Chapter 5

Applications

5.1 Introduction

In the first part of this chapter we review the definition of fuzzy regular expression (fre) [2, 47] and design three algorithms namely LAMBDA, UFFSA, MINIUFFSA to find minimum state uffsa for a given fre.

Algorithm LAMBDA takes as input an fre and produces as output an ffsa with λ-transitions accepting the fuzzy language denoted by the fre. The input of the second algorithm UFFSA is the ffsa with λ-transitions, which is the output of LAMBDA and the output is an equivalent uffsa. Finally algorithm MINIUFFSA produces as output a minimum state uffsa for the input uffsa, which is the output of UFFSA. The final uffsa will accept the fuzzy language denoted by the given regular expression. Note that, these algorithms cannot be applied for arbitrary automaton. Systematically, we apply the three algorithms and obtain an uffsa in an example.

In lexical analysis, it is a common practice to construct a single automa-
5.2 Fuzzy Regular Expression

In order to distinguish strings belonging to different tokens, final states are marked with token names. When we design fuzzy lexical analyzers, for two different strings, we may have the same membership values (degree), and so there is likely to be a conflict to resolve. In order to avoid this eventuality, often, there is a linear order of priorities associated with the token names. Strings belonging to two or more tokens are marked with the name that has the highest priority among them. This idea appears to be especially useful when it is applied to fuzzy languages. Alexandru Mateescu et al. [2] formulated this with the definitions. Finally we propose a model to design fuzzy lexical analyzer with suitable algorithms. The algorithms constructed in this chapter can be implemented in computers.

5.2 Fuzzy Regular Expression

In this section we recall the definition of fuzzy regular expression, its language and give some examples. We refer to [2, 47].

Definition 5.2.1.

Let $\Sigma$ be a finite alphabet. The **fuzzy regular expression** (fre) over $\Sigma$ is defined recursively as follows:

(i) Let $e$ be a regular expression over $\Sigma$ and $m \in [0, 1]$, then $\frac{e}{m}$ is a fuzzy regular expression.

(ii) Let $\tilde{e}_1$ and $\tilde{e}_2$ be fuzzy regular expressions then $\tilde{e}_1 + \tilde{e}_2$, $(\tilde{e}_1)(\tilde{e}_2)$ and $(\tilde{e}_1)^*$ are all fuzzy regular expressions.
5.2 Fuzzy Regular Expression

(iii) A fuzzy regular expression is formed by applying (i) and (ii) a finite number of times.

Definition 5.2.2.
Let $\bar{e}$ be an fre, then the fuzzy regular language denoted by $\bar{e}$ is a fuzzy subset $\overline{L} : \Sigma^* \rightarrow [0,1]$ defined as follows:

1. If $\bar{e} = e$, where $e$ is a regular expression, then
   \[ \overline{L}(x) = \begin{cases} m, & \text{if } x \in L(e) \\ 0, & \text{otherwise} \end{cases} \]

2. Let $\bar{e}_1, \bar{e}_2$ be fre's with fuzzy regular languages $\overline{L}_1$ and $\overline{L}_2$ respectively, then
   (i) if $\bar{e} = \bar{e}_1 + \bar{e}_2$, then $\overline{L} = \overline{L}_1 \cup \overline{L}_2$.
   (ii) if $\bar{e} = \bar{e}_1 \bar{e}_2$, then $\overline{L} = \overline{L}_1 \overline{L}_2$ and
   (iii) if $\bar{e} = \bar{e}_1^*$, then $\overline{L} = \overline{L}_1^*$.

Definition 5.2.3.
An fre over $\Sigma$ is normalized if it is of the form $\frac{e_1}{m_1} + \frac{e_2}{m_2} + \cdots + \frac{e_n}{m_n}$, where $e_1, e_2, \ldots, e_n$ are regular expressions over $\Sigma$ and $m_1, m_2, \ldots, m_n$ are numbers in $[0,1]$, $n \geq 1$.

Note 5.2.4. If $m = 1$, then $\frac{e}{m}$ can simply be written as $e$.

Example 5.2.5.
The following are valid fre's over $\Sigma = \{a, b\}$. 
5.2 Fuzzy Regular Expression

(i) \( a^*ba^* + a^*b + (a+b)^* \)

(ii) \( \left( a^*ba \right) \left( ab^*a^* \right) + ab^* \)

(iii) \( ba^* + \left( \frac{a}{0.3} + \frac{ab}{0.6} \right)^* \)

(iv) \( aab + bab + \frac{(a+b)^*aa(a+b)^*}{0.3} \)

Both (i) and (iv) are normalized.

Definition 5.2.6.

An FREG is called a **strictly normalized FREG** if it is normalized, and

\[
\frac{e_1}{m_1} + \frac{e_2}{m_2} + \cdots + \frac{e_n}{m_n}, \text{ for any } m_i \neq m_j, \ L(e_i) \cap L(e_j) = \phi.
\]

Note 5.2.7. The families of languages represented by FREG’s, normalized FREG’s, and strictly normalized FREG’s respectively are equivalent.

Example 5.2.8.

The following FREG’s are equivalent.

(i) \( (b^*/1)(b/0.5) + (a+b)^*a(a+b)^*/0.5)(b/1) + ((a+b)^*aa(a+b)^*/0.8)(b/1) \).

(ii) \( b^*b/0.5 + (a+b)^*a(a+b)^*b/0.5 + (a+b)^*aa(a+b)^*b/0.8 \).

(iii) \( (a+\lambda)(b + ba)^*b/0.5 + (a + b)^*aa(a + b)^*b/0.8 \).

(ii) and (iii) are normalized and (iii) is strictly normalized.
5.3 Construction of uffsa from a given Fuzzy Regular Expression

In this section, we extend the model of uffsa to include fuzzy transitions on empty word $\lambda$.

Definition 5.3.1.
We define an ffsa with $\lambda$-transitions to be a quintuple $M = (Q, \Sigma, \mu, i, f)$ with all components as before, but $\mu : Q \times \Sigma \cup \{\lambda\} \times Q \rightarrow [0, 1]$ such that
\[
\mu(p, a, q) = \begin{cases} 
 m, m \geq 0, \forall p, q \in Q, a \in \Sigma \\
 1 \text{ or } 0 \forall p, q \in Q, a = \lambda.
\end{cases}
\]

5.3.1 Algorithm LAMBDA($\tilde{e}$)
This algorithm finds an ffsa $M$ with $\lambda$-transitions from a given fre $\tilde{e}$ accepting the fuzzy regular language $\overline{L}$ denoted by $\tilde{e}$.

