Regular Expressions and Languages
Regular Expressions

- Regular expressions are closely related to NFA and can be thought of as a “user-friendly” alternative to NFA notation for describing software components.
- Example: 01* +10*
- Regular expressions are used in e.g.
  1. UNIX grep command
  2. UNIX Lex (Lexical analyzer generator) and Flex (Fast Lex) tools.
- Regular expressions denote languages.
The Operators on Regular Expressions

Operations on languages that the operators of regular expressions represent.

• Union of two languages $L$ and $M$
  \[ L \cup M = \{ w : w \in L \text{ or } w \in M \} \]

• Concatenation of languages $L$ and $M$
  \[ L \cap M = \{ w : w = xy, x \in L, y \in M \} \]

• Powers of a language $L$
  \[ L^0 = \{ \varepsilon \}, \quad L^1 = L, \quad L^{k+1} = L.L^k \]

• The Closure (or star, or Kleene closure) of a language is denoted by $L^*$.
  \[ L^* = \bigcup_{i=0}^{\infty} L^i \]
Examples

• If $L=\{0,1\}$ then $L^*$ is all strings of 0’s and 1’s.
• Let $L=\{0,11\}$. $L^0=\{\varepsilon\}$, $L^1=L$, $L^2=\{00, 011, 110, 1111\}$ etc
• If $L=\{0,11\}$, $L^*$ consists of 0’s and 1’s, but the one’s come in pairs, e.g., 011, 11110, and $\varepsilon$:
  $$L^* = \{\varepsilon, 0, 11, 00, 011, 110, 1111, \ldots\}$$

• $\emptyset^0=?$, $\emptyset^i=?$, $\emptyset^*=?$.
• $\emptyset^0=\{\varepsilon\}$, $\emptyset^i=\emptyset$, $\emptyset^*=\{\varepsilon\}$. 
Building Regular Expressions (Regex)

Inductive definition of regex's:

**Basis:** $\varepsilon$ is a regex and $\emptyset$ is a regex.

$L(\varepsilon) = \{\varepsilon\}$, and $L(\emptyset) = \emptyset$.

If $a \in \Sigma$, then $a$ is a regex. $L(a) = \{a\}$.

**Induction:**

If $E$ is a regex's, then $(E)$ is a regex.

$L((E)) = L(E)$.

If $E$ and $F$ are regex's, then $E + F$ is a regex.

$L(E + F) = L(E) \cup L(F)$.

If $E$ and $F$ are regex's, then $E.F$ is a regex.

$L(E.F) = L(E).L(F)$.

If $E$ is a regex's, then $E^*$ is a regex.

$L(E^*) = (L(E))^*$.  Closure of $L(E)$

Concatenation of languages
Example: Regex for

\[ L = \{ w \in \{0, 1\}^* : 0 \text{ and } 1 \text{ alternate in } w \} \]

\[ (01)^* + (10)^* + 0(10)^* + 1(01)^* \]

or, equivalently,

\[ (\epsilon + 1)(01)^*(\epsilon + 0) \]

Order of precedence for operators:

1. Star
2. Dot
3. Plus

Example: 01* + 1 is grouped (0(1)* + 1
Expressions and their languages

Strictly speaking, a regular expression $E$ is just an expression, not a language. We should use $L(E)$ when we want to refer to the language that $E$ denotes. However, it’s common usage to refer to say “$E$” when we really mean “$L(E)$”.

Finite Automata and Regular Expression

• We have already shown that DFA's, NFA's, and $\varepsilon$-NFA's all are equivalent.

• To show FA's equivalent to regex's we need to establish that

1. For every DFA $A$ we can construct (find) a regex $R$, s.t. $L(R) = L(A)$.

2. For every regex $R$ there is a $\varepsilon$-NFA $A$, s.t. $L(A) = L(R)$. 
Theorem

• For every DFA $A = (Q, \Sigma, \delta, q_0, F)$ there is a regex $R$, s.t. $L(R) = L(A)$.

