Introduction

Research in automata and formal language theory, which is one of the fundamental areas of theoretical computer science, started in the 1950's, when Noam Chomsky in his study on grammars and grammatical structure of a language, proposed a mathematical model of a grammar [12]. The initial investigation of the mathematical structure of language was aimed at trying to understand the basic properties of natural languages. It was found that phrase-structure grammar with a set of rewriting rules, which is essentially quite powerful, can be used as a method for describing languages. In late 1960, it was discovered that the "ALGOL-like" languages defined by Backus are identical with the context-free languages that are generated by a special class of phrase-structure grammars. This finding opens the possibility of investigating programming languages from theoretical points of view instead of by heuristic approaches alone. Since then, extensive research has been done by those concerned with either natural languages or programming languages.

An alternative way of specifying a language is in terms of the set of strings that are accepted by a certain recognition device, described as finite state automaton [12, 19, 42]. The original formal study of finite state systems (neural nets) was carried out by Mc Culloch and Pitts (1943). Kleene (1956) con-
Introduction

considered regular expressions and modeled the neural nets of Mc Culloch and Pitts by finite automata, proving the equivalence of the two concepts. Similar models were considered about that time by Huffman (1954), Moore (1956), and Mealy (1955), the latter two being the sources for the terms "Moore Machine" and "Mealy Machine". Nondeterministic automata were introduced by Rabin and Scott (1959), who proved their equivalence to deterministic automata. Hopcroft and Ullman [19] provided unified expositions in which the significant results concerning the theory of phrase-structure languages and their relation to automata were described. The proof of the equivalence of regular expressions and finite automata was patterned after Mc Naughton and Yamada (1960) [36]. Brzozowski (1962, 1964) developed the theory of regular expressions. Salomaa (1966) gave axiomatizations of regular expressions [42]. The use of the automata theory to design lexical analyzers is treated by Johnson et. al., (1968) and Lesk (1975), as found in [1].

Fuzzy approach is based on the premise that the key elements in human thinking are not just numbers but can be approximated to tables of fuzzy sets, or, in other words, classes of objects in which the transition from membership to non-membership is gradual rather than abrupt. The notion of fuzzy subset of a set as a method of uncertainty was introduced by Zadeh [52]. Since, Zadeh published his classic paper in 1965, fuzzy set theory has been receiving more and more attention from researchers in a wide range of scientific areas. One such area is formal languages and automata theory.

In the case of ordinary languages, a machine $M$ is said to recognize a language $L$ if and only if for every $x \in L$, $M$ decides that it is a member of $L$, and for every $x \notin L$, $M$ either decides that it does not belong to $L$, or
Introduction

loops indefinitely. In effect, this means that in order to recognize a language $L$, a machine must compute the characteristic function $L$. Extending this notion, we can naturally define a machine $M$ to recognize a fuzzy language if and only if $M$ can compute its membership function. Honda and Nasu [18] have shown that the concept of recognition of fuzzy languages by machines like finite automata, pushdown automata, etc. is a reasonable extension of recognition of ordinary languages by these machines. They have developed a formal theory of the recognition of fuzzy languages and have arrived at a number of interesting conclusions.

Apart from serving as models of fuzzy systems, fuzzy automata can act as acceptors and recognizers of fuzzy languages. The earliest definition of a fuzzy automaton was given by Wee [51]. Santos [44] defined maximin and minimax automata while giving a general formulation of fuzzy automata. The maximin automata turned out to include both deterministic and nondeterministic automata as special cases. Moreover, the state transition function and the initial distribution of a maximin automata found interpretation as the grades of membership of fuzzy sets. There are many ways in which the ordinary automata and languages have been fuzzified. There is no universally standardized terminology of defining fuzzy automata in actual practice. However, fuzzy automata have the common property that they have membership values, which are in $[0,1]$, associated with the state transitions as well as initial and final state distributions.

The terms fuzzy finite state automaton (ffsa) and fuzzy finite state machine (ffsm) have been used transposedly, but conceptually they are different. In this research, the term fuzzy finite state automaton (ffsa) refers
Introduction

to: $M = (Q, \Sigma, \mu, i, f)$, where $Q$ is a non-empty finite set of states, $\Sigma$ is a non-empty finite set of input symbols, $\mu$ is a fuzzy transition function, $\mu : Q \times \Sigma \times Q \to [0,1]$, $i : Q \to [0,1]$ and $f : Q \to [0,1]$ are fuzzy subsets of $Q$, called fuzzy initial states and fuzzy final states respectively.

In a fuzzy finite state automaton (ffsa), there may be more than one transitions from a state on an input symbol with a given membership value [44, 51]. This development was followed by the postulation called deterministic fuzzy finite state automaton (dfssa) [28]. i.e., $\forall p \in Q, a \in \Sigma, \mu(p, a, q) = \mu(p, a, q')$ for some $q, q' \in Q$, then $q = q'$. The author considered it as a deterministic fuzzy recognizer. Accordingly, there can be at most one transition on an input, which can be constructed equivalently from an ffisa. However, it only acts as a deterministic fuzzy recognizer, and the fuzzy regular languages accepted by the ffisa and dfssa need not necessarily be equal (i.e., the degree of a string need not be the same). Thus it was felt to evolve a system that is on the one hand simpler than the ffisa while at the same time, maintaining the fuzzy regular language.

