LECTURE 27

12.1 \( \epsilon \) Transitions

In all automata that we have seen so far, every time that it has to change from one state to another, it must use one input symbol. An NFA-\( \epsilon \) is an NFA that allows transitions labeled \( \epsilon \) called \( \epsilon \)-transitions. These transitions are non-deterministic transitions in the sense it the machine might follow one (or several) of those transitions without reading (consuming) any input symbol.

Look carefully at the NFA-\( \epsilon \) N of figure 12.1. Is \( a \) in the language accepted by \( N \)? Is \( b \) in the language accepted by \( N \)? The following computation traces show that both \( a \), and \( b \) are in \( L(N) \).

0, \( a \vdash [1, a] \vdash [2, a] \vdash [3, a] \vdash [4, a] \vdash [5, \epsilon] \vdash [6, \epsilon] \vdash [7, \epsilon] \vdash [8, \epsilon] \vdash [10, \epsilon]

Since there is a computation such that \( [0, a] \not\vdash [10, \epsilon] \) where 10 is a final state, then \( a \in L(N) \). As we can see \( \epsilon \) transitions allow the machine to change state without having to consume any input symbols.

Definition 12.1 The \( \epsilon \)-closure of a state \( q \) is the set of all those states that can be reached from \( q \) using only \( \epsilon \) transitions. Formally,

\[
\epsilon\text{-closure}(q) = \{ r \in Q \mid [q, x] \not\vdash [r, x], \ x \in \Sigma^* \}
\]
The definition of \( \varepsilon \)-closure can be generalized to a set of states.

**Definition 12.2** If \( R \subseteq Q \) then

\[
\varepsilon\text{-closure}(R) = \bigcup_{q \in R} \varepsilon\text{-closure}(q)
\]

**Example.** Let us compute the \( \varepsilon \)-closure(0) in the NFA given above.

\[\varepsilon\text{-closure}(0) = \{0, 1, 2, 3, 4, 6\}\]

Every state is in the \( \varepsilon \)-closure of itself, since it is not necessary to consume any input symbol to remain in the same state.

### 12.2 Closure of Regular Languages and \( \varepsilon \)-closure

We were trying to show that regular languages are closed under concatenation when we decided to explore the idea of NFAs. Now it is time to put the ideas that we developed so far to good use and show that Regular Languages are closed under concatenation. Formally, what we need to show is that if \( L_1 \) and \( L_2 \) are regular languages then \( L_1 L_2 \) is regular too.

Since \( L_1 \) and \( L_2 \) are regular, then there are two DFAs \( M_1 \) and \( M_2 \) that accept the languages \( L_1 \) and \( L_2 \) respectively. We will show that there is an NFA-\( \varepsilon \) \( N \) that accepts \( L_1 L_2 \). We know that NFAs accept regular languages, since we were able to show that given an NFA it can be transformed into an equivalent DFA. We will show in the next section that an NFA-\( \varepsilon \) can also be transformed into an equivalent DFA. For now, assume that NFA-\( \varepsilon \) accept regular languages.

Assume that the following pictures represent \( M_1 \) and \( M_2 \), each having one start state and several accepting states.

To build an NFA-\( \varepsilon \) \( N \) that accepts \( L_1 L_2 \), we just have to add an \( \varepsilon \) transition from the accepting states of \( M_1 \) to the start state of \( M_2 \) as shown below.
Following a similar idea we can show that regular languages are closed under union and Kleene star too. In the case of the Union, we add one new start state and $\epsilon$ transitions from this new state to each one of the original start states of the machine.

In the case of the Kleene star operation we add one new start state and one new final state. We then add $\epsilon$ transitions from the new start state to the original start state, from the original final states to the new final state, from the original final states to the original start state, and from the new start state to the new final state. This last transition makes sure that $\epsilon$ is accepted, as it must, since $\epsilon \in L^\ast$. 
13 Regular Expressions

Regular expressions provide us with a simple and readable way to represent languages. Suppose that we want to represent the language obtained by the following operations:

1. Concatenate the elements of \( \{ab, ba\} \) to the elements of \( \{aa, bb\} \)
2. Apply the kleene star operator to the result
3. Concatenate the answer to the set obtained from the union of \( \{ab, ba\} \) and \( \{aa, bb\} \).

