Definition of GNFA

A **generalized non-deterministic finite automaton** is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where

1. \(Q\) is a finite set of states
2. \(\Sigma\) is a finite alphabet
3. \(\delta: (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \rightarrow \mathcal{R}\) is the transition function
4. \(q_{\text{start}} \in Q\) is the start state
5. \(q_{\text{accept}} \in Q\) is the accept state and \(q_{\text{start}} \neq q_{\text{accept}}\)

A GNFA accepts a string \(w \in \Sigma^*\) if \(w = w_1w_2...w_k\) where each \(w_i \in \Sigma^*\) and a sequence of states \(q_0, q_1, ..., q_k\) exists such that

1. \(q_0 = q_{\text{start}}\)
2. \(q_k = q_{\text{accept}}\)
3. for each \(i\), we have \(w_i \in L(R_i)\), where \(R_i = \delta(q_{i-1}, q_i)\)
Proof: Construction

- Let $M$ be the DFA for language $A$
- Convert $M$ to a GNFA $G$ by adding a new start state and a new accept state and additional transition arrows as necessary
- Use procedure $\text{CONVERT}(G)$, which takes a GNFA and returns an equivalent regular expression
  - $\text{CONVERT}$ is recursive, but calls itself on fewer and fewer states, so no infinite loop

Convert: Construction

- Let $k$ be the number of states of $G$
- If $k = 2$
  - $G$ must consist of a start state, an accept state, and a single arrow connecting them with a regular expression $R$
  - Return $R$
- If $k > 2$
  - Select any state $q_{\text{rip}}$ different from $q_{\text{start}}$ and $q_{\text{accept}}$
  - Let $G'$ be the GNFA $(Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$
    - $Q' = Q - \{q_{\text{rip}}\}$
    - And for any $q_i \in Q' - \{q_{\text{accept}}\}$ and $q_j \in Q' - \{q_{\text{start}}\}$ let
      $\delta'(q_i, q_j) = (R_1)(R_3)^* (R_3) \cup (R_4)$, where $R_1 = \delta(q_i, q_{\text{rip}})$, $R_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$,
      $R_3 = \delta(q_{\text{rip}}, q_i)$, and $R_4 = \delta(q_j, q_j)$
  - Compute $\text{CONVERT}(G')$ and return this value
Continued

- Do $G$ and $G'$ accept the same language?
  - Suppose $G$ accepts an input $w$
  - There is an accepting sequence of states: $q_{\text{start}}, q_1, q_2, \ldots, q_{\text{accept}}$
  - If none of them is $q_{\text{rip}}$, then $G'$ will accept $w$
    + Each of the new regular expression of $G'$ contains the old regular expression as part of a union
  - If $q_{\text{rip}}$ does appear, remove each run of consecutive $q_{\text{rip}}$ states
    + This will be an accepting computation for $G'$
      + The states $q_i$ and $q_j$ that bracket each run have a new regular expression on the arrow between them that describes all strings taking $q_i$ to $q_j$ via $q_{\text{rip}}$. So $G'$ accepts $w$
  - Conversely, say $G'$ accepts an input $w$
    + As each arrow between any two states $q_i$ and $q_j$ in $G'$ describes the collection of strings taking $q_i$ to $q_j$ in $G$, either directly or via $q_{\text{rip}}$, $G$ must also accept $w$
Overview

- Regular Expressions (cont)
  ⇒ Non-regular languages
- Constructing Complex FA

Examples

- To understand the power of finite automata, you must understand their limitations
- What languages are not regular?
  - In the second class, we showed that $a^n b^n$ is not regular
- Consider
  - $\{ w | w \text{ has an equal number of 0s and 1s} \}$
  - $\{ w | w \text{ has an equal number of 01 and 10 as substrings} \}$
How can we prove a Language is not Regular