Input: An fre $\tilde{e}$ over an alphabet $\Sigma$ denoting the fuzzy regular language $\overline{L}$.

Output: An ffsa $M = (Q, \Sigma \cup \{\lambda\}, \mu, i, f)$ such that the fuzzy regular language accepted by $M$ is $\overline{L}$.

Method: We first decompose $\tilde{e}$ into its primitive components. For each component, we construct an uffsa inductively as follows:

Case (i) $\tilde{e} = \frac{e}{m}$.
First, we find an nfa for the regular expression $e$. The method is explained in [1].
Let it be $A = (Q, \Sigma \cup \{\lambda\}, \delta, q_0, F)$ with $\lambda$-transitions such that
$L(A) = L(e) = L$. 

103
5.3 Construction of uffa from a given Fuzzy Regular Expression

Now, define ffsa $M$ with $\lambda$-transitions,

$M = (Q, \Sigma \cup \{\lambda\}, \mu, i, f)$, where $\mu : Q \times \Sigma \cup \{\lambda\} \times Q \to [0,1]$ is defined by

(i) $\forall p, q \in Q, a \in \Sigma, \mu(p, a, q) = \begin{cases} m, & \text{if } q \in \delta(p, a) \\ 0, & \text{otherwise}. \end{cases}$

(ii) $\forall p, q \in Q, \mu(p, \lambda, q) = \begin{cases} 1, & \text{if } q \in \delta(p, \lambda) \\ 0, & \text{otherwise}. \end{cases}$

$i : Q \to [0,1]$ is defined by $i(p) = \begin{cases} 1, & \text{if } p = q_0 \\ 0, & \text{otherwise}. \end{cases}$

$f : Q \to [0,1]$ is defined by $f(p) = \begin{cases} 1, & \text{if } p \in F \\ 0, & \text{otherwise}. \end{cases}$

Let $L_M$ be the fuzzy regular language accepted by $M$.

$\forall x \in \Sigma^*, L_M(x) = \bigvee \left\{ i(p) \land \mu^*(p, x, q) \land f(q) \mid q \in Q \right\}$

Now $\mu^*(p, x, q) = \begin{cases} m, & \text{if } x \in L(e) \text{ and } q \in F \\ 0, & \text{otherwise}. \end{cases}$

Therefore, $L_M(x) = \begin{cases} 1 \land m \land 1 = m, & \text{if } x \in L(e) \\ 0, & \text{otherwise}. \end{cases}$

Thus $L_M = \overline{L}$.

Note that, in $M$ there exists only one state with initial fuzzy value 1 and for all other states, which are 0. Similarly there is only one state with final fuzzy value 1 and for all other states, which are 0.

Having constructed ffsa for $\tilde{e}_1$ and $\tilde{e}_2$, we proceed to combine them in ways that correspond to the way compound fre's are formed.
5.3 Construction of uffsa from a given Fuzzy Regular Expression

Suppose $M_1$ and $M_2$ are the fsa’s with $\lambda$-transitions for $\bar{e}_1$ and $\bar{e}_2$ with fuzzy regular languages $\bar{L}_1$ and $\bar{L}_2$ respectively.

Let $M_1 = (Q_1, \Sigma \cup \{\lambda\}, \mu_1, i_1, f_1)$ and $M_2 = (Q_2, \Sigma \cup \{\lambda\}, \mu_2, i_2, f_2)$.

Suppose $i_1(p_1) = 1$, $f_1(q_1) = 1$ and $i_2(p_2) = 1$, $f_2(q_2) = 1$ where $p_1, q_1 \in Q_1$, $p_2, q_2 \in Q_2$ and zero for all other possibilities, $Q_1 \cap Q_2 = \phi$.

Case (ii): $\bar{e} = \bar{e}_1 + \bar{e}_2$

We now proceed to construct fsa $M$ for $\bar{e}$. Let $q_i$ and $q_f$ be two new states, $Q = Q_1 \cup Q_2 \cup \{q_i, q_f\}$.

Define the fsa $M = (Q, \Sigma, \mu, i, f)$, where $\mu : Q \times \Sigma \cup \{\lambda\} \times Q \rightarrow [0,1]$ is defined as follows:

(i) $\forall q \in Q_1$ with $i_1(q) = 1$, include $\mu(q_i, \lambda, q) = 1$.

(ii) $\forall q \in Q_2$ with $i_2(q) = 1$, include $\mu(q_i, \lambda, q) = 1$.

(iii) $\forall p, q \in Q_1$, $a \in \Sigma \cup \{\lambda\}$, include $\mu(p, a, q) = \mu_1(p, a, q)$.

(iv) $\forall p, q \in Q_2$, $a \in \Sigma \cup \{\lambda\}$, include $\mu(p, a, q) = \mu_2(p, a, q)$.

(v) $\forall p \in Q_1$ with $f_1(p) = 1$, include $\mu(p, \lambda, q_f) = 1$.

(vi) $\forall p \in Q_2$ with $f_2(p) = 1$, include $\mu(p, \lambda, q_f) = 1$.

(vii) $\mu(p, a, q) = 0$, for all other possibilities.

$i : Q \rightarrow [0,1]$ is defined by $i(p) = \begin{cases} 1, & \text{if } p = q_i \\ 0, & \text{otherwise} \end{cases}$

$f : Q \rightarrow [0,1]$ is defined by $f(p) = \begin{cases} 1, & \text{if } p = q_f \\ 0, & \text{otherwise} \end{cases}$
5.3 Construction of ufFsa from a given Fuzzy Regular Expression

Any path in $M$ from $q_i$ begins by going to either $p_1$ or $p_2$ on $\lambda$ with membership value 1 (others value 0). If the path goes to $p_1(p_2)$, it may follow any path in $M_1(M_2)$ to $q_1(q_2)$ and then go to $q_f$ on $\lambda$ with membership value 1 (others the value 0). These are the only paths from $q_i$ to $q_f$, with nonzero membership values, new transitions (at the beginning and at the end) are with membership value 1. For the degree of the strings, we take, the minimum values only, and for $q_i$ only, $i(q_i) = 1$, for $q_f$ only, $f(q_f) = 1$, therefore the degree will not be affected and

$$L_M(x) = \overline{L_1(x)} \lor \overline{L_2(x)} \forall x \in \Sigma^*.\text{ Clearly } L = \overline{L_1} \cup \overline{L_2}.$$  

Case (iii): $\tilde{e} = \tilde{e}_1\tilde{e}_2$

Let $Q = Q_1 \cup Q_2$. Define the ufFsa $M = (Q, \Sigma \cup \{\lambda\}, \mu, i, f)$ where $\mu : Q \times \Sigma \cup \{\lambda\} \times Q \rightarrow [0, 1]$ such that

(i) $\forall p \in Q_1$ with $f_1(p) = 1$ and $\forall q \in Q_2$ with $i_2(q) = 1$, include $\mu(p, \lambda, q) = 1$.