Proof:
• Assume that A’s states are $\{1, 2, \ldots, n\}$ for some integer $n$.
• Our first, and most difficult, task is to construct a collection of regular expressions that progressively broader sets of paths in the transition diagrams of $A$.
• Let $R_{ij}^{(k)}$ be a regex whose language is the set of strings $w$ (such that $w$ is the label of path from state $i$ to state $j$) describing the set of labels of all paths in $A$ from state $i$ to state $j$ going through intermediate states $\{1, 2, \ldots, k\}$ only.
Path $R_{ij}^{(k)}$

To construct the expressions $R_{ij}^{(k)}$, we use the following inductive definition, starting at $k=0$ and finally reaching $k=n$. Notice that there are no states greater than $n$. 
Basis:

The basis is $k=0$.

- **Case $i = j$**

  $R_{ii}^{(0)} = \epsilon + a_1 + a_2 + \ldots$ where $a$'s are the self loop symbols

- **Case $i \neq j$**

  We must examine the DFA $A$ and find those input symbols $a$ such that there is a transition from state $i$ to state $j$ on symbol $a$.
  1. If there is no such symbol $a$, then $R_{ij}^{(0)} = \emptyset$
  2. If there is exactly one symbol $a$, then $R_{ij}^{(0)} = a$
  3. If there are symbols, then $R_{ij}^{(0)} = a_1 + a_2 + \ldots + a_k$
Induction

Suppose there is a path from state $i$ to state $j$ that goes through no state higher than $k$.

- **Case 1:** The path does not go through state $k$ at all. Then the label of the path is $R_{ij}^{(k-1)}$.

- **Case 2:** The path goes through state $k$ at least once. Then we can break the path into several pieces.

\[
R_{ij}^{(k)} = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} (R_{kk}^{(k-1)})^* R_{kj}^{(k-1)}
\]

Eventually, we have $R_{ij}^{(n)}$ for all $i$ and $j$. 
The regular expression for the language of the automaton is then the sum (union) of all expressions $R_{1j}^{(n)}$ such that $1$ is the start state and $j$ is an accepting state.
Simple Example

Let’s find $R$ for $A$, where

$$L(A) = \{x0y: x \in \{1\}^* \text{ and } y \in \{0,1\}^*\}$$
Basic simplification rules

- \((\varepsilon + R)^* = R^*\)
- \(R + RS^* = R(\varepsilon + S)^* = RS^*\)
- \(\emptyset R = R\emptyset = \emptyset\) (Annihilation)
- \(\emptyset + R = R + \emptyset = R\) (Identity)
Example continues

\[
\begin{array}{c|c}
R_{11}^{(0)} & \epsilon + 1 \\
R_{12}^{(0)} & 0 \\
R_{21}^{(0)} & \emptyset \\
R_{22}^{(0)} & \epsilon + 0 + 1 \\
\end{array}
\]

\[
R_{i,j}^{(1)} = R_{i,j}^{(0)} + R_{i,1}^{(0)}(R_{11}^{(0)})^*R_{1,j}^{(0)}
\]

<table>
<thead>
<tr>
<th>\text{By direct substitution}</th>
<th>\text{Simplified}</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_{11}^{(1)})</td>
<td>(\epsilon + 1 + (\epsilon + 1)(\epsilon + 1)^*(\epsilon + 1))</td>
</tr>
<tr>
<td>(R_{12}^{(1)})</td>
<td>(0 + (\epsilon + 1)(\epsilon + 1)^*0)</td>
</tr>
<tr>
<td>(R_{21}^{(1)})</td>
<td>(\emptyset + \emptyset(\epsilon + 1)^*(\epsilon + 1))</td>
</tr>
<tr>
<td>(R_{22}^{(1)})</td>
<td>(\epsilon + 0 + 1 + \emptyset(\epsilon + 1)^*0)</td>
</tr>
</tbody>
</table>
Example continues

