Chapter 7: Kleene’s Theorem

Regular expressions, Finite Automata, transition graphs are all the same!!
Chapter 7: Kleene’s Theorem

- Method of proof

Let $A, B, C$ be sets such that $A \subseteq B$, $B \subseteq C$, $C \subseteq A$. Then $A = B = C$.

Remark: Regular expressions, finite automata, and transition graphs each define a set of languages.
Kleene’s Theorem

Any language that can be defined by a regular expression, or finite automaton, or transition graph can be defined by all three methods.
Proof of Kleene’s theorem: It is enough to prove each of the lemmas below.

**Lemma 1**: Every language that can be defined by a finite automaton can also be defined by a transition graph.

**Lemma 2**: Every language that can be defined by a transition graph can also be defined by a regular expression.

**Lemma 3**: Every language that can be defined by a regular expression can also be defined by a finite automaton.

\[ \text{FA} \subseteq \text{TG} \subseteq \text{RE} \subseteq \text{FA} \]
Lemma 1: Every language that can be defined by a finite automaton can also be defined by a transition graph.

Proof: By definition, every finite automaton is a transition graph.

Lemma 2: Every language that can be defined by a transition graph can also be defined by a regular expression.

Proof: By constructive algorithm, that works
1. for every transition graph
2. in a finite number of steps
1st step: transform to a transition graph with a single start state.
Chapter 7: Kleene’s Theorem

2nd step: and a single final state

Result is an equivalent transition graph, of the form:
3a. Combine edges that have the same starting and ending state.
Chapter 7: Kleene’s Theorem

3b. Eliminate states one by one:
bypass and state elimination operation
bypass state 2,
paths through state 2: $1 \rightarrow 2 \rightarrow 4$, $1 \rightarrow 2 \rightarrow 5$, $3 \rightarrow 2 \rightarrow 4$, $3 \rightarrow 2 \rightarrow 5$
Chapter 7: Kleene’s Theorem

Dr. Nejib Zaguia  CSI3104-W11  12
Chapter 7: Kleene’s Theorem

Even with many paths through state 2, always an equivalent GTG.
Chapter 7: Kleene’s Theorem

Special cases

3 → 2 → 1, 1 → 2 → 1

r_1 r_2^* r_3

r_4 r_2^* r_3
Chapter 7: Kleene’s Theorem

3 \rightarrow 2 \rightarrow 1,
3 \rightarrow 2 \rightarrow 3,
1 \rightarrow 2 \rightarrow 1,
1 \rightarrow 2 \rightarrow 3
Chapter 7: Kleene’s Theorem

3a. Combine edges

\[ r_1 + r_2 + \ldots + r_{35} \]
Chapter 7: Kleene’s Theorem

Example 1

$$(aa+bb)(a+b)^*(aa+bb)$$
Chapter 7: Kleene’s Theorem

EXAMPLE2: EVEN-EVEN

\[(ab+ba)(aa+bb)^*(ab+ba)\]

\[(aa+bb)+(ab+ba)(aa+bb)^*(ab+ba)\]

\[[(aa+bb)+(ab+ba)(aa+bb)^*(ab+ba)]^*\]
Example 3
Chapter 7: Kleene’s Theorem

Transition Graph $\rightarrow$ Regular Expression

- Algorithm (and proof)
  1. Add (if necessary) a unique start state without incoming edges and a unique final state without outgoing edges.
  For each state that is not a start state or a final state, repeat steps 2 and 3.
  2. Do a bypass and state elimination operation.
  3. Combine edges that have the same starting and ending state.
  4. Combine the edges between the start state and the final state. The label on the only remaining edge is the regular expression result. If there is none, the regular expression is $\emptyset$. 
Chapter 7: Kleene’s Theorem

**Lemma 3:** Every language that can be defined by a regular expression can also be defined by a finite automaton.

**Proof:** By constructive algorithm starting from the recursive definition of regular expressions.
Remember: Given an alphabet $\Sigma$, the set of regular expressions is defined by the following rules.