(ii) $\forall p, q \in Q_1, a \in \Sigma \cup \{\lambda\}$, include $\mu(p, a, q) = \mu_1(p, a, q)$.

(iii) $\forall p, q \in Q_2, a \in \Sigma \cup \{\lambda\}$, include $\mu(p, a, q) = \mu_2(p, a, q)$.

$i : Q \rightarrow [0, 1]$ is defined by $i(p) = \begin{cases} 1, & \text{if } i_1(p) = 1, p \in Q_1 \\ 0, & \text{otherwise} \end{cases}$.

$f : Q \rightarrow [0, 1]$ is defined by $f(p) = \begin{cases} 1, & \text{if } f_2(p) = 1, p \in Q_2 \\ 0, & \text{otherwise} \end{cases}$.

Let $x \in \Sigma^*$ be any string. Any path to read $x$ in $M$ from state $p(i(p) = 1)$ to state $r(f_2(r) = 1)$ is in $M_1$ from state $p$ to some state $q$ with $f_1(q) = 1$ by reading a substring $y \in \Sigma^*$, then the path goes to the state $p' \in Q_2$ by $\lambda$-
5.3 Construction of uffsa from a given Fuzzy Regular Expression

transition with membership value 1, therefore it simulates $M_2$ by reading the
suffix $z$ of $x$ (this is the only sequence creates nonzero membership values)
and $x = yz$ for some $y, z \in \Sigma^*$. Thus $L_M(x) = \overline{L_1(y)} \land \overline{L_2(z)}$.
Clearly $L_M(x) = \bigvee \{\overline{L_1(y)} \land \overline{L_2(z)} \mid x = yz, x, y \in \Sigma^*\}$.
Therefore $L_M = \overline{L_1 \land L_2}$.

Case (iv): $\bar{e} = (\bar{e}_1)^*$.

Let $Q = Q_1 \cup \{p_i, q_f\}$, where $p_i$ and $q_f$ are new states which are not in $Q_1$.

Define the ffsa $M = (Q, \Sigma \cup \{\lambda\}, \mu, i, f)$, where $\mu : Q \times \Sigma \cup \{\lambda\} \times Q \rightarrow [0, 1]$ is defined as follows:

(i) $\forall p \in Q_1$ with $i_1(p) = 1$, include $\mu(p_i, \lambda, p) = 1$.

(ii) $\mu(p_i, \lambda, q_f) = 1$.

(iii) $\forall q \in Q_1$ with $f_1(q) = 1$, include $\mu(q, \lambda, q_f) = 1$.

(iv) $\forall p, q \in Q_1$ with $f_1(q) = 1$ and $i_1(p) = 1$, include $\mu(q, \lambda, p) = 1$.

(v) $\forall p, q \in Q_1, a \in \Sigma \cup \{\lambda\}$, include $\mu(p, a, q) = \mu_1(p, a, q)$.

In this new ffsa $M$, for any input string $x \in \Sigma^*$, we can go from $p_i$ to $q_f$ directly on $\lambda$ with membership value 1, representing the fact that $\lambda$ is in
language denoted by $(\bar{e}_1)^*$, or, we can go through $M_1$ one or more times using
$\lambda$-transition (in turn let the string scanned be $x_1, x_2, \ldots$) from $q$ ($f_1(q) = 1$)
to $p(i_1(p) = 1)$ with membership value 1. Thus, there is a path in $M$ from
$p_i$ to $q_f$ with non-zero membership value for the sequence of moves labeled
the string $x \in \Sigma^*$, $L_M(x) > 0$ if and only if $x = x_1x_2 \cdots x_n$, $\overline{L_1(x_i)} \geq 0$,
5.3 Construction of ufssa from a given Fuzzy Regular Expression

\[ x_i \in \Sigma^*, i = 1, 2, \ldots, n. \]

Moreover, \( L_M(x) = \overline{L_1(x_1)} \land \overline{L_1(x_2)} \land \cdots \land \overline{L_1(x_n)}. \)

Clearly \( L_M(x) = \vee \{ \overline{L_1(x_1)} \land \overline{L_1(x_2)} \land \cdots \land \overline{L_1(x_n)} \mid x = x_1x_2 \cdots x_n, \]

\[ x_i \in \Sigma^*, i = 1, 2, \ldots, n. \}

Hence \( L_M = \overline{L_1}. \)

Example 5.3.2.

Consider the fre \( \bar{e} = \frac{ba^*}{0.7} + \left( \frac{a}{0.3} + \frac{ab}{0.6} \right)^*. \)

Systematically, if we follow the Algorithm LAMBDA(\( \bar{e} \)), we obtain the following sequence.

