

Examples of regular languages
over $\{0,1\}$: $L (1 \cup 0^*)$ with strings $1, \epsilon, 0, 00, 000$

Week4

\emptyset (language over any alphabet)
over $\{0,1\}$: $\{ w \mid w \text{ is a string over } \{0,1\} \}$.

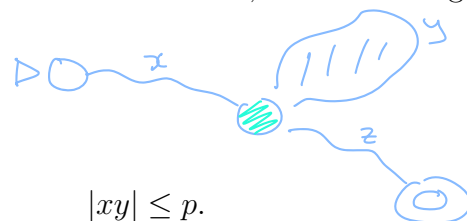
Monday: Pumping Lemma Application

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

long in the language

$$s = xyz$$



and

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

$xz, xy^2z, xy^3z, xy^4z, \dots$

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

$\exists s (|s| \geq p \wedge s \in L \wedge \forall x \forall y \forall z ((s = xyz \wedge |y| > 0 \wedge |xy| \leq p) \rightarrow \exists i (i \geq 0 \wedge xy^iz \notin L)))$
 There is a long string in L which is not pumpable i.e. no matter we split it into three pieces that might work, there's some choice of i for which $xy^iz \notin L$
 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.

Roadmap.

Proof strategy: To prove that a language L is **not** regular,

(WTS : L does not have pumping length)

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

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

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

Pick $s = 0^p 1^p$. This string is in L , choosing $n=p$.
And $|s| = 2p \geq p$.

Suppose $s = xyz$ with $|xy| \leq p$ and $|y| > 0$. By definition of s , $x = 0^m$, $y = 0^r$ for some ints $m \geq 0$, $r > 0$, $m+r \leq p$.

$s = 00 \dots 011 \dots 1$
 xy

Then when $i = 0$, $xy^0z = xy^0z = 0^{p-r} 1^p$ which is not an element of L because $r > 0$ so $p-r \neq p$.

Illustration of family of strings:

$$xz = 0^{p-r} 1^p$$

$$xz \notin L$$

$$s = xyz = 0^m 0^r z$$

$$z = 0^{p-m-r} 1^p$$

$$s \in L$$

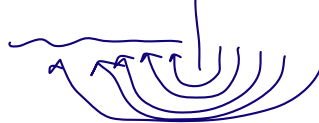
$$xyyz = 0^m 0^{2r} z = 0^m 0^{2r} 0^{p-m-r} 1^p = 0^{p+r} 1^p \quad xyyz \notin L$$

$$xyyyz = 0^{p+2r} 1^p$$

$$xy^3z \notin L$$

$$xyyyyz = 0^{p+3r} 1^p$$

$$xy^4z \notin L$$



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

Fix p an arbitrary positive integer. List strings that are in L and have length greater than or equal to p :
Goal: prove p is not a pumping length for L

* Pick $s = 0^p 11 0^p$ Notice $|s| = p+1+1+p = 2p+2 > p$
and $w = 0^p 1$ witnesses that $s \in L$.
Suppose $s = xyz$ with $|xy| \leq p$ and $|y| > 0$. So $x = 0^m$ $y = 0^r$ $z = 0^{p-m-r} 11 0^p$ ($r > 0$)

$00 \dots 0110 \dots 00$

Then when $i = 2$, $xy^iz = 0^m 0^r 0^r 0^{p-m-r} 11 0^p = 0^{p+r} 11 0^p$
 $xy^iz \notin L$ so s is not pumpable.

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

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

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

Suppose $s = xyz$ with $|xy| \leq p$ and $|y| > 0$. By choice of s , $x = 0^m$ $y = 0^r$ $z = 0^{p-m-r} 1^p$
with $m+r \leq p$, $r > 0$.

$00 \dots 011 \dots 1$

Then when $i = 0$, $xy^iz = 0^{p-r} 1^p \notin L$ because $p-r < p$.
NOTE: Choosing $i = 2, 3, 4, \dots$ would not work for this L !

