

**Computer Science Department -4th Year**

****

*Future Institute of Engineering and Management*

*Sonarpur Station Road,*

*Kolkata-700150*

**Finite State Automata Simulator**

A Dissertation Submitted In Partial Fulfillment of

the Requirements for the “Bachelor of Technology”

(B.Tech.) Degree in Computer Science

By

**Sujit Kanti Biswas**

Reg. No. 148010111029 of 2004-2005

Roll No. 14801041031

**Chandrachur Chakravarti**

Reg. No. 148010111014 of 2004-2005

Roll No. 14801041016

**Sanjib Chakraborty**

Reg. No. 148010111030 of 2004-2005

Roll No. 14801041032

**Pinaki Kundu**

Reg. No. 148010111025 of 2004-2005

Roll No. 14801041027

**Saptarshi Sengupta**

Reg. No. 148010111026 of 2004-2005

Roll No. 14801041028

Under The Sincere Guidance of

**Mr. Abhijeet Saha**

At

**Future Institute of Engineering and Management**

[AICTE approved & Affiliated to West Bengal University of Technology (WBUT)]

To Whom It May Concern

This is to certify that Sujit Kanti Biswas (14801041031), Chandrachur Chakravarti (14801041016), Sanjib Chakraborty (14801041032), Pinaki Kundu (14801041027), and Saptarshi Sengupta (14801041028), students of B.Tech , Computer Science and Engineering of the session 2004-2008 from *Future Institute of Engineering and Management*, have successfully carried out the dissertation titled **“Finite State Automata Simulator”** under my guidance and supervision in the college laboratory. The dissertation is carried out for the partial fulfillment of the requirements of the B.Tech. Degree course.

It is also certified, to the best of my knowledge that, apart from this group, no other students of the current batch has carried out similar work as part of their project assignment during the said period.

They have sucessfully completed the project in time and their performance is satisfactory. I wish them every success in their future life.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Mr. Abhijeet Saha**

**Sr. Lecturer**

[Dept. of Computer Science]

**Future Institute of Engineering and Management**

Acknowledgement

First and foremost we would like to thank our guide Mr. Abhijeet Saha, Lecturer, Dept. of Computer Science, Future Institute of Engineering and Management for his valuable and sincere guidance and utmost patience.

We would also like to acknowledge all other faculty and staff members of the FIEM family for the help and the requirement they had provided in the long journey that ends with this work.

Further we would like to thank Dr. Aloke Ghosh, Principal, and Mr. Abhijeet Bhattacharya, Head of Computer Science and Engineering department, Future Institute of Engineering and Management for the encouragement they have provided. We are also very much thankful to Mr. Sourav Pradhan, for imparting knowledge of this subject and motivating us to continue our study with this subject.

Finally we thank our parents for the patience they had shown during the entire tenure of our work.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Sujit Kanti Biswas**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Chandrachur Chakravarti**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Sanjib Chakraborty**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Pinaki Kundu**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Saptarshi Sengupta**

7th May, 2008

**Future Institute of Engineering and Management**

[AICTE approved & Affiliated to West Bengal University of Technology (WBUT)]

Contents

1. Introduction…………………………………………………………06
   1. Components of the system………………………………… 06
   2. Components Description………………………… ………..06
   3. Technical Specification……………………………………. 07
2. Requirements………………………………………………………08
   1. Acquiring requirements………………………… 09
   2. Model Analysis……………………………………………… 09
   3. Functional Requirements……………………… 10
   4. Non-functional Requirements…………… 14
3. Analysis and Design…………………………………………… 14
   1. Data Flow Diagrams…………………………………… 15
4. Implementation………………………………………………………20
   1. Architecture…………………………………………………… 20
   2. Description……………………………………………………… 21
   3. Technology………………………………………………………22
5. Future Prospects and Stability……………… 22

Section 1: **Introduction**

In today’s scenario, the whole concept of study and acquiring knowledge has changed drastically. The students in the fields of science and technology are very inquisitive now-a-days. The subjects they study, besides knowing the theory part, they also emphasize more and more on the practical woks on that subject too, as by doing the practical only, one can understand the subject of study more vividly, more comprehensively. It makes one understand how a module under the subject works in real life, mainly the functionalities of it. Our Project mainly deals with the theory of Automata and regular expressions. Since this subject is quite vast in its own field, the students find it quite difficult to understand. The Simulator program developed by us will certainly provide some easy way to help make the students understand this subject well for their own benefits only.

