Week 3 at a glance

Textbook reading: Chapter 1

Before Monday, Theorem 1.47 + 1.48, Theorem 1.39 "Proof Idea", Example 1.41, Example 1.56.

Before Wednesday, read Introduction to Section 1.4 (page 77) which introduces nonregularity.

Before Friday, read Example 1.75, Example 1.77.

For Week 4 Monday: read Definition 2.13 (page 111-112) introducing Pushdown Automata.

We will be learning and practicing to:

- Clearly and unambiguously communicate computational ideas using appropriate formalism. Translate across levels of abstraction.
  - 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.**
- Know, select and apply appropriate computing knowledge and problem-solving techniques.
  - Apply classical techniques including pumping lemma, determinization, diagonalization, and reduction to analyze the complexity of languages and problems.
    - **Justify why the Pumping Lemma is true.**
    - **Use the pumping lemma to prove that a given language is not regular.**
- 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)

Homework 2 submitted via Gradescope (https://www.gradescope.com/), due Tuesday 10/15/2024

Review Quiz 2 on PrairieLearn (http://us.prairielearn.com), complete by Sunday 10/20/2024

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 describring patterns in strings. **Finite automata** give a model of computation for processing strings and 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.

Monday: 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.

Same state graph. Flip accept/reject.

run in parallel

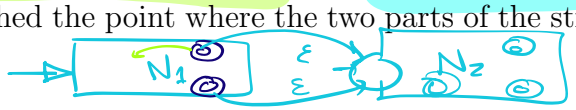
run in parallel

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.

For each L
there is a machine that is a DFA and recognizes L
iff
there is a machine that is a NFA and recognizes L

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

($Q \cup Q_2$; label; state; label of arrow; collection of next states)

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

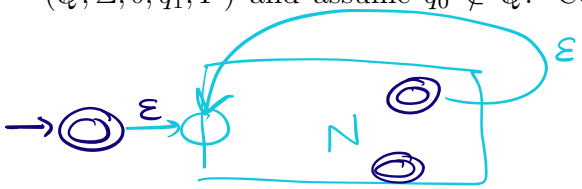
(simulating N_1 and not in an accept state or consume input; simulating N_2; add option to spontaneously move to N_2's start state.)

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.

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



- $Q' = Q \cup \{q_0\}$
- $F' = F \cup \{q_0\}$
- $\delta' : Q' \times \Sigma_\varepsilon \to \mathcal{P}(Q')$ is defined by, for $q \in Q'$ and $a \in \Sigma_\varepsilon$:

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

For any set L, $\varepsilon \in L^*$ because can have zero slots

Question: to build N'' with $L(N'') = L(N)^+$ could use $L(N)^+ = L(N) \circ L(N)^*$

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.

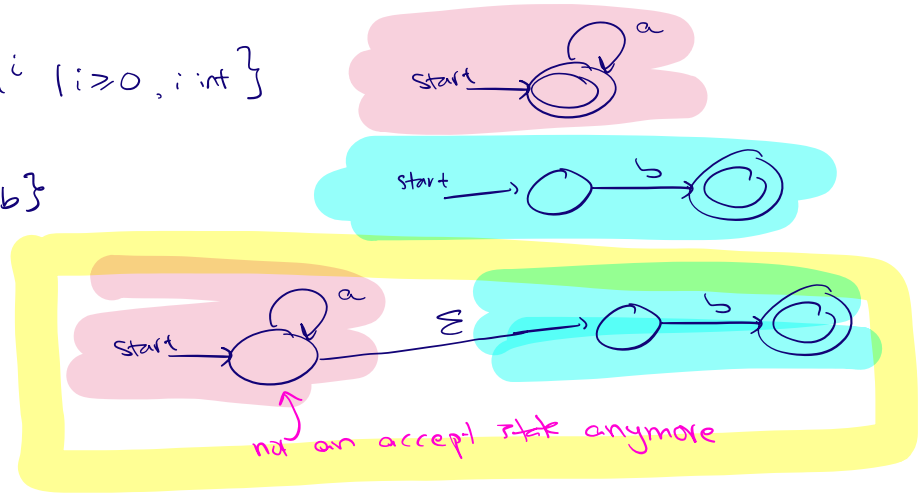
Application: A state diagram for a NFA over $\Sigma = \{a, b\}$ that recognizes $L((a^*b)^*)$:

