Current Trends in Technology and Science
ISSN : 2279-0535. Volume : 04, Issue : 06 (Oct- Nov 2015)

A Pilot Study on Natural Language Processing – Applications of Finite State Automation

Shubham Chatterjee
Student, Department of Computer Science, St. Xavier's College (Autonomous), Kolkata, India
shubham.chatterjee94@gmail.com

Kasturi Paul
Student, Department of Computer Science, St. Xavier's College (Autonomous), Kolkata, India
paulkasturi15@gmail.com

Reek Roy
Student, Department of Computer Science, St. Xavier's College (Autonomous), Kolkata, India
reekroy1@gmail.com

Asoke Nath
Associate Professor, Department of Computer Science, St. Xavier's College (Autonomous), Kolkata, India
asokejoy1@gmail.com

***Abstract* — Natural language processing (NLP) may be defined as the automatic or semi-automatic processing of human language. The term 'NLP' is sometimes considered to be a process which excludes information retrieval and sometimes even machine translation. Sometimes NLP is contrasted with 'computational linguistics'. The alternative terms of NLP are often preferred, like 'Language Technology' or 'Language Engineering'. Language is often used in contrast with speech (e.g., Speech and Language Technology). In the present paper the authors refer the term NLP in much more broader sense. NLP is related to linguistics and also has links to research in cognitive science, psychology, philosophy and mathematical logic. In computer science, it relates to formal language theory, compiler techniques, theorem proving, machine learning and human-computer interaction and also to AI. In the present paper, the authors have given an introduction to natural language processing and then application of theory of finite state automation. The basic principles of Finite State Automata Theory including DFAs and NFAs are also discussed and finally applications of FSM in NLP.**

***Keyword* — Deterministic Automata, Finite state transducer, Natural Language Processing, Non-Deterministic Automata.**

1 INTRODUCTION TO NATURAL LANGUAGE PROCESSING

Natural Language Processing (NLP) is a field of computer science, artificial intelligence and computational linguistics concerned with the interactions between computers and human (natural) languages. NLP involves the natural language understanding, that is enabling computers to derive meaning from human or natural language input, and others involve natural language generation. [3]

NLP is a theoretically motivated range of computational techniques for analyzing and representing naturally occurring texts at one or more levels of linguistic analysis for the purpose of achieving human like language processing for a range of tasks or applications. [4]

The goal of NLP is to accomplish human-like language processing. NLP has made serious inroads into jobs like:

1) Paraphrase an input text.
2) Translate the text into another language.
3) Answer questions about the contents of the text.

There are more practical goals for NLP, mostly related to the certain applications for which it is being applied. As for example, an NLP based Information Retrieval System has the goal of providing more accurate and complete information in response to a user's actual information need. The goal of NLP here is to represent the real meaning and intent of the user's query, which can be expressed as naturally in everyday language as if they were conversing with a reference librarian. All the contents of the documents that are being searched will be represented at every level of meaning so that a correct match between the requirement and response can be discovered, no matter how either is represented in their surface form. [4]

The lineage of NLP is mixed. Main contributors to the discipline and practice of NLP are:

1) *Linguistics*-focuses on formal, structural models of language and the discovery of language universals.
2) *ComputerScience*-is concerned with developing internal representations of data and efficient processing of these structures.
3) *CognitivePsychology*-looks at language usage as a window into human cognitive processes, and has the goal of modeling the usage of language in a psychologically feasible way.

The central point of any NLP job is an important issue of natural language understanding. The method of creating computer programs that understand natural language includes three major problems-the first one relates to thought process, the second one to the representation and meaning of linguistic input, and the third one to the world knowledge. Thus an NLP system may start at the "word" level-to determine the morphological structure, nature etc. of the word, and then move on to the sentence level-to determine the word order, grammar, meaning of the entire sentence, etc.-and then to the context and the overall environment or domain. A given word or a sentence may have a particular meaning or connotation in a given context or domain, and may be related to many different other words and/or sentences in the given context. [4].

2 ISSUES IN NATURAL LANGUAGE PROCESSING

The tasks taking place within an NLP system generally occurs in certain levels.

