Find the regular expressions for the following languages:

• 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.
CS/ECE-374: Lecture 5 - RegExp-DFA-NFA Equivalence

Lecturer: Nickvash Kani
Chat moderator: Samir Khan
February 09, 2021

University of Illinois at Urbana-Champaign
Find the regular expressions for the following languages:

- All strings that end in \texttt{1011}

- All strings that contain \texttt{101} or \texttt{010} as a substring.

- All strings that do not contain \texttt{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)
Thompson’s algorithm

Given two NFAs $s$ and $t$:

$L = L_s \cap L_t$

$L = L_s \cup L_t$

$L = (L_s)^*$
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 DFA for \((0 + 1)^*\)
Find DFA for \((0 + 1)^*\)
Find DFA for \((0 + 1)^*\)
Find DFA for \((101 + 010)\)
Find DFA for \((101 + 010)\)
Regular expression to DFA example

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

**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?

0

1
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?

- Start state: $q_0$
- Transitions:
  - $0$: From $q_0$ to $q_1$
  - $0$: From $q_1$ to $q_2$
  - $0$, $1$: From $q_2$ to $q_3$

Graph:

```
  start   0,1
    q0     1   q1
    0,1   0   q2
    0,1   ε   1
    0,1   ε   q3
```
The idea of the conversion of NFA to DFA
Theorem
For every NFA $N$ there is a DFA $M$ such that $L(M) = L(N)$. 
DFAAs 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

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

Possible configurations:
- $P(q_0) = \emptyset$
- $\{q_0\}$
- $\{q_0, q_1\}$

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

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

State space of the DFA is $P(Q)$.
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 $w$?
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
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,
- \( \delta : Q \times \Sigma \cup \{\epsilon\} \to P(Q) \) is the transition function (here \( P(Q) \) 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.
Algorithm for converting NFA to DFA
Extending the transition function to strings

**Definition**
For NFA $N = (Q, \Sigma, \delta, s, A)$ and $q \in Q$ the $\epsilon$-reach($q$) is the set of all states that $q$ can reach using only $\epsilon$-transitions.

**Definition**
Inductive definition of $\delta^*: Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$:

- if $w = \epsilon$, $\delta^*(q, w) = \epsilon$-reach($q$)
- if $w = a$ where $a \in \Sigma$:
  $$\delta^*(q, a) = \epsilon$-$\text{reach}\left( \bigcup_{p \in \epsilon$-$\text{reach}(q)} \delta(p, a) \right)$$
- if $w = ax$:
  $$\delta^*(q, w) = \epsilon$-$\text{reach}\left( \bigcup_{p \in \epsilon$-$\text{reach}(q)} \bigcup_{r \in \delta^*(p, a)} \delta^*(r, x) \right)$$
Formal definition of language accepted by $N$

**Definition**
A string $w$ is accepted by NFA $N$ if $\delta^*_N(s, w) \cap A \neq \emptyset$.

**Definition**
The language $L(N)$ accepted by a NFA $N = (Q, \Sigma, \delta, s, A)$ is

$$\{w \in \Sigma^* \mid \delta^*(s, w) \cap A \neq \emptyset\}.$$
NFA $N = (Q, \Sigma, s, \delta, A)$. We create a DFA $D = (Q', \Sigma, \delta', s', A')$ as follows:

- $Q' =$
- $s' =$
- $A' =$
- $\delta'(X, a) =$
Algorithm for converting NFA into regular expression
Stage 0: Input
Stage 1: Normalizing

A \xrightarrow{a} B
C \xrightarrow{a, b} A
init \xrightarrow{\epsilon} A \xrightarrow{a} B \xrightarrow{b} C \xrightarrow{\epsilon} AC
\epsilon \xrightarrow{a + b} A
\epsilon \xrightarrow{a} B
\epsilon \xrightarrow{b} C
\epsilon \xrightarrow{AC}
Stage 2: Remove state A
Stage 4: Redrawn without old edges
Stage 4: Removing B

\[ \text{init} \rightarrow \text{a} \rightarrow \text{B} \rightarrow \text{b} \]

\[ \text{C} \rightarrow \epsilon \rightarrow \text{AC} \]

\[ a + b \]

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

\[
\begin{array}{c}
\text{init} \\
\hline
ab^*a + b \\
C \\
\epsilon \\
\text{AC} \\
a + b
\end{array}
\]
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

\[(ab^*a + b)(a + b)^*\].