Bonus (recommending at home)

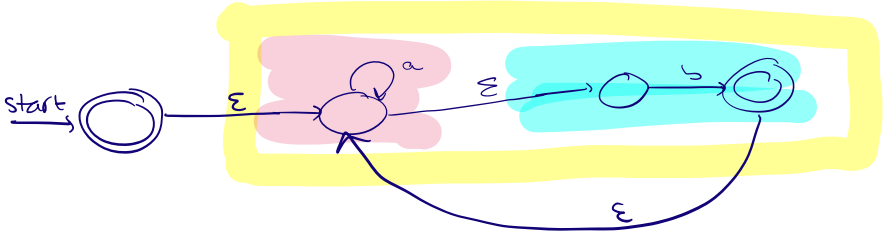
$L(a^*) = \{a^i \mid i \geq 0, i \text{ int}\}$

$L(b) = \{b\}$

$L(a^* \circ b)$



$L((a^* \circ b)^*)$



Note: If have DFA, we can immediately build NFA that recognizes its language.

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.

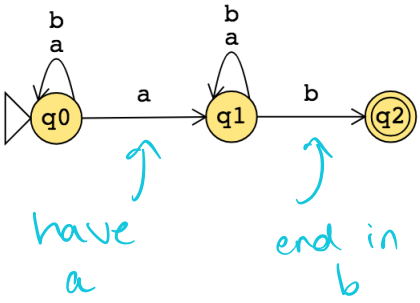
Formal construction: Let $N = (Q, \Sigma, \delta, q_0, F)$. Define

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

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

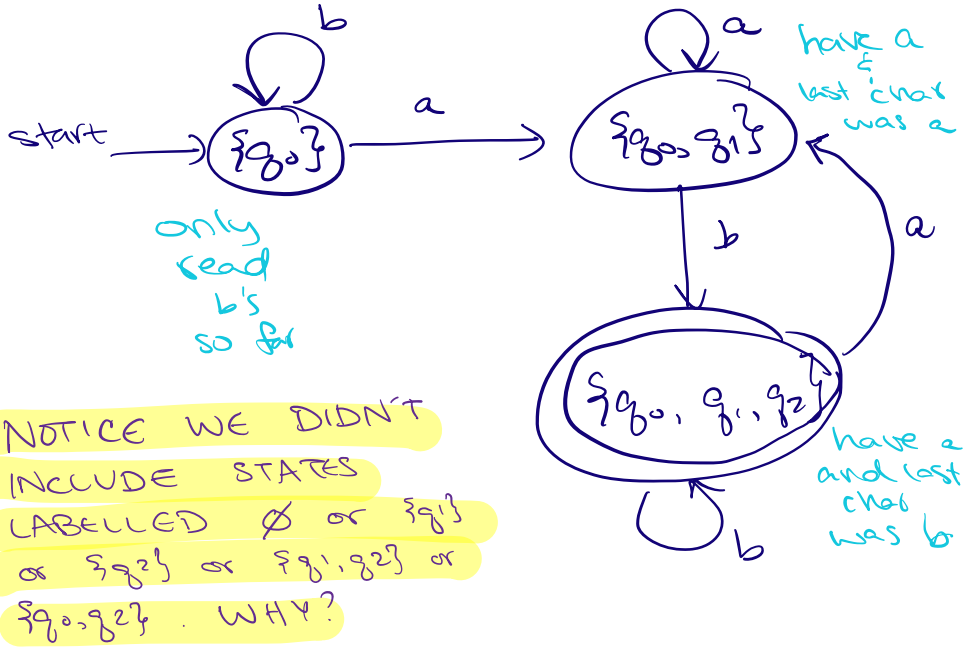
$\delta'((X, x)) = \{q \in Q \mid q \in \delta((r, x))$ for some $r \in X$ or is accessible from such an r by spontaneous moves in $N\}$

