Pre-lecture brain teaser

Prove at the following languages are regular:

• All strings that end in **1011**

• All strings that contain **101** or **010** as a substring.

• All strings that do **not** contain **111** as a substring.
Pre-lecture brain teaser

Prove at the following languages are regular:

• All strings that end in 1011

• All strings that contain 101 or 010 as a substring.

• All strings that do not contain 111 as a substring.
Theorem
Languages accepted by DFAs, NFAs, and regular expressions are the same.
Theorem
Languages accepted by DFAs, NFAs, and regular expressions are the same.

- DFAs are special cases of NFAs (easy)
- NFAs accept regular expressions (seen)
- DFAs accept languages accepted by NFAs (shortly)
- Regular expressions for languages accepted by DFAs (shown previously)
Theorem
Languages accepted by DFAs, NFAs, and regular expressions are the same.
Theorem
Languages accepted by DFAs, NFAs, and regular expressions are the same.
Regular Expression to NFA
Proving equivalence
Thompson’s algorithm

Given two NFAs $s$ and $t$:

$L = L_s \cdot L_t$

$L = L_s \cup L_t$

$L = (L_s)^*$
Let’s take a regular expression and convert it to a DFA.

**Example:** \((0 + 1)^*(101 + 010)(0 + 1)^*\)
Let's take a regular expression and convert it to a DFA.

**Example:** \((0 + 1)^* (101 + 010) (0 + 1)^*\)

Using the concatenation rule:
Find NFA for \((0 + 1)^*\)
Find NFA for \((0 + 1)^*\)
Regular expression to NFA example

Find NFA for \((0 + 1)^*\)
Find NFA for \((101 + 010)\)
Regular expression to NFA example

Find NFA for \((101 + 010)\)
Regular expression to NFA example

Find NFA for \((101 + 010)\)
Let’s take a regular expression and convert it to a NFA.

Example: \((0 + 1)^* (101 + 010)(0 + 1)^*\)
Let’s take a regular expression and convert it to a NFA. 

Example: \((0 + 1)^* (101 + 010)(0 + 1)^*\)

Using the concatenation rule:
Let’s take a regular expression and convert it to a NFA.

Example: \((0 + 1)^* (101 + 010)(0 + 1)^*\)

Using the concatenation rule:

What does Thompson’s algorithm mean?!
Equivalence of NFAs and DFAs
Another Way to look at NFAs

Is 010110 accepted?
Another Way to look at NFAs

Is 010110 accepted?
Another Way to look at NFAs

Is 010110 accepted?
Another Way to look at NFAs

Is 010110 accepted?
Another Way to look at NFAs

Is 010110 accepted?
Another Way to look at NFAs

Is 010110 accepted?

\[
\begin{align*}
\text{start} & \quad 0, 1 \\
q_0 & \quad 0, 1 \\
q_1 & \quad 0 \\
q_2 & \quad 1 \\
q_3 & \quad 0, 1 \\
\end{align*}
\]
Another Way to look at NFAs

Is 010110 accepted?
Another Way to look at NFAs

Is 010110 accepted?

![Diagram of NFA](image)

1. Start at state q0.
2. Read 0: Move to state q1.
3. Read 1: Move to state q2.
4. Read 0: Move to state q3.
5. Read 1: Move to state q3.
6. Read 0: Move to state q3.

Since we end in an accepted state (q3), the string 010110 is accepted by the NFA.
Conversion of NFA to DFA
Proving equivalence

- Regular expressions
- DFAs
- NFAs
- Thompson’s Alg.
Theorem
For every NFA \( N \) there is a DFA \( M \) such that \( L(M) = L(N) \).
DFAs are memoryless...

• **DFA** knows only its current state.
• The state is the memory.
• To design a **DFA**, answer the question: What minimal info needed to solve problem.
Simulating NFA

NFAs know many states at once on input 010110.
It is easy to state that the state of the automata is the states that it might be situated at.
The state of the NFA

It is easy to state that the state of the automata is the states that it might be situated at.

configuration: A set of states the automata might be in.
It is easy to state that the state of the automata is the states that it might be situated at.

configuration: A set of states the automata might be in.

Possible configurations: $\mathcal{P}(q) = \emptyset, \{q_0\}, \{q_0, q_1\}...$
The state of the NFA

It is easy to state that the state of the automata is the states that it might be situated at.

configuration: A set of states the automata might be in.

