Introduction to Computability Theory

Lecture 2: Non-Deterministic Finite Automata

Prof. Amos Israeli

Roadmap for Lecture

In this lecture we:
- Present the three regular operations.
- Present Non-Deterministic Finite Automata.
- Prove that NFA-s and DFA-s are equivalent.
- Use NFA-s to show that when each of the regular operation is applied on regular languages it yields yet another regular language.

The Regular Operations

Let \( A \) and \( B \) be 2 regular languages above the same alphabet, \( \Sigma \). We define the 3 Regular Operations:

- **Union**: \( A \cup B = \{ x \mid x \in A \text{ or } x \in B \} \).
- **Concatenation**: \( A \circ B = \{ xy \mid x \in A \text{ and } y \in B \} \).
- **Star**: \( A^* = \{ x_1, x_2, \ldots, x_k \mid k \geq 0 \text{ and } x_k \in A \} \).

Elaboration

- **Union** is straightforward.
- **Concatenation** is the operation in which each word in \( A \) is concatenated with every word in \( B \).
- **Star** is a unary operation in which each word in \( A \) is concatenated with every other word in \( A \) and this happens any finite number of times.
The Regular Operations - Examples

\[ A = \{ \text{good, bad} \} \quad B = \{ \text{girl, boy} \} \]

- \( A \cup B = \{ \text{good, bad, girl, boy} \} \)
- \( A \circ B = \{ \text{goodgirl, goodboy, badgirl, badboy} \} \)
- \( A^* = \{ \varepsilon, \text{good, bad, goodgood, goodbad}, \ldots \} \)

Motivation for Nondeterminism

The Regular Operations give us a way to construct all regular languages systematically. In the previous lecture, we showed that the union operation preserves regularity:

Given two regular languages \( L_1 \) and \( L_2 \) and their recognizing automata, \( M_1 \) and \( M_2 \), we constructed an automaton that recognizes \( L_1 \cup L_2 \).

Motivation for Nondeterminism

This approach fails when trying to prove that concatenation and star preserve regularity. To overcome this problem we introduce nondeterminism.

Example of an NFA

NFA – Nondeterministic Finite Automaton

1. A state may have 0-many transitions labeled with the same symbol.
2. \( \varepsilon \) transitions are possible.
Computation of an NFA

• When several transitions with the same label exist, an input word may induce **several** paths.
• When 0 transition is possible a computation may get “stuck”.

Q: Which words are accepted and which are not?
A: If word \( w \) induces (at least) a single accepting computation, the automaton “chooses” this **accepting path** and \( w \) is accepted.

Why do we Care About NFAs?

• NFA-s and DFA-s are **equivalent** (Meaning: They recognize the same set of languages). In other words: Each NFA recognizing language \( L \) has an equivalent DFA recognizing \( L \).
  (Note: This statement must be proved.)
• But usually, the NFA is much simpler.
• Enables the proof of theorems. (e.g. about regular operations)

Example

• An automaton over unary alphabet accepting words whose length is divided either by 2 or by 3.
What is the language of this DFA?

Bit Strings that have 1 in position third from the end

Can you verify it now?
DFA – A Formal Definition

A finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where:
1. \(Q\) is a finite set called the states.
2. \(\Sigma\) is a finite set called the alphabet.
3. \(\delta: Q \times \Sigma \rightarrow Q\) is the transition function.
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accepting states.

NFA – A Formal Definition

A finite automaton is a 5-tuple \((Q, \Sigma, \delta, q_0, F)\) where:
1. \(Q\) is a finite set called the states.
2. \(\Sigma\) is a finite set called the alphabet.
3. \(\delta: Q \times \Sigma^* \rightarrow P(Q)\) is the transition function.
4. \(q_0 \in Q\) is the start state, and
5. \(F \subseteq Q\) is the set of accept states.

Differences between NFA-s and DFA-s

There are two differences:
1. The range of the transition function \(\delta\) is now \(P(Q)\). (The set of subsets of the state set \(Q\))
2. The transition function allows \(\varepsilon\) transitions.

