Review of regular languages

**Def. 1:** DFA  $M = (Q, \Sigma, \delta, q_0, F)$

- $Q$: states
- $\Sigma$: alphabet
- $\delta$: transition function
- $q_0$: start state
- $F$: final states

$\delta: Q \times \Sigma \rightarrow Q$  $\Rightarrow \hat{\delta}: Q \times \Sigma^* \rightarrow Q$

$L(M) = \{ x \in \Sigma^* | \hat{\delta}(q_0, x) \in F \}$

Deterministic finite automata model physical computing devices.

**Def. 2:** NFA  $N = (Q, \Sigma, \delta, q_0, F)$

- $Q': Q \times \Sigma \rightarrow 2^Q$

$L(N) = \{ x \in \Sigma^* | \delta(q_0, x) \cap F \neq \emptyset \}$

**Def. 3:** E-NFA  $\hat{\delta}: Q \times (\{ \epsilon \} \cup \Sigma) \rightarrow 2^Q$

$\hat{\delta}$ defined by $\varepsilon$-closure

**Def. 4:** Regular expression

- Literals: $\epsilon, a \in \Sigma$
- Operations: $\cdot, +, ^*$

$L(r \cdot s) = L(r) \cdot L(s)$, etc.

**Theorem:** Consider $L \subseteq \Sigma^*$. The following four conditions are equivalent:

1. $\exists$ DFA $M : L(M) = L$
2. $\exists$ NFA $N : L(N) = L$
3. $\exists$ E-NFA $N : L(N) = L$
4. $\exists$ regular expression $r : L(r) = L$

**Def.** $L$ is a regular language if any of the above conditions hold.

**Morals:**

1. Regular languages can be specified either operationally (by what a FA recognizes) or syntactically (by what a regex expression generates)
2. Adding non-determinism
   - Does not increase the set of languages that can be decided, i.e., the set of problems that can be solved
   - Can be exponentially more efficient, in terms of number of states
3. To prove regularity, can use closure properties
   **Exercise:** For each closure property we studied, what is the overhead for applying it? (In terms of # of states for an automaton, or # of operators for a regular expression) Eg., complement
4. To prove that a language is not regular, can use Pumping Lemma
   In English: If $L$ is a regular language and $x \in L$ is sufficiently long, then there is a nonempty substring of $x$ (sufficiently close to the start) that can be pumped: $x = uvw$ and $uv^n w \in L$. 

...
Pumping Lemma (review)

For any regular language \( L \), there exists a constant \( N \) such that for any string \( x \in L \) with \( |x| > N \), there is a decomposition \( x = uvw \) with:
- \( uv^iw \in L \) for all \( i \geq 0 \)
- \( |v| > 0 \) (i.e., \( v \neq \epsilon \))
- \( |uv| \leq N+1 \).

In English: If \( L \) is a regular language, then any sufficiently long string \( x \in L \) has a nonempty substring that can be pumped.

How small can \( N \) be, as a function of \( L \)?

From the proof, \( N = \) longest path in a DFA or NFA without any cycles (not an \( \epsilon \)-NFA)

\[ \Rightarrow N = \min \# \text{states in any NFA for } L \text{ certainly works} \]

The Pumping Lemma can also be proved based on a regular expression for \( L \).

Eg., say
\[ r = \epsilon + 000(\epsilon + \underbrace{1 + 111}(\epsilon + \epsilon)0^*) \]

The longest \( x \in L(r) \) that cannot be pumped is \( x = 000111011 \)

Any longer string would have to use the \( 0^* \) portion of \( r \) and that part could be pumped.

\[ \Rightarrow N = \# \text{of alphabet characters in } r \text{ certainly works, as a rather crude bound} \]
Context-free grammars (Ch.5)

Example: If the rules are \( A \rightarrow OA1, A \rightarrow \varepsilon \)
then can derive:
\[
\begin{align*}
A & \Rightarrow OA1 \\
& \Rightarrow O(OA1)1 = OOA111 \\
& \Rightarrow OOOA1111 \\
& \Rightarrow OOO\varepsilon 111 = 0^31^3
\end{align*}
\]
\[L(A) = \{ \text{every string derivable from } A \} = \{ 0^n1^n | n \geq 0 \} \quad \text{not regular} \]

Def: Context-free grammar \( G = (V, T, P, S) \)
- \( V = \) "variables", a finite set (each representing a set of strings)
- \( T = \) "terminal symbols" or alphabet, finite \& disjoint from \( V \)
  (the symbols that compose the strings)
- \( P = \) productions = a set of rules each of the form
  \[
  \text{Head } \rightarrow \text{ Body}
  \]
  \[\begin{align*}
  \text{\small V} & \quad \text{\small (VUT)}^* \\
  \text{\small \& variable} & \quad \text{\small \& string in variables \& terminals}
  \end{align*}\]
- \( T \)

Example: \( G = (\{A,B\}, \{0,1\}, \{A \rightarrow OA1, A \rightarrow \varepsilon \}, A) \)
Compact notation: \( A \rightarrow OA1 | \varepsilon \) means \( A \rightarrow OA1 \) and \( A \rightarrow \varepsilon \).

Derivations:
\[\Rightarrow\] means one application of a rule
- if \( A \rightarrow \beta \) is a production, then
  \[\Lambda A \beta \Rightarrow \alpha \gamma \beta \quad \text{for all } \alpha, \beta, \gamma \in (T \cup V)^*\]
\[\Rightarrow^*\] means zero or more steps of \( \Rightarrow \) (same as \( \Rightarrow^* \) vs. \( \Rightarrow \) for \( FA \)).

