Algebraic Laws for languages

• \( L \cup M = M \cup L \).

Union is commutative.

• \((L \cup M) \cup N = L \cup (M \cup N)\).

Union is associative.

• \((LM)N = L(MN)\).

Concatenation is associative

Note: Concatenation is not commutative, \( i.e., \) there are \( L \) and \( M \) such that \( LM \neq ML \).
• $\emptyset \cup L = L \cup \emptyset = L$.

$\emptyset$ is identity for union.

• $\{\epsilon\}L = L\{\epsilon\} = L$.

$\{\epsilon\}$ is left and right identity for concatenation.

• $\emptyset L = L\emptyset = \emptyset$.

$\emptyset$ is left and right annihilator for concatenation.
• \( L(M \cup N) = LM \cup LN. \)

Concatenation is \textit{left distributive} over union.

• \( (M \cup N)L = ML \cup NL. \)

Concatenation is \textit{right distributive} over union.

• \( L \cup L = L. \)

Union is \textit{idempotent}.

• \( \emptyset^* = \{\epsilon\}, \ \{\epsilon\}^* = \{\epsilon\}. \)

• \( L^+ = LL^* = L^*L, \ L^* = L^+ \cup \{\epsilon\} \)
• $(L^*)^* = L^*$. Closure is idempotent

Proof:

$$w \in (L^*)^* \iff w \in \bigcup_{i=0}^{\infty} \left( \bigcup_{j=0}^{\infty} L^j \right)^i$$

$$\iff \exists k, m_1, \ldots, m_k \in \mathbb{N} : w = w_1 \ldots w_k \text{ with } w_1 \text{ in } L^{m_1}, \ldots, w_k \text{ in } L^{m_k}$$

$$\iff \exists p \in \mathbb{N} : w \in L^p \text{ where } p = m_1 + \ldots + m_k$$

$$\iff w \in \bigcup_{i=0}^{\infty} L^i$$

$$\iff w \in L^*$$

Claim. $(L \cup M)^* = (L^* M^*)^*$.

Proof. It is easy to see that $L \cup M$ is contained in $L^* M^*$, since $L$ is contained in $L^*$ which is contained in $L^* M^*$, and similarly $M$ is contained in $L^* M^*$. Thus, the LHS is contained in the RHS.

To see that the RHS is also contained in the LHS, take any $w$ in $(L^* M^*)^*$. Then, $w = w_1 w_2 \ldots w_n$, where each substring $w_i$ is an element of $L^* M^*$ and can thus be written as $x_{i1} \ldots x_{ik} y_{i1} \ldots y_{ih}$, where each sub-substring $x_{ij}$ is an element of $L$ and each $y_{ij}$ an element of $M$. Thus, $w$ is the concatenation of a sequence of strings, each of which is an element of $L \cup M$. Therefore, it is a string in $(L \cup M)^*$. 
The above language laws all concern regex operations and can also be written as, e.g., $L + M = M + L$ and $L(M+N) = LM + LN$.

**Algebraic Laws for regex’s**

Evidently e.g. $L((0 + 1)1) = L(01 + 11)$

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 $G$ or more generally, any languages $E$, $F$, and $G$.

- How do we verify that a general identity like above is true?

1. Prove it by hand.

2. Let the computer prove it.
In Chapter 4 we will learn how to test automatically if $E = F$, for any concrete regex’s $E$ and $F$.

We want to test general identities, such as $\mathcal{E} + \mathcal{F} = \mathcal{F} + \mathcal{E}$, for any regex’s $\mathcal{E}$ and $\mathcal{F}$.

Method:

1. “Freeze” $\mathcal{E}$ to $a_1$, and $\mathcal{F}$ 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.

Let’s denote a generalized regex, such as $(\mathcal{E} + \mathcal{F})\mathcal{E}$ by

$$E(\mathcal{E}, \mathcal{F})$$

Now we can for instance make the substitution $S = \{\mathcal{E}/0, \mathcal{F}/11\}$ to obtain

$$S(E(\mathcal{E}, \mathcal{F})) = (0 + 11)0$$
Theorem 3.13: Fix a “freezing” substitution

\[ \star = \{ \varepsilon_1/a_1, \varepsilon_2/a_2, \ldots, \varepsilon_m/a_m \} \].