Possible Computations

At each step of the computation:
- DFA - A single state is occupied.
- NFA - Several states may be occupied.
Computations of NFA-s

In general a computation of an NFA, $N$, on input $w$, induces a **computation tree**.
Each path of the computation tree represents a possible computation of $N$.
The NFA $N$ accepts $w$, if its computation tree includes at least one path ending with an accepting state.

Equivalence Between DFAs and NFAs

Now we prove that the class of NFAs is **Equivalent** to the class of DFA:

**Theorem:** For every NFA $N$, there exists a DFA $M = M(N)$, such that $L(N) = L(M(N))$.

**Proof Idea:** The proof is **Constructive:** We assume that we know $N$, and construct a simulating DFA, $M$.

Proof

Let $N = (Q, \Sigma, \delta, q_0, F)$ recognizing some language $A$. the state set of the simulating DFA $M$, should reflect the fact that at each step of the computation, $N$ may occupy several states.
Thus we define the state set of $M$ as $P(Q)$ the **power-set** of the state set of $N$. 
Let \( N = (Q, \Sigma, \delta, q_0, F) \) recognizing some language \( A \). First we assume that \( N \) has no \( \varepsilon \) - transitions.

Define \( M = (Q', \Sigma, \delta', q_0', F) \) where \( Q' = P(Q) \).

Our next task is to define \( M \)'s transition function, \( \delta' \):

For any \( R \in Q' \) and \( a \in \Sigma \) define

\[
\delta'(R, a) = \{ q \in Q | q \in \delta(r, a) \text{ for some } r \in R \}
\]

If \( R \) is a state of \( M \), then it is a set of states of \( N \).

When \( M \) in state \( R \) processes an input symbol \( a \), \( M \) goes to the set of states to which \( N \) will go in any of the branches of its computation.

An alternative way to write the definition of \( M \)'s transition function, \( \delta' \) is:

For any \( R \in Q' \) and \( a \in \Sigma \) define

\[
\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)
\]

And the explanation is just the same.

**Note:** if \( \bigcup_{r \in R} \delta(r, a) = \phi \) than \( \delta'(R, a) = \phi \)

Which is OK since \( \phi \in P(Q) \).

The initial state of \( M \) is:

\[
q_0' = \{q_0\}
\]

Finally, the final state of \( M \) is:

\[
F' = \{R \in Q' | R \text{ contains a finite state of } N\}
\]

Since \( M \) accepts if \( N \) reaches **at least one** accepting state.

The reader can verify for her/him self that \( M \) indeed **simulates** \( N \).
Proof (cont.)

It remains to consider $\varepsilon$-transitions.

For any state $R$ of $M$ define $E(R)$ to be the collection of states of $R$ unified with the states reachable from $R$ by $\varepsilon$-transitions. The old definition of $\delta'(R,a)$ is:

$\delta'(R,a) = \{ q \in Q | q \in \delta(r,a) \text{ for some } r \in R \}$

And the new definition is:

$\delta'(R,a) = \{ q \in Q | q \in E(\delta(r,a)) \text{ for some } r \in R \}$

Proof (end)

In addition, we have to change the definition of $q_0'$, the initial state of $M$. The previous definition, $q_0' = \{q_0\}$, is replaced with $q_0' = E(\{q_0\})$.

Once again the reader can verify that the new definition of $M$ satisfies all requirements.

Corollary

A language $L$ is regular if and only if there exists an NFA recognizing $L$.

The Regular Operations (Rerun)

Let $A$ and $B$ be 2 regular languages above the same alphabet, $\Sigma$. We define the 3 Regular Operations:

- Union: $A \cup B = \{ x \mid x \in A \text{ or } x \in B \}$.
- Concatenation: $A \circ B = \{ xy \mid x \in A \text{ and } y \in B \}$.
- Star: $A^* = \{ x_1, x_2, \ldots, x_k \mid k \geq 0 \text{ and } x_k \in A \}$. 
The Regular Operations - Examples

\[ A = \{ \text{good, bad} \} \quad B = \{ \text{girl, boy} \} \]
- \( A \cup B = \{ \text{good, bad, girl, boy} \} \)
- \( A \circ B = \{ \text{goodgirl, goodboy, badgirl, badboy} \} \)
- \( A^* = \{ \varepsilon, \text{good, bad, goodgood, goodbad, goodgoodgoodbad, badbadgoodbad,...} \} \)

Theorem (rerun)

The class of Regular languages is **closed** under the all three regular operations.

