International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue1 | ISSN : 2456-3307

Application Review of Automata Theory

Aparna*[1], Dr. Gulshan Goyal[2]
[*1]Student, Department of CSE, Chandigarh College of Engineering and Technology, Chandigarh, India
[2]Assistant Professor, Department of CSE, Chandigarh College of Engineering and technology, Chandigarh, India

ABSTRACT

A formal language is described as a set of strings following a defined pattern over a given alphabet. An automaton is a machine, which is used to process the formal languages. The field of automata theory finds number of applications in literature. Present paper reviews few of them. The application domain considered in present study includes compiler design, time granularity, deep packet inspection and DNA evolution. The aim of present study is to explore the applicability of concept in described field. For this, research papers have been reviewed to infer that the principles and concepts of automata are being used in fields as diverse as networking to a field like biology and bio-informatics. From the review, it has been concluded that each automaton, which is available, is a representation of a real-life scenario and they can be used to solve other problems. The review is quite helpful for novel researchers in the field of formal languages and automata theory to understand applicability of field in variety of applications.

Keywords: Automata, Compiler, Cellular Automata, Time Granularity, Packet Inspection.

I. INTRODUCTION

'Automata' is a Greek word, which means self-acting. In automata theory, an automaton is a machine, which does some processing by moving through a series of states on providing some inputs and by following some rules [8]. A simple automaton has generally five parameters:

Set of states (Q): This is a finite set, which represents the possible states in which an automaton can be at a particular time.
Input alphabet (∑): This set contains the input symbols using which the strings that can be processed by the automaton.
Initial state (qo): This is an element of set Q and represents the state from which string processing starts.
Set of final states (A): This is a subset of the set of states. The states in this set are called final states and a string, which stops at one of these states, is said to be accepted by the automaton.
Transition function (δ): This is a rule that describes the transition to a state if given a particular input symbol on a particular state [8].

There are many other types of automaton, which are modifications of this basic automaton. Some basic types of automaton are:

a) Finite automaton
b) Push-down automaton
c) Linear-bounded automaton
d) Turing machine

Taking the example of a machine like fan, the process of constructing and understanding a finite state machine is described.
Set of States: A fan can have two possible states i.e. **'ON'** and **'OFF'.** Thus,

Q= {ON, OFF}.

Input Alphabet: Consider an input 1 to switch on the fan and input 0 to switch off fan. Thus,

$$\sum = \{0, 1\}.$$

Initial state: Let initially the fan is in 'OFF' state. Thus, qo= OFF.

Set of final states: Let the final state of the fan be 'ON'.
Thus, A= {ON}.

Transition function: The transition function is described as:

$$\delta: (Q \times \sum) \rightarrow Q$$

This means that on any state q, if an input is given, a possible transition to another state is possible. The finite state machine for fan is shown in Figure 1 and corresponding transition table is given in Table 1.

This means that if 1 is given on state OFF state, fan will move to ON state and on providing 0 input, it will move to O. Otherwise, it will remain on same state. In this way, the fan can be represented as a finite automaton.

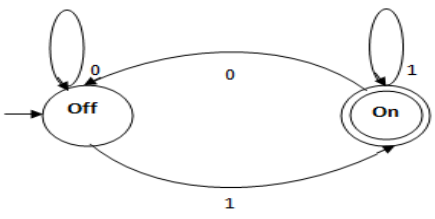

Figure 1: Finite State Machine for a Fan

Table 1. Transition table for Fan Automaton

State	0	1
→OFF	OFF	ON
(ON)	OFF	ON

II. APPLICATIONS OF AUTOMATON

Automata theory is the study of machines, which can be used to represent any computational system as well as computational problems that can be solved using them. It is important to study this because it allows scientists to understand how machines solve problems. Some important fields in which automata theory is applicable are:

1. Compiler design [13]
2. Time granularity representation in databases [1]
3. Deep packet inspection in network [2]
4. Bio-Informatics [11]

Different papers have been reviewed which describe use or application of automata theory in some way or the other. The main emphasis is on how and what concepts of automata theory is being used in different fields discussed in each paper.

