

# Regular expressions, languages, and the pumping lemma for regular languages

[illegible]

Textbook: 1.3 and 1.4

## Regular languages

**Def:** Regular languages. A language is said to be “regular” iff there exists a DFA recognizing it and only it.

**Corollary:** Since DFA are closed under the regular operations, so to are regular languages.

**Corollary:** Since DFA and NFA are equivalent, a language is regular iff there exists an NFA recognizing it.

**Corollary:** All finite sets are regular. Since the language consisting of exactly one string is trivially decidable by a DFA and regular languages are closed under union, any finite set of strings is regular.

# Regular expressions

- ▶ Invented by Kleene (of the Kleene star, Kleene plus, and Kleene recursion theorem)
- ▶ Formal notation for expressing DFAs
- ▶ Commonly used in non-theoretical computer science

# Formal definition of regular expressions

**Def:** Regular expressions. We say  $R$  is a regular expression on alphabet  $\Sigma$  if  $R$  is:

- 1) a character  $\in \Sigma$
- 2) the **empty string**  $\epsilon$
- 3) the **empty set**  $\emptyset$
- 4)  $(R_1 \cup R_2)$  where  $R_1, R_2$  are regular expressions
- 5)  $(R_1 \circ R_2)$  where  $R_1, R_2$  are regular expressions
- 6)  $(R_1)^*$  (zero or more repetitions of  $R_1$ ) where  $R_1$  is a regular expression

# RegEx shorthands

- ▶ For convenience,  $A \circ B$  is written  $AB$
- ▶  $A^+$  is shorthand for  $A \circ A^*$
- ▶ **Informally**,  $A \cup B$  is given by  $(A|B)$ . This is used in real life, but not here
- ▶  $\Sigma$  is usually inferred

**Thm:** Regular expressions and NFA are equivalent. This will be informally proved after this slide.

**Corollary:** A language is regular iff there exists a regular expression recognizing it.

## RegEx-NFA equivalence pt. 1

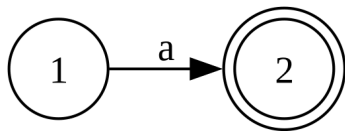


Figure 1:  $a$  for some  $a \in \Sigma$



Figure 2:  $\epsilon$



Figure 3:  $\emptyset$

## RegEx-NFA equivalence pt. 2

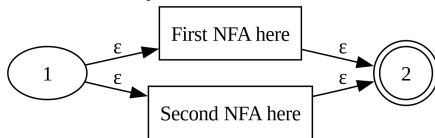


Figure 4:  $(R_1 \cup R_2)$

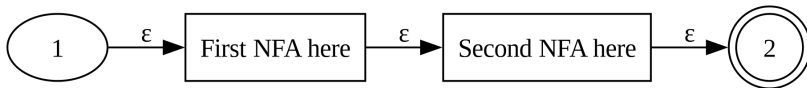


Figure 5:  $(R_1 \circ R_2)$

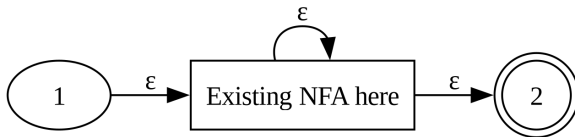


Figure 6:  $R_1^*$

## GNFAs pt. 1

- ▶ We can use the previous diagrams to convert REs to NFAs (and thus to DFAs)
- ▶ **How do we convert NFAs to REs?**

**Def:** Generalized nondeterministic finite automata (**GNFAs**). A GNFA is a NFA where edges are allowed to be any RE, rather than a character  $a \in \Sigma$  for alphabet  $\Sigma$ . For convenience, we make the following assumptions:

- 1) The start state has transition arrows going to every other state but no arrows coming in from any other state
- 2) There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state
- 3) The accept state is not the start state
- 4) Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself



## GNFAs pt. 2

- It is trivial to show that any NFA can be converted to an equivalent one in this form

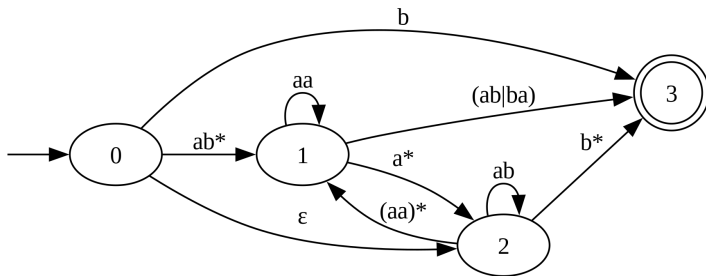


Figure 7: A generalized nondeterministic finite automaton

## GNFA reduction

- ▶ Want to reduce a GNFA to a single regular expression
- ▶ Will reduce the graph to an equivalent one of size 2, with one entry node and one acceptance node
- ▶ By definition, this will have one edge ( $q_0 \rightarrow q_f$ )
- ▶ The label of this edge will be a regular expression equivalent to the GNFA