- *Phonology* deals with interpretation of speech sounds within and across the word
- *Morphology* includes componential nature of the words which are composed of morphemes (smallest units of meaning);
- *Lexical* level includes both humans and NLP systems determining the meaning of individual words;
- *Syntactic* level deals with the analysis of the words in a sentence in order to discover the grammatical structure of the sentence
- *Semantic* level determines feasible meaning of a sentence by focusing on the word-level meanings' interactions within the sentence.
- Other levels such as *Disclosure*, *Pragmatic* deals with longer lengths of textual matter and more intricate meaning deciphering. [4]

3 INTRODUCTION TO FINITE STATE AUTOMATA

Automata theory is the study of abstract computing devices or machines. During the time when computers had still not developed and use of computers was still not widespread, Allen Turing studied an abstract machine that had all the capabilities of today's computers, at least as far as in what they could compute. Turing wanted to explain clearly what such a machine was capable of doing or not doing. Later on, many simpler kinds of machines, which we today call finite automata, were developed. [2]

Finite State Automata (FSA) and Finite State Transducers (FST) are Finite State devices which have been used widely in the field of Computer Science since the beginning of Computer Science period. These Finite State devices used in a applications ranging from compilation of programs to hardware modeling or database management and other domains like speech processing, Optical Character Recognition (OCR), matching and recognition of patterns and many more. Recently many mathematical and algorithmic results have shown that Finite State technologies such as FSA and FST have great impact on the representation of electronic dictionaries and Natural Language Processing (NLP) which as a result is leading to a new language technology in academic and industrial research. [1]

Both FSA and FST operate on sets of strings (alphabets) which are actually sets of sequences of symbols or characters. These characters are either finite for example the English alphabet or infinite like the Real Numbers. A string is a finite sequence of symbols. '*Free monoid*'(Σ^*) is the set of strings which are built on an alphabet $\sum$. [1]

Finite automata are a useful model for many important kinds of hardware and software. Some of the important applications of the Finite State Automata are in [1]:

i) Switching theory
ii) Pattern recognition
iii) Speech processing
iv) Optical character recognition
v) Data compression
vi) Compiler theory

Many systems such as those listed above can be supposed to be in one of a finite number of "states". A state remembers the relevant portion of the system's history. The entire history cannot be remembered since there are only a finite number of states. Hence, one of the "Eq. important design issues for such systems to remember what is important and to forget the unimportant parts. Since there are a finite number of states, the system can be implemented using a fixed set of resources.

The simplest example that one could give of a FSA is an ON/OFF switch [2]. Such a device would remember when it is in the ON state and when it is in the OFF state. Depending on the state of the switch, the effect of a button push would be different. If the switch is in ON state, then it would go to an OFF state and vice-versa. Such a switch could be modeled as in the figure below:

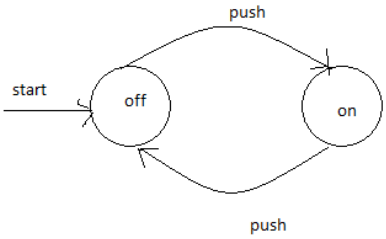

Fig. 1. A FSA for a switch

The states are represented by circles. In this example, the states have been named as ON and OFF. The arcs are labeled with the inputs. One of the states is the "start" state and is identified by an arrow as shown in the figure.

One of the crucial distinctions among classes of finite automata is whether that control is *deterministic* or *non-*

Current Trends in Technology and Science
ISSN : 2279-0535. Volume : 04, Issue : 06 (Oct- Nov 2015)

deterministic. Deterministic Automata cannot be in more than one state at any given point in time whereas Non-deterministic Automata can be.

3.1 Deterministic Finite Automata

3.1.1 What is DFA*?*

A Deterministic Finite Automata is a Finite State Machine that accepts or rejects finite strings of symbols and only produces a unique computation of the automaton for each input string. The term "Deterministic" refers to the uniqueness of the computation.

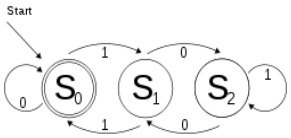

Fig. 2. An example DFA

The above Fig. 2 illustrates a DFA using a state diagram. There are three states in the DFA, namely S0, S1 and S2. S0 is the start state. The DFA accepts strings of the form 011, 01001, 010101, 00101101, etc. Whenever the DFA reads a symbol, it jumps *deterministically* from one state to the other, depending on the transition function. For example $\delta(S0,1)=S1$, so whenever the DFA reads the symbol 1 and it is in state S0 at that moment, it will jump to state S1. The DFA has a start state S0, represented in the diagram by an arrow and a set of accepting states. In this case, there is only one accepting state, which is S0.