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



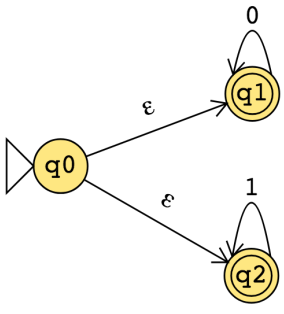
have a

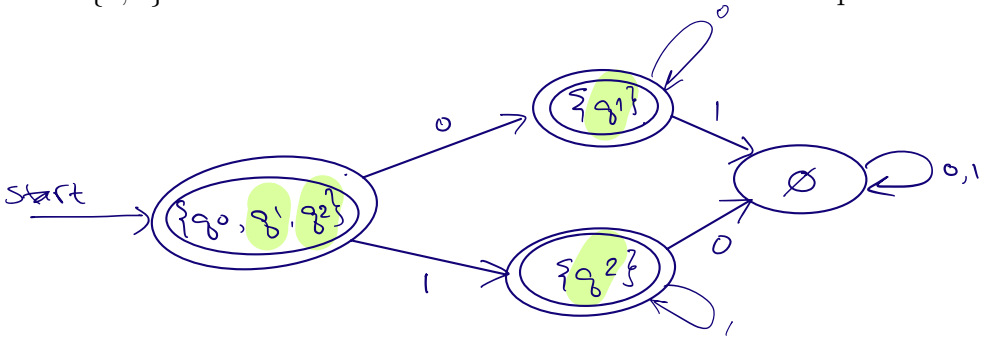
end in b



NOTICE WE DIDN'T INCLUDE STATES LABELLED $\emptyset$ or $\{q_1\}$ or $\{q_2\}$ or $\{q_1, q_2\}$ or $\{q_0, q_2\}$. WHY?

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

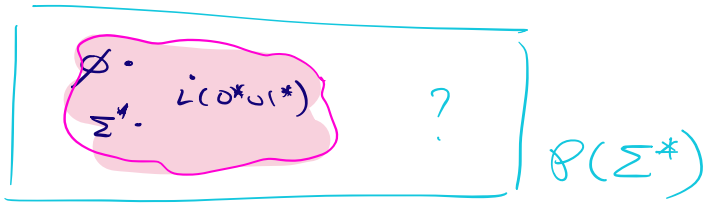




NOT INCLUDING STATES LABELLED $\{q_0\}$, $\{q_0, q_1\}$, $\{q_0, q_2\}$, $\{q_1, q_2\}$. WHY?

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	$\exists M$ (M is a DFA and $L(M) = A$)
if and only if	
There is a NFA over Σ that recognizes L	$\exists N$ (N is a NFA and $L(N) = A$)
if and only if	
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.

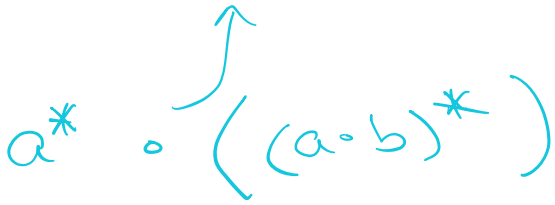
Part 1: Suppose A is a language over an alphabet Σ. If there is a regular expression R such that $L(R) = A$, then there is a NFA, let's call it N, such that $L(N) = A$.

~ R1 ~ R2 ~

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 $L(a^*(ab)^*)$:

see page 4.



Once we have NFA, can build DFA, then run it in code.

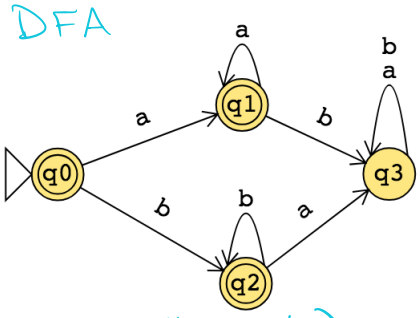
Part 2: Suppose A is a language over an alphabet Σ. If there is a DFA M such that $L(M) = A$, then there is a regular expression, let's call it R, such that $L(R) = A$.

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.