Let \( \bar{e} = \bar{e}_1 + \bar{e}_2, \) where \( \bar{e}_1 = \frac{ba^*}{0.7}, \bar{e}_2 = \left( \frac{a}{0.3} + \frac{ab}{0.6} \right)^*. \)

For \( \bar{e}_1 \) and \( \bar{e}_2 \) we have the ffss's \( M_1 \) and \( M_2 \) such that

\[ M_1 = (Q_1, \Sigma \cup \{\lambda\}, \mu_1, i_1, f_1), \]

where \( Q_1 = \{1, 2, 3, 4, 5\}, \Sigma = \{a, b\}, i_1(1) = 1, f_1(5) = 1 \) and \( \mu_1 \) is shown in the following fuzzy transition diagram.

\[ \lambda/1 \]

\[ 1 \quad 2 \quad 3 \quad 4 \quad 5 \]

\[ \lambda/1 \quad a/0.7 \quad \lambda/1 \quad \lambda/1 \]

\[ b/0.7 \]

\[ M_2 = (Q_2, \Sigma \cup \{\lambda\}, \mu_2, i_2, f_2), \]

where \( Q_2 = \{j \mid 6 \leq j \leq 14\}, \Sigma = \{a, b\}, i_2(6) = 1, f_2(14) = 1 \) and \( \mu_2 \) is shown in the following transition diagram.
5.3 Construction of uffsa from a given Fuzzy Regular Expression

Finally, for $\tilde{e} = \tilde{e}_1 + \tilde{e}_2$, we obtain the required ffsa $M = (Q, \Sigma \cup \{\lambda\}, \mu, i, f)$, where $Q = \{j \mid 1 \leq j \leq 15\}$, $\Sigma = \{a, b\}$, $i(15) = 1$, $f(16) = 1$ and $\mu$ is shown in the following transition diagram.

Clearly, the fuzzy regular language accepted by $M$ and the fuzzy regular language denoted by the fre $\tilde{e}$ are the same.
5.3 Construction of uffsa from a given Fuzzy Regular Expression

5.3.2 Algorithm UFFSA(M)

This algorithm constructs an uffsa $M'$ from the ffsa $M$ which is obtained using Algorithm \textsc{Lambda}($\bar{e}$).

**Input:** Let $M = (Q, \Sigma \cup \{\lambda\}, \mu, i, f)$ be the ffsa with $\lambda$-transitions, which is the output of Algorithm \textsc{Lambda}(\bar{e}). i.e., $M := \textsc{Lambda}(\bar{e})$. In ffsa $M$ there is only one state with non-zero initial fuzzy value, i.e., 1 and only one state with non-zero final fuzzy value, i.e., 1. Let $q_0, q_f \in Q$ such that $i(q_0) = 1$ and $f(q_f) = 1$.

**Output:** uffsa $M'$ with $L_{M'} = L_M$.

**Method:** For $s \in Q$, we define $s^\lambda = \{q \in Q \mid \mu^*(s, \lambda, q) = 1\}$, i.e., the set of all reachable states from $s$ by means of $\lambda$-transitions with membership value 1. Clearly, $s$ is a member of $s^\lambda$, $\forall s \in Q$.

If $T$ is a set of states, then $T^\lambda = \cup\{s^\lambda \mid s \in T\}$. We present the procedure in Figure 5.1 to obtain $T^\lambda$, using the data structure stack called \textsc{STACK}.

```
begin
push all states in $T$ into \textsc{STACK};
$T^\lambda := T$;
while \textsc{STACK} $\neq \phi$ do
begin
    pop $s$; the top element of \textsc{STACK};
    for each $t \in Q$ with $\mu(s, \lambda, t) = 1$ do
    begin
        if $t$ is not in $T^\lambda$ then
        begin
            add $t$ to $T^\lambda$;
            push $t$ onto \textsc{STACK};
        end
    end
end
```

Figure 5.1:
5.3 Construction of uffsa from a given Fuzzy Regular Expression

Now, we proceed to construct $M^1 = (Q^1, \Sigma, \mu^1, i^1, f^1)$. Each state of $M^1$ is a set of states of $M$. $Q^1$, the states of $M^1$ and their fuzzy transitions on input symbols are defined as follows:

**Step 1:** Let $s_0 = q_0^0$. We assume that $s_0 \in Q^1$ and is initially unmarked.

**Step 2:** Do the procedure given in Figure 5.2. Thus $Q^1$ and

$$\mu^1 : Q^1 \times \Sigma \times Q^1 \to [0, 1]$$

are defined.

```plaintext
begin
  while there is an unmarked state $s = \{s_1, s_2, \ldots, s_n\}$ do
    mark $s$;
    for each input symbol $a$ and for each membership value $m$
      do
        begin
          $T := \{ r_j | \mu(s_i, a, r_j) = m, s_i \in s \}$;
          $y := T^\lambda$;
          if $y \notin Q^1$ then include $y$ an unmarked state of $Q^1$;
          set $\mu^1(s, a, y) := m$;
        end
  end
end
```

Figure 5.2:

**Step 3:** Define $i^1 : Q^1 \to [0, 1]$ by

$$i^1(p) = \begin{cases} 
1, & \text{if } p = s_0 \\
0, & \text{otherwise.}
\end{cases}$$

and $f^1 : Q^1 \to [0, 1]$ by

$$f^1(p) = \begin{cases} 
1, & \text{if } q_f \in p \\
0, & \text{otherwise.}
\end{cases}$$

Clearly, $M^1$ is an uffsa without $\lambda$-transitions with $L_{M^1} = L_M$. 

111
5.3 Construction of uffsa from a given Fuzzy Regular Expression

Example 5.3.3.
Consider the ffsa $M = (Q, \Sigma \cup \{\lambda\}, \mu, \tau, f)$ which is obtained in Example 5.3.2:
$15^\lambda = \{15, 1, 6, 7, 8, 10, 14, 16\} = q_0$.

From $q_0$ on $b/0.7$, $T = \{2\}$, $T^\lambda = \{2, 3, 5, 16\} = q_1$, therefore

$$\mu^1(q_0, b, q_1) = 0.7, \text{ i.e., } \quad q_0 \xrightarrow{b/0.7} q_1$$

From $q_0$ on $a/0.3$, $T = \{9\}$, $T^\lambda = \{9, 13, 7, 8, 10, 14, 16\} = q_2$, therefore

$$\mu^1(q_0, a, q_2) = 0.3, \text{ i.e., } \quad q_0 \xrightarrow{a/0.3} q_2$$

From $q_0$ on $a/0.6$, $T = \{11\}$, $T^\lambda = \{11\} = q_3$, therefore

$$\mu^1(q_0, a, q_3) = 0.6, \text{ i.e., } \quad q_0 \xrightarrow{a/0.6} q_3$$

From $q_1$ on $a/0.7$, $T = \{4\}$, $T^\lambda = \{4, 3, 5, 16\} = q_4$, therefore