• Regular language is recognizable by a DFA
  - Say DFA has \( p \) states
  - DFA can only be in one of those \( p \) states
  - So, if input has more than \( p \) length, DFA must repeat a state
  - Everything that it saw between the two instances of that state
    + Could be taken out
    + Or repeated as many times as you want!
    + It can be \textit{pumped down} or \textit{pumped up}
  - If string that is pumped up or pumped down is not part of language,
    then language must not be regular

Pumping Lemma

If \( A \) is a regular language, then there is a number \( p \) (the pumping length) where, if \( s \) is any string in \( A \) of length at least \( p \), then \( s \) can be divided into three parts, \( s = xyz \), satisfying the following conditions:

1. for each \( i \geq 0 \), \( xy^i z \in A \)
2. \( |y| > 0 \)
3. \( |xy| \leq p \)
Proof of Pumping Lemma

- Let $M$ be a DFA that recognizes $A$ and $p$ be number of states
  - Use a DFA to prove this, much easier than using a NFA!
- Case 1: There is no string in $A$ of length at least $p$
- Case 2: Let $s \in A$ and have length $n \geq p$
  - In accepting $s$, $M$ will go through at least $n + 1$ states
    - $s = s_1 s_2 s_3 s_4 s_5 \ldots s_{n-1} s_n$
      - Pigeonhole principle: two of $r_1 \ldots r_{p+1}$ must be the same; say $r_j$ and $r_l$
      - Let $x$ be the string $s_1 \ldots s_{j-1}$. $x$ takes $M$ from $r_0$ to $r_j$
      - Let $y$ be the string $s_{j+1} \ldots s_l$. $y$ takes $M$ from $r_j$ to $r_l$
      - Let $z$ be the string $s_{l+1} \ldots s_n$. $z$ takes $M$ from $r_l$ to $r_n$
      - $M$ must accept $xy^i z$ as it can go through the cycle $r_j \ldots r_l$ times

Example: $\{0^n1^n | n \geq 0\}$

- To use pumping lemma, assume that language is regular, and show there is a contradiction
- Assume it is regular
  - Let $p$ be the pumping length for the language
  - Choose $s$ to be $0^p1^p$
    - By the pumping lemma, there must be a $x, y, z$, in which $|xy| \leq p$
      + $x = 0^j$ for some $j \geq 0$
      + $y = 0^k$ for some $k > 0$
      + $z = 0^l1^{k+l}$ where $l \geq 0$
    - $xy^i z$ must be in the language by the pumping lemma
      + But, for $i \neq 1$, the string will not have the same number of 0s and 1s
Example: \( \{w \mid w \text{ has an equal number of 0s and 1s}\} \)

- Assume it is regular, so must be a \( p \)
- But what string should we choose?
  - If we pick \( (01)p \), we cannot find contradiction

- Assume it is regular, so must be a \( p \)
- But what string should we choose?
  - How about \( 0^p1^p \)?
    - Won't work, as we can pump \( 00 \)
    - How about \( 0^p1^p1 \)?
      - This forces us to pump in the first set of 0's
Final Words on Pumping Lemma

- All regular languages obey the pumping lemma
- But not all non-regular languages do not obey pumping lemma
  - Some non-regular languages obey the pumping lemma
  - Cannot use pumping lemma to show a language is regular
  - See Question 1.54
- How can you prove a language is regular?

Overview

- Regular Expressions (cont)
- Non-regular languages
  ⇒ Constructing Complex FA
Constructing Complex FA

• To prove a language is regular, construct a DFA/NFA/RE
  - Might need to modify a DFA(s) for another language
  - Seen this for intersection, union, complement, star, concatenation

• Other Examples
  - Prefix: \( L = \{ w | wx \in A \} \) where \( A \) is regular
  - Suffix: \( L = \{ w | xw \in A \} \) where \( A \) is regular
  - Splicing: \( L = \{ xy | xw \in A \text{ and } uy \in B \} \) where \( A \) and \( B \) are both regular

