Closure Properties of Regular Languages

Let $L$ and $M$ be regular languages. Then the following languages are all regular:

- **Union**: $L \cup M$
- **Intersection**: $L \cap M$
- **Complement**: $\overline{N}$
- **Difference**: $L \setminus M$
- **Reversal**: $L^R = \{ w^R : w \in L \}$
- **Closure**: $L^*$.
- **Concatenation**: $L.M$
- **Homomorphism**:
  \[ h(L) = \{ h(w) : w \in L, h \text{ is a homom.} \} \]
- **Inverse homomorphism**:
  \[ h^{-1}(L) = \{ w \in \Sigma : h(w) \in L, h : \Sigma \rightarrow \Delta^* \text{ is a homom.} \} \]
**Theorem 4.4.** For any regular \( L \) and \( M \), \( L \cup M \) is regular.

**Proof.** Let \( L = L(E) \) and \( M = L(F') \). Then \( L(E + F') = L \cup M \) by definition.

**Theorem 4.5.** If \( L \) is a regular language over \( \Sigma \), then so is \( \overline{L} = \Sigma^* \setminus L \).

**Proof.** Let \( L \) be recognized by a DFA

\[
A = (Q, \Sigma, \delta, q_0, F).
\]

Let \( B = (Q, \Sigma, \delta, q_0, Q \setminus F') \). Now \( L(B) = \overline{L} \).
Example:

Let $L$ be recognized by the DFA below

Then $\overline{L}$ is recognized by

Question: What are the regex’s for $L$ and $\overline{L}$
**Theorem 4.8.** If $L$ and $M$ are regular, then so is $L \cap M$.

**Proof.** By DeMorgan’s law $L \cap M = \overline{L \cup M}$. We already that regular languages are closed under complement and union.

We shall also give a nice direct proof, the *Cartesian* construction from the e-commerce example.
Theorem 4.8. If $L$ and $M$ are regular, then so is $L \cap M$.

Proof. Let $L$ be the language of

$$A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$$

and $M$ be the language of

$$A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$$

We assume w.l.o.g. that both automata are deterministic.

We shall construct an automaton that simulates $A_L$ and $A_M$ in parallel, and accepts if and only if both $A_L$ and $A_M$ accept.
If $A_L$ goes from state $p$ to state $s$ on reading $a$, and $A_M$ goes from state $q$ to state $t$ on reading $a$, then $A_L \cap M$ will go from state $(p, q)$ to state $(s, t)$ on reading $a$. 

![Diagram showing the process](image-url)
Formally

\[ A_{L \cap M} = (Q_L \times Q_M, \Sigma, \delta_{L \cap M}, (q_L, q_M), F_L \times F_M), \]

where

\[ \delta_{L \cap M}((p, q), a) = (\delta_L(p, a), \delta_M(q, a)) \]

It will be shown in the tutorial by an induction on \(|w|\) that

\[ \tilde{\delta}_{L \cap M}((q_L, q_M), w) = (\tilde{\delta}_L(q_L, w), \tilde{\delta}_M(q_M, w)) \]

The claim then follows.

**Question:** Why?
Example: \((c) = (a) \times (b)\)

\[(a)
\begin{array}{c}
\text{Start} \\
p
\end{array} \\
\begin{array}{c}
1 \\
0 \\
\downarrow \\
0,1 \\
\end{array} \\
\begin{array}{c}
0 \\
\rightarrow \\
qu \\
o,1 \\
\end{array}
\]

\[(b)
\begin{array}{c}
\text{Start} \\
r
\end{array} \\
\begin{array}{c}
0 \\
1 \\
\downarrow \\
0,1 \\
\end{array} \\
\begin{array}{c}
1 \\
\rightarrow \\
s \\
o,1 \\
\end{array}
\]

\[(c)
\begin{array}{c}
\text{Start} \\
pr
\end{array} \\
\begin{array}{c}
1 \\
0 \\
\downarrow \\
0,1 \\
\end{array} \\
\begin{array}{c}
1 \\
\rightarrow \\
ps \\
o,1 \\
\end{array}
\]

Another example?
Theorem 4.10. If $L$ and $M$ are regular languages, then so is $L \setminus M$.

Proof. Observe that $L \setminus M = L \cap \overline{M}$. We already know that regular languages are closed under complement and intersection.
Theorem 4.11. If $L$ is a regular language, then so is $L^R$.

Proof 1: Let $L$ be recognized by an FA $A$. Turn $A$ into an FA for $L^R$, by

1. Reversing all arcs.

2. Make the old start state the new sole accepting state.

3. Create a new start state $p_0$, with $\delta(p_0, \epsilon) = F$ (the old accepting states).
**Theorem 4.11.** If $L$ is a regular language, then so is $L^R$.