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

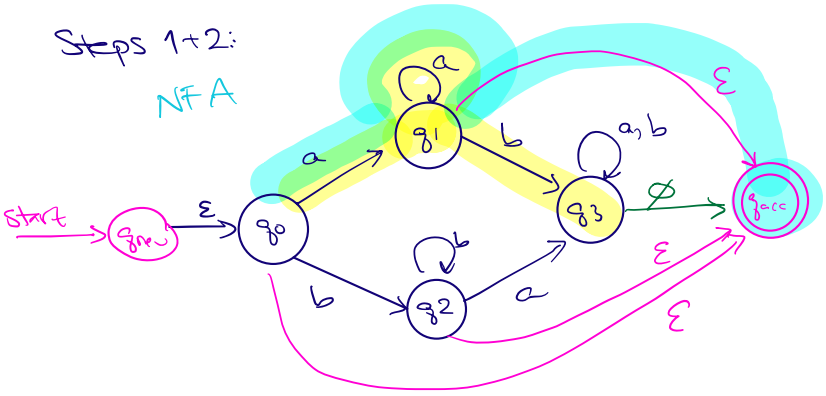
DFA



$L(a^* \cup b^*)$

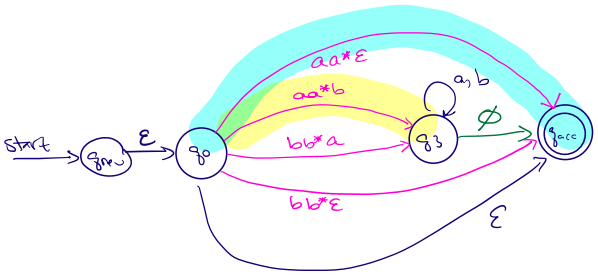
Steps 1+2:

NFA

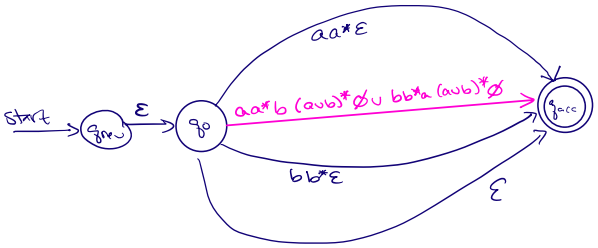


Step 3: Create intermediate representations "GNFA"

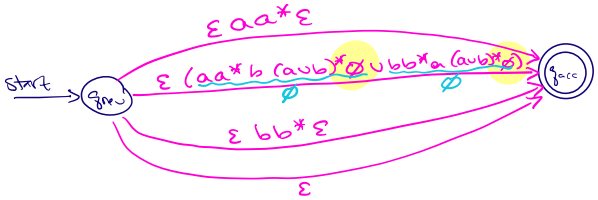
Removing q_1, q_2



Removing q_3



Removing q_0



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

$\varepsilon\, aa^*\varepsilon \cup \varepsilon(aa^*b(a\cup b)^*\emptyset \cup bb^*a(a\cup b)^*\emptyset) \cup \varepsilon bb^*\varepsilon \cup \varepsilon$

top; second from bottom; bottom

or equivalently, $aa^* \cup bb^* \cup \varepsilon$

$a^* \cup b^*$

2 regular expressions describing same language.

$L(R_1 \circ R_2) = \{xy \mid x \in L(R_1) \text{ and } y \in L(R_2)\}$

$L(R_1 \circ \emptyset) = \{xy \mid x \in L(R_1) \text{ and FALSE}\} = \emptyset$

Wednesday: Nonregular languages

Definition and Theorem: For an alphabet Σ, a language L over Σ is called **regular** exactly when L is recognized by some DFA, which happens exactly when L is recognized by some NFA, and happens exactly when L is described by some regular expression

We saw that: The class of regular languages is closed under complementation, union, intersection, set-wise concatenation, and Kleene star.

Prove or Disprove: There is some alphabet Σ for which there is some language recognized by an NFA but not by any DFA.

ex

Prove or Disprove: There is some alphabet Σ for which there is some finite language not described by any regular expression over Σ.

ex

Prove or Disprove: If a language is recognized by an NFA then the complement of this language is not recognized by any DFA.

ex

Fix alphabet Σ. Is every language L over Σ regular?