Def: Language of a CFG
- If \( G = (V, T, P, S) \) is a CFG, then
  \[L(G) = \{ \omega \in T^* | S \Rightarrow^* \omega \}\]
  i.e. the set of all terminal strings that can be derived from the start.

Example: If \( A \rightarrow OA1 | A1 | \varepsilon \) and \( S \rightarrow AS | \varepsilon \)
then \( L(A) = \{ 0^k1^k | k \geq 0 \} \)
\[L(S) = L(A)^* = \{ 0^{k_1}1^{k_1}0^{k_2}1^{k_2} \cdots 0^{k_m}1^{k_m} | m \geq 0, k_1 \leq k_2 \leq \cdots \leq k_m \}\]
**Claim:** \( L(A) = \{0^k1^k : 1 \leq k \leq 3\} \) if \( A \rightarrow OA1|A1|\)  

**Proof:** Let \( K = \{0^k1^k : 1 \leq k \leq 3\} \). \( \text{WTS} \) \( K \subseteq L(A) \), \( L(A) \subseteq K \).  

1. \( K \subseteq L(A) \): Let \( x = 0^k1^k \in K \) with \( 1 \leq k \leq 3 \). \( \text{Want to show} \ x \in L(A) \), i.e., \( A \overset{*}{\Rightarrow} x \).  

**Claim:** For all \( m, n \) with \( m \leq n \), \( A \overset{*}{\Rightarrow} 0^m1^n \)  

**Proof:** By induction in \( n \).  

- **Base case** \( n = 0 \):  
  Then \( m = 0 \), \( A \overset{*}{\Rightarrow} A \) (zero steps) \( \checkmark \)  

- **Induction step:**  
  If \( m = 0 \), use \( A \overset{*}{\Rightarrow} A_1^{n-1} \) (by induction)  
  \( \Rightarrow A_1^n \) (by \( A \rightarrow A_1 \)) \( \checkmark \)  

  If \( m > 0 \), use \( A \overset{*}{\Rightarrow} 0^m1^{n-1} \) (by induction)  
  \( \Rightarrow 0^m1^n \) (by \( A \rightarrow OA_1 \)) \( \checkmark \)  

Hence \( K \subseteq L(A) \).  

2. \( L(A) \subseteq K \): Proof by a similar induction. Main claim is  

**Claim:** \( \{x \in \{0,1\}^* : A \overset{*}{\Rightarrow} x \text{ in } m \leq N \text{ steps}\} \)  

\( \subseteq \{0^m1^n : m \leq n \leq N\} \cup \{0^m1^n : m \leq n \leq N\} \).  

---

**Example:** \( S \overset{*}{\Rightarrow} SS | (s) | [s] | \varepsilon \)  

generates \( L(S) = \) strings of matching parentheses  

- not regular  

  \( \text{e.g.,} \ (\text{(}} (\text{(}} (\text{)}) (\text{))} \in L(S) \)  

  \( (\text{)}} (\text{) \notin L(S) \)  

**Example** For the regular language \( L(0^+1(0+1)^+) \), it is \( L(S) \) where  

\[ 
\begin{align*}
S & \rightarrow A1B \\
A & \rightarrow \varepsilon | 0A \\
B & \rightarrow \varepsilon | OB | 1B
\end{align*}
\]  

**Theorem:** Every regular language has a context-free grammar.
Theorem. Every regular language is generated by a context-free grammar.

In other words, if $r$ is a regular expression, then there exists a CFG $G$ such that $L(r) = L(G)$.

Proof: By the obvious induction in the number of operators of $r$.

- If $r = \epsilon$, use $S_r \rightarrow \epsilon$, no productions, or $S_r \rightarrow \epsilon$, respectively, to get $L(S_r) = L(r)$.

Induction step $r = r_1 r_2$.

- If $r = \epsilon$, use $S_r \rightarrow S_r S_r$.
- If $r = s + \epsilon$, use $S_r \rightarrow S_r | S_r$.
- If $r = s^*$, use $S_r \rightarrow S_r S_r | \epsilon$.

Parse Trees represent a sequence of derivations hierarchically.

Example: $G = (\{A, S\}, \{\circ, \rightarrow\}, S \rightarrow AS, S \rightarrow E, A \rightarrow E, A \rightarrow AA, A \rightarrow \epsilon, S \rightarrow AS, S \rightarrow E, S \rightarrow A | \epsilon)$

In general,

- root must be start variable
- internal nodes are variables
- leaves are terminals or $\epsilon$
- for each internal node, it is the head and its children from left to right the body of some production

yield is $01100111E = 01100111$.

Important in applications: eg, software source code, XML document structure, STD, document type definitions.

(read Ch.5.3) parsers for programming languages: Bison & Yacc.

- Explains "Context-free" because you can expand a subtree without worrying about anything outside that subtree (its context).

- Theorem: For a CFG $G = (V, T, P, S)$, a variable $A \in V$ and a string $x \in T^*$,

$$A \Rightarrow x \quad \text{if and only if} \quad \text{there is a parse tree with root } A \text{ and yield } x.$$ 

(Proof by induction on tree size/# derivation steps, see [5.2.4&5.2.6].)