$$\mu^1(q_1, a, q_4) = 0.7, \text{ i.e., } \quad q_1 \xrightarrow{a/0.7} q_4$$

From $q_2$ on $a/0.3$, $T = \{9\}$, $T^\lambda = q_2$, therefore

$$\mu^1(q_2, a, q_2) = 0.3, \text{ i.e., } \quad q_2 \xrightarrow{a/0.3} q_2$$

From $q_2$ on $a/0.6$, $T = \{11\}$, $T^\lambda = q_3$, therefore

$$\mu^1(q_2, a, q_3) = 0.6, \text{ i.e., } \quad q_2 \xrightarrow{a/0.6} q_3$$

From $q_3$ on $b/0.6$, $T = \{12\}$, $T^\lambda = \{12, 13, 7, 8, 10, 14, 16\} = q_5$, therefore

$$\mu^1(q_3, b, q_5) = 0.6, \text{ i.e., } \quad q_3 \xrightarrow{b/0.6} q_5$$

From $q_4$ on $a/0.7$, $T = \{4\}$, $T^\lambda = q_4$, therefore
5.3 Construction of uffsa from a given Fuzzy Regular Expression

\[ \mu^1(q_4, a, q_5) = 0.7, \text{ i.e.,} \quad \begin{array}{c}
q_4 \\
\downarrow a/0.7
\end{array} \]

From \( q_5 \) on \( a/0.3 \), \( T = \{9\}, T^\lambda = q_2 \), therefore

\[ \mu^1(q_5, a, q_2) = 0.3, \text{ i.e.,} \quad \begin{array}{c}
q_5 \\
\downarrow a/0.3
\end{array} \quad q_2 \]

From \( q_5 \) on \( a/0.6 \), \( T = \{11\}, T^\lambda = q_3 \), therefore

\[ \mu^1(q_5, a, q_3) = 0.6, \text{ i.e.,} \quad \begin{array}{c}
q_5 \\
\downarrow a/0.6
\end{array} \quad q_3 \]

Now we have the uffsa, \( M^1 = (Q^1, \Sigma, \mu^1, i^1, f^1) \), where \( Q^1 = \{q_0, q_1, q_2, q_3, q_4, q_5\}, i^1(q_0) = 1, f^1(q_i) = 1, i = 0, 1, 2, 4, 5 \) and \( \mu^1 : Q^1 \times \Sigma \times Q^1 \to [0, 1] \) is shown below in the fuzzy transition diagram.

Figure 5.3:
5.3 Construction of uffsa from a given Fuzzy Regular Expression

5.3.3 Algorithm MINUFFSA($M^1$)

The number of states in uffsa $M^1$ which is constructed using Algorithm UFFSA($M$) is not the smallest possible. This algorithm gives a more general way of reducing the number of states of $M^1$ as few as possible.

**Input:** The uffsa $M^1 = \left( Q^1, \Sigma, \mu^1, i^1, f^1 \right)$ which is the output of Algorithm UFFSA($M$). i.e., $M^1 := $ UFFSA($M$).

**Output:** An uffsa $M^{11} = \left( Q^{11}, \Sigma, \mu^{11}, i^{11}, f^{11} \right)$ such that $L_{M^{11}} = L_{M^1}$ and having as few states as possible.

**Method:**

**Step 1.** We construct a partition $\Pi$ of the set states of $Q^1$. Initially, $\Pi$ consists of three groups, $Q^1_1, Q^1_2, Q^1_3$ such that

\[
Q^1_1 = \{ q \in Q^1 \mid i^1(q) = 1 \}, \quad Q^1_2 = \{ q \in Q^1 \mid f^1(q) = 1 \} - Q^1_1, \\
Q^1_3 = Q^1 - Q^1_1 \cup Q^1_2.
\]

Then we construct a new partition $\Pi_{\text{new}}$ using the procedure given in Figure 5.4 and followed by the procedure given in Figure 5.5.

**Step 2.** From the final partition $\Pi$, which is the output of the procedure given in Figure 5.5, pick a representative for each group. The representatives will be the states of the reduced uffsa $M^{11}$. Let $s$ be a representative state and $\mu^1(s, a, t) = m$ in $M^1$, then set $\mu^{11}(s, a, r) := m$, if $r$ is the representative of $t$'s group ($r$ may be $t$).
5.3 Construction of uffsa from a given Fuzzy Regular Expression

begin
  \(\Pi_{\text{new}} := \Pi\);
repeat
  begin
    \(\Pi := \Pi_{\text{new}}\);
    make all groups of \(\Pi\) are unmarked;
    while there is an unmarked group \(G\) in \(\Pi\) do
    begin
      mark \(G\);
      \(\text{new} := \emptyset\);
      for each input \(a \in \Sigma\) do begin
        if \((\mu^1(p, a, r) = m_1 \text{ and } \\
        \mu^1(p', a, r') = m_2), p, p' \in G, m_1 \neq m_2, \\
m_1, m_2 > 0 \text{ and } r, r' \text{ are in the same} \\
group \text{ of } \Pi \text{ (G}_1 \text{ may be } G)\) then
        \(\text{new} := \text{new} \cup \{r\}\);
        /* thus \(r\) and \(r'\) are placed in \\
different groups */.
      end
      place \(\text{new}\) as a group in \(\Pi_{\text{new}}\)
    end
  until \((\Pi = \Pi_{\text{new}})\);
end

Figure 5.4:
5.3 Construction of uffsa from a given Fuzzy Regular Expression

begin
  repeat
    \( \Pi := \Pi_{\text{new}}; \)
    for each group \( G \) of \( \Pi \) do
      begin
        partition \( G \) into subgroups such that two states 
        \( s \) and \( t \) of \( G \) are in the same group if and 
        only if for each input symbol \( a \) and each 
        \( m > 0 \) such that \( \mu^{11}(s, a, s') = \mu^{11}(t, a, t') = m \) 
        and \( s', t' \) are in the same group of \( \Pi \).
      end
    Place all subgroups so formed in \( \Pi_{\text{new}}; \)
  until \( (\Pi = \Pi_{\text{new}}) \)
end

Figure 5.5:

Step 3. We now formally define the initial fuzzy values \( i^{11} \) and final fuzzy 
values \( f^{11} \), using the procedure given in Figure 5.6.