Let \( E(\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_m) \) be a generalized regex. Then for any regex's \( E_1, E_2, \ldots, E_m \), or languages

\[ w \in L(E(E_1, E_2, \ldots, E_m)) \]

if and only if there are strings \( w_i \in L(E_{ji}) \), s.t.

\[ w = w_1 w_2 \cdots w_k \]

and

\[ a_{j_1} a_{j_2} \cdots a_{j_k} \in L(E(a_1, a_2, \ldots, a_m)) \]

Informally, to obtain \( w \), we can first pick \( a_{j_1} a_{j_2} \cdots a_{j_k} \) in \( L(E(a_1, a_2, \ldots, a_m)) \) and then substitute for each \( a_{ji} \) any string from \( L(E_{ji}) \).

For example, suppose \( E(\varepsilon_1, \varepsilon_2) = (\varepsilon_1 + \varepsilon_2)^* \). Then string \( w \) is in \( L((E_1 + E_2)^*) \) iff \( w = w_1 w_2 \cdots w_k \) such that \( a_{j_1} a_{j_2} \cdots a_{j_k} \) is in \( L((a_1 + a_2)^*) \) and \( w_i \) is in \( L(E_{ji}) \).
For example: Suppose the alphabet is \( \{1, 2\} \).
Let \( E(\mathcal{E}_1, \mathcal{E}_2) \) be \((\mathcal{E}_1 + \mathcal{E}_2)\mathcal{E}_1\), and let \( E_1 \) be 1, and \( E_2 \) be 2. Then

\[
w \in L(E(E_1, E_2)) = L((E_1 + E_2)E_1) = (\{1\} \cup \{2\})\{1\} = \{11, 21\}
\]

if and only if

\[
\exists w_1 \in L(E_{j_1}) \quad , \quad \exists w_2 \in L(E_{j_2}) \quad : \quad w = w_1 w_2
\]

and

\[
a_{j_1}a_{j_2} \in L(E(a_1, a_2))) = L((a_1+a_2)a_1) = \{a_1a_1, a_2a_1\}
\]

if and only if

\( j_1 = j_2 = 1 \), or \( j_1 = 2 \), and \( j_2 = 1 \)

Another example, suppose \( E_1 = 1^* \) and \( E_2 = 2^* \). Then

\[
L_0 = L((E_1 + E_2)E_1) = L((1^* + 2^*)1^*) = L(1^* + 2^*1^*).
\]

\[
L((a_1 + a_2)a_1) = \{a_1a_1 + a_2a_1\}.
\]

String \( w \) is in \( L_0 \) iff there exist \( w_1 \) in \( L(E_{j_1}) \) and \( w_2 \) in \( L(E_{j_2}) \) such that \( w = w_1 w_2 \) and \( a_{j_1}a_{j_2} \) is in \( \{a_1a_1 + a_2a_1\} \).
Proof of Theorem 3.13: We do a structural induction of $E$.

**Basis:** If $E = \epsilon$, the frozen expression is also $\epsilon$.

If $E = \emptyset$, the frozen expression is also $\emptyset$.

If $E = a$, the frozen expression is also $a$. Now

$w \in L(E(E_1))$ if and only if $w$ is in $L(E_1)$, since $L(E(a_1)) = \{a_1\}$. 

See page 120 of the textbook.
**Induction:**

**Case 1:** \( E = F + G \).

Then \( \mathtt{\star}(E) = \mathtt{\star}(F) + \mathtt{\star}(G) \), and
\[
L(\mathtt{\star}(E)) = L(\mathtt{\star}(F)) \cup L(\mathtt{\star}(G))
\]

Let \( F \)' and \( G \)' be regex's. Then \( w \in L( F + G' ) \) if and only if \( w \in L( F' ) \) or \( w \in L( G' ) \). Also, a string \( u \) is in \( E(a_1, ..., a_m) \) iff it is in \( F(a_1, ..., a_m) \) or in \( G(a_1, ..., a_m) \). See the book for the rest of the proof using the I.H.

