Last Lecture Covered...

- DFAs, NFAs, $\epsilon$-NFAs and the equivalence of the language classes they accept

This Lecture will Cover...

- Introduction to regular expressions and regular languages
- Equivalence of classes of regular languages and languages accepted by DFAs
- Algebraic laws of (abstract) regular expressions

Background Reading: Chapter 3 of HMU.
Regular Expressions

- So far, DFAs, NFAs were given a machine-like description

- Regular expressions are *user-friendly* and *declarative* formulation

- Regular expressions find extensive use.
  - Searching/finding strings/pattern matching or conformance in text-formatting systems (e.g., UNIX `grep`, `egrep`, `fgrep`)
  - Lexical analyzers (in compilers) use regular expressions to identify tokens (e.g., `Lex`, `Flex`)
  - In Web forms to (structurally) validate entries (passwords, dates, email IDs)
Regular Expressions

• Given an alphabet \( \Sigma \) of symbols disjoint from \( \{+, *, (, )\} \):

  - A regular expression over \( \Sigma \) is a string over \( \Sigma \cup \{*, +, (, )\} \)
  
  i.e., a regular expression consists *only* of:

  1. constants: \( \emptyset, \epsilon \)
  2. symbols from \( \Sigma \)
  3. operators: \( +, * \)
  4. parantheses: \( (, ) \)

• Regular expressions are defined via induction.
Regular Expressions

- Regular expressions are defined inductively as follows:

  **Basis:** (B1) \( \emptyset \) and \( \epsilon \) are regular expressions.
  (B2) For each \( a \in \Sigma \), \( a \) is a regular expression.

  **Induction:** If \( E \) and \( F \) are regular expressions:

  - (I1) So is \( E^* \)
  - (I2) So is \( E+F \)
  - (I3) So is \( EF \)
  - (I4) So is \( (E) \)

- *Only* those generated by the above induction are regular.

**Remark 1:** Some authors/texts use \( | \) instead of \( + \). HMU uses +.

**Remark 2:** All expressions generated by Option 2 are also generated by Option 1

\[
I_1 + I_4 \Rightarrow I_1' \quad I_2 + I_4 \Rightarrow I_2' \quad I_3 + I_4 \Rightarrow I_3'
\]

**Remark 3:** Some expressions are regular according to Option 1 but not Option 1
E.g., \((0))\), \(0 + 11^*\)
• Let $\Sigma = \{0, 1\}$.

• $(((0 + 1)1)*0)$ is a regular expression

• $0 + 11^*0$ is a regular expression

<table>
<thead>
<tr>
<th>Expression</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(B2)</td>
</tr>
<tr>
<td>1</td>
<td>(B2)</td>
</tr>
<tr>
<td>(0+1)</td>
<td>(I2')</td>
</tr>
<tr>
<td>((0+1)1)</td>
<td>(I3')</td>
</tr>
<tr>
<td>(((0+1)1)*)</td>
<td>(I1')</td>
</tr>
<tr>
<td>(((((0+1)1)*)0)</td>
<td>(I3')</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Expression</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(B2)</td>
</tr>
<tr>
<td>1</td>
<td>(B2)</td>
</tr>
<tr>
<td>0 + 1</td>
<td>(I2)</td>
</tr>
<tr>
<td>0 + 11</td>
<td>(I3)</td>
</tr>
<tr>
<td>0 + 11*</td>
<td>(I1)</td>
</tr>
<tr>
<td>0 + 11*0</td>
<td>(I3)</td>
</tr>
</tbody>
</table>
What Do Regular Expressions Stand For?

- Each **properly parenthesized** regular expression $E$ (i.e., a regular expression that is generated by Option 2) is a shorthand for a language $L(E)$.

- A language is said to be **regular** if it corresponds to a regular expression.