Discussion of these applications is done in the following sections.

III. COMPILER DESIGN

A compiler is translator that converts a program written in a high-level language into low-level language. It takes as input a program in a specific language, translates it into machine language, and executes it. Before translation, it checks various factors like tokens, syntax, semantics etc. The need of a compiler is because of the reason that computer understands only binary language, which is difficult to code in [13].

The process of translation is consists of several phases or stages. Each stage takes input from previous stage, does some processing, and gives output to the next stage. Through these stages, the source code is converted into object code. The various phases of a compiler are the following:

a) Lexical analysis
b) Syntax analysis
c) Semantic analysis

d) Intermediate code generation

e) Code optimization

f) Code generation

The diagrammatic representation of the phases of compiler is given in Figure 2.

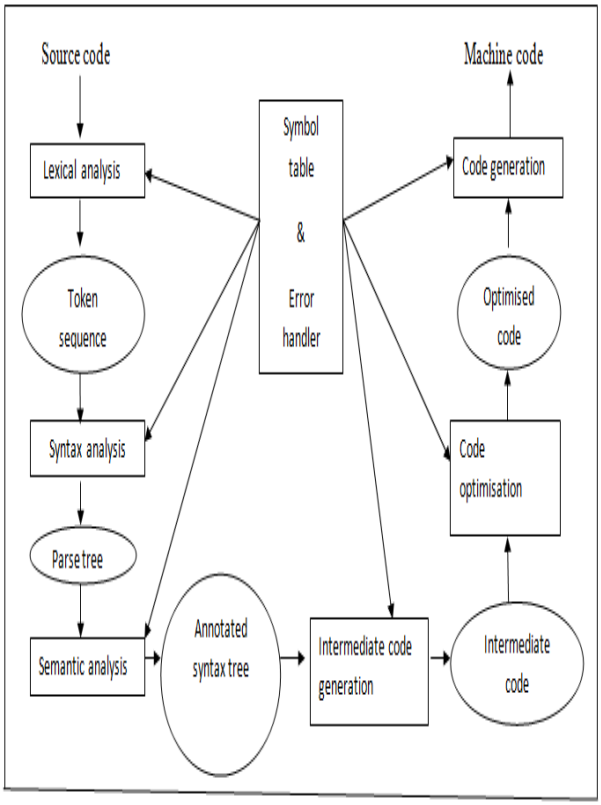

Figure 2: Phases of a compiler

The first phase of compiler is lexical analysis. In this phase the source code that comprised of large sentences are broken into small tokens. A token is the indivisible unit in a program. It can be a constant, identifier, string, operator etc.

Then it checks the tokens for their validity i.e. whether the token is a valid token or not. In order to check the validity of the tokens the programming language specifies some rules. Here, the role of automata theory and regular expressions comes into effect. The rules to check the validity are written in the form of mathematical expressions called regular expressions. The regular expressions are used for description of tokens. The tokens described using regular expressions are further recognized using finite automata.

The program that simulates the DFA for these regular expressions can be used to check whether a given token belongs to the language or not. If the token string is accepted by the DFA, then it is a valid token and can be passed to the next state, otherwise it is invalid.

The second phase of the compiler i.e. syntax analysis also has an application of another automaton. This phase is also called parsing. In this phase, the sequence of tokens which forms an expression are checked for the syntax specified by the grammar for that language and generates a parse tree or syntax tree. Due to the limitations of regular grammar, (e.g., regular grammar cannot check balancing parenthesis, since it does not have memory), in this phase we use context free grammar for syntax checking [13]. The rest of phases of compiler derive their output from the output of syntax analysis.