**Proof 2:** Let $L$ be described by a regex $E$. We shall construct a regex $E^R$, such that $L(E^R) = (L(E))^R$.

We proceed by a structural induction on $E$.

**Basis:** If $E$ is $\epsilon$, $\emptyset$, or $a$, then $E^R = E$.

**Induction:**

1. $E = F + G$. Then $E^R = F^R + G^R$

2. $E = F.G$. Then $E^R = G^R.F^R$

3. $E = F^*$. Then $E^R = (F^R)^*$

We will show by structural induction on $E$ on blackboard in class that

$$L(E^R) = (L(E))^R$$
A homomorphism on $\Sigma$ is a function $h : \Sigma \rightarrow \Theta^*$, where $\Sigma$ and $\Theta$ are alphabets.

Let $w = a_1a_2\cdots a_n \in \Sigma^*$. Then

$$h(w) = h(a_1)h(a_2)\cdots h(a_n)$$

and

$$h(L) = \{h(w) : w \in L\}$$

Example: Let $h : \{0, 1\}^* \rightarrow \{a, b\}^*$ be defined by $h(0) = ab$, and $h(1) = \epsilon$. Now $h(0011) = abab$.

Example: $h(L(10^*1)) = L((ab)^*)$. 
Theorem 4.14: \( h(L) \) is regular, whenever \( L \) is.

Proof: E.g., \( h(0^*1+(0+1)^*0) = h(0)^*h(1)+(h(0)+h(1))^*h(0) \)

Let \( L = L(E) \) for a regex \( E \). We claim that \( L(h(E)) = h(L) \).

Basis: If \( E \) is \( \epsilon \) or \( \emptyset \). Then \( h(E) = E \), and \( L(h(E)) = L(E) = h(L(E)) \).

If \( E \) is \( a \), then \( L(E) = \{a\} \), \( L(h(E)) = L(h(a)) = \{h(a)\} = h(L(E)) \).

Induction:

Case 1: \( G = E + F \). Now \( L(h(E + F)) = L(h(E)\cup h(F)) = L(h(E)) \cup L(h(F)) = h(L(E)) \cup h(L(F)) = h(L(E) \cup L(F)) = h(L(E + F)) \).

Case 2: \( G = E.F \). Now \( L(h(E.F)) = L(h(E)) \cdot L(h(F)) = h(L(E)) \cdot h(L(F)) = h(L(E) \cdot L(F)) = h(L(E.F)) \)

Case 3: \( G = E^* \). Now \( L(h(E^*)) = L(h(E))^* = L(h(E))^* = h(L(E))^* = h(L(E)^*) \)
Inverse Homomorphism

Let \( h : \Sigma \rightarrow \Theta^* \) be a homom. Let \( L \subseteq \Theta^* \), and define

\[
h^{-1}(L) = \{ w \in \Sigma^* : h(w) \in L \}\]

(a)

(b)
Example: Let $h : \{a, b\} \rightarrow \{0, 1\}^*$ be defined by $h(a) = 01$, and $h(b) = 10$. If $L = L((00 + 1)^*)$, then $h^{-1}(L) = L((ba)^*)$.

Claim: $h(w) \in L$ if and only if $w = (ba)^n$

Proof: Let $w = (ba)^n$. Then $h(w) = (1001)^n \in L$.

Let $h(w) \in L$, and suppose $w \not\in L((ba)^*)$. There are four cases to consider.

1. $w$ begins with $a$. Then $h(w)$ begins with $01$ and $\not\in L((00 + 1)^*)$.

2. $w$ ends in $b$. Then $h(w)$ ends in $10$ and $\not\in L((00 + 1)^*)$.

3. $w = xaay$. Then $h(w) = z0101v$ and $\not\in L((00 + 1)^*)$.

4. $w = xbbby$. Then $h(w) = z1010v$ and $\not\in L((00 + 1)^*)$.
**Theorem 4.16:** Let $h : \Sigma \rightarrow \Theta^*$ be a homom., and $L \subseteq \Theta^*$ regular. Then $h^{-1}(L)$ is regular.

**Proof:** Let $L$ be the language of $A = (Q, \Theta, \delta, q_0, F)$. We define $B = (Q, \Sigma, \gamma, q_0, F)$, where

$$\gamma(q, a) = \hat{\delta}(q, h(a))$$

It will be shown by induction on $|w|$ in the tutorial that $\hat{\gamma}(q_0, w) = \hat{\delta}(q_0, h(w))$.
We consider the following:

1. Converting among representations for regular languages.

2. Is \( L = \emptyset \)?

3. Is \( w \in L \)?

4. Do two descriptions define the same language?
From NFA’s to DFA’s

Suppose the \( \epsilon \)-NFA has \( n \) states.

To compute \( \text{ECLOSE}(p) \) we follow at most \( n^2 \) arcs.