**Basis:**

- $(B1)$ $L(\emptyset) = \emptyset$ [Empty Language]
- $L(\epsilon) = \{\epsilon\}$ [Language with only the empty string]
- $(B2)$ $L(a) = \{a\}, a \in \Sigma$ [Language with only the symbol $a$]

**Induction:**

- $(I1') L((E^*)) = L(E)^* \overset{\Delta}{=} \{\epsilon\} \cup L(E) \cup L(E)^2 \cup L(E)^3 \cdots$ [Kleen-* closure of $L(E)$]
- $(I2') L((E + F)) = L(E) \cup L(F)$ [Union]
- $(I3') L((EF)) = L(E)L(F)$ [Concatenation]

What if a regular expression is generated by Option 1?
What if an Expression isn’t Bracketed Properly?

• Improperly parenthesized regular expressions =

• Is $0 + 11$ the same as $((0 + 1)1)$? Or is it equal to $(0 + (11))$?
What if an Expression isn’t Bracketed Properly?

- Improperly parenthesized regular expressions (generated by Option 1) must be converted to properly paranthesized expressions.

1. We remove unwanted parentheses by replacing \(((E))\) by \(E\) inductively.
   - Additionally, if \(E\) is a symbol or a constant, we replace it by \(E + \emptyset\)
     e.g., \(((0)) \equiv (0 + \emptyset)\), \(((0 + 1)) \equiv (0 + 1)\)

2. Apply precedence rules:
   First: \(*\) applies to the smallest (properly bracketable) expression preceding \(*\).
     e.g., \(01* \equiv (0(1*))\)
   Second: concatenation applies from left to right.
     e.g., \(010 \equiv ((01)0)\)
   Third: \(+\) applies from left to right
     e.g., \(a + b + c \equiv ((a + b) + c)\)

Example:

\[
0 + 11* \equiv (0 + (1(1*)))
\]
\[
((0)) + 11* \equiv ((0 + \emptyset) + (1(1*)))
\]

\[
L(0 + 11*) = L(((0)) + 11*) = (L(0) \cup (L(1)L(1)*))
\]

\[
= \{0, 1, 11, 111, 1111, \ldots\}
\]
Theorem 1: Let $w \in \Sigma^*$. Then $\{w\}$ is regular.

Proof: Languages $\{\epsilon\}$ and $\{a\}$ for $a \in \Sigma$ are regular (B1, B2).

For $w = s_1 s_2 \cdots s_k \in \Sigma^k$ for $k \geq 2$, $\{w\} = L(s_1 s_2 \cdots s_k)$ [Induction]

Theorem 2: Let $L_1$ and $L_2$ be regular languages. Then, $L_1^*$, $L_1 \cup L_2$ and $L_1 L_2$ are also regular languages.

Proof: Let $L_i = L(E_i)$ for $i = 1, 2$. Then, $L_1^* = L((E_1^*))$, $L_1 \cup L_2 = L((E_1 + E_2))$ and $L_1 L_2 = L((E_1 E_2))$. Since $E_1^*$, $(E_1 + E_2)$ and $(E_1 E_2)$ are regular expressions, the claim holds.

Corollary 1: The class of regular languages is closed under finite union and concatenation, i.e., if $L_1, \ldots, L_k$ are regular languages for any $k \in \mathbb{N}$, then $L_1 \cup \cdots \cup L_k$ and $L_1 \cdots L_k$ are also regular languages.

Corollary 2: Any finite language is regular.
Theorem 3: For every regular language $M$, there exists a DFA $A$ such that $M = L(A)$.

Proof: WLOG, let $\Sigma = \{0, 1\}$. Let $M$ be a regular language. Then, $M = L(E)$.

For each regular expression, we will devise an $\epsilon$-NFA.