IV. TIME GRANULARITY

Granularity is defined as the level of refinedness or minuteness of a quantity. It means how much a system is composed of distinct pieces. For example, an hour can be broken into smaller granules like minutes, seconds, and milliseconds. The term time granularity is related to database and is called temporal granularity or timestamp granularity. In database, the time of occurring of any event is recorded by the computer and is called timestamp. Chronons are the basic elements of granularity. A chronon is an indivisible time interval of some fixed duration. So, they group together to form granules. If the timestamp granularity is one second, then the duration of each chronon is one second (and vice-versa).To store valid time date like September 12 2017 with a valid timestamp granularity of a second, we need to store it as a particular second during that day, for example, midnight September 12th, 2017 [1,5].

Regular ώ-languages are a variation of regular languages as these are those languages, which contain infinite length strings. Their accepting device is calledώ-automaton. Out of many types of ώ-automaton, one is Buchi automaton, which accepts infinite length word if on parsing the string on the automaton, it visits one of the final states infinitely often. A new class of automaton called ultimately periodic automaton, which accepts ultimately periodic words, is used. to form a subclass of Buchi automata. Since, a finite sequence of words is repeated in this language, thus, it also has inherent properties of automata for finite words like NFA.

The similarities between UPA, Buchi automaton and NFA are used to devise solution for the basic problems like inclusion problem, equivalence, optimization, emptiness, membership problem etc. for the ultimately periodic languages. Figure 3 shows the various levels of inheritance of properties in these languages.

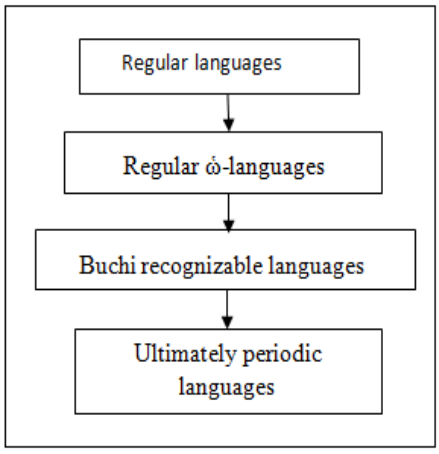

Figure 3. Levels of inheritance for Language Properties

Using this theory, we can represent the sets of time granularities in very compact form on which algorithms for manipulation can be applied easily. Then the UPA is applied to a concrete time granularity scenario taken from medical world.

Ultimately periodic languages can be expressed as finite unions of languages of the form $U\{v\}^{\omega}$, where U is a regular language of finite words and v is a non-empty finite word.[1] Such language contains strings which contain infinite length but a finite number of non-equivalent repeating patterns repeated infinitely.

The paper takes a real world example of clinical medicine and applies the theory of Ultimately Periodic Languages to solve a problem. The problem is taken from the medical domain of heart transplant patients. Post transplantation guidelines are given to the patients which tell them about their required drug dosage and periodical visits for life. These requirements are collected in the form of protocols, which specify the schedules for the therapies and the frequency of the check-ups. An excerpt of the guidelines for a heart transplant patient is used in the paper [6]. Depending on the physical conditions of the patient, the guidelines can require, together with other treatments, an estimation of the glomerular filtration rate (GFR) with one of the following schedules [1]:

a) 3 months and 12 months post transplantation and every year thereafter;
b) 3 months and 12 months post transplantation and every 2 years thereafter.

Here, the therapies are in the form of time granularities of months or years. These protocols involve unanchored granularities (independent of the starting point of therapy), and sets of granularities with different repeating patterns, which represent the set of different periodicities of the therapies. Therefore, the two problems are:

1. Different protocols can be specified for the same class of patients by different institutions. In such case, we need to check whether two protocols specify same set of therapies. This is similar to the equivalence problem where we check if two languages contain same strings.
2. Checking whether a given therapy/granularity assigned to a patient belongs to the set of therapies/granularities of the protocol. This can be solved using the consistency checking problem algorithm.