Possible configurations: $\mathcal{P}(q) = \emptyset, \{q_0\}, \{q_0, q_1\}...$

Big idea: Build a DFA on the configurations.
Example

If receives 0 : 

If receives 1 :
Example

If receives **0**:

If receives **1**:
Simulating an NFA by a DFA

- Think of a program with fixed memory that needs to simulate NFA $N$ on input $w$.
- What does it need to store after seeing a prefix $x$ of $w$?
- It needs to know at least $\delta^*(s, x)$, the set of states that $N$ could be in after reading $x$
- Is it sufficient?
Simulating an NFA by a DFA

- Think of a program with fixed memory that needs to simulate NFA $N$ on input $w$.
- What does it need to store after seeing a prefix $x$ of $w$?
- It needs to know at least $\delta^*(s, x)$, the set of states that $N$ could be in after reading $x$.
- Is it sufficient? Yes, if it can compute $\delta^*(s, xa)$ after seeing another symbol $a$ in the input.
- When should the program accept a string $w$? If $\delta^*(s, w) \cap A \neq \emptyset$.

**Key Observation:** DFA $M$ simulating $N$ should know current configuration of $N$.

State space of the DFA is $\mathcal{P}(Q)$.
DFA from NFA
Definition
A non-deterministic finite automata (NFA) \( N = (Q, \Sigma, \delta, s, A) \) is a five tuple where

- \( Q \) is a finite set whose elements are called states,
- \( \Sigma \) is a finite set called the input alphabet,
- \( \delta : Q \times \Sigma \cup \{\epsilon\} \rightarrow \mathcal{P}(Q) \) is the transition function (here \( \mathcal{P}(Q) \) is the power set of \( Q \)),
- \( s \in Q \) is the start state,
- \( A \subseteq Q \) is the set of accepting/final states.

\( \delta(q, a) \) for \( a \in \Sigma \cup \{\epsilon\} \) is a subset of \( Q \) — a set of states.
NFA $N = (Q, \Sigma, s, \delta, A)$. We create a DFA $D = (Q', \Sigma, \delta', s', A')$ as follows:

- $Q' =$
- $s' =$
- $A' =$
- $\delta'(X, a) =$
DFAs to Regular expressions
Proving equivalence

- Regular expressions
- DFAs
- NFAs
- Thompson's Alg.
- Subset Construction
State Removal method

If $q_1 = \delta(q_0, x)$ and $q_2 = \delta(q_1, y)$

then $q_2 = \delta(q_1, y) = \delta(\delta(q_0, x), y) = \delta(q_0, xy)$
State Removal method - Example

\[ q_0 \rightarrow q_1 \rightarrow q_2 \]

Transitions:
- From \( q_0 \) to \( q_1 \) on input 0
- From \( q_1 \) to \( q_2 \) on input 1
- From \( q_2 \) to \( q_0 \) on input 0

Start state: \( q_0 \)

Final state: \( q_2 \)
State Removal method - Example

\[
\begin{align*}
q_0 & \xrightarrow{0} q_1 \\
q_0 & \xrightarrow{1} q_2 \\
q_1 & \xrightarrow{0} q_0 \\
q_1 & \xrightarrow{1} q_2 \\
q_2 & \xrightarrow{0} q_1 \\
q_2 & \xrightarrow{1} q_0
\end{align*}
\]
State Removal method - Example
State Removal method - Example

01 + (1 + 00)(10)^* (0 + 11)
State Removal method - Example

$01 + (1 + 00)(10)^*(0 + 11)$

$(01 + (1 + 00)(10)^*(0 + 11))^*$
Transition functions are themselves algebraic expressions!

Demarcate states as variables.

Can rewrite \( q_1 = \delta(q_0, x) \) as \( q_1 = q_0x \)

Solve for accepting state.
Algebraic method - Example

\[ q_0 = \epsilon + q_1 1 + q_2 0 \]

\[ q_1 = q_0 0 \]

\[ q_2 = q_0 1 + q_3 (0 + 1) \]

\[ q_3 = q_1 0 + q_2 1 \]
\[ q_0 = \epsilon + q_1 1 + q_2 0 \]
\[ q_1 = q_0 0 \]
\[ q_2 = q_0 1 \]
\[ q_3 = q_1 0 + q_2 1 + q_3 (0 + 1) \]
Algebraic method - Example

\[ q_0 = \epsilon + q_1 1 + q_2 0 \]
\[ q_1 = q_0 0 \]
\[ q_2 = q_0 1 \]
\[ q_3 = q_1 0 + q_2 1 + q_3 (0 + 1) \]

Now we simple solve the system of equations for \( q_0 \):

\[ q_0 = \epsilon + q_1 1 + q_2 0 \]
\[ q_0 = \epsilon + q_0 01 + q_0 10 \]
\[ q_0 = \epsilon + q_0 (01 + 10) \]

**Theorem (Arden’s Theorem)**
\[ R = Q + RP = QP^* \]
Algebraic method - Example

- $q_0 = \epsilon + q_11 + q_20$
- $q_1 = q_00$
- $q_2 = q_01$
- $q_3 = q_10 + q_21 + q_3(0 + 1)$

Now we simply solve the system of equations for $q_0$:

