigma^* \exists v \in \Sigma^* (w = uv \text{ and } u \in A_1 \text{ and } v \in A_2) \}$$

Suppose A_1, A_2 are languages over an alphabet Σ . **Claim:** if there is a NFA N_1 such that $L(N_1) = A_1$ and NFA N_2 such that $L(N_2) = A_2$, then there is another NFA, let's call it N , such that $L(N) = A_1 \circ A_2$.

Proof idea: Allow computation to move between N_1 and N_2 "spontaneously" when reach an accepting state of N_1 , guessing that we've reached the point where the two parts of the string in the set-wise concatenation are glued together.

Formal construction: Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ and assume $Q_1 \cap Q_2 = \emptyset$. Construct $N = (Q, \Sigma, \delta, q_0, F)$ where

- $Q = Q_1 \cup Q_2$

- $q_0 = q_1$

- $F = F_2$

- $\delta : Q \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q)$ is defined by, for $q \in Q$ and $a \in \Sigma_\epsilon$:

Domain

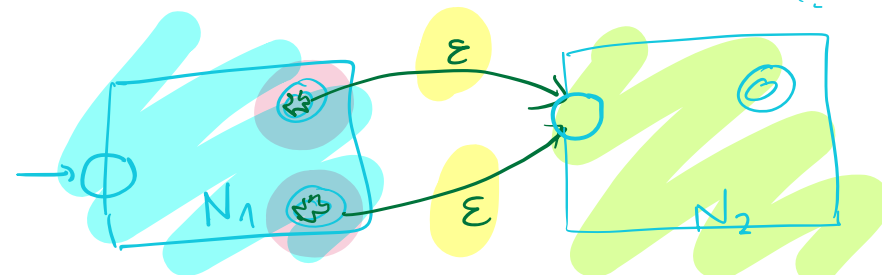
Codomain

$$\delta((q, a)) = \begin{cases} \delta_1((q, a)) & \text{if } q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1((q, a)) & \text{if } q \in F_1 \text{ and } a \in \Sigma \\ \delta_1((q, a)) \cup \{q_2\} & \text{if } q \in F_1 \text{ and } a = \epsilon \\ \delta_2((q, a)) & \text{if } q \in Q_2 \end{cases}$$

Well defined

simulating N_1

also allow spontaneous jump to N_2

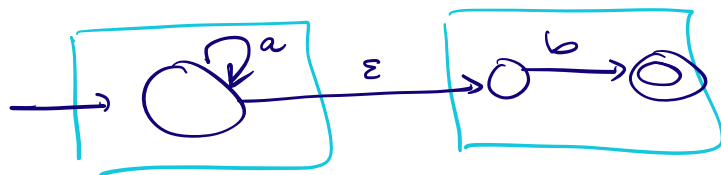


Example: build NFA recognizing $L(a^*b)$

$A_1 = L(a^*)$



$A_2 = L(b)$



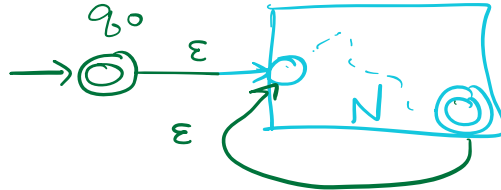
Proof of correctness would prove that $L(N) = A_1 \circ A_2$ by considering an arbitrary string accepted by N , tracing an accepting computation of N on it, and using that trace to prove the string can be written as the result of concatenating two strings, the first in A_1 and the second in A_2 ; then, taking an arbitrary string in $A_1 \circ A_2$ and proving that it is accepted by N . Details left for extra practice.

$$A^* = \{ w_1 \cdots w_k \mid k \geq 0, w_i \in A \}$$

Suppose A is a language over an alphabet Σ . **Claim:** if there is a NFA N such that $L(N) = A$, then there is another NFA, let's call it N' , such that $L(N') = A^*$.

Proof idea: Add a fresh start state, which is an accept state. Add spontaneous moves from each (old) accept state to the old start state.