To achieve this objective, we define an ffisa with a unique membership transition on an input symbol, which is denoted by uffsa. An uffsa is an ordered quintuple $M = (Q, \Sigma, \mu, i, f)$, where $Q$ is a non-empty finite set of states, $\Sigma$ is a non-empty finite set of input symbols, $\mu$ is a fuzzy subset of $Q \times \Sigma \times Q$, called the fuzzy transition function, i.e., $\mu : Q \times \Sigma \times Q \to [0,1]$ and such that $\mu$ is a fuzzy function of $Q \times \Sigma \times [0,1]$ into $Q$, $i : Q \to [0,1]$ and $f : Q \to [0,1]$ are fuzzy subsets of $Q$, called fuzzy initial states and fuzzy final states respectively.

This new system brings forth many results and properties, which are
Introduction

discussed in the thesis. The use of uffsa is widely applied in this thesis, which had led to many interesting results suitable for uffsa. The details of the study undertaken in the thesis are now summarized.

The thesis begins with Chapter 1, comprising of preliminaries. This includes known basic definitions and concepts, which are essential to establish the assertions proved in the subsequent chapters. The key definitions are finite state automaton and regular expressions [19], fuzzy sets [22, 43, 52], fuzzy languages [25, 26], algebraic system [27, 40], fuzzy automaton [44, 48, 51], fuzzy recognizers and fuzzy finite state machines [28].

In Chapter 2, we formally define fuzzy finite state automaton with unique membership transition on an input symbol (uffsa), where 'u' refers to unique membership transition. It is known that for any fuzzy recognizer $M$ there is a deterministic fuzzy recognizer $M_1$ with the same behaviour [28] in the sense $L(M) = L(M_1)$. However, for any string $x \in \Sigma^*$, $\deg(x)$ in $M$ and that in $M_1$ need not be the same. We prove that for any ffsm there exists an equivalent ufissa such that the language accepted by the two are the same. The existence is illustrated with an example. It is also proved that for incomplete ufissa, there exists an equivalent complete ufissa. Also this chapter includes some of the closure properties of fuzzy regular languages such as union, intersection, concatenation and Kleene's closure.

A study on monoids (underlying semigroups) and transformation semigroups is done in Chapter 3. Malik, Mordeson and Sen [31] have generated two distinct semigroups from the fuzzy transition function of an ffsm. We, on our part, have been successful in generating two other monoids namely $F(M)$ and $S_M$ for an ufisa $M$. These monoids (underlying semigroups) are
Introduction

not only different from the two semigroups mentioned above, but also that they could only be generated in uffsa’s, which are not possible in flsm’s. The existence of the monoids have been illustrated with example. We are also able to prove in the reverse process, the result that for a given finite fuzzy monoid $S$ there exists an uffsa $M$ such that $F(M)$ is isomorphic to $S$. A new concept called fuzzy anti-transformation semigroup (fats) is defined and is linked with an uffsa in the sense that, given an uffsa $M$, we can construct a faithful fats $(Q, F(M), \rho)$. A similar result holds good for anti-polytransformation semigroup as well.

Chapter 4 begins with the definition of sub uffsa’s of a given uffsa, $M = (Q, \Sigma, \mu, 1, f)$ which are uffsa’s and as set of states that are subsets of $Q$ with restriction mappings $\mu'$, $i'$ and $f'$. The sub uffsa’s are closed under intersection and union. It is interesting to note that if $M_1$ is a sub uffsa of the uffsa $M$, then $F(M)$ is a homomorphic image of $F(M_1)$. We define homomorphism and strong homomorphism of uffsa’s and prove that if $(\alpha, \beta) : M_1 \rightarrow M_2$ is a homomorphism, then $L_1(x) \leq L_2(\beta^*(x)) \forall x \in \Sigma_1^*$, the equality occurs if $(\alpha, \beta)$ is a strong homomorphism and $\alpha$ is bijective. We prove another important result that if two uffsa’s $M_1$ and $M_2$ are strong isomorphic, then their monoids $F(M_1)$ and $F(M_2)$ are also isomorphic as monoids. Admissible relation on the set of states of uffsa is defined and some results are obtained.

Finally we arrive at applications in Chapter 5. A compiler [1] is a program, which takes as input, a program (source program) written in one of the high-level languages, such as FORTRAN, C, C++, etc. and produces as output a program (object program) in machine language. Conceptually, a
Introduction

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. Lexical analyzer is a dfa, 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:

1. the string is accepted as a token
2. with warming message, the string is accepted as a token
3. an on-line question is given to user, the string is accepted if the user answers 'yes'; rejected if the user answers 'no' and
4. the string is rejected.

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 the minimum state uffsa for a given fre.

Algorithm LAMBDA takes as input an fre and produces as output an ffsa with \( \lambda \)-transitions accepting the language denoted by the fre. The input of the second algorithm UFFSA is the ffsa with \( \lambda \)-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
Introduction

output of UFFSA. The final uffsa will accept the fuzzy language denoted by the given regular expression. Systematically we apply the three algorithms and obtain an uffsa in an example.

In lexical analysis, it is a common practice to construct one automaton for accepting many different tokens (i.e., regular languages). In order to distinguish strings belonging to different tokens, final states are marked with token names. When we design fuzzy lexical analyzers, i.e., fuzzy finite state automaton, for two different strings, we may have the same membership values (degree), so there is likely to be a conflict to resolve it. 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.