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

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 $i =$, $xy^iz =$

Extra practice:

Language	$s \in L$	$s \notin L$	Is the language regular or nonregular?
$\{a^n b^n \mid 0 \leq n \leq 5\}$	ε	ba	regular
$\{b^n a^n \mid n \geq 2\}$	$bbaa$	ba	nonregular
$\{a^m b^n \mid 0 \leq m \leq n\}$	b	aab	nonregular
$\{a^m b^n \mid m \geq n + 3, n \geq 0\}$	aaa	ab	nonregular
$\{b^m a^n \mid m \geq 1, n \geq 3\}$	$baaa$	ba	regular
$\{w \in \{a, b\}^* \mid w = w^R\}$	aba	ab	nonregular
$\{ww^R \mid w \in \{a, b\}^*\}$	$abba$	aba	nonregular

Wednesday: Pushdown Automata

$\{0^n 1^n \mid n \geq 0\}$ nonregular

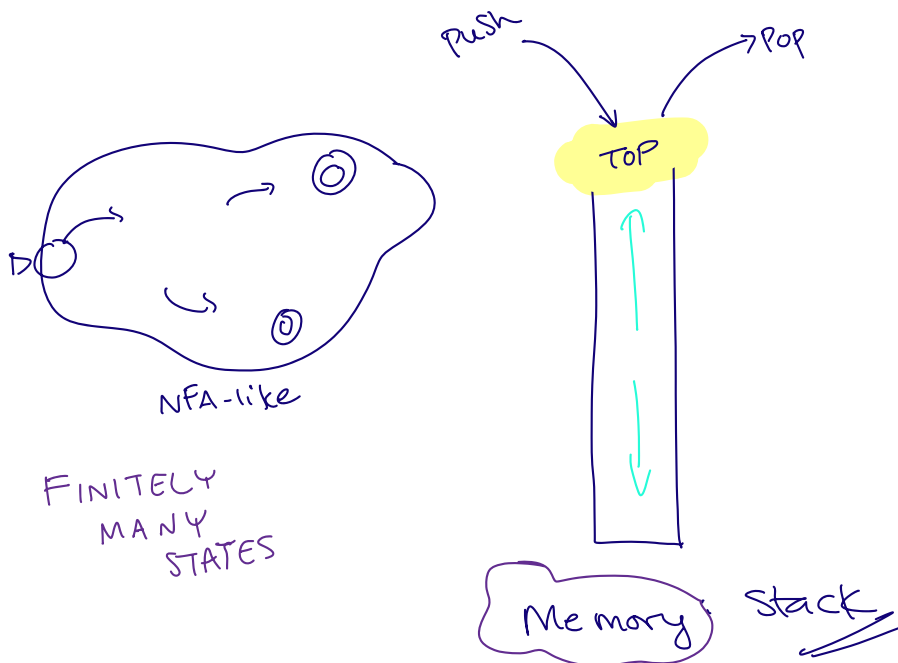
Regular sets are not the end of the story

- Many nice / simple / important sets are not regular
- Limitation of the finite-state automaton model: Can't "count", Can only remember finitely far into the past, Can't backtrack, Must make decisions in "real-time"
- We know actual computers are more powerful than this model...

The **next** model of computation. Idea: allow some memory of unbounded size. How?

- Sec 2.1
- To generalize regular expressions: **context-free grammars**

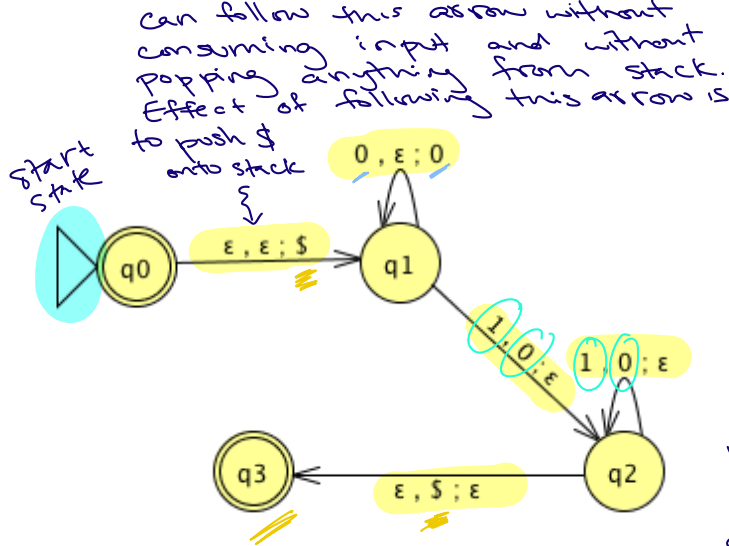
- Sec 2.2
- To generalize NFA: **Pushdown automata**, which is like an NFA with access to a stack: Number of states is fixed, number of entries in stack is unbounded. At each step (1) Transition to new state based on current state, letter read, and top letter of stack, then (2) (Possibly) push or pop a letter to (or from) top of stack. Accept a string iff there is some sequence of states and some sequence of stack contents which helps the PDA process the entire input string and ends in an accepting state.