Formal construction: Let $N = (Q, \Sigma, \delta, q_1, F)$ and assume $q_0 \notin Q$. Construct $N' = (Q', \Sigma, \delta', q_0, F')$ where



- $Q' = Q \cup \{q_0\}$

- $F' = F \cup \{q_0\}$

- $\delta' : Q' \times \Sigma_\epsilon \rightarrow \mathcal{P}(Q')$ is defined by, for $q \in Q'$ and $a \in \Sigma_\epsilon$:

$$\delta'((q, a)) = \begin{cases} \delta((q, a)) & \text{if } q \in Q \text{ and } q \notin F \\ \delta((q, a)) & \text{if } q \in F \text{ and } a \in \Sigma \\ \delta((q, a)) \cup \{q_1\} & \text{if } q \in F \text{ and } a = \epsilon \\ \{q_1\} & \text{if } q = q_0 \text{ and } a = \epsilon \\ \emptyset & \text{if } q = q_0 \text{ and } a \in \Sigma \end{cases}$$

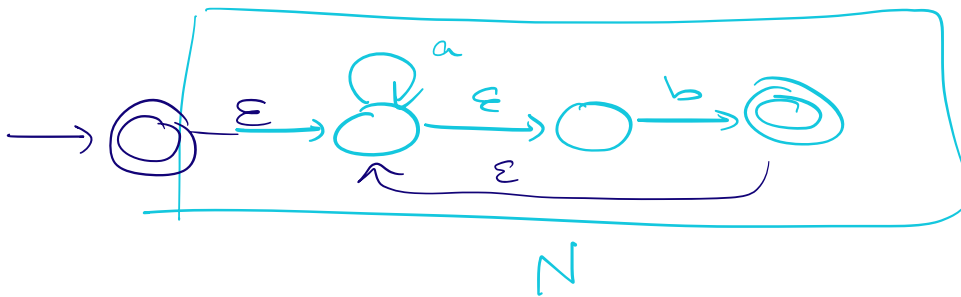
simulate N

add spontaneous move from accept states

spontaneously move from new start state to old start

Proof of correctness would prove that $L(N') = A^$ by considering an arbitrary string accepted by N' , tracing an accepting computation of N' on it, and using that trace to prove the string can be written as the result of concatenating some number of strings, each of which is in A ; then, taking an arbitrary string in A^* and proving that it is accepted by N' . Details left for extra practice.*

Application: A state diagram for a NFA over $\Sigma = \{a, b\}$ that recognizes $L((a^*b)^*)$:



Know: DFA can give NFA spontaneous moves $\xrightarrow{\epsilon}$
 BUT NFAs can have missing arrows
 have extra arrows

Suppose A is a language over an alphabet Σ . **Claim:** if there is a NFA N such that $L(N) = A$ then there is a DFA M such that $L(M) = A$.

Proof idea: States in M are "macro-states" - collections of states from N - that represent the set of possible states a computation of N might be in.

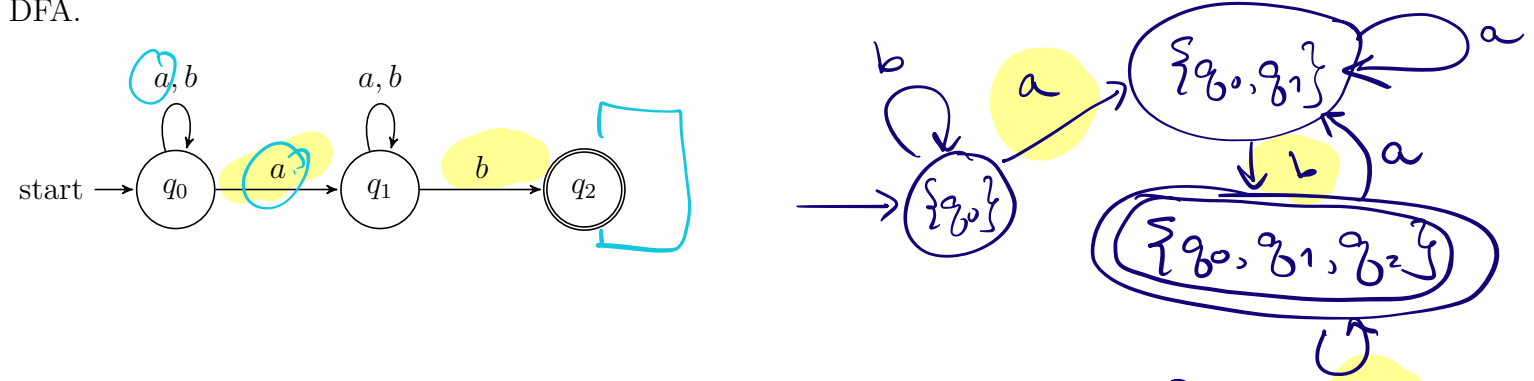
Formal construction: Let $N = (Q, \Sigma, \delta, q_0, F)$. Define