1. For every letter in $\Sigma$, the letter written in bold is a regular expression. $\Lambda$ is a regular expression.
2. If $r_1$ and $r_2$ are regular expressions, so is $r_1 + r_2$.
3. If $r_1$ and $r_2$ are regular expressions, so is $r_1 r_2$.
4. If $r_1$ is a regular expression, so is $r_1^*$. 
5. Nothing else is a regular expression.
Chapter 7: Kleene’s Theorem

Build a finite automaton that accepts the language: \((a+b)^*(aa+bb)(a+b)^*\)

Rule:

1. The letter \(a\)  
2. The letter \(b\)  
3. The word \(aa\) (using 1)  
4. The word \(bb\) (using 2)  
5. The expression \(aa+bb\) (using 3 and 4)  
6. The expression \(a+b\) (using 1 and 2)  
7. The expression \((a+b)^*\) (using 6)  
8. The expression \((a+b)^*(aa+bb)\) (using 7 and 5)  
9. The expression \((a+b)^*(aa+bb)(a+b)^*\) (using 8 and 7)
Chapter 7: Kleene’s Theorem

Rule 1

```
The language Λ

Only the letter x
```

Dr. Nejib Zaguia

CSI3104-W11
Chapter 7: Kleene’s Theorem

Rule 2: \( r_1 + r_2 \), Example 1

All word containing \( aa \)

EVEN-EVEN

\[ \begin{array}{c|cc}
\text{a} & \text{b} \\
\hline
-x_1 & x_2 & x_1 \\
x_2 & x_3 & x_1 \\
x_3 & x_3 & x_3 \\
\end{array} \]

\[ \begin{array}{c|cc}
\text{a} & \text{b} \\
\hline
\pm y_1 & y_3 & y_2 \\
y_2 & y_4 & y_1 \\
y_3 & y_4 & y_1 \\
y_4 & y_2 & y_3 \\
\end{array} \]
Chapter 7: Kleene’s Theorem

Result: r1+r2
Example 2.

(words ending in a)

(words ending in b)

(\begin{array}{c|cc}
\text{a} & \text{b} \\
\hline
-x_1 & x_2 & x_1 \\
+x_2 & x_2 & x_1 \\
\end{array})

(\begin{array}{c|cc}
\text{a} & \text{b} \\
\hline
-y_1 & y_1 & y_2 \\
+y_2 & y_1 & y_2 \\
\end{array})
Chapter 7: Kleene’s Theorem

**Lemma 3:** Every language that can be defined by a regular expression can also be defined by a finite automaton.

**Proof:** By constructive algorithm starting from the recursive definition of regular expressions

1. There is an FA that accepts only the empty word (Λ) and an FA that accepts only a single letter.
2. If there is an FA that accepts the language defined by \( r_1 \) and an FA that accepts the language defined by \( r_2 \), then there is an FA that accepts the language \( r_1 + r_2 \).
3. If there is an FA that accepts the language defined by \( r_1 \) and an FA that accepts the language defined by \( r_2 \), then there is an FA that accepts the language defined by their concatenation \( r_1 r_2 \).
4. If there is an FA that accepts the language defined by \( r \) then there is an FA that accepts the language defined by \( r^* \).

Thus for every regular expression, we can construct a FA.
Chapter 7: Kleene’s Theorem

Algorithm 1: \( r_1 + r_2 \)

Input:
FA 1: alphabet: \( \Sigma \)   states: \( x_1, x_2, x_3, \ldots \)    start state: \( x_1 \)
FA 2: alphabet: \( \Sigma \)   states: \( y_1, y_2, y_3, \ldots \)    start state: \( y_1 \)

plus final states and transitions

The new FA:

alphabet: \( \Sigma \)   states: \( z_1, z_2, z_3, \ldots \)    start state: \( x_1 \) or \( y_1 \)

transitions: if \( z_i = x_j \) or \( y_k \) and \( x_j \rightarrow x_{\text{new}} \) and \( y_k \rightarrow y_{\text{new}} \) (for input \( p \))    then \( z_{\text{new}} = (x_{\text{new}} \text{ or } y_{\text{new}}) \) for input \( p \).