V. DEEP PACKET INSPECTION

Deep packet inspection is a method of filtering the packets of a network and managing their flow. It examines, unlike other inspection methods, not only the header part of the packet but also the data part. Thus, it looks for anything in the packet, which is not according to the protocol, or anything hindering the protocol like viruses, intrusions, etc. and decides whether the packet may pass or if there is a need to divert to a different destination. It is used in the application layer of the Open Systems Interconnection model. [2]

In a network, messages are sent in the form of packets of data. Packet content scanning is important process due to factors like network security, network monitoring, HTTP load balancing, etc. In this process, the packet payload is matched against a set of regular expressions to check whether it contains valid content (that is allowed to pass through the network) or not. Thus, we get another application of automata theory in the field of networking in order to scan the packets.

Regular expressions are used in pattern matching because they can easily express any string pattern. A finite automaton can be drawn for a regular expression. This paper first tells us various existing solutions for regular expression matching using these automata.

In networking applications, Z(set of input symbols) contains the *28* symbols from the extended ASCII code in dfa as well as nfa. According to a study on worst-case scenario, [8] a regular expression whose length is n can have an NFA with $O(n)$ states. It's corresponding DFA may have $O(Z^n)$ states since for each input symbol in Z, there is a possibility on n different transitions. Processing complexity for each input character is $O(1)$ in a DFA, but is $O(n^2)$ for an NFA with all n states active at the same time, because for each input character there can be n transitions at maximum. To handle m regular expressions, two choices are possible:

a) Processing them individually in p automata
b) Compiling them into a single automaton.

If we process p automata individually, this creates large numbers of active states which leads to worst case complexity as the sum of p separate NFAs. This method can be slow, since for each input symbol, each active state must be serially examined to obtain new states. In DFA-based systems, we can easily create the combined regular expression for the p regular expressions and thus a composite DFA can be produced. This provides better performance over running p individual DFA. Specifically, a composite DFA reduces processing cost from $O(p)$ ($O(1)$ for each automaton) to $O(1)$, i.e., a single lookup to obtain the next state for any given character since in a DFA, for each input symbol on a state ,there can be only one transition. However, the number of states in the composite automaton grows to $O(Z^{pn})$ in the theoretical worst case. [2]

Between DFA and NFA, there is what is called lazy DFA. Lazy DFA are designed so as to reduce memory consumption of DFA [9][10]: a lazy DFA keeps a subset of the DFA such that it contains the commonly used strings in memory. Thus, it provides good performance for common input strings. But a disadvantage of this approach is that malicious senders can easily construct packets that keep the system busy and slow down the matching process by knowing the common strings.

Using parallel processing techniques we can speed up the regular expression matching process. But these approaches have limitations like the cost of hardware and embedding. In this paper a new DFA based approach on general purpose processor has been devised. The focus of the study is to reduce the space complexity of DFA and getting the processing speed of $O(1)$ per character. In order to achieve this, a method of regular expression rewrite techniques has been devised. In this method, instead of exhaustive matching we use non-overlapping matching.

Exhaustive Matching: In this type of matching, the string is scanned for any substring that matches with the given regular expression. This type of matching is expensive and often unnecessary to report all matching substrings.

Non overlapping: In this matching process, all non-overlapping substrings that match the pattern are given as output.

For the pattern *10** and the input *1000*, the three matches overlap by sharing the prefix *10* (*10, 100, 1000*). Thus, here non-overlapping matching will give only one match rather than giving three overlapping matches.

In order to avoid interaction between DFAs (which is the phenomenon in which the composite DFA of two DFAs contains more number of states than the sum of the states of the two DFAs), so that an improvement of speed is there, a grouping scheme has been developed. In this scheme, the p patterns are divided into k groups such that patterns in each group do not interact with each other. These algorithms reduce the time complexity of computation from $O(p)$ to $O(k)$ without causing extra memory usage. Thus, a new DFA-based packet scanner using the above techniques is implemented. Figure 4 shows the methodologies used for deep packet inspection in the paper being discussed.