$$M = (\mathcal{P}(Q), \Sigma, \delta', q', \{X \subseteq Q \mid X \cap F \neq \emptyset\})$$

where $q' = \{q \in Q \mid q = q_0 \text{ or is accessible from } q_0 \text{ by spontaneous moves in } N\}$ and

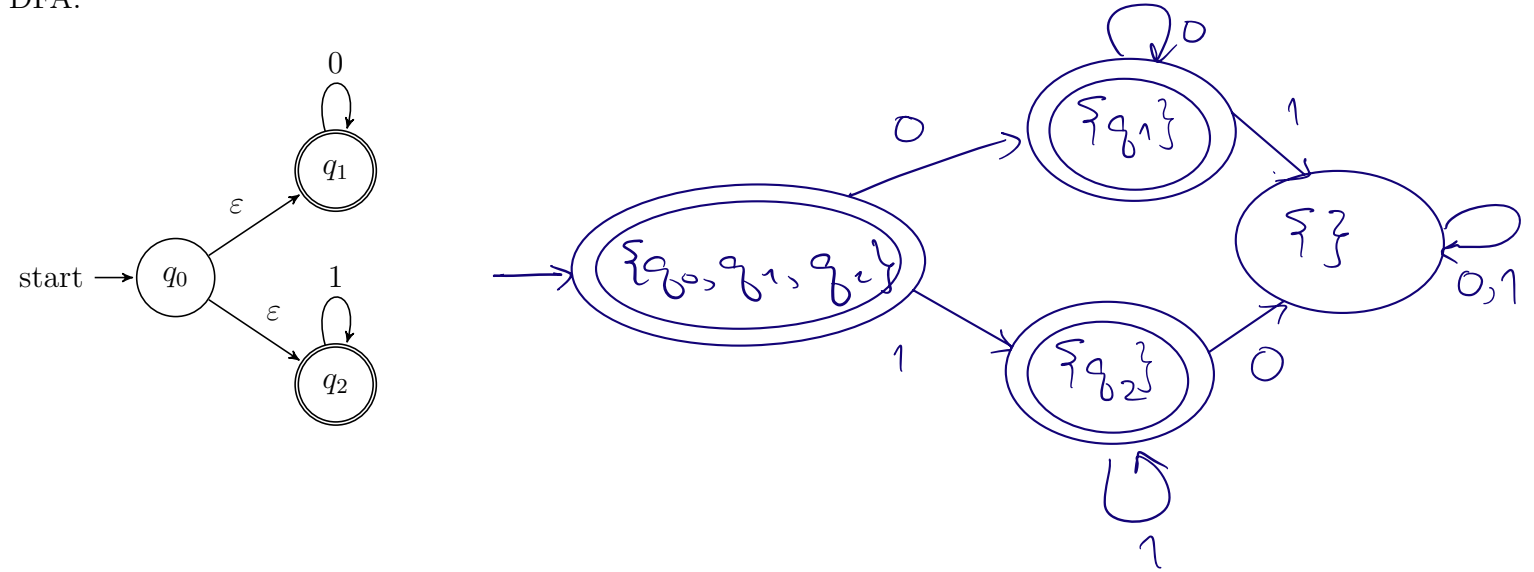
$\delta'((X, x)) = \{q \in Q \mid q \in \delta(r, x) \text{ for some } r \in X \text{ or is accessible from such an } r \text{ by spontaneous moves in } N\}$

Consider the state diagram of an NFA over $\{a, b\}$. Use the "macro-state" construction to find an equivalent DFA.