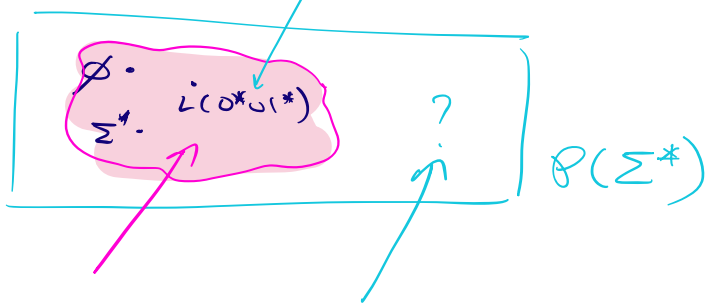
Compare size of collection of regular languages with the size of collection of all languages.

Set	Cardinality
$\{0,1\}$	
$\{0,1\}^*$	countably infinite
$\mathcal{P}(\{0,1\})$	
The set of all languages over $\{0,1\}$	uncountable.
The set of all regular expressions over $\{0,1\}$	countably infinite
The set of all regular languages over $\{0,1\}$	countably infinite

$\varepsilon, 0, 1, 00, 01, 10, 11, 000, \ldots$

$\mathcal{P}(\{0,1\}^*)$

can list all and only



Strategy: Find an **invariant** property that is true of all regular languages. When analyzing a given language, if the invariant is not true about it, then the language is not regular.

Pumping Lemma (Sipser Theorem 1.70): If A is a regular language, then there is a number p (a *pumping length*) where, if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$ such that

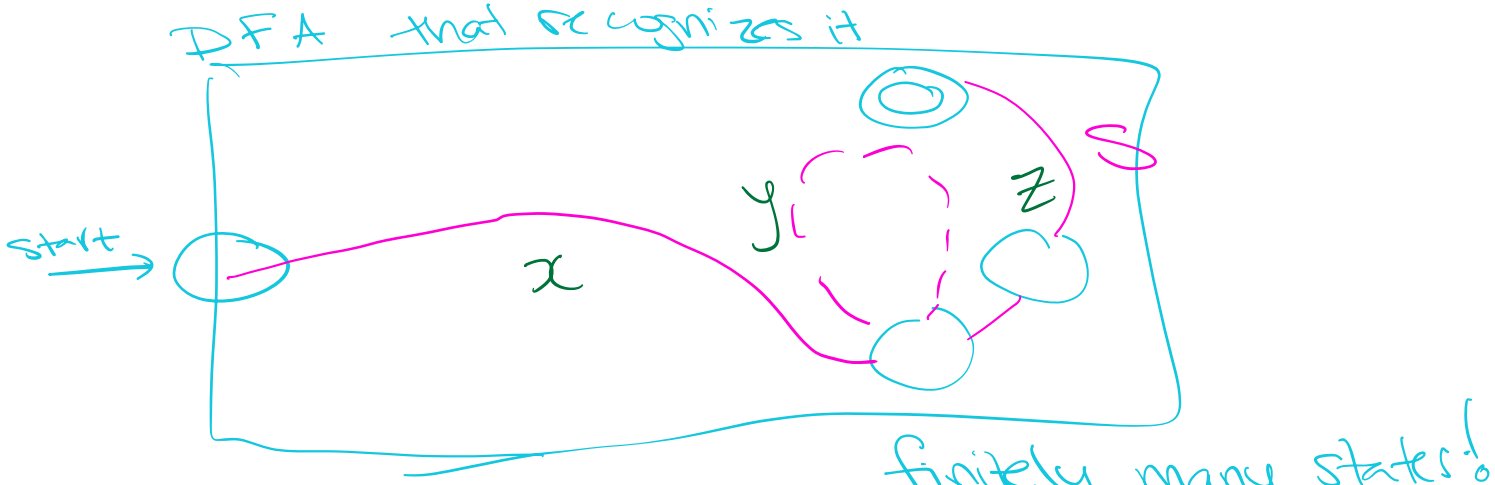
- $|y| > 0$
- for each $i \geq 0$, $xy^iz \in A$
- $|xy| \leq p$.

long

xz
xyz
xyyz
xyyyz
} different strings

Proof illustration

A regular language
- regex that describes A
- DFA that recognizes A
- NFA that recognizes A.