### **Algorithm:** GNFA to RE.

- ▶ Take in some GNFA
- ▶ For as long as the GNFA is larger than 2 nodes:
  - ▶ Select a state which is not the entry or accepting state
  - ▶ Rip that node out
  - ▶ Repair the GNFA so that it recognizes the same language while maintaining the aforementioned GNFA assumptions
- ▶ Once we are at 2 nodes, output the label of the edge connecting them

# Non-regular languages

- ▶ **Not all languages are regular**
- ▶ Ex: The set of all properly-matched series of parenthesis
  - ▶ Like `((()((())))`, but not like `))(`
  - ▶ This takes a potentially infinite amount of information to decide, meaning it cannot be encoded into a DFA/NFA/RegEx
- ▶ We can prove that a language *is*, but how do we prove that a language *isn't*?

# The pumping lemma for regular languages

- ▶ For a regular language  $\mathcal{L}$ , if a string  $s \in \mathcal{L}$  is greater than some special length (the **pumping length**), they can be “pumped”
- ▶ A string can be “pumped” if it contains a section that can be repeated *any number* of times while keeping the string  $\in \mathcal{L}$

$$(\mathcal{L} \text{ is regular}) \rightarrow (\mathcal{L} \text{ can be pumped})$$

$$\neg(\mathcal{L} \text{ can be pumped}) \rightarrow \neg(\mathcal{L} \text{ is regular})$$

## Pumping lemma (formally)

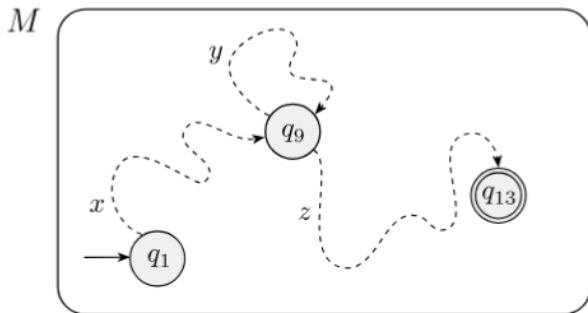
**Def:** If  $A$  is a regular language, then there exists a natural number  $p$  (the pumping length) where, if  $s \in A$  such that  $|s| \geq p$ , then  $s$  may be divided into three pieces,  $s = xyz$  such that the following conditions are met.

- 1) For each  $i \geq 0$ ,  $xy^iz \in A$
- 2)  $|y| > 0$
- 3)  $|xy| \leq p$

Either  $x$  or  $z$  can be  $\epsilon$ , but condition 2 implies  $y \neq \epsilon$ .

# Pumping lemma (informal proof)

- From the textbook (page 79):



**FIGURE 1.72**

Example showing how the strings  $x$ ,  $y$ , and  $z$  affect  $M$

## Example: Proving nonregularity

**Ex:** Consider the language  $B = \{0^n 1^n : n \geq 0\}$ . We will assume  $B$  is regular, then apply the pumping lemma to derive a contradiction (proving  $B$  is nonregular).

Assume  $B$  is regular. Let  $p$  be the pumping length guaranteed by the pumping lemma. We will examine  $s = 0^p 1^p$ . Since  $|s| > p$ , there must exist some  $s = xyz$  where:

- 1) For each  $i \geq 0$ ,  $xy^i z \in A$
- 2)  $|y| > 0$
- 3)  $|xy| \leq p$

Condition 3 imposes that  $xy = 0^j$  for some  $1 \leq j \leq p$ . Since  $x$  can be  $\epsilon$  but  $y$  cannot, we can say  $y = 0^k$ ,  $1 \leq k \leq p$ . However, the removal or addition of any nonzero number of 0s causes the resulting string  $\notin B$ : Thus, condition 1 is always violated when the others are applied. This is a contradiction, proving  $B$  is not regular. End of proof.

## Next up: Context-free grammars

*Although his last name is commonly pronounced KLEE-nee or KLEEN, Kleene himself pronounced it KLAY-nee. His son, Ken Kleene, wrote: "As far as I am aware this pronunciation is incorrect in all known languages. I believe that this novel pronunciation was invented by my father."  
(From Wikipedia)*