Step 4. If \( M^{11} \) has a dead state \( d \) i.e., there exists no \( d' \in Q^{11} \) such that 
\( \mu^{11}(d, a, d') > 0 \) for each \( a \in \Sigma \) and \( f^{11}(d) = 0 \), remove \( d \) from \( Q^{11} \) and 
set \( \mu^{11}(p, a, d) := 0 \) \( \forall p \in Q^{11}, a \in \Sigma \).

Step 5. If \( s \in Q^{11} \) and for any string \( x \in \Sigma^* \), \( \mu^{11*}(q, x, s) = 0 \), 
for \( q \in Q^{11} \), \( i^{11}(q) = 1 \), then \( s \) is not reachable, remove \( s \) from \( Q^{11} \) and 
set \( \mu^{11}(p, a, s) := 0 \) \( \forall p \in Q^{11}, a \in \Sigma \).
5.3 Construction of ufFsa from a given Fuzzy Regular Expression

\begin{verbatim}
begin
  for each group G in \Pi do
    begin
      let s be the representative of group G;
      if i^1(r) = 1 for each r \in G then set i''(s) := 1;
      else set i^11(s) := 0;
      if f^1(r) = 1 for each r \in G then set f^11(s) := 1;
      else set f^11(s) := 0
      /* thus i^11 and f^11 are defined */
    end
end
\end{verbatim}

Figure 5.6:

Example 5.3.4.

Consider the ufFsa \( M^1 \) which is obtained in Example 5.3.3.

\[ \Pi = \{(q_0), (q_1, q_2, q_4, q_5), (q_3)\} \]

Using the procedure given in Figure 5.4, we get \( \Pi_{new} = \{q_2, q_5\} \) in the first interation.

\[ \Pi_{new} = \{(q_0), (q_3), (q_1, q_4), (q_2, q_5)\} \]. At the end of the procedure, \( \Pi = \Pi_{new} \).

By the procedure given in Figure 5.5, we get, \( \Pi = \{(q_0), (q_3), (q_1, q_4), (q_2, q_5)\} \).

At the end of the procedure, \( \Pi = \Pi_{new} \) for the above \( \Pi \). Let \( q_1 \) be the representative of group \( \{q_1, q_4\} \) and \( q_2 \) be the representative of group \( \{q_2, q_5\} \).

Finally, we have the reduced ufFsa \( M^{11} = (Q^{11}, \Sigma, \mu^{11}, i^{11}, f^{11}) \), where

\[ Q^{11} = \{q_0, q_1, q_2, q_3\}, \Sigma = \{a, b\}, i^{11}(q_0) = 1, f^{11}(q_i) = 1, i = 0, 1, 2 \text{ and } \mu^{11} : Q^{11} \times \Sigma \times Q^{11} \to [0,1] \text{ is shown in the following transition diagram.} \]
Clearly the fuzzy regular language accepted by $M^{11}$ is the fuzzy language denoted by the given fre in Example 5.3.2.

5.4 Fuzzy Lexical Analyzer

In this section, first we review lexical analyzer [1], then we recall marked fuzzy languages [2] and finally we propose two algorithms to design fuzzy lexical analyzer.

5.4.1 Lexical Analyzer

A compiler is a program that takes as input a program (source program) written in one of the high-level languages such as C, C++, FORTRAN, etc. and
5.4 Fuzzy Lexical Analyzer

produces as output a program (object program) in machine language.

Executing a user program written in a high level programming language is basically a two-step process, as illustrated in the following figures.

Source Program \( \rightarrow \) Compiler \( \rightarrow \) Object Program

Object program input \( \rightarrow \) Object program output

The source program must be compiled, that is translated into the object program, then the resulting object program is loaded into memory and executed. Conceptually, a compiler operates, in phases, each of which transforms the source program from one representation to another. The first phase of the compiler is the lexical analyzer (scanner), which is the interface between the source program and the compiler. The function of the lexical analyzer is to read the source program, one character at a time, and produces a stream of primitive units called tokens (single logical entity). Keywords, identifiers, constants, operators and delimiters are examples to tokens.

Lexical analyzer is a dfa which can be constructed in the following way, as found in [1]: Regular expressions are used to describe the tokens. One advantage of using regular expressions to specify tokens is that from a regular expression, we can construct an nfa with \( \lambda \)-moves and then obtain an equivalent dfa such that the language accepted by the dfa is the language denoted by the regular expression. For each token after getting the dfa, combine them and find a single dfa. Associate each final state of the dfa an action specifying the name of the token. Thus the lexical analyzer is a single dfa.
5.4 Fuzzy Lexical Analyzer

Read the source program one character at a time and move on to the states of the dfa according to the input symbols scanned, from the initial state. After reading all the input symbols of the input string $x$ (the end is identified by implementing the 'lookahead' feature), if the dfa halts at a final state, then it invokes the action associated with the final state and returns the token name; otherwise, the string $x$ is rejected by saying that $x$ is not a valid token.

5.4.2 Fuzzy Lexical Analyzer

In lexical analyzer, input strings are treated as crisp tokens. A string is either a token or non-token, there is no middle ground. We propose the fuzzy lexical analyzer to recognize strings with degrees in $[0,1]$. According to the degree of the input string, any of the following four actions may be taken:

(i) the string is accepted as a token.

(ii) with warning message, the string is accepted as a token.

(iii) an on-line question is given to user, the string is accepted if the user answers "yes"; rejected if the user answers "no" and

(iv) the string is rejected.

In the following example, we illustrate it for a single token named identifier. In general, identifier is a string beginning with a letter $a$ to $z$, followed by combination of letters ($a - z$) or digits ($0 - 9$) plus underscore symbol, here we fuzzify the identifier.
Example 5.4.1.

Consider the token “identifier” which is described by the following fre.

Let $\Sigma_i = \{a, b, \ldots, z\}$, $\Sigma_d = \{0, 1, 2, \ldots, 9\}$, $\Sigma_0 = \Sigma_i \cup \Sigma_d$.

\[
\Sigma_i \Sigma_0^* + \frac{\Sigma_i(+)\Sigma_0^*}{0.9} + \frac{\Sigma_d\Sigma_i\Sigma_0^*}{0.7} + \frac{\Sigma_d\Sigma_d\Sigma_0^*}{0.3}
\]

If we follow the three algorithms, \texttt{LAMBDA}, \texttt{UFFSA}, \texttt{MINUFFSA} systematically, we obtain the uffsa corresponding to the fre, which is shown below.