Basis:

- $E = \emptyset$
  - $A : q_0 \rightarrow 0, 1 q_1$
  - $A : q_0 \rightarrow 0, 1 q_1$

- $E = \epsilon$
  - $A : q_0 \rightarrow 0, 1 q_1$
  - $A : q_0 \rightarrow 0, 1 q_1$

- $E = 0$
  - $A : q_0 \rightarrow 1 q_1$
  - $A : q_0 \rightarrow 0 q_1$

- $E = 1$
  - $A : q_0 \rightarrow 1 q_1$
  - $A : q_0 \rightarrow 0 q_1$
Induction: [I1']
Proof of Theorem 3 [Continued]

Induction: [I2']

\[ E \]

\[ (E + F) \]
Proof of Theorem 3 [Continued]

Induction: [I3']
So far...

Regular Languages

Languages accepted by DFAs, NFAs, $\epsilon$-NFAs

Finite languages

Is the inclusion strict?

Are there languages accepted by DFAs that are not regular?
Theorem 4: For every DFA $A$, there is a regular expression $E$ such that $L(A) = L(E)$.

Proof:
1) Let DFA $A = (Q, \Sigma, \delta, q_0, F)$ be given.
2) Let us rename the states so that $Q = \{q_0, q_1, q_2, \ldots, q_{n-1}\}$
3) For any string $s_1 \ldots s_k \in L(A)$, there is a path
   \[
   q_0 \xrightarrow{s_1} q_{i_1} \xrightarrow{s_2} q_{i_2} \cdots \xrightarrow{s_k} q_{i_k} \in F
   \]
4) Let $R(i, j, k)$ be the set of all input strings that move the internal state of $A$ from $q_i$ to $q_j$ using paths whose intermediate nodes comprise only of $q_\ell$, $\ell < k$. 

\[\text{States } q_k, \ldots, q_{n-1}\]
\[\text{States } q_0, \ldots, q_{k-1}\]
5) Then $L(A) = \bigcup_{j:q_j \in F} R(0, j, n)$.
[i.e., paths that start in $q_0$ and end in an accepting state with intermediate nodes $q_0, q_1, \ldots, q_{n-1}$ (all nodes)]

6) $L(A)$ will be regular if each $R(i, j, k)$ to be regular. We now proceed to show that each $R(i, j, k)$ is regular.

7) Induction:

Base: Consider $R(i, j, 0)$ for $i, j \in \{0, 1, \ldots, n - 1\}$.

$R(i, j, 0)$ consists of strings whose corresponding paths start in $q_i$ and end in $q_j$ with intermediate nodes $q_\ell, \ell < 0$.

$\Rightarrow$ NO INTERMEDIATE NODES!
$\Rightarrow$ $R(i, j, 0)$ contains strings that change state $q_i$ to $q_j$ directly
$\Rightarrow$ $R(i, j, 0) \subseteq \{\epsilon\} \cup \Sigma$
$\Rightarrow$ $R(i, j, 0)$ is a regular language [Corollary 2]
Proof of Theorem 4 [Continued]

Induction: Let $R(i, j, \ell)$ be regular for $i, j \in \{0, \ldots, n - 1\}$ and $0 \leq \ell < k$.

Consider $R(i, j, k)$ for $i, j \in \{0, \ldots, n - 1\}$.

The strings in $R(i, j, k)$ correspond either to paths whose intermediate nodes $q_0, \ldots, q_{k-1}$.

Partition $R(i, j, k)$ as follows:

Case (a): Strings whose paths do not have $q_{k-1}$ as an intermediate node

Case (b): Strings whose paths do pass through $q_{k-1}$ as an intermediate node
Proof of Theorem 4 [Continued]

\[ R(i, j, k) = \{ \text{Case (a) strings} \} \cup \{ \text{Case (b) strings} \} \]

Case (a) Strings are exactly those in \( R(i, j, k - 1) \)

Hence,

\[ R(i, j, k) = R(i, k - 1, k - 1) \cup \{ \text{Case (b) strings} \} \]
Proof of Theorem 4 [Continued]

Each case (b) string is the concatenation of 3 strings:

