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.

Wednesday: Finite automaton constructions

Review: Formal definition of finite automaton: $M = (Q, \Sigma, \delta, q_0, F)$

 \bullet Finite set of states Q

• Start state q_0

• Alphabet Σ

• Accept (final) states F

• Transition function δ

In the state diagram of M, how many outgoing arrows are there from each state?

 $M = (\{q, r, s\}, \{a, b\}, \delta, q, \{q\})$ where δ is (rows labelled by states and columns labelled by symbols):

$$\begin{array}{c|cccc} \delta & a & b \\ \hline q & r & r \\ r & s & s \\ s & q & q \end{array}$$

The state diagram for M is

Give two examples of strings that are accepted by M and two examples of strings that are rejected by M:

$$L(M) =$$

A regular expression describing L(M) is



Using this terminology, M accepts a string w over Σ if and only if $\delta^*((q_0, w)) \in F$.



Friday: Nondeterministic automata

Nondeterministic finite automaton (Sipser Page 53) Given as $M = (Q, \Sigma, \delta, q_0, F)$

Finite set of states Q Can be labelled by any collection of distinct names. Default: $q0, q1, \ldots$

Alphabet Σ Each input to the automaton is a string over Σ .

Arrow labels Σ_{ε} $\Sigma_{\varepsilon} = \Sigma \cup \{\varepsilon\}.$

Arrows in the state diagram are labelled either by symbols from Σ or by ε

Transition function $\delta = \delta : Q \times \Sigma_{\varepsilon} \to \mathcal{P}(Q)$ gives the **set of possible next states** for a transition

from the current state upon reading a symbol or spontaneously moving.

Start state q_0 Element of Q. Each computation of the machine starts at the start state.

Accept (final) states $F ext{ } F \subseteq Q$.

M accepts the input string $w \in \Sigma^*$ if and only if **there is** a computation of M on w that processes the whole string and ends in an accept state.

The formal definition of the NFA over $\{0,1\}$ given by this state diagram is:



The language over $\{0,1\}$ recognized by this NFA is:

Change the transition function to get a different NFA which accepts the empty string (and potentially other strings too).

The state diagram of an NFA over $\{a,b\}$ is below. The formal definition of this NFA is:



The language recognized by this NFA is:

Week 2 at a glance

Textbook reading: Section 1.1, 1.2

For Wednesday: Pages 41-43 (Figures 1.18, 1.19, 1.20) (examples of automata and languages).

For Friday: Pages 48-50 (Figures 1.27, 1.29) (introduction to nondeterminism).

For Week 3 Monday: Pages 60-61 Theorem 1.47 and Theorem 1.48 (closure proofs).

Make sure you can:

- Use regular expressions and relate them to languages and automata
 - Write and debug regular expressions using correct syntax
- Use precise notation to formally define the state diagram of DFA, NFA and use clear English to describe computations of DFA, NFA informally.
 - Design an automaton that recognizes a given language
 - Specify a general construction for DFA based on parameters
 - Design general constructions for DFA
 - Motivate the use of nondeterminism
 - Trace the computation(s) of a nondeterministic finite automaton

TODO:

#FinAid Assignment on Canvas https://canvas.ucsd.edu/courses/51649/quizzes/158899

Review guizzes based on class material each day.

Homework assignment 1 due Thursday.