A DFA is defined as an abstract mathematical concept but it is widely implemented in hardware and software terms owing to its deterministic nature. For example, a DFA can model software that decides whether or not online user-input such as email addresses are valid.

3.1.2 Formal Definition of a DFA

A *Deterministic Finite Automata (DFA) M is a 5-tuple*

1) A finite set of *states*, denoted by *Q*.
2) A finite set of *input symbols*, denoted by $\sum$.
3) δ is the *transition function* that takes a state in Q and an input symbol in $\sum$ as arguments and returns a subset of Q.
4) A start state ($q_0 \in Q$).
5) A set of final states F such that $F \subseteq Q$.

Let $w = a_1a_2 \ldots a_n$ be a string over the alphabet Σ. The automaton *M* accepts the string *w* if a sequence of states, $r_0, r_1, \ldots, r_n$, exists in *Q* with the following conditions:

1) $r_0 = q_0$
2) $r_{i+1} = \delta(r_i, a_{i+1})$, for $i = 0, \ldots, n-1$
3) $r_n \in F$.

The above conditions mean that the DFA starts form the initial state q_0. Then it moves from one state to another depending on the transition function. The DFA is said to *accept* a string if it halts at one of the accepting states. Otherwise we say that the DFA does not accept the string. The set of string that the DFA *M* accepts is called the *language* recognized by *M* and it is denoted by *L(M).*

3.1.3 An example of a DFA

Let us take the example of a DFA that accepts all and only the strings of 0s and 1s that have the sequence 01 somewhere in the string [2]. We can write this language L as:

{ w | w is of the form x01y for some strings x and y consisting of 0s and 1s only }

Examples of strings in this language include 01, 11010, 100011, etc. examples of strings *not* in the language include ∈, 0, and 111000. We can define the automaton that accepts this language as follows:

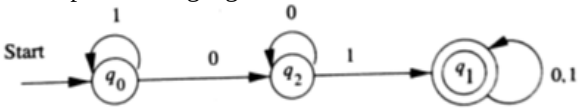

Fig. 3. DFA

3.2 Non-Deterministic Finite Automata

3.2.1 What is NDFA?

A "Non-deterministic" finite automaton (NFA) can be in several states at given point of time. The ability is often expressed as an ability to "guess" something about its input [2]. For example, when the automaton is searching for certain sequences of characters in a long string, it would help to make a guess that it is at the beginning of one of those strings and use a sequence of states to do nothing but check that the string appears , character by character. It can be shown that a language accepted by an NFA is also acceptable to some DFA. We can convert any NFA to an equivalent DFA using the *subset construction algorithm*. Like DFAs, NFAs also accept only regular languages.

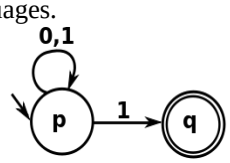

Fig. 4. NDFA

The above figure illustrates an NFA using state diagram. There are two states in the NFA, *p* and *q*. *p* is the start state and *q* is the final state. As can be seen from the figure, this automata is non-deterministic in the sense that $\delta(p, 1) = p \text{ or } q$, that is, on encountering the input symbol 1, if the NFA is in state *p*, it can go to either *p* or *q*.

3.2.2 Formal Definition of a NDFA

A *Non-Deterministic Finite AutomataA is a 5-tuple* $(Q,\sum,\delta,q_0,F)$ *where*

1) A finite set of *states*, denoted by *Q*.
2) A finite set of *input symbols*, denoted by $\sum$.
3) A *transition function*, that takes as a state and an input symbol as arguments and returns a state. The transition function is commonly denoted by δ. If *q* is a state and *a* is an input symbol, then $\delta(q,a)$ is that state *p* such that here is an arc labeled a from q to p.
4) A start state ($q_0 \in Q$).
5) A set of final states F such that $F \subseteq Q$.