$\{w \in \{a, b\}^* \mid w \text{ has an } a \text{ and ends in } b\}$.

Consider the state diagram of an NFA over $\{0, 1\}$. Use the "macro-state" construction to find an equivalent DFA.



Note: We can often prune the DFAs that result from the "macro-state" constructions to get an equivalent DFA with fewer states (e.g. only the "macro-states" reachable from the start state).

The class of regular languages

Fix an alphabet Σ . For each language L over Σ :

There is a DFA over Σ that recognizes L
if and only if

$\exists M$ (M is a DFA and $L(M) = A$)

There is a NFA over Σ that recognizes L
if and only if

$\exists N$ (N is a NFA and $L(N) = A$)

There is a regular expression over Σ that describes L $\exists R$ (R is a regular expression and $L(R) = A$)

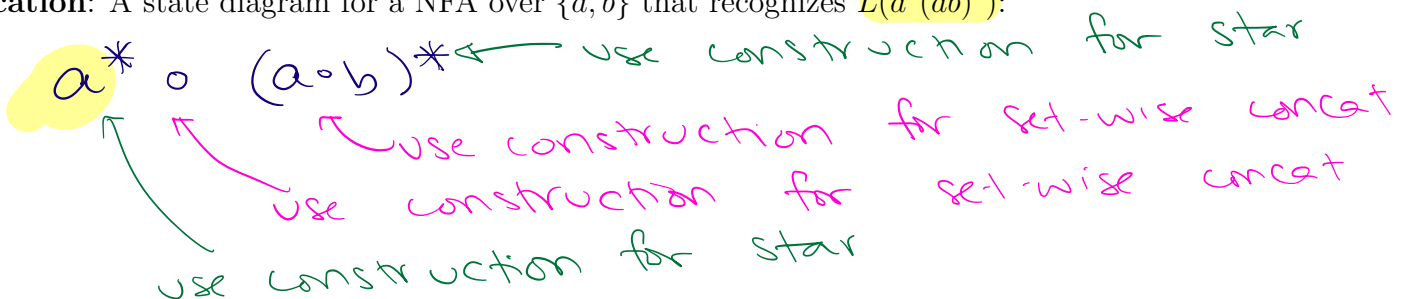
A language is called **regular** when any (hence all) of the above three conditions are met.

We already proved that DFAs and NFAs are equally expressive. It remains to prove that regular expressions are too.

Part 1: Suppose A is a language over an alphabet Σ . If there is a regular expression R such that $L(R) = A$, then there is a NFA, let's call it N , such that $L(N) = A$.

Structural induction: Regular expression is built from basis regular expressions using inductive steps (union, concatenation, Kleene star symbols). Use constructions to mirror these in NFAs.

Application: A state diagram for a NFA over $\{a, b\}$ that recognizes $L(a^*(ab)^*)$:



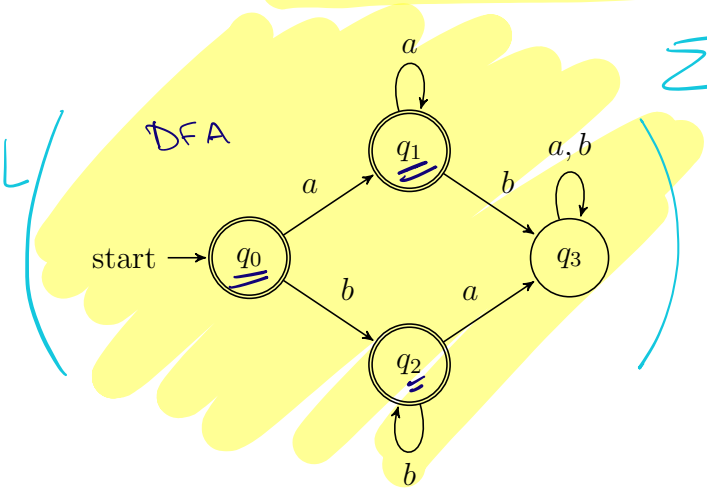
Part 2: Suppose A is a language over an alphabet Σ . If there is a DFA M such that $L(M) = A$, then there is a regular expression, let's call it R , such that $L(R) = A$.