<table>
<thead>
<tr>
<th>Simplified</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^{(1)}_{11}$</td>
</tr>
<tr>
<td>$R^{(1)}_{12}$</td>
</tr>
<tr>
<td>$R^{(1)}_{21}$</td>
</tr>
<tr>
<td>$R^{(1)}_{22}$</td>
</tr>
</tbody>
</table>

$$R^{(2)}_{i,j} = R^{(1)}_{i,j} + R^{(1)}_{i2} (R^{(1)}_{22})^* R^{(1)}_{2j}$$

<table>
<thead>
<tr>
<th>By direct substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^{(2)}_{11}$</td>
</tr>
<tr>
<td>$R^{(2)}_{12}$</td>
</tr>
<tr>
<td>$R^{(2)}_{21}$</td>
</tr>
<tr>
<td>$R^{(2)}_{22}$</td>
</tr>
</tbody>
</table>
Example continues

<table>
<thead>
<tr>
<th>By direct substitution</th>
<th>Simplified</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_{11}^{(2)}$</td>
<td>$1^*$</td>
</tr>
<tr>
<td>$R_{12}^{(2)}$</td>
<td>$1^<em>0(0 + 1)^</em>$</td>
</tr>
<tr>
<td>$R_{21}^{(2)}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$R_{22}^{(2)}$</td>
<td>$(0 + 1)^*$</td>
</tr>
</tbody>
</table>

The final regex for $A$ is

$$R_{12}^{(2)} = 1^*0(0 + 1)^*$$
Example continues

- The final regular expression for $A$ is
  $$1^*0(0+1)^*$$
  $$L(A) = \{x0y: x \in \{1\}^* \text{ and } y \in \{0,1\}^*\}$$
Observations

• There are $n^3$ expressions $R_{ij}^{(n)}$
• Each inductive step grows the expression 4-fold
• $R_{ij}^{(n)}$ could have size $4n$
• For all $\{i, j\} \subseteq \{1, \ldots, n\}$, $R_{ij}^{(k)}$ uses $R_{kk}^{(k-1)}$ so we have to write $n^2$ times regex $R_{kk}^{(k-1)}$.
• We need more efficient technique.
State elimination technique

- Let’s label the arcs by with redex’s instead of symbols
- Now, let's eliminate state s.
Rules

1. For each accepting state $q$, apply the above reduction process to produce an equivalent automaton with regular expression labels on the arcs. Eliminate from the original automaton all states except $q_0$ and $q$.

2. If $q \neq q_0$, then we shall left with two state automaton like which corresponds to regex:

$$E_q = (R+SU^*T)^*SU^*$$
Rules

3. If the start state is also an accepting state, then with performing a state-elimination from the original, we must end up with the start state only. Corresponding regex is then

$$E_q = R^*$$

4. The desired regular expression is the sum (union) of all the expressions derived from the reduced automata for each accepting state.
Example
Let us consider the NFA that accepts all strings of 0’s and 1’s such that either the second or third position from the end has a 1.

Replace the labels with equivalent reqex’s.
• Let’s eliminate B first:

- Let's eliminate B first:

• Then eliminate C (which gives the regex of D accepting state only)

• Final D accepting regex is: \((0+1)^*1(0+1) (0+1)\)
• Or eliminate D (which gives the regex of C accepting state only)

• Final C accepting regex is: \((0+1)^*1(0+1)\)
• Automaton regex is:
  \[(0+1)^*1(0+1) (0+1) + (0+1)^*1(0+1)\]
Converting Regular Expressions to Automata

Theorem: For every regex $R$ we can construct an $\epsilon$-NFA $E$, s.t. $L(E) = L(R)$.

Proof: We show for a regex $R$ that $L(E)$ with

1. Exactly one accepting state.
2. No arcs into the initial state.
3. No arcs out of the accepting state.

The proof by structural induction on $R$. 
Proof - Basis

• Automata for (a) $\varepsilon$, (b) $\emptyset$ and (c) regex $a$. 