Current Trends in Technology and Science
ISSN : 2279-0535. Volume : 04, Issue : 06 (Oct- Nov 2015)

Let $w = a_1a_2 \ldots a_n$ be a string over the alphabet Σ. The automaton A accepts the string w if a sequence of states, $r_0, r_1, \ldots, r_n$, exists in Q with the following conditions:

1) $r_0 = q_0$
2) $r_{i+1} = \delta(r_i, a_{i+1})$, for $i = 0, \ldots, n-1$
3) $r_n \in F$.

3.2.3 An example of an NDFA

Let us take the example of an NFA which accepts exactly those strings that have any number of either the symbols *a* or *b* in the starting and always end with the string *abb*. Examples of such strings include *aabb, babb, aabbabb* etc. The NFA has been shown below:

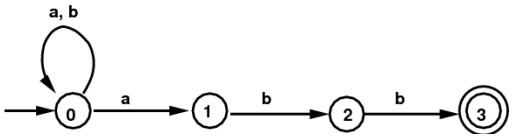

Fig. 5. An example of NFA

3.3 Finite State Transducers

3.3.1 What is an FST?

Finite-State Transducers can be conceptually thought of as defining a class of graphs, a class of relations on strings, or a class of transductions on strings [1]. A **finite state transducer** (**FST**) is a finite state machine with two tapes: an input tape and an output tape. This contrasts with an ordinary finite state automaton (or finite state acceptor), which has a single tape. The two tapes of a transducer are typically viewed as an input tape and an output tape. On this view, a transducer is said to *transduce* (i.e., translate) the contents of its input tape to its output tape, by accepting a string on its input tape and generating another string on its output tape. It may do so non-deterministically and it may produce more than one output for each input string. A transducer may also produce no output for a given input string, in which case it is said to *reject* the input. In general, a transducer computes a relation between two formal languages.[6]

Each string-to-string finite state transducer relates the input alphabet Σ to the output alphabet Γ. Relations *R* on Σ*×Γ* that can be implemented as finite state transducers are called **rational relations**. Rational relations that are partial functions, i.e. that relate every input string from Σ* to at most one Γ*, are called **rational functions**.[6]

Finite-state transducers are often used for phonological and morphological analysis in natural language processing research and applications.[6]

3.3.2 Formal Definition of an FST

A Finite-State Transducer T is a 6-tuple $(Q, \Sigma, \Gamma, i, F, \delta)$ where:

- Q is a finite set of *states*
- Σis finite set, called the *input alphabet*
- Γis finite set, called the *output alphabet*
- $i \in Q$ is *the initial state*.
- $F \subset Q$ is the set of *final states.*
- $\delta \subset Q \times \Sigma^* \times \Gamma^* \times Q$ is the *set of transition relations.*

We can view (Q, δ) as a labeled directed graph, known as the *Transition Graph* of T. The set of vertices is Q and $(q, a, b, r) \in \delta$ means that there is a labeled edge going from vertex q to vertex r. Also, *a* is the input label and *b* is the output label of that edge. [6]

3.3.3 Some other definitions

- **Path:** An *Extended Transition Function* δ^* is defined as the smallest set such that :
  - $\delta \subseteq \delta^*$
  - $(q, \varepsilon, \varepsilon, q) \in \delta^* \; \forall q \in Q$ and
  - $whenever\ (q, x, y, r) \in \delta^*\ and\ (r, a, b, s) \in \delta\ then\ (q, xa, yb, s) \in \delta^*$

  The extended transition relation is essentially the reflexive transitive closure of the transition graph that has been augmented to take edge labels into account. The elements of $\delta^*$are known as ***paths***. The edge labels of a path are obtained by concatenating the edge labels of its constituent transitions in order. *A successful path* is a path that starts from an initial state and ends in a final state.
- **Underlying finite-state automaton*:*** If $T=(Q, \Sigma, \Gamma, i, F, \delta)$ is a FST, then its underlying FSA (Σ', Q, i, F, E) is defined as follows:

$$\Sigma' = \Sigma \times \Gamma$$
$$(q_1, (a, b), q_2) \in E \; iff \; (q_1, a, b, q_2) \in \delta$$

- **Rational Functions*:*** A **regular (or rational) relation** over the alphabets Σ, Γ is formed from a finite combination of the following rules:

i) $\forall (x, y) \in \Sigma \cup \{\varepsilon\} \times \Gamma \cup \{\varepsilon\}$
ii) $\emptyset$ is a regular relation
iii) If *R*, *S* are regular relations, then so are $R \circ S$, $R \cup S$, and R^*
iv) Each string-to-string finite state transducer relates the input alphabet Σ to the output alphabet Γ. Relations *R* on Σ*×Γ* that can be implemented as finite state transducers are called ***rational relations***. Rational relations that are partial functions, i.e. that relate every input string from Σ* to at most one Γ*, are called rational functions.