Is there a PDA that recognizes the nonregular language $\{0^n 1^n \mid n \geq 0\}$?

Store 0s as we read them,
match them to 1s.

Need to detect when stack is
empty at end of computation



accept state

nondeterminism.

input label, trigger vis-a-vis stack, effect vis-a-vis stack

eg. a, b; c label means We can follow arrow when reading a from input and current top letter of stack is b effect of following this arrow is to pop b off and push c.

The PDA with state diagram above can be informally described as:

Read symbols from the input. As each 0 is read, push it onto the stack. As soon as 1s are seen, pop a 0 off the stack for each 1 read. If the stack becomes empty and we are at the end of the input string, accept the input. If the stack becomes empty and there are 1s left to read, or if 1s are finished while the stack still contains 0s, or if any 0s appear in the string following 1s, reject the input.

Trace the computation of this PDA on the input string 01.

accepted

Trace the computation of this PDA on the input string 011.

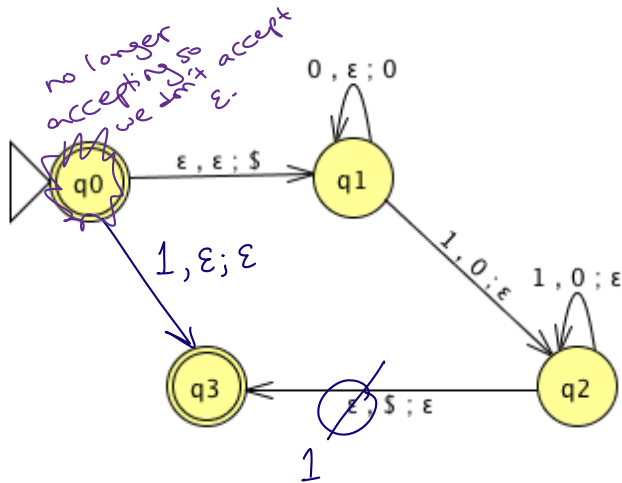
rejected.

	011	input string needed to read
q ₀	[TOP]	011
q ₁	[TOP \$]	011
q ₁	[TOP 0]	11
q ₂	[TOP \$]	1
can't follow loop at q ₂ .		
q ₃	[TOP]	1
can't follow any arrows b/c no outgoing arrows from q ₃ !		

A PDA recognizing the set $\{0^n 1^{n+1} \mid n \geq 0\}$ can be informally described as:

English description
 Read symbols from the input. As each 0 is read, push it onto the stack. As soon as 1s are seen, pop a 0 off the stack for each 1 read. If the stack becomes empty and there is exactly one 1 left to read, read that 1 and accept the input. If the stack becomes empty and there are either zero or more than one 1s left to read, or if the 1s are finished while the stack still contains 0s, or if any 0s appear in the input following 1s, reject the input.

Modify the state diagram below to get a PDA that implements this description:



state diagram.

$$\delta \left(\underset{\substack{\text{current} \\ \text{state}}}{q}, \underset{\substack{\text{input} \\ \text{label}}}{x}, \underset{\substack{\text{top of} \\ \text{stack} \\ \text{label}}}{y} \right) = \left\{ \underset{\substack{\text{next} \\ \text{state}}}{(q', z)}, \left(\quad, \dots \right) \right\}$$

action of stack label

Γ is the Greek capital letter Gamma.

Definition A **pushdown automaton** (PDA) is specified by a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where Q is the finite set of states, Σ is the input alphabet, Γ is the stack alphabet,

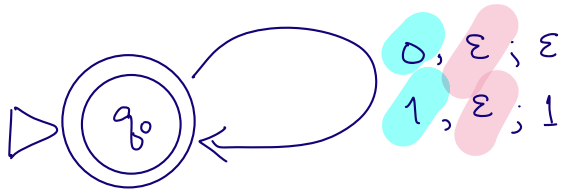
$$\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$$

is the transition function, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of accept states.