If \( x_{\text{new}} \text{ or } y_{\text{new}} \) is a final state, then \( z_{\text{new}} \) is a final state.
Chapter 7: Kleene’s Theorem

(words ending in a)

(words ending in b)

algorithm 1

(words ending in a)

(words ending in b)
Chapter 7: Kleene’s Theorem

- **Algorithm 2**: Put all pairs \((x_i \text{ or } x_j)\) in the transition table

```
<table>
<thead>
<tr>
<th>x_1 or y_1</th>
<th>x_1 or y_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>x_2 or y_1</td>
<td>x_2 or y_2</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>
```

Diagram:
```
\(x_1 \text{ or } y_1\)  \(x_1 \text{ or } y_2\)  \\
\(x_2 \text{ or } y_1\)  \(x_2 \text{ or } y_2\)  \\
```

 transitions:
- \(a\):
  - \(x_1 \text{ or } y_1\) to \(x_1 \text{ or } y_2\)
  - \(x_1 \text{ or } y_2\) to \(x_1 \text{ or } y_1\)

- \(b\):
  - \(x_2 \text{ or } y_1\) to \(x_2 \text{ or } y_2\)
  - \(x_2 \text{ or } y_2\) to \(x_2 \text{ or } y_1\)
Chapter 7: Kleene’s Theorem

Rule 3: $r_1r_2$, Example 1

- a,b

b

+ a,b

second letter is b

a

a,b

a,b

odd number of a’s
Chapter 7: Kleene’s Theorem
Chapter 7: Kleene’s Theorem

Example 2

\[ x_1 \rightarrow a \rightarrow x_2 \rightarrow x_3 \]

\[ y_1 \rightarrow b \rightarrow y_2 \rightarrow + \]

\[ z_1 \rightarrow a \rightarrow z_2 \rightarrow a \rightarrow z_3 \rightarrow b \rightarrow z_4 + \]

\( r_1 \): all with aa

\( r_2 \): words ending in b

\( r_1 r_2 \)
Chapter 7: Kleene’s Theorem

We start by creating the states $z_1 = x_1$ and $z_2 = x_2$

$$(z_2, a) = x_3 \text{ “we continue on AF1” OR}
\quad y_1 \text{ ”we move to AF2, since } x_3 \text{ is a final state in AF1”}$$

$$(z_2, a) = x_3 \text{ or } y_1 = z_3$$

$$(z_3, a) = x_3 \text{ or } y_1 = z_3$$
$$(z_3, b) = x_3 \text{ or } y_1 \text{ or } y_2 = z_4 +$$

$$(z_4, a) = x_3 \text{ or } y_1 = z_3$$
$$(z_4, b) = x_3 \text{ or } y_1 \text{ or } y_2 = z_4 +$$
Rule 3: Concatenation: Summary of the Algorithm

1. Add a state $z$ for every state of the first automaton that is possible to go through before arriving at a final state.

2. For each final state $x$, add a state $z = (x \text{ or } y_1)$, where $y_1$ is the start state of the second automaton.

3. Starting from the states added at step 2, add states:

   $z = \begin{cases} 
   x \text{ (state such that execution continues on 1}\text{st automaton)} \\
   \text{OR} \\
   y \text{ (state such that execution moves on 2}\text{nd automaton)} 
   \end{cases}$

4. Label every state that contains a final state from the second automaton as a final state.
Chapter 7: Kleene’s Theorem

Example 3

\[ x_1 + \xrightarrow{a} a \xrightarrow{b} x_2+ \xrightarrow{a} x_3 \]

\[ y_1 \xrightarrow{a,b} y_2 \xleftarrow{a,b} y_1 \xrightarrow{a} y_2 \xleftarrow{a} y_1 \]

- \( r_1 \): no double a
- \( r_2 \): odd number of letters
Chapter 7: Kleene’s Theorem

\[ r_1 r_2: \text{all words except } \Lambda \]
Example 4

\[ r_1: \text{words starting with } b \]

\[ r_2: \text{words ending in } b \]
Chapter 7: Kleene’s Theorem

\[ r_1 r_2 \]

\[ r_2 r_1 \]
Chapter 7: Kleene’s Theorem

Rule 4: $r^*$, Example 1

$$r: \quad a^* + aa*b$$

$r^*$: words without double $b$, and that do not start with $b$. 
Chapter 7: Kleene’s Theorem

\[ z_1 = x_1 \pm (z_1, a) = x_1 \text{ or } x_2 = z_2 + (z_1, b) = x_4 = z_3 \]
\[ (z_2, a) = x_1 \text{ or } x_2 = z_2 \]
\[ (z_3, a) = z_3 \]
\[ (z_4, a) = x_1 \text{ or } x_2 \text{ or } x_4 = z_5 + (z_4, b) = x_4 = z_3 \]
\[ (z_5, a) = x_1 \text{ or } x_2 \text{ or } x_4 = z_5 \]
\[ (z_5, b) = x_1 \text{ or } x_3 \text{ or } x_4 = z_4 \]
Example 2