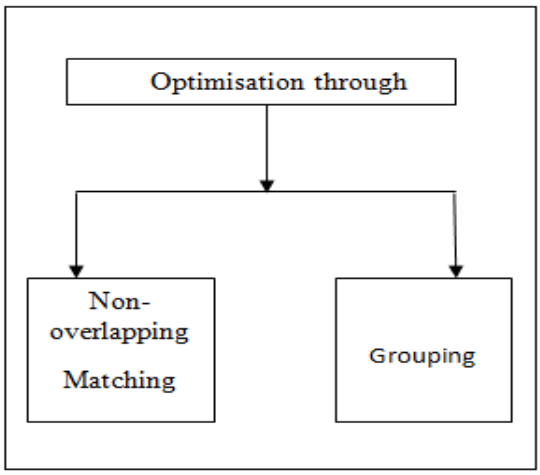

Figure 4. Methodologies used for fast and memory efficient packet inspection

VI. BIO INFORMATICS

Cellular automaton is a type of automaton, which consists of a grid of cells. Here, each cell acts like an individual automaton and thus it is in one of the finite number of states. The grid can be one dimensional, two dimensional or n dimensional. The set of cells present near a cell are called its neighbourhood. Each cell is assigned an initial state at time t=0. By applying some rules on each state and advancing time by 1, a new state is obtained in terms of the current state for each cell and its neighbourhood. This creates a new generation of the cells. Generally, the rules for updating the state of cells are the same for the entire grid [3].

Cellular automata can be used to represent the behaviour of complex systems in nature. They perfectly depict the behaviour of self-reproducing systems. Thus, they have been studied extensively in the natural sciences, mathematics, and in computer science. [4]

According to some studies conducted, it has been found that mutagenesis is not completely random phenomenon. Thus, the changes in the DNA sequence that it leads to, are somewhat predictable [11][12]. Using this observation, this paper talks about a software tool called DNA_EVO that can determine the rules of DNA evolution using algorithms which take as input the DNA sequences of different generations, and if the evolution rules are available then it can give the DNA sequence in any previous evolution step for which the exact sequence information was not known.

Two main assumptions have been made in this paper:

1. **No role of natural selection:** All the evolution that happens is due to mutation only and there is no role of natural selection.
2. **Effect of neighbor nucleotides:** Mutagenesis of a nucleotide is also influenced by the identity of the surrounding nucleotides.

One dimensional cellular automaton has been used to model the DNA sequence. A collection of cells of cellular automata forms the DNA with each cell representing a nucleotide. Four states are possible of each cell, namely, A, T, G and C and quaternary numbers are used to represent these states. The cellular automaton representing DNA evolves at discrete time step. In evolution, the state of one or more cells changes. The time interval between two stages of evolution is called a time step. According to assumption 2, the state of a cell during evolution will be affected by the neighbour cells. So, the state of a cell at any time step t is a function (evolution rule) of the states of neighbour cells at t-1 and its own state at t-1.

$$State(j,t) = f(State(j-r,t-1),...,State(j-2,t-1),State(j-1,t-1),State(j,t-1),State(j+1,t-1),..,State(j+r,t-1))$$

Where, State (j, t) represents state of jth cell at time step t, f is the evolution rule and r is the neighbourhood size.

Also, in this paper, linear evolution rules have been considered and thus, square matrices are used to represent f i.e. the evolution rule here is a square matrix of states of cells at t-1 stage. This matrix contains only 0 or 1 as elements and has dimensions n x n, where n is the number of cells in the DNA. Thus, if we have the DNA sequence of various generations, then we can find the evolution rule i.e. matrix f. Using this rule, we can predict future generation sequence of DNA or previous generation's sequence if it was not known. In order to find the matrix f, we have various possible options of placing 0 and 1. If we consider four state CA and two cells in neighbour (one on left and one on right) and one cell itself, the number of possible rules is 4^{64}, which is quite large. In order to find suitable rule from the search space, the paper has used genetic algorithm.