4 Some Applications of Finite State Automata and Discussions the subject

1) This work which was published in ***Introduction to Finite-State Devices in Natural Language Processing June 1996, MITSUBISHI ELECTRIC RESEARCH LABORATORIES.*** The authors have tried to show how the closure property of FSA exhibits their use, uniformity and flexibility on constraints operating on a local context. In these examples the input string, the lexicon, the local syntactic rules are all represented as Finite State Automata and the combination of rules corresponding to operations on FSA produce other FSA. We give an overview of the method here. Interested readers can refer to the published work for more details.

Current Trends in Technology and Science
ISSN : 2279-0535. Volume : 04, Issue : 06 (Oct- Nov 2015)

Let us assume that we have to analyze the below given statement:

He hopes that this works

i) If we look up in the dictionary we can easily identify that the word '*He*' is a pronoun, '*hopes*' is a noun or a verb, '*that*' is a determiner or a pronoun or a conjunction, '*this*' is a determiner or a pronoun and the word '*works*' is a noun or a verb. These kinds of morphological information that are already encoded within a dictionary are easily represented by a Finite State Automata in Fig. 6 below:

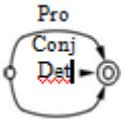

Fig. 6

ii) The automaton of the word 'that' could be easily done since the dictionary lookup gave us such representations. Similarly the morphological information of all the other words in the sentence that we took to analyze can be also represented by individual automaton for each word. Hence the morphological analysis of the whole input sentence can also be represented by a FSA as shown in Fig. 7 below. As we can see that the dictionary and the morphological analysis of a sentence can be represented by a FSA, the morphological disambiguating rules can also be represented by a FSA.

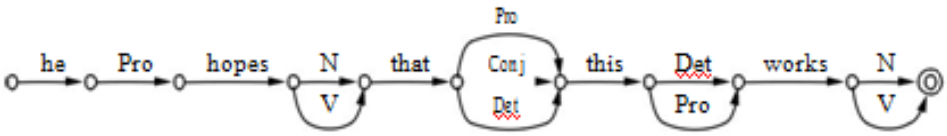

Fig. 7

iii) If we have to encode a negative rule stating the partial analysis "*that Det this Det*" is not possible, then the negative constraint can be encoded by the automaton as shown in Fig. 8 below.

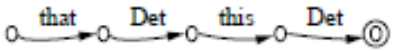

Fig. 8

iv) We see that applying the constraint of Fig. 8 to the Fig. 7 gives the automaton shown in Fig. 9 .

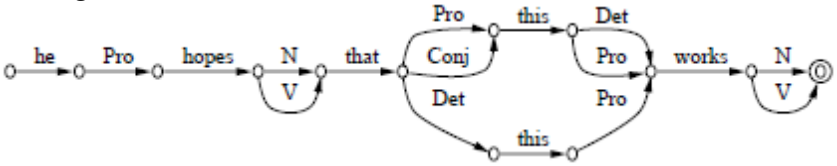

Fig. 9

2) FSTs are particularly useful in the implementation of certain natural language processing tasks [8]. Context-sensitive rewriting rules (e.g. **ax → bx**) are adequate for implementing certain computational linguistic tasks such as morphological stemming and part-of-speech tagging. Such rewrite rules are also computationally equivalent to finite-state transducers, providing a unique path for optimizing rule based systems.