Words ending in a

Problem with $\Lambda$

$\begin{align*}
   z_1 &= x_1 - \\
   (z_1, a) &= x_1 \text{ ou } x_2 = z_2 + \\
   (z_1, b) &= z_1 \\
   (z_2, a) &= x_1 \text{ ou } x_2 = z_2 \\
   (z_2, b) &= x_1 = z_1
\end{align*}$
Rule 4: Kleene Star: Algorithm

Given: an FA whose states are \{x_1, x_2, x_3, \ldots\}

- For every subset of states, create a state of the new FA. Remove any subset that contains a final state but not the start state.
- Make the transition table for all the new states.
- Add a \( + \) state. Connect it to the same states as the original start state was connected to using the same transitions.
- The final states must be those that contain at least one final state from the original FA.
Chapter 7: Kleene’s Theorem

Example 3

Words with an odd number of b’s.

Λ and words with at least one b.
Chapter 7: Kleene’s Theorem

- A **nondeterministic finite automaton (NFA)** is:
  1. a finite set of states, one of which is designated as the start state, and some (maybe none) of which are designated the final states
  2. an **alphabet** $\Sigma$ of input letters
  3. a finite set of **transitions** that show how to go to a new state, for some pairs of state and letters

Remark: Every finite automaton is a nondeterministic finite automaton. Every nondeterministic finite automaton is a transition graph.
Chapter 7: Kleene’s Theorem

2 Examples of Nondeterministic Finite Automata

\[(a+b)^*aaa(a+b)^*bbb(a+b)^*\]

\[bb+bbb\]
Chapter 7: Kleene’s Theorem

Theorem: Every language that can be defined by a nondeterministic finite automaton can also be defined by a deterministic finite automaton.

Proof (1): Every nondeterministic finite automaton is a transition graph.
By lemma 2: transition graph $\rightarrow$ regular expression
By lemma 3: regular expression $\rightarrow$ finite automaton

Proof (2): By constructive algorithm (See Manuel page 135)
Chapter 7: Kleene’s Theorem

Algorithm: nondeterministic automaton $\rightarrow$ deterministic automaton (FA)
Given: a nondeterministic automaton whose states are

\[ \{x_1, x_2, x_3, \ldots\} \]

1. For every subset of states, create a state of the new FA.
2. Make the transition table for all the new states (or just the new states that can be entered).
3. Add a state $\emptyset$. Add transitions that loop back to itself for all letters of the alphabet. For each new state, if there is no transition for letter $p$, add one that goes to the $\emptyset$ state.
4. The final states must be those that contain at least one final state from the original nondeterministic finite automaton.
Chapter 7: Kleene’s Theorem

\[
bb+bbb
\]
Chapter 7: Kleene’s Theorem

\[(x_1, a) = x_1\]
\[(x_1 \text{ or } x_2, a) = x_1\]
\[(x_2 \text{ or } x_3, a) = x_2 \text{ or } x_3\]

\[(x_1, b) = x_1 \text{ or } x_2\]
\[(x_1 \text{ or } x_2, b) = x_2 \text{ or } x_3\]
\[(x_2 \text{ or } x_3, b) = x_2 \text{ or } x_3\]
Lemma 3: Every language that can be defined by a regular expression can also be defined by a finite automaton.

Proof: By constructive algorithm starting from the recursive definition of regular expressions, we build a nondeterministic finite automaton. Then, by the most recent theorem, it is possible to then build a finite automaton.
Chapter 7: Kleene’s Theorem

Rule 1: \( \Lambda \) and the letters in \( \Sigma \)

\[
\begin{align*}
&\text{Rule 1: } \Lambda \text{ and the letters in } \Sigma \\
&\begin{array}{c}
\text{Rule 1: } \Lambda \text{ and the letters in } \Sigma \\
\end{array}
\end{align*}
\]
Chapter 7: Kleene’s Theorem

Rule 2: r1+r2
Chapter 7: Kleene’s Theorem

Rule 3: \( r_1r_2 \)

\[
\begin{align*}
\text{Rule 3: } & r_1r_2 \\
& w_1 \rightarrow w_2 \\
& x_1 \rightarrow x_2 \rightarrow x_3 \\
& w_1 \rightarrow w_2 \rightarrow x_1 \rightarrow x_2 \rightarrow x_3
\end{align*}
\]