If we wanted to write it down using set notation and regular operations, we would need to write lots of \{ and \}. So, instead of writing:

\[
(\{ab, ba\}\{aa, bb\})^*(\{ab, ba\} \cup \{aa, bb\})
\]

We will write the regular expression: \((ab \cup ba)(aa \cup bb))^*((ab \cup ba) \cup (aa \cup bb))\).

**Definition 13.1** Assume that \(x\) and \(y\) are regular expressions that describe the languages \(X\) and \(Y\) respectively.

1. \(\emptyset\) is a regular expression that represents the language \(\emptyset\).
2. \(\epsilon\) is a regular expression that represents the language \(\{\epsilon\}\).
3. \(a\) is a regular expression that represents the language \(\{a\}\), where \(a \in \Sigma\).
4. \(x \cup y\) is a regular expression that represents the language \(X \cup Y\).
5. \(xy\) is a regular expression that represents the language \(XY\).
6. \(x^*\) is a regular expression that represents the language \(X^*\)

The type of definition given above is usually called a *recursive* definition, since the regular expressions are used to define regular expressions. However, the definition can be applied to generate longer regular expressions based on shorter ones, with the base cases being the \(\emptyset\) and \(a\). As a matter of notation, the regular expression \(xx^*\) is abbreviated as \(x^+\). This means that \(x\) appears at least once.

Operator precedence is similar to that of regular algebraic expressions, parentheses are computed first, then kleene star, then concatenation, and finally union. For example, the regular expression \(ab \cup c\), represents the language \(\{ab, c\}\), not \(\{ab, ac\}\).

**Example.** Some examples of regular expressions:
1. Language of those strings that have exactly one $b$
$$a^*ba^*$$

2. Language of those strings that have at least one $a$
$$(a \cup b)^*a(a \cup b)^*$$
   It can also be represented by the regular expression
   $$\Sigma^*a\Sigma^*$$

3. Language of those strings that have $aab$ as a substring
$$\Sigma^*aab\Sigma^*$$

4. Language of those strings that start and end with the same symbol
$$a\Sigma^*a \cup b\Sigma^*b \cup a \cup b$$

There are many regular expression identities. We will give only a few of them:

- $x\emptyset = \emptyset$
- $x \cup \emptyset = x$
- $x\epsilon = x$
- $\emptyset^* = \epsilon$

We must be very careful to distinguish between the empty set $\emptyset$, which has NO elements, and the regular expression $\epsilon$, which denotes a set that has exactly one element, which is the empty string, i.e. $\{\epsilon\}$.

**Example.** There are many applications where regular expressions are very useful. One of them is in the area of compilers, where we can describe tokens by using regular expressions.

Assume that the following regular expressions represent the given sets

- $d_0$ represents $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $d$ represents $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $x$ represents $\{a, \ldots, z, A, \ldots, Z, \_\}$
1. A regular expression for integers (assuming that they don’t begin with a 0) is
   \[ dd^* \]

2. A regular expression for floats (without scientific notation) is
   \[ dd_0.d^* \]

13.1 Regular Expressions and Regular Languages

In this section we will show that the languages represented by regular expressions are regular languages. To show this we will show the following theorem (in two parts):

**Theorem 13.2**
1. Given a regular expression \( x \) that represents language \( L \), there is an NFA-\( \epsilon \) \( M \) such that \( L(M) = L \).
2. Given an NFA-\( \epsilon \) \( M \), there is a regular expression \( x \) that represents \( L(M) \).

**Proof.** (Part 1) The proof is done by induction, using the definition of regular expressions. In each one of the cases show that there is an NFA-\( \epsilon \) that accepts the same language. Assume that \( M_x \) is an NFA-\( \epsilon \) that accepts the language represented by \( x \) and that \( M_y \) is an NFA-\( \epsilon \) that accepts the language represented by \( y \).

1. \( \emptyset \) is accepted by the following machine:

   ![Diagram for \( \emptyset \)]

2. \( \epsilon \) is accepted by the following machine:

   ![Diagram for \( \epsilon \)]

3. \( a \) is accepted by the following machine:

   ![Diagram for \( a \)]
4. \( x \cup y \) is accepted by the following machine:

![Diagram for \( x \cup y \)]

5. \( xy \) is accepted by the following machine:

![Diagram for \( xy \)]

6. \( x^* \) is accepted by the following machine:

![Diagram for \( x^* \)]