Genetic algorithms (GA) can efficiently find solutions for problems with complex search space. These algorithms are based on the concepts of natural selection and genetics. In this algorithm, the solutions to the optimisation problem are representedin the form of a fixed length tuple, which represents a 'chromosome'. Each solution in this tuple/chromosome is a 'gene'. A set of chromosomes form the population.

A target function is used to evaluate the solution represented by each tuple. So, the target function evaluates the 'fitness' of each chromosome and selects chromosome with highest fitness as the optimal solution. Then it produces next generation using operations like, natural selection, crossover and mutations.

1. **Natural selection:** This process selects the solutions with high fitness values and discards solutions with low fitness values.
2. **Crossover:** In crossover, if z is the length of the chromosome, then a crossover point is chosen anywhere between 1 to z-1 and the portions of the two chromosomes beyond the crossover point are exchanged. This is done to get better combinations from two highly fit chromosomes.
3. **Mutation:** A random gene of a chromosome is selected and its base is changed or in terms of CA, the state of the cell is changed.

Figure 5 shows the flowchart for the genetic algorithm.

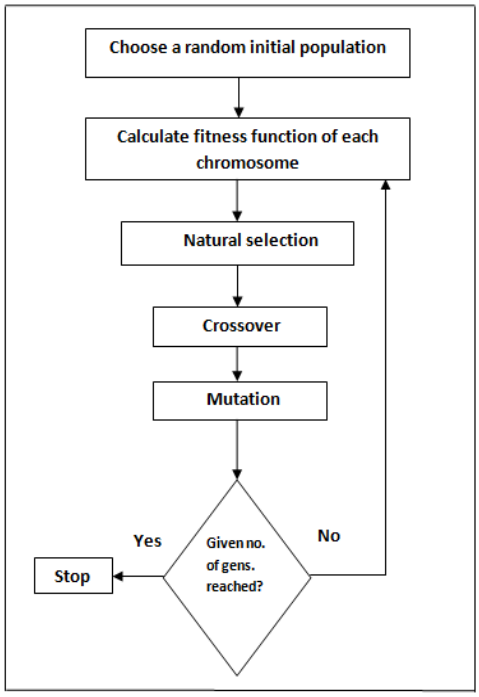

Figure 5. Flowchart for Genetic Algorithm

In this way, the tool finds the evolution rule and thus can be employed to predict sequences for future generations. The developed tool can also be used to test various parameters that could influence evolution.

VII. COMPARISION AND CONCLUSION

This paper is aimed at giving an idea of application of Theory of computation in various fields. Review of four papers, which show application of theory of computation in fields like compiler design, database systems, Deep packet Inspection and bio informatics, is done.

Table 2 shows the comparison between different parameters of various applications of automata discussed. From the various applications discussed, the common thing in all of them is that there are some states, some input and some computation to be done. Thus, Theory of computation is applicable anywhere there is computation to be done. Discussion about how various types of automata have been used in different applications to solve the problem is done and also about some newly devised automata like ultimately periodic language automata.

Also, from the comparisons done between the three applications, it can be concluded that each automaton can be exploited to represent a real life computation problem and using the existing machines, new machines can be devised to solve our purpose.And from these applications, an important thing to be learnt is that how to correlate the theoretical concepts with real life. There are many other applications of automata theory for example in image processing, in chemistry, biology and earthquake measuring and sensing etc. in which researches are going on. They have been used as models of physical and biological phenomena, such as fluid flow, galaxy formation, earthquakes, and biological pattern formation.