Friday: Pushdown Automata Constructions

Draw the state diagram and give the formal definition of a PDA with $\Sigma = \Gamma$.

input alphabet
stack alphabet



$$\Sigma = \{0, 1\} = \Gamma$$

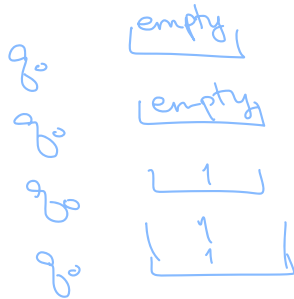
Formal definition

$$(\{q_0\}, \{0, 1\}, \{0, 1\}, \delta, q_0, \{q_0\})$$

where $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$

$$\delta(q, a, b) = \begin{cases} \{(q, \epsilon)\} & \begin{matrix} a=0 \\ q=q_0 \\ b=\epsilon \end{matrix} \\ \{(q, 1)\} & \begin{matrix} a=1 \\ q=q_0 \\ b=\epsilon \end{matrix} \\ \emptyset & \text{otherwise} \end{cases}$$

accepts $\epsilon, 011$

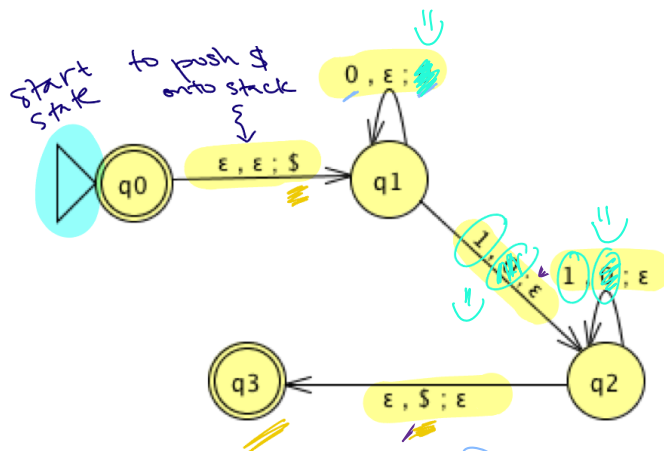


left to read
011

Done!

The language recognized by this PDA to be $\{0, 1\}^*$

Draw the state diagram and give the formal definition of a PDA with $\Sigma \cap \Gamma = \emptyset$.



$$\Sigma = \{0, 1\}$$

$$\Gamma = \{\epsilon, \$\}$$

$$\Sigma \cap \Gamma = \{0\}$$

Formal definition: $(\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \{\epsilon, \$\}, \delta, q_0, \{q_3\})$

with δ given by $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$ with

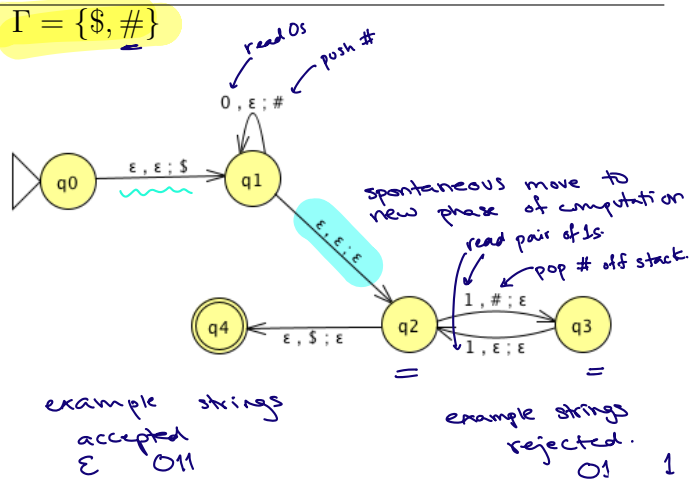
$$\delta(q, a, b) = \begin{cases} \{(q_1, \$)\} & \text{when } q=q_0, a=\epsilon, b=\epsilon \\ \{(q_1, \epsilon)\} & \text{when } q=q_1, a=0, b=\epsilon \\ \{(q_2, \epsilon)\} & \text{when } q=q_1, a=1, b=\epsilon \\ \{(q_2, \epsilon)\} & \text{when } q=q_2, a=1, b=\epsilon \\ \{(q_3, \epsilon)\} & \text{when } q=q_2, a=\epsilon, b=\$ \\ \emptyset & \text{otherwise} \end{cases}$$

For the PDA state diagrams below, $\Sigma = \{0, 1\}$.

