Week 1

Regular Languages Pt. 1

Anakin

Outline

Regex

Automata

Updates

- Why Sipser?
- Combinational Algorithms
- Discord

Section 1

Regex

Regular Expressions

$$\Sigma^* = \texttt{(0 + 1)}^* = \text{The set of all finite binary strings}$$

- Many of us have probably seen regex appear in programming when we want to do **string matching**
- Turns out that **languages** (a set of strings) are an extremely powerful idea at the center of CS
- Languages recognized by regular expressions are called regular languages
- Let's look at a simplified version of regex with only 3 operations

Recursively build regex:

- Base Cases: The emptyset: $\emptyset$, the empty string ε, or a string s
- Operations:
    - **Concatenation:** We can attach strings in one language R_1 to strings in another R_2 and form R_1R_2
        - Example: $R_1 = \{\texttt{0, 00}\}$, $R_2 = \{\texttt{1, 11}\}$, $R_1R_2 = \{\texttt{01, 011, 001, 0011}\}$
    - **Union:** We take one language R_1 and add all the words of another language another $R2$ and form $R_1 + R_2$
        - Example: $R_1 = \{\texttt{0, 00}\}$, $R_2 = \{\texttt{1, 11}\}$, $R_1 + R_2 = \{\texttt{0, 00, 1, 11}\}$
    - **Star:** We take a language R and string words in itself 0 or more times and form R^*
        - Example: $R = \{\texttt{0, 1}\}$, $R^* =$ every possible finite binary string!

Questions?

Questions!

Try to write out some strings in the following languages and come up with a more intuitive way to understand the language
Example:

$$\begin{aligned} \texttt{0}^* + \texttt{0}^*\texttt{1}\texttt{0}^* &= \{\varepsilon, \texttt{0},\ \texttt{00},\ \texttt{000},\ \ldots,\ \texttt{1},\ \texttt{01},\ \texttt{10},\ \texttt{0010},\ \ldots\} \\ &= \text{Strings with at most a single } \texttt{1} \end{aligned}$$

Note: We use $\Sigma = \texttt{(0 + 1)}$

1. $(\Sigma\Sigma\Sigma)^*$
2. $\texttt{0 + 1 + 0}\Sigma^*\texttt{0 + 1}\Sigma^*\texttt{1}$

Section 2

Automata

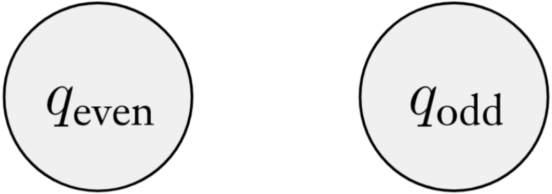
Deterministic Finite Automata

Problem: How can we tell if a string is in some regular language?
Solution: Deterministic Finite Automata (DFA)

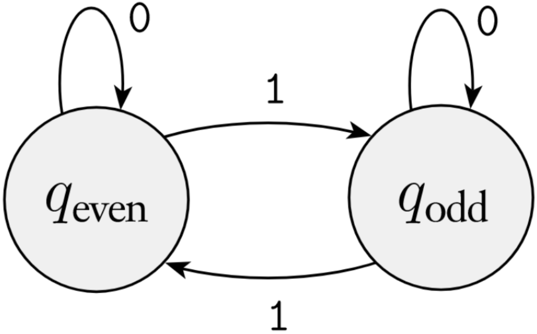
We could define this fully formally but these are better introduced an intuitive level

GOAL: Identify binary strings containing an odd number of `1`'s

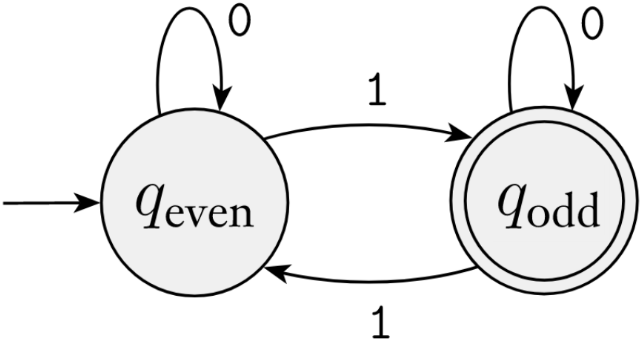
Odd number of 1's



Odd number of 1's



Odd number of 1's



Questions?

Questions!

- Remember those regex from before?
- Turns out regex are **equivalent** to DFAs!!
  Every regex can be turned into a DFA, and vice versa

Try to come up with DFAs for some of the following regular expressions:

1. $0^* + 0^*10^*$
2. $(\Sigma\Sigma\Sigma)^*$
3. $0 + 1 + 0\Sigma^*0 + 1\Sigma^*1$

See y'all next week!