Take, for example, the following set of ordered context-sensitive rules:

- *change 'c' to 'b' if followed by 'x'*
  cx → bx
- *change 'a' to 'b' if preceded by 'rs'*
  rsa → rsb
- *change 'b' to 'a' if preceded by 'rs' and followed by 'xy'* **rsbxy → rsaxy**

Given the following string on the input tape:

rsaxyrscxy

the application of the given rule set would proceed as follows:

- **rsaxyrscxy→rsaxyrs<u>b</u>xy**
- **rsaxyrsbxy→rs<u>b</u>xyrsbxy**
- **rsbxyrsbxy → rs<u>a</u>xyrs<u>a</u>xy**

The time required to apply a sequence of context-sensitive rules is dependent upon the number of rules, the size of the context window, and the number of characters in the input string. The inefficiencies in such an implementation are highlighted in this example by the multi-step transformation required to translate '**c**' to '**b**' then '**b**' to '**a**' by rules 1 and 3, and the redundant transformation of '**a**' to '**b**' and back to '**a**' by rules 2 and3. Finite State Transducers provide a path to eliminate these inefficiencies. But first we need to convert the rules to State Machines.

Current Trends in Technology and Science
ISSN : 2279-0535. Volume : 04, Issue : 06 (Oct- Nov 2015)

Converting Rules to FSTs

To do this, we simple represent each rule as an FST where each link between states represent the acceptance of an input character and the expression of the corresponding output character. This input / output combination is denoted within an FST by labeling edges with both the input and output character separated by the '/' character. Following is an FST for each of the above rules:

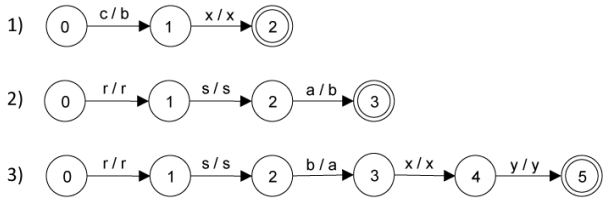

Fig. 10

Extending the FSTs

While the FSTs above represent our set of context-sensitive rules, they would be of little use in matching against an input string as each is designed to process exactly the context widow described in its corresponding rule. To make each FST applicable to a string of arbitrary length and perform the necessary translation each time the rule is fired, we will need to extend each of the Transducers. We do this by allowing for every possible input in our language at each state. For example, rule 1 must be able to handle the string **rsaxyrscxy**. Since rule 1 matches the string '**cx**' and outputs the string '**bx**', it must handle the characters '**r**', '**s**', '**a**', '**x**', '**y**', '**r**', and '**s**' before finally encountering '**c**' and '**x**'. Then it must handle the final '**y**' character. Each of these characters (and all other possible characters) could be explicitly listed on its own individual edge, but to simplify, we can create a single edge labeled with '**?/?**', to match and output any character not already represented on another edge leading from that state. FST extension of rules with a trailing context will also require the use of edges with an input but no output (labeled with '**ε**' for output) and edges with multiple outputs. Following is the extension for each of the above FSTs:

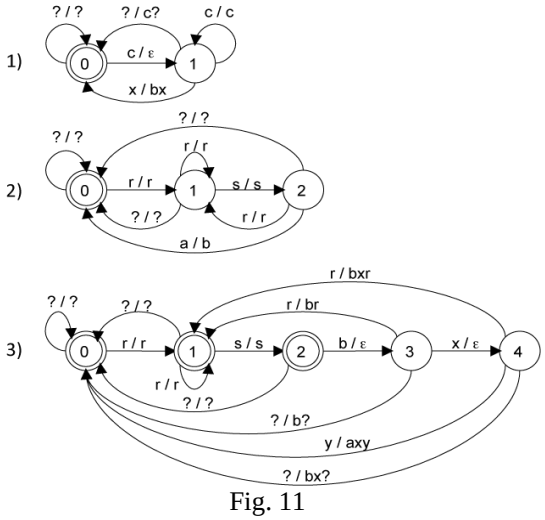

Fig. 11

5 CONCLUSION

We know that Natural Language Processing is a field that is part of both computer science and linguistics. It aims to process natural languages automatically with the less human supervision possible. Hence a lot of its domains use Finite State Machines as seen above with a lot of different requirements.