True or False: A pumping length for $A = \{0,1\}^*$ is $p = 5$.

Consider any string $s \in \{0,1\}^*$ with $|s| \geq 5$. WTS $\exists x,y,z$ with (1) $s=xyz$ (2) $|xy| \leq 5$ (3) $|y| > 0$ so that $xy^iz \in \{0,1\}^*$

Consider $x = \varepsilon$ y: first 5 chars of s z: rest of s

WTS for each nonnegative int i $xy^iz \in \{0,1\}^*$

By definition $xy^iz = \varepsilon (s_1s_2s_3s_4s_5)^i (s_6 \cdots s_{|s|})$

each char is 0 or 1

so xy^iz is a string over $\{0,1\}$, ie. in $\{0,1\}^*$ ☐ ($|xy| \leq p$)

Pumping length means all strings in language whose length is more than this can be split into x,y,z y nonempty and xy^iz in language for all integers $i \geq 0$.

Friday: Pumping Lemma

Recap so far: In DFA, the only memory available is in the states. Automata can only "remember" finitely far in the past and finitely much information, because they can have only finitely many states. If a computation path of a DFA visits the same state more than once, the machine can't tell the difference between the first time and future times it visits this state. Thus, if a DFA accepts one long string, then it must accept (infinitely) many similar strings.

Definition A positive integer p is a **pumping length** of a language L over Σ means that, for each string $s \in \Sigma^*$, if $|s| \geq p$ and $s \in L$, then there are strings x, y, z such that

$$s = xyz$$

and

$$|y| > 0, \qquad \text{for each } i \geq 0,\ xy^iz \in L, \qquad \text{and} \qquad |xy| \leq p.$$

Negation: A positive integer p is **not a pumping length** of a language L over Σ iff

counterexample consider all cuts counterexample

$$\exists s \big(|s| \geq p \wedge s \in L \wedge \forall x \forall y \forall z \big(\underbrace{(s = xyz \wedge |y| > 0 \wedge |xy| \leq p)}_{\text{that are valid}} \rightarrow \exists i (i \geq 0 \wedge xy^iz \notin L) \big) \big)$$

Informally:

Restating **Pumping Lemma**: If L is a regular language, then it has a pumping length.

Contrapositive: If L has no pumping length, then it is nonregular.

The Pumping Lemma *cannot* be used to prove that a language *is* regular.

The Pumping Lemma **can** be used to prove that a language *is not* regular.

Extra practice: Exercise 1.49 in the book.

Proof strategy: To prove that a language L is **not** regular, we want to show it has no pumping length.

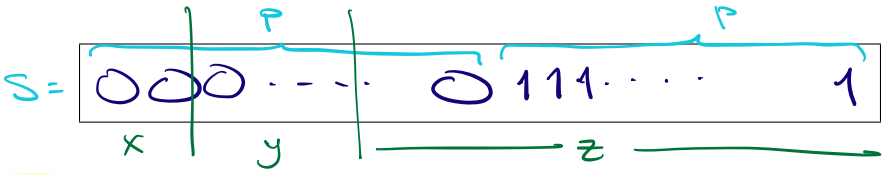
- Consider an arbitrary positive integer p
- Prove that p is not a pumping length for L. Therefore each positive int is not a pumping length for L.
- Conclude that L does not have *any* pumping length, and therefore it is not regular.

Example: $\Sigma = \{0, 1\}$, $L = \{0^n 1^n \mid n \geq 0\}$.

Fix p an arbitrary positive integer. List strings that are in L and have length greater than or equal to p:

Pick $s =$.

Suppose $s = xyz$ with $|xy| \leq p$ and $|y| > 0$.



