Outline

- FA, Regular Expression equivalence
- Non-regular languages: Pumping Lemma
  ** first problem set out today

Regular expressions and FA

- **Theorem**: a language \( L \) is recognized by a FA if and only if \( L \) is described by a regular expression.
  Must prove two directions:
  \((\Rightarrow)\) L is recognized by a FA implies \( L \) is described by a regular expression
  \((\Leftarrow)\) L is described by a regular expression implies L is recognized by a FA.

Proof: given FA \( M \) that recognizes \( L \), we will

1. build an equivalent machine "Generalized Nondeterministic Finite Automaton" (GNFA)
2. convert the GNFA into a regular expression

Regular expressions and FA

- GNFA definition:
  - it is a NFA, but may have regular expressions labeling its transitions
  - GNFA accepts string \( w \in \Sigma^* \) if can be written
    \[ w = w_1w_2\ldots w_k \]
    where each \( w_i \in \Sigma^* \), and there is a path from the start state to an accept state in which the \( i \)th transition traversed is labeled with \( R \) for which \( w_i \in L(R) \)

Recall step 1: build an equivalent GNFA

- Our FA \( M \) is a GNFA.
- We will require "normal form" for GNFA to make the proof easier:
  - single accept state \( q_{\text{accept}} \) that has all possible incoming arrows
  - every state has all possible outgoing arrows; exception: start state \( q_0 \) has no self-loop
Regular expressions and FA

• converting our FA $M$ into GNFA in normal form:

1. Convert $M$ into a GNFA with $k$ states.
2. Convert the GNFA into a regular expression $R$.

- if normal-form GNFA has two states:
  the regular expression $R$ labeling the single transition describes the language recognized by the GNFA.

- if GNFA has more than 2 states:
  - select one "$q_{rip}\)"; delete it; repair transitions so that machine still recognizes same language.
  - repeat until only 2 states.

- how to repair the transitions:
  - for every pair of states $q_i$ and $q_j$ do
    
    \[
    q_i \to q_j \text{ in } M \\text{ allows all strings taking } M \text{ from } q_i \text{ to } q_j \text{ using } q_{rip} \text{ (see slide)}
    \]
    
    \[
    \text{thus } G' \text{ accepts } w
    \]

- summary:
  FA $M \to k$-state GNFA $\rightarrow (k-1)$-state GNFA $\rightarrow (k-2)$-state GNFA $\rightarrow \ldots \rightarrow 2$-state GNFA $\rightarrow R$
  want to prove that this procedure is correct, i.e. $L(R) = \text{language recognized by } M$

  - FA $M$ equivalent to $k$-state GNFA
  - $i$-state GNFA equivalent to $(i-1)$-state GNFA (we will prove...)
  - 2-state GFNA equivalent to $R$

- Claim: $i$-state GNFA $G$ equivalent to $(i-1)$-state GNFA $G'$ (obtained by removing $q_{rip}$)
- Proof:
  - if $G$ accepts string $w$, then it does so by entering states: $q_0$, $q_1$, $q_2$, ... $q_{accept}$
  - else, break state sequence into runs of $q_0$:
    $q_0 \rightarrow q_0q_0 \rightarrow \ldots \rightarrow q_0q_{accept}$
  - transition from $q_0$ to $q_i$ in $G'$ allows all strings taking $G$ from $q_0$ to $q_i$ using $q_{rip}$ (see slide)
  - thus $G'$ accepts $w$
— **Proof** (continued):
  - if $G'$ accepts string $w$, then every transition from $q_i$ to $q_j$ traversed in $G'$ corresponds to either a transition from $q_i$ to $q_j$ in $G$ or transitions from $q_i$ to $q_j$ via $q_{ri}$ in $G$.
  - In both cases $G$ accepts $w$.
  - Conclude: $G$ and $G'$ recognize the same language.

---

**Theorem**: a language $L$ is recognized by a FA iff $L$ is described by a regular expression.

Languages recognized by a FA are called regular languages.

Rephrasing what we know so far:
- regular languages closed under 3 operations
- NFA recognize exactly the regular languages
- regular expressions describe exactly the regular languages

---

**Limits on the power of FA**

- Is every language describable by a sufficiently complex regular expression?
  - If someone asks you to design a FA for a language that seems hard, how do you know when to give up?

**Is this language regular?**

$w$ has an equal # of "01" and "10" substrings} = 0^*1^* \cup 1^*0^*$

but $w$ has an equal # of "0" and "1" substrings} is not regular!
How do you prove that there is no Finite Automaton recognizing a given language?

**Pumping Lemma:** Let $L$ be a regular language. There exists an integer $p$ ("pumping length") for which every $w \in L$ with $|w| \geq p$ can be written as $w = xyz$ such that
1. for every $i \geq 0$, $xy^iz \in L$, and
2. $|y| > 0$, and
3. $|xy| \leq p$.  

Using the Pumping Lemma to prove $L$ is not regular:

- Assume $L$ is regular
- Then there exists a pumping length $p$
- Select a string $w \in L$ of length at least $p$
- Argue that for every way of writing $w = xyz$ that satisfies (2) and (3) of the Lemma, pumping on $y$ yields a string not in $L$.
- Contradiction.

**Theorem:** $L = \{0^n1^n : n \geq 0\}$ is not regular.

**Proof:**
- Let $p$ be the pumping length for $L$
- Choose $w = 0^p1^p$
  - $w = 00000000...01111111...1$
  - $w = xyz$, with $|y| > 0$ and $|xy| \leq p$.

3 possibilities:

- $w = 00000000...01111111...1$
- $w = 00000000...01111111...1$
- $w = 00000000...01111111...1$

- In each case, pumping on $y$ gives a string not in language $L$.  

**Theorem:** $L = \{w : w$ has an equal # of 0s and 1s $\}$ is not regular.

**Proof:**
- Let $p$ be the pumping length for $L$
- Choose $w = 0^p1^p$
  - $w = 00000000...01111111...1$
  - $w = xyz$, with $|y| > 0$ and $|xy| \leq p$.  

3 possibilities:

- $w = 00000000...01111111...1$
- $w = 00000000...01111111...1$
- $w = 00000000...01111111...1$

- In each case, pumping on $y$ gives a string not in language $L$.  

**Theorem:** $L = \{w : w$ has an equal # of 0s and 1s $\}$ is not regular.

**Proof:**
- Let $p$ be the pumping length for $L$
- Choose $w = 0^p1^p$
  - $w = 00000000...01111111...1$
  - $w = xyz$, with $|y| > 0$ and $|xy| \leq p$.  

3 possibilities:

- $w = 00000000...01111111...1$
- $w = 00000000...01111111...1$
- $w = 00000000...01111111...1$

- In each case, pumping on $y$ gives a string not in language $L$.  

Pumping Lemma Examples

– 3 possibilities:
  \[w = 000000000...0111111111...1\]
  \[w = 000000000...01111111111...1\]
  \[w = 000000000...01111111111...1\]

– first 2 cases, pumping on \(y\) gives a string not in language \(L\); 3rd case a problem!

– recall condition 3: \(|xy| \leq p\)

– since \(w = 0^p1^p\) we know more about how it can be divided, and this case cannot arise:
  \[w = 000000000...01111111111...1\]

– so we do get a contradiction.

– conclude that \(L\) is not regular.