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 $1011$.

\[(0+1)^*1011\]

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

\[(0+1)^*101(0+1)^*010(0+1)^*11\]

• 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

- Regular expressions
- NFAs
- DFAs
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:
Regular expression to NFA example

Find NFA for $(0 + 1)^*$
Regular expression to NFA example

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

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

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

Find NFA for \((101 + 010)\)
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:

\[q_i \rightarrow q_F \rightarrow q_i \rightarrow (010+101) \rightarrow q_F \rightarrow (0+1)^* \rightarrow q_F\]
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?

1

0
Another Way to look at NFAs

Is 010110 accepted?

1

0

0
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?

0

1

0

1

0

1

0
Conversion of NFA to DFA
Proving equivalence

- Regular expressions
- NFAs
- DFAs
- 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 $010110$. 
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.

![Diagram of an NFA]

Big idea: Build a DFA on the configurations.
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.
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: \( \mathcal{P}(q) = \emptyset, \{q_0\}, \{q_0, q_1\}, \ldots \)
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: $\mathcal{P}(q) = \emptyset, \{q_0\}, \{q_0, q_1\}, \ldots$

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 $\mathcal{P}(Q)$. 
Formal Tuple Notation for 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,
- $\delta : Q \times \Sigma \cup \{\epsilon\} \to \mathcal{P}(Q)$ 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 $M = (Q', \Sigma, \delta', s', A')$ as follows:

- $Q' = 2^Q$
- $s' = \text{Ereach}(s)$
- $A' = \{ X \subseteq Q | X \cap A \neq \emptyset \}$
- $\delta'(X, a) = \bigcup_{q \in X} \delta^*(q, a)$ for every $X \subseteq Q$ and $a \in \Sigma$
DFAs to Regular expressions
Proving equivalence

- Regular expressions
- NFAs
- DFAs
- 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
State Removal method - Example

Diagrams showing state transitions with labels for each transition.
State Removal method - Example

```
q_0^{start} \quad 01 \quad 1+00 \quad 0+11 \quad 10

\begin{align*}
\text{start} \quad q_0 & \quad \rightarrow \quad q_2 \quad \leftarrow \\
\quad & 01 \quad 1+00 \quad 0+11 \quad 10
\end{align*}
```
State Removal method - Example

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

\[
01 + (1 + 00)(10)^*(0 + 11)
\]

\[
(01^* + (1 + 00)(10)^*(0 + 11))^*
\]
Algebraic method

Transition functions are themselves algebraic expressions!

Demarcate states as variables.

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

Solve for accepting state.
Algebraic method - Example

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

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

- $q_0 = \varepsilon + q_11 + q_20$
- $q_1 = q_00$
- $q_2 = q_01$
- $q_3 = q_10 + q_21 + q_3(0 + 1)$
Algebraic method - Example

\[
\begin{align*}
q_0 &= \epsilon + q_11 + q_20 \\
q_1 &= q_00 \\
q_2 &= q_01 \\
q_3 &= q_10 + q_21 + q_3(0 + 1)
\end{align*}
\]

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

\[
\begin{align*}
q_0 &= \epsilon + q_11 + q_20 \\
q_0 &= \epsilon + q_001 + q_010 \\
q_0 &= \epsilon + q_0(01 + 10)
\end{align*}
\]

Theorem (Arden’s Theorem)

\[
R = Q + RP = QP^*
\]
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 simply solve the system of equations for $q_0$:

- $q_0 = \epsilon + q_1 1 + q_2 0$
- $q_0 = \epsilon + q_0 01 + q_0 10$
- $q_0 = \epsilon + q_0 (01 + 10)$
- $q_0 = \epsilon(01 + 10)^* = (01 + 10)^*$
Converting NFAs to Regular Expression
Proving equivalence

- Regular expressions
- NFAs
- DFAs

- Thompson’s Alg.
- Algebraic Method
- Subset Construction
Stage 0: Input

A \rightarrow C \quad b \quad C \rightarrow B \quad a \quad B \rightarrow C \quad b

A \rightarrow B \quad a \quad B \rightarrow C \quad a, b
Stage 1: Normalizing

A \rightarrow B

b \rightarrow a

A \rightarrow \epsilon

init

a \rightarrow b

AC

\epsilon

b

a + b
Stage 2: Remove state A
Stage 4: Redrawn without old edges
Stage 4: Removing B
Stage 5: Redraw

\[
\begin{align*}
\text{init} & \quad \text{ab}^*a + b \\
C & \quad \epsilon \\
\Rightarrow & \quad a + b
\end{align*}
\]
Stage 6: Removing C

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

\[ (ab^*a + b)(a + b)^* \]
Thus, this automata is equivalent to the regular expression $(ab^*a + b)(a + b)^*$. 
Regular expressions to DFAs
Proving equivalence

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

Lemma
Many regular expressions cannot be easily converted to DFAs.
**Lemma**

*Many regular expressions cannot be easily converted to DFAs.*

Consider $L = \{ w \in \Sigma^* | w$ has a substring 010 or 101 $\}$
Lemma
Many regular expressions cannot be easily converted to DFAs.

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

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

Draw the DFA for \( L = \{ w \in \Sigma^* | w \text{ has a substring } 010 \} \). 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>

```diagram```

- **Start**: \(q_0\)
- **States**: \(q_0, q_1, q_2\)
- **Transitions**:
  - \(q_0\) to \(q_1\) on \(a\)
  - \(q_0\) to \(q_1\) on \(b\)
  - \(q_1\) to \(q_0\) on \(c\)
  - \(q_1\) to \(q_2\) on \(a, c\)
```
Lemma
Many regular expressions cannot be easily converted to DFAs.

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

- 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
- NFAs
- DFAs
- State removal
- Thompson’s Alg.
- Algebraic Method
- Subset Construction

But what about the expressions at aren't regular?!
Proving equivalence

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