$M = (Q, \Sigma, \mu, i, f)$ with fuzzy regular language $L$, where

$Q = \{j \mid 1 \leq j \leq 8\}$, $\Sigma$ is the character set of the programming language,

$i : Q \rightarrow [0, 1]$ is defined by $i(j) =\begin{cases} 1, & \text{if } j = 1 \\ 0, & \text{otherwise.} \end{cases}$

$f : Q \rightarrow [0, 1]$ is defined by $f(j) =\begin{cases} 1, & \text{if } j = 2, 4, 6, 8 \\ 0, & \text{otherwise.} \end{cases}$

$\mu : Q \times \Sigma \times Q \rightarrow [0, 1]$ is shown in the following fuzzy transition diagram, where $l$ denotes any letter belonging to $\Sigma_i$ and $d$ denotes any digit belonging to $\Sigma_d$. 

121
5.4 Fuzzy Lexical Analyzer

We assume the following actions for different ranges of degrees:

\[ (0.9, 1] : \{ \text{accept} \} \]
\[ [0.8, 0.9] : \{ \text{warning; accept} \} \]
\[ [0.7, 0.8) : \{ \text{question} \} \]
\[ [0, 0.7) : \{ \text{reject} \} \]

We now illustrate the classification of strings with some examples, to be an identifier.

(i) Let \( x = \text{max} \), then \( L(x) = 1 \). Therefore \( \text{max} \) is accepted as the token identifier.

(ii) Let \( x = m(ax) \), then \( L(x) = 0.9 \). Therefore \( m(ax) \) is accepted as the token identifier with warning.

(iii) Let \( x = 1_y \), then \( L(x) = 0.7 \). Therefore \( 1_y \) is accepted if the user wants to accept it as an identifier, by answering “yes”; otherwise, \( 1_y \) will be
rejected as non-identifier.

(iv) Let $x = 123x$, then $L(x) = 0.3$. Therefore $123x$ is rejected.

When we consider all tokens of the programming language together with
the single uffsa, input string may have the same highest degree of different
tokens. Therefore there will be a conflict to assign the name of the token. To
overcome this problem, a linear order of priorities associated with the token
names and the marked fuzzy regular languages are introduced [2]. This idea
is formulated in the following definitions:

Definition 5.4.2.
Let $L$ be a fuzzy language, $T$ is a finite set of names with a linear order $<$,
and $\gamma : \Sigma^* \rightarrow T$ a function. Then we call the set $\{(x, L(x), \gamma(x)) | x \in \Sigma^*\}$
a marked fuzzy language, denoted by $(L, \gamma)$, where $\gamma$ is the marking
function of the language.

Definition 5.4.3.
Let $(L, \gamma)$ be a marked fuzzy language then $(L, \gamma)$ is called a marked fuzzy
regular language if $L$ is a fuzzy regular language. Note that if $L(x) = 0$
for some $x \in \Sigma^*$, then the value of $\gamma(x)$ is unimportant.

Definition 5.4.4.
Let $(L_1, \gamma_1)$ and $(L_2, \gamma_2)$ be two marked fuzzy regular languages over $\Sigma$. We
say that $(L_1, \gamma_1)$ and $(L_2, \gamma_2)$ are equivalent, denoted by $(L_1, \gamma_1) = (L_2, \gamma_2)$
if
(i) $L_1 = L_2$ and (ii) $\gamma_1(x) = \gamma_2(x) \forall x \in \Sigma^*$ such that $L_1(x) = L_2(x) > 0$. 

123
Definition 5.4.5.

Let \((L_1, \gamma_1)\) and \((L_2, \gamma_2)\) be two marked fuzzy languages over \(\Sigma\). We say that \(\gamma_1 \subseteq \gamma_2\) if

(i) \(L_1 \subseteq L_2\) and

(ii) \(\gamma_1(x) = \gamma_2(x) \quad \forall x \in \Sigma^*\) such that \(L_1(x) > 0\).

Definition 5.4.6.

Let \((L_1, \gamma_1)\) and \((L_2, \gamma_2)\) be two marked fuzzy regular languages. A marked fuzzy regular language \((L, \gamma)\) is called the marked union of \((L_1, \gamma_1)\) and \((L_2, \gamma_2)\) if \(L = L_1 \cup L_2\) and \(\gamma : \Sigma^* \to T\) is defined by

\[
\gamma(x) = \begin{cases} 
\gamma_1(x), & \text{if } (L_1(x) > L_2(x)) \text{ or } (L_1(x) = L_2(x) \text{ and } \gamma_2(x) < \gamma_1(x)) \\
\gamma_2(x), & \text{otherwise}
\end{cases}
\]

\(\forall x \in \Sigma^*\).

The marked union can be extended for any finite number of marked fuzzy regular languages.

We illustrate the concept of marked union of fuzzy regular languages in the following example by considering two tokens—integer and identifier.

Example 5.4.7.

Let \(T = \{\text{integer, identifier}\}\) with the linear order integer < identifier.

\(\Sigma_t = \{a, b, \ldots, z\}, \Sigma_d = \{0, 1, \ldots, 9\}, \Sigma = \Sigma_t \cup \Sigma_d\).

Consider the fuzzy regular expression \(\bar{e}_1 = \Sigma_t \Sigma^* + \frac{\Sigma_d \Sigma^* \Sigma_t \Sigma^*}{0.9}\), let \(L_1\) be the fuzzy regular language denoted by \(\bar{e}_1\).

\(\gamma_1 : \Sigma^* \to T\) is defined by \(\gamma_1(x) = \text{identifier} \quad \forall x \in \Sigma^*\). Therefore \((L_1, \gamma_1)\) is a marked fuzzy regular language.

Consider the fuzzy regular expression \(\bar{e}_2 = \Sigma_d \Sigma^*_d + \frac{\Sigma_d \Sigma^*_d \Sigma_t \Sigma^*_d}{0.9}\).
5.4 Fuzzy Lexical Analyzer

Let $L_2$ be the fuzzy regular language denoted by $\tilde{e}_2$.

$\gamma_2 : \Sigma^* \rightarrow T$ is defined by $\gamma_2(x) = \text{integer} \ \forall x \in \Sigma^*$.

Hence $(L_2, \gamma_2)$ is marked fuzzy regular language.

Let $(L, \gamma)$ be the marked union of $(L_1, \gamma_1)$ and $(L_2, \gamma_2)$.