The DFA has \( 2^n \) states, for each state \( S \) and each \( a \in \Sigma \) we compute \( \delta_D(S, a) \) in \( n^3 \) steps. Grand total is \( O(n^3 2^n) \) steps.

If we compute \( \delta \) for reachable states only, we need to compute \( \delta_D(S, a) \) only \( s \) times, where \( s \) is the number of reachable states. Grand total is \( O(n^3 s) \) steps.
From DFA to NFA

All we need to do is to put set brackets around the states. Total $O(n)$ steps.

From FA to regex

We need to compute $n^3$ entries of size up to $4^n$. Total is $O(n^34^n)$.

The FA is allowed to be a NFA. If we first wanted to convert the NFA to a DFA, the total time would be doubly exponential

From regex to FA’s

We can build an expression tree for the regex in $n$ steps.

We can construct the automaton in $n$ steps.

Eliminating $\epsilon$-transitions takes $O(n^3)$ steps.

If you want a DFA, you might need an exponential number of steps.
Testing emptiness

$L(A) \neq \emptyset$ for FA $A$ if and only if a final state is reachable from the start state in $A$. Total $O(n^2)$ steps.

Alternatively, we can inspect a regex $E$ and tell if $L(E) = \emptyset$. We use the following method:

$E = F + G$. Now $L(E)$ is empty if and only if both $L(F)$ and $L(G)$ are empty.

$E = F \cdot G$. Now $L(E)$ is empty if and only if either $L(F)$ or $L(G)$ is empty.

$E = F^*$. Now $L(E)$ is never empty, since $\epsilon \in L(E)$.

$E = \epsilon$. Now $L(E)$ is not empty.

$E = a$. Now $L(E)$ is not empty.

$E = \emptyset$. Now $L(E)$ is empty.
Testing membership

To test \( w \in L(A) \) for DFA \( A \), simulate \( A \) on \( w \). If \( |w| = n \), this takes \( O(n) \) steps.

If \( A \) is an NFA and has \( s \) states, simulating \( A \) on \( w \) takes \( O(ns^2) \) steps.

If \( A \) is an \( \epsilon \)-NFA and has \( s \) states, simulating \( A \) on \( w \) takes \( O(ns^3) \) steps.

If \( L = L(E) \), for regex \( E \) of length \( s \), we first convert \( E \) to an \( \epsilon \)-NFA with \( 2s \) states. Then we simulate \( w \) on this machine, in \( O(ns^3) \) steps.

Does \( L((0+1)^*0(0+1)^31^*) \) contain 10101011 or 101011101?

**Finiteness:** How to decide if \( L(A) \) is finite for DFA \( A \)?
Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA, and $\{p, q\} \subseteq Q$. We define

$$p \equiv q \iff \forall w \in \Sigma^* : \hat{\delta}(p, w) \in F \text{ iff } \hat{\delta}(q, w) \in F$$

- If $p \equiv q$ we say that $p$ and $q$ are equivalent
- If $p \not\equiv q$ we say that $p$ and $q$ are distinguishable

IOW (in other words) $p$ and $q$ are distinguishable iff

$$\exists w : \hat{\delta}(p, w) \in F \text{ and } \hat{\delta}(q, w) \notin F, \text{ or vice versa}$$
Example:

\[ \hat{\delta}(C, \epsilon) \in F, \hat{\delta}(G, \epsilon) \notin F \Rightarrow C \neq G \]

\[ \hat{\delta}(A, 01) = C \in F, \hat{\delta}(G, 01) = E \notin F \Rightarrow A \neq G \]
What about $A$ and $E$?

\[ \hat{\delta}(A, \epsilon) = A \notin F, \hat{\delta}(E, \epsilon) = E \notin F \]
\[ \hat{\delta}(A, 1) = K = \hat{\delta}(E, 1) \]

Therefore $\hat{\delta}(A, 1x) = \hat{\delta}(E, 1x) = \hat{\delta}(K, x)$
\[ \hat{\delta}(A, 00) = G = \hat{\delta}(E, 00) \]
\[ \hat{\delta}(A, 01) = C = \hat{\delta}(E, 01) \]

Conclusion: $A \equiv E$. 
We can compute distinguishable pairs with the following inductive \textit{table filling algorithm}:

\textbf{Basis}: If $p \in F$ and $q \notin F$, then $p \not\equiv q$.

\textbf{Induction}: If $\exists a \in \Sigma : \delta(p, a) \not\equiv \delta(q, a)$, then $p \not\equiv q$.

Example: Applying the table filling algo to A:
**Theorem 4.20:** If $p$ and $q$ are not distinguished by the TF-algo, then $p \equiv q$.