Prefix DFA

\( L = \{ w | wx \in A \} \) where \( A \) is regular

• Option 1:
  - Since \( A \) is regular, there is a DFA \( M \) that recognizes it
  - Define garbage states:
    + states that do not have a path on any string to an accept state
    + don’t have to worry about states not reachable from start:
      can never get to them anyways
  - To construct a DFA for \( L \), use \( M \)
    + but change all of its non-garbage states to accept states
Epsilon Transitions

- \( A \) is a regular language, and \( M \) a DFA that recognizes it.
- Let’s build a FA \( N \) based on \( M \) but where we change all transitions so that rather than read characters, they read \( \epsilon \)
  - \( \delta_N(q, \epsilon) \in \{ q’ \mid \text{there is } a \in \Sigma \text{ s.t. } \delta_M(q, a) = q’ \} \)
  - Note that \( M \) is now a NFA, but that is fine.
- Claim: if \( A \) is empty, so is \( L(N) \) other \( L(N) = \{ \epsilon \} \)
  - If \( A \) accepts a string, must be a path from start to an accept state in \( M \)
    + That same path is in \( N \), so \( N \) must accept \( \epsilon \)
  - If \( A \) does not accept any string, there is no path from start to accept state in \( M \)
    + So there is no path in \( N \), so \( N \) does not accept anything.

Second Way to Show Prefix is Regular

\( L = \{ w \mid wx \in A \} \) where \( A \) is regular.

- Let \( M \) be a DFA for \( A \)
  - Make a copy of \( M \), call it \( N \)
  - Have \( \delta_N \)'s transitions be same as \( \delta_M \)'s, but read \( \epsilon \) instead of characters
  - Make a new machine \( P \) that joins \( M \) and \( N \) together
    + Add transitions from states of \( M \) to corresponding states of \( N \) on reading \( \epsilon \)
    + Start state is \( M \)'s start state, and accept states are \( N \)'s accept states.
More formally

- First $N$
  - Construct $N$ similar to $M$, same states, start state, and accept states
  - $\delta_N(q, \varepsilon) = \{q' \mid \text{there is } a \in \Sigma \text{ s.t. } \delta_M(q, a) = q'\}$

- Now $P$
  - States of $P$ are the union of states of $M$ and $N$ (not cross product)
  - Call $M$’s states $m_1...m_k$ and $N$’s states $n_1...n_k$
  - $\delta_P$ has all the transitions of $M$ and $N$
  - $\delta_P$ also has the following transitions:
    $\delta_P(m_i, \varepsilon) \rightarrow n_i$ for all corresponding states $m_i$ and $n_i$
  - Start state of $P$ is $M$’s start state, and accept states are $N$’s accept states

Splicing

$L = \{xy \mid xw \in A \text{ and } uy \in B\}$ where $A$ and $B$ are both regular

- Let $M$ be DFA s.t. $L(M) = A$ and $N$ be a DFA s.t. $L(N) = B$
  - Join the two machines together
    - Add transitions from all states of $M$ to all states of $N$ with $\varepsilon$ transition
    - Start state is $M$’s start state, and accept states are $N$’s accept states

- More formally
  - Create $P$ as follows
  - $P$ has union of states of $M$ and $N$
  - Start state of $P$ is $M$’s start state
  - Accept states of $P$ are $N$’s accept states
  - Transitions of $P$ are union of $M$ and $N$’s transitions
    along with $\delta(q, \varepsilon) \rightarrow q'$ for all of $q \in M$’s states and $q' \in N$’s states
Common Prefix

\[ L = \{ x | xw \in A \text{ and } xy \in B \} \] where \( A \) and \( B \) are both regular

- Let \( M \) be a DFA that recognizes \( A \) and \( N \) be a DFA that recognizes \( B \)
  - Build machine \( P \) in the same way we built a DFA that does intersection (cross product of states)
  - Turn all non-garbage states into accept states