- $q_0 = \epsilon + q_11 + q_20$
- $q_0 = \epsilon + q_001 + q_010$
- $q_0 = \epsilon + q_0(01 + 10)$
- $q_0 = \epsilon(01 + 10)^* = (01 + 10)^*$
Converting NFAs to Regular Expression
Proving equivalence

- Regular expressions
- NFAs
- DFAs
- Algebraic Method
- Thompson’s Alg.
- Subset Construction
Stage 0: Input
Stage 1: Normalizing

Diagram showing transitions and states with labels: A, B, C, init, a, b, and ε.
Stage 2: Remove state A
Stage 4: Redrawn without old edges
Stage 4: Removing B

\[ \text{init} \rightarrow B \rightarrow \text{AC} \]

\[ a + b \]

\[ ab^*a \]
Stage 5: Redraw

\[
\begin{align*}
\text{init} & \rightarrow ab^*a + b \\
\text{C} & \rightarrow \varepsilon \\
a + b & \rightarrow AC
\end{align*}
\]
Stage 6: Removing C

\[(ab^*a + b)(a + b)^* \epsilon\]
Stage 7: Redraw

\[(ab \cdot a + b)(a + b)^*\]
Thus, this automata is equivalent to the regular expression

\[(ab^*a + b)(a + b)^*\].
Regular expressions to DFAs
Proving equivalence

- Regular expressions
- NFAs
- DFAs
- State removal
- Thompson’s Alg.
- Algebraic Method
- Subset Construction
Difficulty going from RegEx’s to DFAs

Lemma

Many regular expressions cannot be easily converted to DFAs.
Difficulty going from RegEx’s to DFAs

Lemma
Many regular expressions cannot be easily converted to DFAs.

Consider $= \{ w \in \Sigma^* | w$ has a substring 010 or 101 $\}$
Difficulty going from RegEx’s to DFAs

Lemma
Many regular expressions cannot be easily converted to DFAs.

Consider $\equiv \{ w \in \Sigma^* | w \text{ has a substring } 010 \text{ or } 101 \}$

- Is possible using Brzozowski\(^1\) algorithm. Not needed for this course.
But here’s the idea anyway....

Draw the DFA for $\{w \in \Sigma^* | w \text{ has a substring } 010\}$. What does each state represent?
Brzozowski Method

Brings us to the Brzozowski derivative where \((u^{-1}S)\) of a set \(S\) of strings and a string \(u\) is the set of strings obtainable from a string in \(S\) by cutting of the prefixing \(u\).

Consider the language \(R = (ab + c)^*\)
Brzozowski Method

Brings us to the Brzozowski derivative where \((u^{-1}S)\) of a set \(S\) of strings and a string \(u\) is the set of strings obtainable from a string in \(S\) by cutting of the prefixing \(u\).

Consider the language \(R = (ab + c)^*\)

<table>
<thead>
<tr>
<th>(R)</th>
<th>(a^{-1}R)</th>
<th>(b^{-1}R)</th>
<th>(c^{-1}R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0 = \varepsilon^{-1}R = (ab + c)^*)</td>
<td>(b (ab + c)^*)</td>
<td>(\emptyset)</td>
<td>((ab + c)^*)</td>
</tr>
<tr>
<td>(q_1 = b (ab + c)^*)</td>
<td>(\emptyset)</td>
<td>((ab + c)^*)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(q_2 = \emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>
Brzozowski Method

Brings us to the Brzozowski derivative where \((u^{-1}S)\) of a set \(S\) of strings and a string \(u\) is the set of strings obtainable from a string in \(S\) by cutting off the prefixing \(u\).

Consider the language \(R = (ab + c)^*\)

<table>
<thead>
<tr>
<th>(R)</th>
<th>(a^{-1}R)</th>
<th>(b^{-1}R)</th>
<th>(c^{-1}R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0 = \varepsilon^{-1}R = (ab + c)^*)</td>
<td>(b (ab + c)^*)</td>
<td>(\emptyset)</td>
<td>((ab + c)^*)</td>
</tr>
<tr>
<td>(q_1 = b (ab + c)^*)</td>
<td>(\emptyset)</td>
<td>((ab + c)^*)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>(q_2 = \emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>
Difficulty going from RegEx’s to DFAs

Lemma
Many regular expressions cannot be easily converted to DFAs.

Consider \( = \{w \in \Sigma^* | w \text{ has a substring } 010 \text{ or } 010 \} \)

- Is possible using Brzozowski\(^2\) algorithm. Not needed for this course.
- Easier to just convert RegEx \(\rightarrow\) NFA \(\rightarrow\) DFA.
Conclusion
Proving equivalence

Regular expressions

\[ \text{DFAs} \rightarrow \text{NFA} \rightarrow \text{DFA} \]

State removal

Thompson’s Alg.

Algebraic Method

Subset Construction

NFAs

DFAs

But what about the expressions that aren’t regular?! See on Thursday.
Proving equivalence

But what about the expressions at aren’t regular?! See on Thursday