1. A string that changes the state from $q_i$ to $q_{k-1}$ through a path whose intermediate nodes are $q_0, \ldots, q_{k-2}$ \( \text{i.e.}, R(i, k-1, k-1) \)

2. A finite concatenation of strings, each of which take $q_{k-1}$ back to $q_{k-1}$ via paths that use only $q_0, \ldots, q_{k-2}$ as intermediate nodes. \( \text{i.e.}, R(k-1, k-1, k-1)^* \)

3. A string that takes $q_{k-1}$ back to $q_j$ via a path that uses only $q_0, \ldots, q_{k-2}$ as intermediate nodes. \( \text{i.e.}, (R(k-1, j, k-1) \)

\[
R(i, j, k) = R(i, j, k-1) \cup [R(i, k-1, k-1)R(k-1, k-1, k-1)^*R(k-1, j, k-1)]
\]

From Theorem 2, it follows that $R(i, j, k)$ is regular for any $i, j, k$. Consequently, $L(A)$ is regular.
The following are indeed equivalent:

- The class of regular languages
- The class of languages accepted by DFAs
- The class of languages accepted by NFAs
- The class of languages accepted by $\varepsilon$-NFAs
Properties of Regular Languages

• Regular languages are closed under finite union, concatenation, and Kleene-* operation. [Theorem 2]

• They are also closed under:
  
  Complementation
  
  [Given DFA $A = (Q, \Sigma, \delta, q_0, F)$, DFA $A' = (Q, \Sigma, \delta, q_0, F^c)$ accepts $L(A)^c$]

  Intersection
  
  [De Morgan’s Law: $R_1 \cap R_2 = (R_1^c \cup R_2^c)^c$]
Abstract Regular Expressions

- We can also define **abstract** regular expressions over languages over $\Sigma$.

Let $\mathcal{V}$ be a set of **variables** (which will be interpreted as languages).

Use the induction definition for regular languages replacing $B_2$ alone by:

\[(B2) \ M \text{ is an (abstract) regular expression for every } M \in \mathcal{V}\]

**Remark:** Even though $\mathcal{V}$ could be infinite, every regular expression consists only of finitely many variables.

- Unlike **concrete** regular expressions (such as $1^*$, $0 + 1$), **abstract** regular expressions (such as $M^*$, $M + N$) don’t stand for a **unique** language.

- However, we can **evaluate** abstract regular expressions by **assigning** any languages to variables, and inductively interpreting:

  Variable$^* \rightarrow$ Kleene-$*$ closure of its language

  Sum of variables $\rightarrow$ union of the languages assigned to them

  Concatenation of variables $\rightarrow$ concatenation of their languages

- We can introduce a notion of equality of (abstract) regular expression:

  Abstract regular expressions $E_1 = E_2 \iff$ For any assignment of languages to the variables contained in $E_1, E_2$, their evaluations equal (i.e., $L(E_1) = L(E_2)$)
**Algebraic Laws of Abstract Regular Expressions**

- **Commutativity**: $L + M = M + L$
  - $LM \neq ML$  
    - [Union is commutative]
    - [Concatenation is not commutative]

- **Associativity**: $(L + M) + N = L + (M + N)$
  - $(LM)N = L(MN)$
    - [Union is associative]
    - [Concatenation is associative]

- **Identity**: $\emptyset + L = L + \emptyset = L$
  - $\varepsilon L = L \varepsilon = L$
    - [$\emptyset$ is the identity element for $+$]
    - [$\varepsilon$ is the identity element for concatenation]

- **Annihilator**: $\emptyset L = L \emptyset = \emptyset$

- **Idempotent**: $L + L = L$

- **Distributive**: $L(M + N) = LM + LN$
  - $(M + N)L = ML + NL$
    - [Concatenation distributes over $+$]

- **Kleene $*$**: $(L^*)^* = L^*$;  $\emptyset^* = \varepsilon$;  $\varepsilon^* = \varepsilon$. 

23