Automata are seen as an efficient representation of data such as morphological rules or lexicons. Through this report we have seen that automata can be used in all stages of Natural Language Processing (NLP) even at various stages of translation. Automata can be used to represent the morphological analysis and phonological rules very efficiently as compared to other required techniques (Kaplan and Kay).It has been useful in grammars and syntax, for example in case of Context Free Grammars (Chomsky). They outperform in indexation in terms of speed (consumption time) and memory space consumption than other techniques. Automata can handle all the alphabets (set of strings), from boolean to real, that may be needed, by implementing both Finite State Automata and Finite State Transducers.

REFERENCES

[1] Emmanuel Roche and Yves Schabes, *Introduction to Finite-State Devices in Natural Language Processing June 1996, MITSUBISHI ELECTRIC RESEARCH LABORATORIES*

[2] Hopcroft, John E.; Motwani, Rajeev; Ullman, Jeffrey D. *(2001).* "Introduction to Automata Theory, Languages, and Computation (3 ed.). Pearson Publications".

[3] *https://en.wikipedia.org/wiki/ Natural_language_processing*

[4] Liddy, E. D, "Encyclopedia of Library and Information Science, 2nd Ed. Marcel Decker, Inc".

[5] ANS S I YLI - JYR ¨A, ANDR ´A S KORNAI, JACQUES SAKAROVITCH, "Finite-State Methods and Models in Natural Language Processing, 2010 Cambridge University Press"

[6] *https://en.wikipedia.org/wiki/Finite_ state_ transducer*

[7] Jimmy Ma, "Automata in Natural Language Processing , Technical Report n°0834, December 2008, Revision 2002"

[8] http://infolocata.com/mirovia/ finite-state-transducers-for-natural-language-processing

[9] Mehryar Mohri, "On Some Applications of Finite State Automata Theory to Natural Language Processing, Cambridge University Press, 1995"

[10] Shuly Wintner, "Formal Language Theory for Natural Language Processing"

[11] Daniel Jurafsky and James H. Martin, "Speech and Language Processing: An introduction to natural language processing, computational linguistics, and speech recognition, 2006"

[12] Lauri Karttunen, "Applications of Finite State Transducers in Natural Language Processing."
[13] Anne Kao and Steve Poteet, "Text Minining and Natural Language Processing-Introduction for the Special Issue"
[14] Harvey J Greenberg, "A Natural Language Discourse Model to explain Linear Programming Models and Solutions"
[15] Anssi Yli-Jyra, Andras Kornai, Jacques Sakarovitch, "Finite-State Methods and Models in Natural Language Processing"
[16] Gobinda C Chowdhury, "Natural language processing" University of Strathclyde
[17] Jasmeen Kaur, Bhawna Chauhan, Jatinder Kaur Korepal, "Implementation of Query Processor Using Automata and Natural Language Processing" International Journal of Scientific and Research Publications, Volume 3, Issue 5, May 2013, ISSN 2250-3153
[18] Mengqiu Wangand Christopher D. Manning, "Probabilistic Finite State Machines for Regression-based MT Evaluation"
[19] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, Pavel Kuksa, "Natural Language Processing (Almost) from Scratch" Journal of Machine Learning Research 12 (2011) ISSN-2493-2537
[20] Matthew Lease, "Natural Language Processing for Information Retrieval: the time is ripe (again)"

AUTHOR'S PROFILE

Asoke Nath is Associate Professor in the Department of Computer Science, St. Xavier's College (Autonomous), Kolkata, India. Apart from his teaching assignment he is actively associated with research work in field of Cryptography and Network Security, Steganography, Green Computing, E-learning, MOOCs, Big data analytics, Cognitive Radio etc. He has already published **157** research papers in International Journals and Conference Proceedings. He is the Life member of MIR Labs(USA), CSI Kolkata Chapter.

Shubham Chatterjee is a student of MSc Computer Science at St. Xavier's College (Autonoumous), Kolkata. Along with normal studies, he is engaged in research work in the field of Computer Vision, Natural Language Processing, Li-Fi Technology, Android Security, Etc. He has published 4 research papers till date in above areas.

Kasturi Paul a student of MSc Computer Science at St. Xavier's College (Autonoumous), Kolkata. Along with normal studies, she is engaged in research work in the field of Computer Natural Language Processing.

Reek Roy is a student of MSc Computer Science at St. Xavier's College (Autonoumous), Kolkata. Along with normal studies, she is engaged in research work in the field of Computer Natural Language Processing.