**Case 2:** \( E = F . G \).

Then \( \mathtt{\star}(E) = \mathtt{\star}(F) . \mathtt{\star}(G) \), and
\[
L(\mathtt{\star}(E)) = L(\mathtt{\star}(F)).L(\mathtt{\star}(G))
\]

Let \( F' \) and \( G' \) be regex's. Then \( w \in L( F'.G' ) \) if and only if \( w = w_1 w_2 \), \( w_1 \in L( F' ) \) and \( w_2 \in L( G' ) \). Also, a string \( u \) is in \( E(a_1, ..., a_m) \) iff \( u = u_1 u_2 \) where \( u_1 \) is in \( F(a_1, ..., a_m) \) and \( u_2 \) is in \( G(a_1, ..., a_m) \). The rest is similar to the above case.

**Case 3:** \( E = F^* \).

Prove this case at home.
The test would not work if the operation intersection were included in the regular expressions. E.g. consider $E \land \mathcal{I} = \phi$.

**The test for regular expressions and languages**

Examples:

To prove $(\mathcal{L} + \mathcal{M})^* = (\mathcal{L}^* \mathcal{M}^*)^*$ it is enough to determine if $(a_1 + a_2)^*$ is equivalent to $(a_1^* a_2^*)^*$

To verify $\mathcal{L}^* = \mathcal{L}^* \mathcal{L}^*$ test if $a_1^*$ is equivalent to $a_1^* a_1^*$.

**Question:** Does $\mathcal{L} + \mathcal{M} \mathcal{L} = (\mathcal{L} + \mathcal{M}) \mathcal{L}$ hold?

To prove $(a_1 + a_2)^* = (a_1^* a_2^*)^*$, we first notice that, $L((a_1^* a_2^*)^*)$ is a subset of $L((a_1 + a_2)^*)$.

Since $L(a_1 + a_2)$ is a subset of $L(a_1^* a_2^*)$, $L((a_1 + a_2)^*)$ is a subset of $L((a_1^* a_2^*)^*)$.

**Does $a + ba = (a + b)a$ hold?**
**Theorem 3.14:** \( E(\mathcal{E}_1, \ldots, \mathcal{E}_m) = F(\mathcal{E}_1, \ldots, \mathcal{E}_m) \Leftrightarrow L(\diamondsuit(E)) = L(\diamondsuit(F)) \)

**Proof:**

*(Only if direction)* \( E(\mathcal{E}_1, \ldots, \mathcal{E}_m) = F(\mathcal{E}_1, \ldots, \mathcal{E}_m) \) means that \( L(E(E_1, \ldots, E_m)) = L(F(E_1, \ldots, E_m)) \) for any concrete regex's \( E_1, \ldots, E_m \). In particular then \( L(\diamondsuit(E)) = L(\diamondsuit(F)) \)

*(If direction)* Let \( E_1, \ldots, E_m \) be concrete regex's. Suppose \( L(\diamondsuit(E)) = L(\diamondsuit(F)) \). Then by Theorem 3.13,

\[
  w \in L(E(E_1, \ldots, E_m)) \Leftrightarrow \exists w_i \in L(E_i), w = w_{j_1} \cdots w_{j_m}, a_{j_1} \cdots a_{j_m} \in L(\diamondsuit(E)) \Leftrightarrow \exists w_i \in L(E_i), w = w_{j_1} \cdots w_{j_m}, a_{j_1} \cdots a_{j_m} \in L(\diamondsuit(F)) \Leftrightarrow w \in L(F(E_1, \ldots, E_m))
\]

See page 121 of the textbook.
Properties of Regular Languages

• *Pumping Lemma*. Every regular language satisfies the pumping lemma. If somebody presents you with fake regular language, use the pumping lemma to show a contradiction.

• *Closure properties*. Building automata from components through operations, e.g. given $L$ and $M$ we can build an automaton for $L \cap M$.