Mathematical description of language

State diagram of PDA recognizing language

$$\{0^n 1^{2n} \mid n \geq 0\}$$

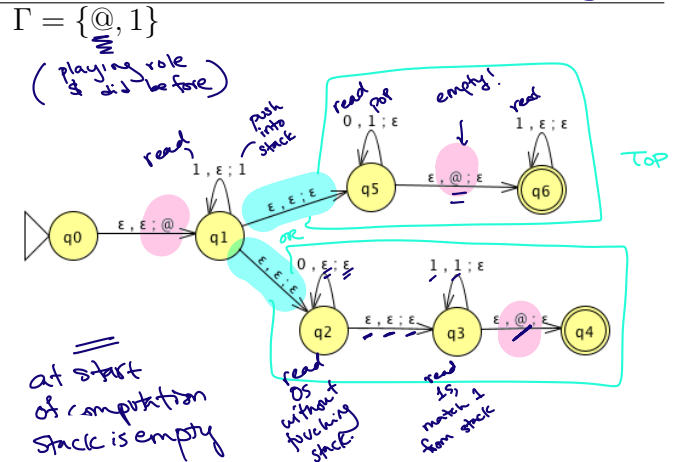


$$\{1^n 0^m 1^n \mid n \geq 0, m \geq 0\}$$

string

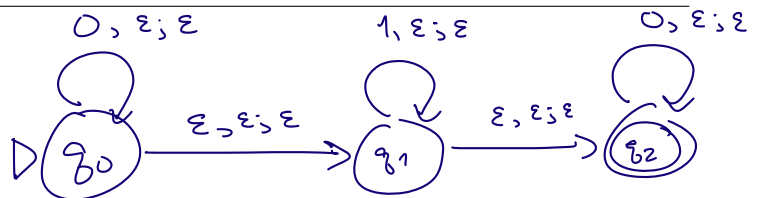
$$\{1^n 0^m 1^n \mid n \geq 0, m \geq 0\}$$

string



$$\{0^i 1^j 0^k \mid i, j, k \geq 0\}$$

(a regular language!)



Note: alternate notation is to replace ; with \rightarrow

Big picture: PDAs were motivated by wanting to add some memory of unbounded size to NFA. How do we accomplish a similar enhancement of regular expressions to get a syntactic model that is more expressive?

DFA, NFA, PDA: Machines process one input string at a time; the computation of a machine on its input string reads the input from left to right.

Regular expressions: Syntactic descriptions of all strings that match a particular pattern; the language described by a regular expression is built up recursively according to the expression's syntax

Context-free grammars: Rules to produce one string at a time, adding characters from the middle, beginning, or end of the final string as the derivation proceeds.

Week 4 at a glance

Textbook reading: Section 1.4 and section 2.2

For Monday: Example 1.75, Example 1.77

For Wednesday: Definition 2.13 (page 111-112)

For Friday: Example 2.18 (page 114)

For Week 5 Monday: Introduction to Section 2.1 (page 102)

Make sure you can:

- Classify the computational complexity of a set of strings by determining whether it is regular
 - Explain the limits of the class of regular languages
 - Identify some nonregular sets
- Use the pumping lemma to prove that a given language is not regular.
 - Justify why the Pumping Lemma is true
 - Apply the Pumping Lemma in proofs of nonregularity
- Use precise notation to formally define the state diagram of PDA and use clear English to describe computations of PDA informally.
 - Define push-down automata informally and formally
 - Trace the computation of a push-down automaton
 - Determine the language recognized by a given PDA
 - Design push-down automata to recognize specific languages
 - Determine whether a language is recognizable by a (D or N) FA and/or a PDA

TODO:

Review quizzes based on class material each day.

Homework assignment 2 due Thursday.

Test next week on Friday in Discussion section.