Table 2. Comparison of the above discussed applications

Application	Compiler design	Time granularity	Packet inspection	DNA evolution
Automaton used	DFA & PDA	Buchi automaton	DFA	Cellular automaton
Mode of implementation	Algorithm	Algorithm	Algorithm	Software tool
Property of automaton exploited	Pattern checking using regular expressions(DFA) & memory element (PDA)	Accepting infinite length strings.	Compact and easily constructible from regular expressions	Represents self-reproducing systems efficiently
Algorithm used/developed	Algorithm for pattern recognition and grammar checking	Algorithms for inclusion, equivalence and size optimisation for UPA	Algorithms for non-overlapping pattern matching and grouping	Genetic algorithm for finding evolution rule
Benefit	Conversion of one language to other ,which is an ease to user	Representing and reasoning of sets of time granularities in database	Fast and memory efficient method devised	Prediction of DNA sequences

VIII. FUTURE SCOPE

A natural development of the present work of theory of ultimately periodic languages is the definition of a high level logical language, e.g., a variant of propositional linear temporal logic [7] that allows one to represent all UPA-recognizable languages by means of suitable formulas. In the second application, the new DFA based method devised can prove to be really cost efficient and productive and thus can become widely used. Related to application in packet scanning, in the future, further study to apply different DFA compression techniques and explore tradeoffs between the overhead of compression and the savings in memory usage would be useful. Since this tool has made DNA predictions possible, it can further be used to generate DNA sequences with highly desirable properties for an organism.

IX. REFERENCES

[1]. Bresolin, D., Montanari, A., & Puppis, G. (2009), "A theory of ultimately periodic languages and automata with an application to time granularity", Acta Informatica, Vol. 46, Issue 5, pp. 331-360.

[2]. Yu, F., Chen, Z., Diao, Y., Lakshman, T. V., & Katz, R. H. (2006), "Fast and memory-efficient

regular expression matching for deep packet inspection. ACM/IEEE Symposium Architecture for Networking and Communications systems (ANCS), pp. 93-102.

[3]. Andrews, G., & Dobrin, A. (2005), "Cellular Automata and Applications", pp. 1-6.

[4]. Mitchell, M. (1996). "Computation in cellular automata: A selected review". Nonstandard Computation, pp. 95-140.

[5]. Bettini, C., Jajodia, S., & Wang, S. (2000), " Time granularities in databases, data mining, and temporal reasoning", edition 1, Springer Science & Business Media, pp. 11-18.

[6]. Loma Linda International Heart Institute: Paediatric heart transplantation protocol. Tech. rep., International Heart Institute, Loma Linda University Medical Center, Loma Linda, CA (2002). Available at: http://www.llu.edu/ihi/pedproto.pdf

[7]. Emerson, E. A. (1990), "Temporal and modal logic", Handbook of Theoretical Computer Science: Formal Models and Semantics, MIT press, pp. 995-1072.

[8]. Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2001), "Introduction to automata theory, languages, and computation". ACM Sigact News, Vol. 32, Issue 1, pp. 60-65.

[9]. Green, T. J., Gupta, A., Miklau, G., Onizuka, M., & Suciu, D. (2004), "Processing XML streams with deterministic automata and stream indexes" ACM Transactions on Database Systems (TODS), Vol. 29, Issue 4, pp. 752-788.

[10]. Sommer, R., & Paxson, V. (2003), "Enhancing byte-level network intrusion detection signatures with context" Proceedings of the 10th ACM conference on Computer and communications security, pp. 262-271.

[11]. Mizas, C., Sirakoulis, G. C., Mardiris, V., Karafyllidis, I., Glykos, N., & Sandaltzopoulos, R. (2008), "Reconstruction of DNA sequences using genetic algorithms and cellular automata: Towards mutation prediction" Biosystems, Vol. 92, Issue 1, pp. 61-68.

[12]. McFadden, J., & Al-Khalili, J. (1999), "A quantum mechanical model of adaptive mutation" Biosystems, Vol. 50, Issue 3, pp.203-211.

[13]. Ullman, J. D. (1972), "Applications of language Theory to Compiler Design", Proceedings of the May 16-18, 1972, spring joint computer conference, pp. 235-242(ACM).