• *Decision properties*. Computational analysis of automata, e.g. are two automata equivalent.

• *Minimization techniques*. We can save money since we can build smaller machines.
Suppose $L_{01} = \{0^n1^n : n \geq 1\}$ were regular.

Then it would be recognized by some DFA $A$, with, say, $k$ states.

Let $A$ read $0^k$. On the way it will travel as follows:

<table>
<thead>
<tr>
<th>Input</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\epsilon)</td>
<td>$p_0$</td>
</tr>
<tr>
<td>0</td>
<td>$p_1$</td>
</tr>
<tr>
<td>00</td>
<td>$p_2$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$0^k$</td>
<td>$p_k$</td>
</tr>
</tbody>
</table>

\[ \Rightarrow \exists i < j : p_i = p_j \] Call this state $q$. 
Now you can fool $A$:

If $\hat{\delta}(q, 1^i) \in F$ the machine will foolishly accept $0^i 1^i$.

If $\hat{\delta}(q, 1^i) \notin F$ the machine will foolishly reject $0^i 1^i$.

Therefore $L_{01}$ cannot be regular.

- Let’s generalize the above reasoning.
Theorem 4.1.

The Pumping Lemma for Regular Languages.

Let $L$ be regular.

Then there exists $n$, such that for all $w \in L$:

1. $|y| \neq \varepsilon$
2. $|xy| \leq n$
3. For all $k \geq 0$, $xy^kz \in L$
Proof: Suppose $L$ is regular

Then $L$ is recognized by some DFA $A$ with, say, $n$ states.

Let $w = a_1a_2\ldots a_m \in L$, $m \geq n$.

Let $p_i = \delta(q_0, a_1a_2\ldots a_i)$.

$\Rightarrow \exists i < j : p_i = p_j, j \leq n$
Now \( w = xyz \), where

1. \( x = a_1a_2 \cdots a_i \)

2. \( y = a_{i+1}a_{i+2} \cdots a_j \)

3. \( z = a_{j+1}a_{j+2} \cdots a_m \)

Evidently \( xy^kz \in L \), for any \( k \geq 0 \). \( Q.E.D. \)
Example: Let $L_{eq}$ be the language of strings with equal number of zero’s and one’s.

Suppose $L_{eq}$ is regular. Then $w = 0^n 1^n \in L$. 

By the pumping lemma $w = xyz$, $|xy| \leq n$, $y \neq \epsilon$ and $xy^k z \in L_{eq}$.

$$w = \underbrace{000 \cdots 0}_{x} \underbrace{011 \cdots 1}_{y} \underbrace{011 \cdots 1}_{z}$$

In particular, $xz \in L_{eq}$, but $xz$ has fewer 0’s than 1’s.

$L = \{0^i 1^j \mid i > j\}$

Consider string $w = 0^{n+1} 1^n$. By the pumping lemma, we can partition $w$ as $w = xyz$ such that $|xy| \leq n$, $y \neq \epsilon$, and $xy^k z \in L$. But $xz = 0^{n+1 - |y|} 1^n$ is not in $L$. 

95
Suppose \( L_{pr} = \{1^p : p \text{ is prime} \} \) were regular.

Let \( n \) be given by the pumping lemma.

Choose a prime \( p \geq n + 2 \).

\[
\begin{align*}
\text{w} &= \underbrace{111 \cdots 1}_{x} \underbrace{y}_{1111 \cdots 11} \underbrace{11 \cdots 1}_{z} \\
|y| &= m
\end{align*}
\]

Now \( xy^{p-m}z \in L_{pr} \)

\[
|xy^{p-m}z| = |xz| + (p - m)|y| = p - m + (p - m)m = (1 + m)(p - m)
\]

which is not prime unless one of the factors is 1.

\[
\begin{align*}
\bullet \ y \neq \epsilon &\Rightarrow 1 + m > 1 \\
\bullet \ m = |y| &\leq |xy| \leq n, \ p \geq n + 2 \\
&\Rightarrow p - m \geq n + 2 - n = 2.
\end{align*}
\]