Proof idea: Trace all possible paths from start state to accept state. Express labels of these paths as regular expressions, and union them all.

1. Add new start state with ϵ arrow to old start state.
2. Add new accept state with ϵ arrow from old accept states. Make old accept states non-accept.
3. Remove one (of the old) states at a time: modify regular expressions on arrows that went through removed state to restore language recognized by machine.

"GNFA" this gives an intermediate representation

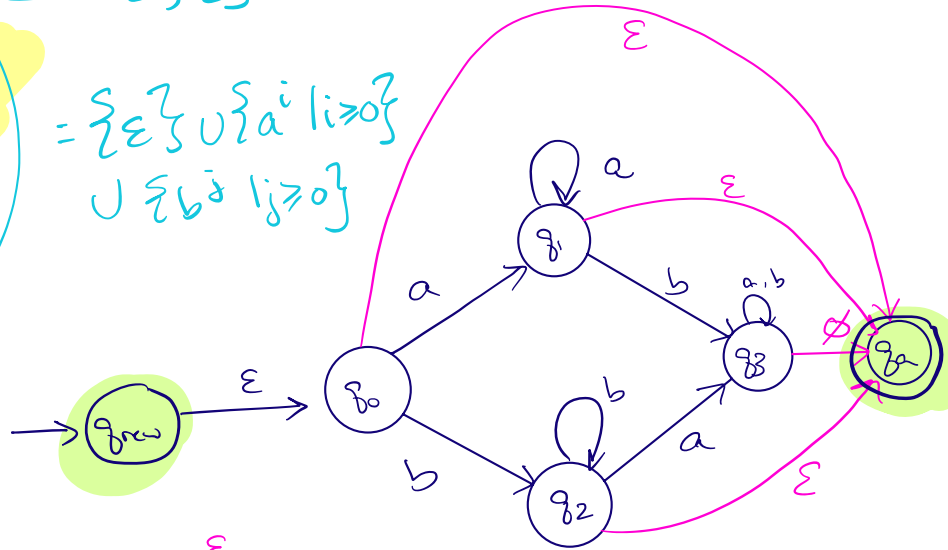
Application: Find a regular expression describing the language recognized by the DFA with state diagram