(i) Let $x = 52 \ e \ 01$

$L_1(x) = 0.9, \ \gamma_1(x) = \text{identifier}$

$L_2(x) = 0.9, \ \gamma_2(x) = \text{integer}$

Since integer $< \text{identifier}$,

$L(x) = 0.9$ and $\gamma(x) = \text{identifier}$ (token name).

(ii) Let $x = \text{max}$

$L_1(x) = 1, \ \gamma_1(x) = \text{identifier}$

$L_2(x) = 0, \ \gamma_2(x) = \text{integer}$

Therefore $L(x) = 1$ and $\gamma(x) = \text{identifier}$.

(iii) Let $x = 1250$

$L_1(x) = 0, \ \gamma_1(x) = \text{identifier}$

$L_2(x) = 1, \ \gamma_2(x) = \text{integer}$

Therefore $L(x) = 1$ and $\gamma(x) = \text{integer}$.

Now we formally present algorithms to implement the lexical analyzer.
5.4 Fuzzy Lexical Analyzer

5.4.3 Algorithm MFRL\( (t_1, t_2, \ldots, t_n) \)

The algorithm uses the input alphabet \( \Sigma \), the character set of the programming language and \( \{t_1, t_2, \ldots, t_n\}, n > 0 \), the set of all tokens with linear order of priorities of their names \( \gamma_i, i = 1, 2, \ldots, n \). It computes uffsa for each token and then marked fuzzy regular languages are generated. It returns the marked fuzzy language for each token \( t_i, i = 1, 2, \ldots, n \).

\[
\begin{align*}
\text{begin} \\
  & \text{for } i := 1 \text{ to } n \text{ do} \\
  & \quad \text{begin} \\
  & \quad \quad \text{let } c_i \text{ be the fre that denotes } t_i; \\
  & \quad \quad /* \text{Now find ffsa for the fre } c*/ \\
  & \quad \quad M_i := LAMBDA(c) ; \\
  & \quad \quad /* \text{Now find uffsa for the ffsa } M_i */ \\
  & \quad \quad M_i^L := UFFSA(M_i) ; \\
  & \quad \quad /* \text{Find the minimum state uffsa for the ffsa } M_i^L */ \\
  & \quad \quad M_i^{L^1} := \text{MINUFFSA}(M_i^L) ; \\
  & \quad \quad /* \text{We consider the final uffsa for } c \text{ as } M_i */ \\
  & \quad \quad \text{let } M_i := M_i^{L^1} \text{ and } L_i \text{ be the fuzzy regular language}; \\
  & \quad \text{end} \\
  & \text{for } i := 1 \text{ to } n \text{ do} \\
  & \quad \text{let } (L_i, \gamma_i) \text{ be the marked fuzzy regular language for token } t_i; \\
\end{align*}
\]

Figure 5.9:
5.4 Fuzzy Lexical Analyzer

5.4.4 Algorithm FLEX

The algorithm uses the marked fuzzy regular languages \( (L_i, \gamma_i) \) for each token \( t_i, i = 1, 2, \ldots, n \). These are generated by calling the algorithm \text{mfrl}. It determines the degree of the input string \( x \in \Sigma^* \) and decides the action to be taken.

\begin{verbatim}
1 begin
2 call MFRL\( (t_1, t_2, \ldots, t_n) \); /* It returns the marked fuzzy regular
   language for each token */
3 let \( L_k(x) := \bigvee \{ L_i(x) \mid i = 1, 2, \ldots, n \} \), for some \( k, 1 \leq k \leq n \); /* thus the degree of \( x \) is defined */
4 if \( (L_k(x) \neq L_j(x), j \neq k) \) then
5   begin
6     /* no conflict, maximum is unique */
7     \( L(x) := L_k(x) \);
8     \( \gamma(x) := \gamma_k(x) \);
9   end
10 else
11   begin
12     /* finding the marked union */
13     \( (L(x), \gamma(x)) := \bigcup \{ (L_j(x), \gamma_j(x)) \mid L_j(x) = L_k(x), j = 1, 2, \ldots, n \} \);
14   end
15 /* deciding the action to be taken */
16 if \( (0.9 < L(x) \leq 1) \) then
17   begin
18     write ("accepted");
19     write ("Token name =", \( \gamma(x) \));
20   end
21 else if \( (0.8 \leq L(x) \leq 0.9) \) then
22   begin
23     write ("warning; accepted");
24 end
\end{verbatim}

write ("Token name =", \( \gamma(x) \));
end
else if \((0.7 \leq L(x) \leq 0.8)\) then
begin
write ("degree is low");
write ("Do you want to accept? yes/no");
if "yes" then
begin
write ("accepted");
write ("Token name =", \( \gamma(x) \));
end
else
write ("rejected");
end
else
write ("ERROR, the string is rejected")
end

Figure 5.10
Conclusion

This dissertation is thus a study on fuzzy finite state automata. It is based on formal languages and automata theory, which is an integral part of theoretical computer science. It seeks to elucidate the application and use of fuzzy finite state automaton with unique membership transition on an input symbol (uffsa), which is defined in this project.

The research findings are: It is proved and illustrated that for a given ffsa, there exists an equivalent uffsa. Also proved that for incomplete uffsa, there exists an equivalent complete uffsa. uffsa’s are designed to accept the union, intersection, concatenation and Kleene’s closure of fuzzy regular languages accepted by uffsa’s.

The generation of two different monoids from a given uffsa is proved and illustrated and some results are obtained. The underlying semigroups of the monoids that have been generated have become fuzzy anti-transformation semigroups and anti-polytransformation semigroups.

Admissible relation is defined on the set of states of an uffsa and some results are obtained.
5.4 Fuzzy Lexical Analyzer

The sub uffsa of an uffsa is defined and it is proved that the monoid corresponding to an uffsa is a homomorphic image of the monoid corresponding to the sub uffsa.

The homomorphism between uffsa’s has been defined, and some theorems have been proved. One of them is, if two uffsa’s are strong isomorphic, then their corresponding monoids are isomorphic.

Finally as a vital application, the uffsa’s which are defined in this work are useful to design fuzzy lexical analyzer (one of the phases of a compiler), which will be of a great use to the software industry. A model is proposed to design fuzzy lexical analyzer with suitable algorithms.