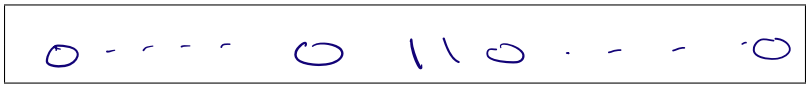
Then when $i =$, $xy^i z =$

Example: $\Sigma = \{0,1\}$, $L = \{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}$. Remember that the reverse of a string w is denoted $w^{\mathcal{R}}$ and means to write w in the opposite order, if $w = w_1 \cdots w_n$ then $w^{\mathcal{R}} = w_n \cdots w_1$. Note: $\varepsilon^{\mathcal{R}} = \varepsilon$.

Fix p an arbitrary positive integer. List strings that are in L and have length greater than or equal to p:

Pick $s =$ $0^p 1 1 0^p$

Suppose $s = xyz$ with $|xy| \leq p$ and $|y| > 0$. Then there are ints $m \geq 0$ and $k > 0$ with $x = 0^m$, $y = 0^k$, $z = 0^{p-m-k} 1 1 0^p$



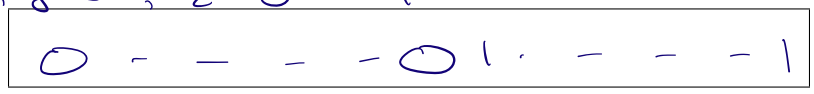
Then when $i =$ 3, $xy^iz = xyyyz = 0^m 0^k 0^k 0^k 0^{p-m-k} 1 1 0^p = 0^{p+2k} 1 1 0^p$ which is even length string whose first half is 0^{p+k+1} and second half is $0^{k-1} 1 1 0^p$. These halves are not reverses of one another because first has all 0s and second half has 1s.

Example: $\Sigma = \{0,1\}$, $L = \{0^j 1^k \mid j \geq k \geq 0\}$.

Fix p an arbitrary positive integer. List strings that are in L and have length greater than or equal to p:

Pick $s =$ $0^p 1^p$

Suppose $s = xyz$ with $|xy| \leq p$ and $|y| > 0$. Then there are ints $m \geq 0$ and $t > 0$ with $x = 0^m$, $y = 0^t$, $z = 0^{p-m-t} 1^p$



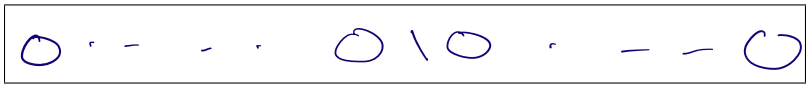
Then when $i =$ 0, $xy^iz = xz = 0^m 0^{p-m-t} 1^p = 0^{p-t} 1^p$ which is not in L because $p - t < p$, since $t > 0$.

Example: $\Sigma = \{0,1\}$, $L = \{0^n 1^m 0^n \mid m, n \geq 0\}$.

Fix p an arbitrary positive integer. List strings that are in L and have length greater than or equal to p:

Pick $s =$ $0^p 1 0^p$

Suppose $s = xyz$ with $|xy| \leq p$ and $|y| > 0$. Then there are ints $r \geq 0$ and $k > 0$ with $x = 0^r$ $y = 0^k$ $z = 0^{p-r-k} 1 0^p$



Then when $i =$ 2, $xy^iz = xyyz = 0^r 0^k 0^k 0^{p-r-k} 1 0^p = 0^{p+k} 1 0^p$ which is not in L because the number of 0s on either side of the 1 is not balanced (since $k > 0$).

Extra practice:

Language	$s \in L$	$s \notin L$	Is the language regular or nonregular?
$\{a^n b^n \mid 0 \leq n \leq 5\}$			
$\{b^n a^n \mid n \geq 2\}$			
$\{a^m b^n \mid 0 \leq m \leq n\}$			
$\{a^m b^n \mid m \geq n+3, n \geq 0\}$			
$\{b^m a^n \mid m \geq 1, n \geq 3\}$			
$\{w \in \{a,b\}^* \mid w = w^{\mathcal{R}}\}$			
$\{ww^{\mathcal{R}} \mid w \in \{a,b\}^*\}$			