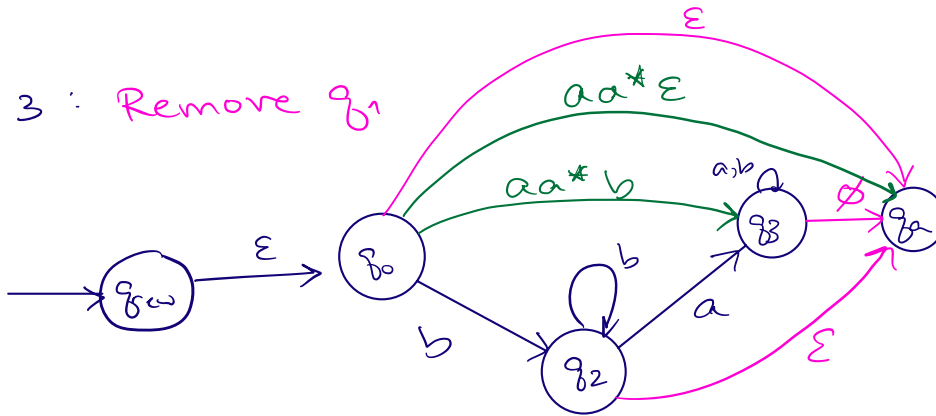
$$\Sigma = \{a, b\}$$

$$= \{\epsilon\} \cup \{a^i | i \geq 0\} \cup \{b^j | j \geq 0\}$$

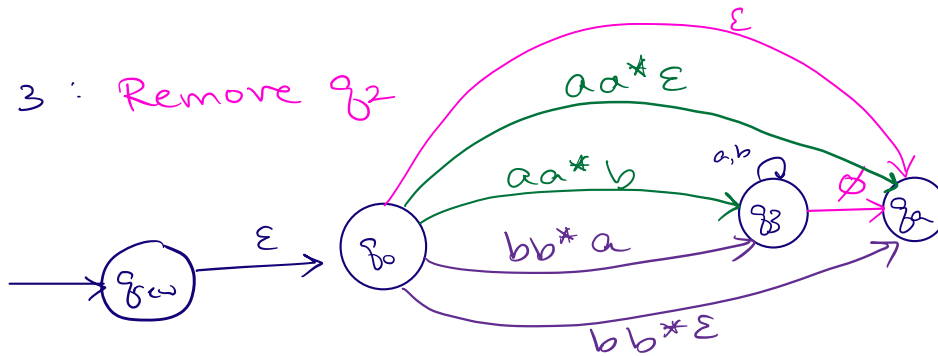
Steps 1+2 : GNFA



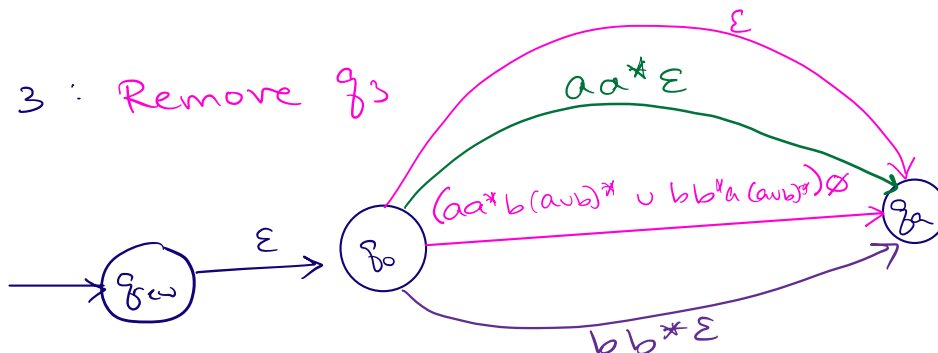
Step 3 : Remove q_1



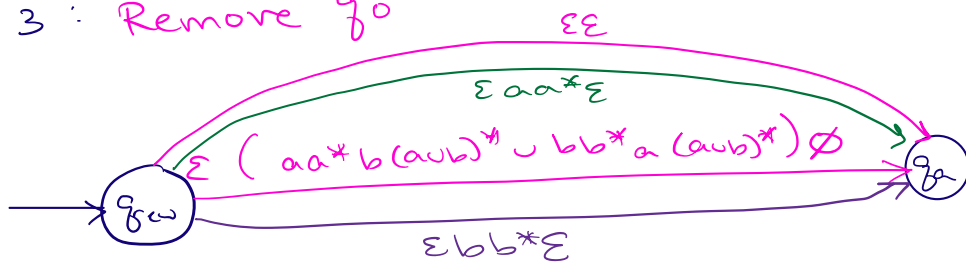
Step 3 : Remove q_2



Step 3 : Remove q_3



Step 3: Remove q_0



Possible paths from q_{new} to q_a are described by

$$\underline{\varepsilon\varepsilon} \cup \underline{\varepsilon aa^* \varepsilon} \cup \underline{\varepsilon (aa^* b (aub)^* \cup bb^* a (aub)^*) \emptyset} \cup \underline{\varepsilon bb^* \varepsilon}$$

describes same language as ε describes same language as aa^* describes same language as \emptyset describes same language as bb^*

or equivalently

$$\varepsilon \cup aa^* \cup bb^*$$

or equivalently

$$a^* \cup b^*$$