Proof for union Using NFA-s

If \( A_1 \) and \( A_2 \) are regular, each has its own recognizing automaton \( N_1 \) and \( N_2 \), respectively.

In order to prove that the language \( A_1 \cup A_2 \) is regular we have to construct an FA that accepts exactly the words in \( A_1 \cup A_2 \).

A Pictorial proof

\[
\begin{align*}
& N_1 \quad q_1 \quad N_2 \quad q_2 \\
& f_1 \quad f_2 \quad f_3 \quad f_4 \quad f_5
\end{align*}
\]
**Proof for union Using NFA-s**

Let \( N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1) \) recognizing \( A_1 \), and \( N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2) \) recognizing \( A_2 \).

Construct \( N = (Q, \Sigma, \delta, q, F) \) to recognize \( A_1 \cup A_2 \),

Where \( Q = \{q_0\} \cup Q_1 \cup Q_2 \), \( F = F_1 \cup F_2 \),

\[
\delta(q,a) = \begin{cases} 
\delta_1(q,a) & q \in Q_1 \\
\delta_2(q,a) & q \in Q_2 \\
\{q_1,q_2\} & q = Q_i \text{ and } a = \varepsilon \\
\phi & q = Q_i \text{ and } a \neq \varepsilon 
\end{cases}
\]

**Theorem**

The class of Regular languages is **closed** under the \textit{concatenation} operation.

**Proof idea**

Given an input word to be checked whether it belongs to \( A_1 \circ A_2 \), we may want to run \( N_1 \) until it reaches an accepting state and then to move to \( N_2 \).

**Proof idea**

\textbf{The problem:} Whenever an accepting state is reached, we cannot be sure whether the word of \( A_1 \) is finished yet.

\textbf{The idea:} Use non-determinism to choose the right point in which the word of \( A_1 \) is finished and the word of \( A_2 \) starts.
**A Pictorial proof**

Let $N_1 = (Q_1, \Sigma, \delta, q_1, F_1)$ recognizing $A_1$, and

$N_2 = (Q_2, \Sigma, \delta, q_2, F_2)$ recognizing $A_2$.

Construct $N = (Q, \Sigma, \delta, q, F)$ to recognize $A_1 \circ A_2$,

Where $Q = Q_1 \cup Q_2$, $F = F_2$,

$$\delta(q,a) = \begin{cases} 
\delta_1(q,a) & q \in Q_1 \text{ and } q \notin F_1 \\
\delta_1(q,a) & q \in F_1 \text{ and } a \neq \varepsilon \\
\delta_1(q,a) \cup q_2 & q = F_1 \text{ and } a = \varepsilon \\
\delta_1(q,a) & q = Q_2
\end{cases}$$

**Proof using NFAs**

The class of Regular languages is **closed** under the **star** operation.
Proof using NFAs

Let $N_1 = (Q_1, \Sigma, \delta, q_1, F_1)$ recognizing $A_1$.

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1^*$

Where $Q = \{q_0\} \cup Q_1$, $F = \{q_0\} \cup F_1$, and

$$\delta(q,a) = \begin{cases} 
\delta_1(q,a) & q \in Q_1 \text{ and } q \notin F_1 \\
\delta_1(q,a) & q \in F_1 \text{ and } a = \varepsilon \\
\delta(q,a) \cup \{q_1\} & q \in F_1 \text{ and } a \neq \varepsilon \\
q_1 & q = q_0 \text{ and } a = \varepsilon \\
\phi & q = q_0 \text{ and } a \neq \varepsilon 
\end{cases}$$

Wrap Up

In this lecture we:

• Motivated and defined the three Regular Operations.
• Introduced NonDeterministic Finite Automatons.
• Proved equivalence between DFA-s and NFA-s.
• Proved that the regular operations preserve regularity.