(a)

(b)

(c)
Proof - Induction

• The Expression $R+S$ for some smaller expressions $R$ and $S$.

• Thus the language of the automaton is $L(R) \cup L(S)$.

• The Expression $RS$ for some smaller expressions $R$ and $S$.

• Thus the language of the automaton is $L(R)L(S)$. 
Proof - Induction

• The expression is $R^*$. The automaton allows us to go either:
  1. Directly from the start state to the accepting state.
  2. From start state, through that automaton one or more times, to the accepting state.
Example: convert \((0+1)^*1(0+1)\) to Automaton
Example: convert \((0+1)^*1(0+1)\) to Automaton

Or just remove redundant \(\varepsilon\) transitions.
Regular expressions in Unix

• Try “man grep” command, which is fundamental. Grep stands for “Global (search for) Regular Expression and Print”.

• You can find pdf version “grep.pdf” at the given web site heraklit.physics.metu.edu.tr. Regular expressions in Unix start at page 4.
Algebraic Laws for Regular Expressions

• Associativity and Commutativity
  1. \( L + M = M + L \)  commutative law for Union.
  2. \( (L + M) + N = L + (M + N) = L + M + N \)  associative law for union.
  3. \( (LM)N = L(MN) = LMN \)  associative law for concatenation.
  4. \( LM = ML \)  commutative concatenation
     Example: \( L = 0, M = 1 \). It is clear that \( 01 \neq 10 \)
Identities and Annihilators

- $\emptyset L = L\emptyset = \emptyset$. $\emptyset$ is the annihilator for concatenation.
- $\emptyset + L = L + \emptyset = L$. $\emptyset$ is the identity for union.
- $\{\varepsilon\}L = L\{\varepsilon\} = L$. $\varepsilon$ is the identity for concatenation.
- $L(M + N) = LM + LN$   Left distributive law
- $(M + N)L = ML + NL$   Right distributive law

- Example:
  \[
  0 + 01^* = 0(\varepsilon + 1^*) \quad \text{since} \quad 0 \varepsilon = 0 \\
  = 01^* \quad \text{since} \quad \varepsilon \text{ is in } L(1^*)
  \]
Laws for Regular Expressions

- $L \cup L = L$  
  Idempotence law for union. $(R + R = R)$
- $(L^*)^* = L^*$
- $\emptyset^* = \varepsilon$  
  Closure of empty set contains only the empty string.
- $L^+ = LL^* = L^*L$
- $L^* = L^+ \cup \varepsilon$
- $L^? = \varepsilon + L$  
  This rule is the definition of the $?$ operator.
The test for a regular expression algebraic law

- For example $L((0+1)1) = L(01+11)$
- Also e.g. $L((00+101)11) = L(0011+10111)$.
- More generally

$$L((E +F)G) = L(EG+FG)$$

for any regex's E, F, and G.
The test for a regular expression algebraic law

- We want to test general identities, such as $E + F = F + E$, for any regex's $E$ and $F$.

**Method:**

1. “Freeze" $E$ to $a_1$, and $F$ to $a_2$
2. Test automatically if the frozen identity is true, e.g. if $L(a_1 + a_2) = L(a_2 + a_1)$

- Question: Does this always work?
- Answer: Yes, as long as the identities use only plus, dot, and star.

- Consider $(L + M)^* = (L^* M^*)^*$ in $\{0,1\}$.
- Replace: $(1 + 0)^* = (1^* 0^*)^*$ and check whether the law hold.
Example

• Consider \((L + M)^* = (L^*M^*)^*\) in arbitrary alphabet. Then it is enough to determine if \((a_1 + a_2)^*\) is equivalent to \((a_1^*a_2^*)^*\).

• To verify \(L^* = L^*L^*\) test if \(a^*\) is equivalent to \(a^*a^*\).

• Does \(L + ML = (L+M)L\) hold?