**Proof:** Suppose to the contrary that that there is a bad pair $\{p, q\}$, s.t.

1. $\exists w : \hat{\delta}(p, w) \in F, \hat{\delta}(q, w) \notin F$, or vice versa.

2. The TF-algo does not distinguish between $p$ and $q$.

Let $w = a_1a_2\cdots a_n$ be the shortest string that identifies a bad pair $\{p, q\}$.

Now $w \neq \epsilon$ since otherwise the TF-algo would in the basis distinguish $p$ from $q$. Thus $n \geq 1$. 
Consider states \( r = \delta(p, a_1) \) and \( s = \delta(q, a_1) \). Now \( \{r, s\} \) cannot be a bad pair since \( \{r, s\} \) would be indentified by a string shorter than \( w \). Therefore, the TF-algo must have discovered that \( r \) and \( s \) are distinguishable.

But then the TF-algo would distinguish \( p \) from \( q \) in the inductive part.

Thus there are no bad pairs and the theorem is true.
Testing Equivalence of Regular Languages

Let $L$ and $M$ be reg langs (each given in some form).

To test if $L = M$

1. Convert both $L$ and $M$ to DFA’s.

2. Imagine a DFA that is the union of the two DFA’s (never mind there are two start states)

3. If TF-algo says that the two start states are distinguishable, then $L \neq M$, otherwise $L = M$. 
Example:

We can "see" that both DFA accept $L(\epsilon + (0 + 1)^*0)$. The result of the TF-algo is

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>E</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Therefore the two automata are equivalent.
Minimization of DFA’s

We can use the TF-algo to minimize a DFA by merging all equivalent states. IOW, replace each state \( p \) by \( p/\equiv \).

Example: The DFA on slide 119 has equivalence classes \( \{\{A, E\}, \{B, H\}, \{C\}, \{D, K\}, \{G\}\} \).

The “union” DFA on slide 125 has equivalence classes \( \{\{A, C, D\}, \{B, E\}\} \).

Note: In order for \( p/\equiv \) to be an equivalence class, the relation \( \equiv \) has to be an equivalence relation (reflexive, symmetric, and transitive).
Theorem 4.23: If $p \equiv q$ and $q \equiv r$, then $p \equiv r$.

Proof: Suppose to the contrary that $p \not\equiv r$. Then $\exists w$ such that $\hat{\delta}(p, w) \in F$ and $\hat{\delta}(r, w) \not\in F$, or vice versa.

OTH, $\hat{\delta}(q, w)$ is either accepting or not.

Case 1: $\hat{\delta}(q, w)$ is accepting. Then $q \not\equiv r$.

Case 2: $\hat{\delta}(q, w)$ is not accepting. Then $p \not\equiv q$.

The vice versa case is proved symmetrically.

Therefore it must be that $p \equiv r$. 
Assume A has no inaccessible states.

To minimize a DFA \( A = (Q, \Sigma, \delta, q_0, F) \) construct a DFA \( B = (Q/\equiv, \Sigma, \gamma, q_0/\equiv, F/\equiv) \), where

\[
\gamma(p/\equiv, a) = \delta(p, a)/\equiv
\]

In order for \( B \) to be well defined we have to show that

If \( p \equiv q \) then \( \delta(p, a) \equiv \delta(q, a) \)

If \( \delta(p, a) \not\equiv \delta(q, a) \), then the TF-algo would conclude \( p \not\equiv q \), so \( B \) is indeed well defined. Note also that \( F/\equiv \) contains all and only the accepting states of \( A \).
Example: We can minimize

to obtain
NOTE: We cannot apply the TF-algo to NFA’s.

For example, to minimize

we simply remove state $C$.

However, $A \not\equiv C$. 
Let $B$ be the minimized DFA obtained by applying the TF-algo to DFA $A$.

We already know that $L(A) = L(B)$.

What if there existed a DFA $C$, with $L(C) = L(B)$ and fewer states than $B$?

Then run the TF-algo on $B$ “union” $C$.

Since $L(B) = L(C)$ we have $q^B_0 \equiv q^C_0$.

Also, $\delta(q^B_0, a) \equiv \delta(q^C_0, a)$, for any $a$. 
Claim: For each state $p$ in $B$ there is at least one state $q$ in $C$, s.t. $p \equiv q$.

Proof of claim: There are no inaccessible states, so $p = \hat{\delta}(q_0^B, a_1a_2\cdots a_k)$, for some string $a_1a_2\cdots a_k$. Now $q = \hat{\delta}(q_0^C, a_1a_2\cdots a_k)$, and $p \equiv q$.

Since $C$ has fewer states than $B$, there must be two states $r$ and $s$ of $B$ such that $r \equiv t \equiv s$, for some state $t$ of $C$. But then $r \equiv s$ (why?) which is a contradiction, since $B$ was constructed by the TF-algo.