Our project guide and teacher, Mr. Abhijeet Saha, gave us proper guidance and support towards completion of this project in due course of time.

Section 2**: Objective**

1. Demonstrate a sound understanding of the algorithms for: (i) Constructing an NFA for a formal language from the definition of it as a regular expression. (ii) Converting an NFA to it’s equivalent DFA. (iii) Optimizing the given DFA.

2. Implementing the above algorithms with a sufficiently friendly “Graphical User Interface” to serve as a tool for simulation purpose in the field of education & research.

Section 3**: Innovativeness & Usefulness**:-

This Project can be considered to be innovative in the following respects:-

1. Can be used for classroom & laboratory demonstration to the students for better understanding of the subjects considered (Automata Theory, Language Processor, etc).

2. The scope of this simulator can be extended by implementing other Automations like Pushdown Automata, Turing Machine, etc. for use in the field of R & D for various simulation purposes.

**Description of the Project**

A detailed and in-depth analysis of the theory of Regular expressions and Automata resulted in the whole package to consist of the following components. The list below is anything but exhaustive, as the need to expand it can come up at any point of time depending on an even deeper analysis of the system. The project is at its budding state so it’s bearable to some extent.

We built a Simulator Program which will provide the following services:

* We will be able to provide as input a valid Regular expression in a user-friendly Graphical interface.
* There will be an error message for any type of wrong input or inputs violating the theory of Regular expressions (like the entry of “ ^ “ character is not allowed in the regular expressions as it indicates “ NULL “ state in proper defination).
* The output window will provide the user with 3 graphs, viz. First showing the NFA states, the Second one showing non-optimized DFA graph and the 3 rd one showing the final Optimized DFA graph.
* The detailed Graphs with the colored nodes will be comprehensive for the users to understand the gradual process of development of how the Null steps get minimized in due process to the final optimized state.
* The colored nodes will help users to distinguish between different types of Automata states.
* The new “ Drag-and-Drop “ feature of the Graphical window will enable the user to modify the simulated graphs and arrange the different nodes created in any position within the window domain to their own convenience.
  + Algorithms Used for Project Development:-
    1. Thompson’s Construction
    2. Subset Construction & Epsilon Closure
    3. DFA optimization algorithm

**1. THOMPSON’S CONSTRUCTION**

(i) Input:–A regular expression ‘E’ over an alphabet ‘∑’.

(ii)Output: - An NFA ‘N’ accepting L (E).

(iv)Steps:-

 (a) For E=Ø,



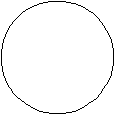
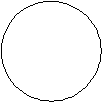
 (b)For E=**^**, **^**

***ending state***

****** (c)For ‘a’ in ∑, construct the NFA, a ***ending state***

(d)Suppose N (F) and N (G) are NFA’s for regular expressions F and G:

(1) For regular expression F+G, construct NFA

 **^** **^**

****

****

**** **^** **^**

Mg

(2)For regular expression=F.G, construct composite NFA, N (FG):

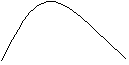
Mf

Mg

 ^

(3) For the regular expression F\*, construct composite NFA, N (F\*):



Mf

   ^

 ^ ^



^

(4) For the parenthesized regular expression (F), use N (F) itself as the NFA.

**2. SUBSET CONSTRUCTION (Construction a DFA from an NFA) & EPSILON CLOSURE.**

*Input:*  An NFA N

*Output:* A DFA D accepting the same language

*Method:*  This algorithm constructs a transition table *Dtran* for D. Each DFA state is a set of NFA states and we construct *Dtran* so that D will simulate “in parallel“ all possible moves N can make on a given input string.

We use the operations shown in fig. below to keep track of sets of NFA states (s represents an NFA state and T a set of NFA states).

|  |  |
| --- | --- |
| OPERATION | DESCRIPTION |
| *ε-closure(s)* | Set of NFA states reachable from NFA state *s* on *ε*-transitions alone. |
| *ε-closure(T)* | Set of NFA states reachable from some NFA state *s* in T on *ε*-transitions alone. |
| *move(T,a)* | Set of NFA states to which there is a transition on input symbol *a* from NFA state *s* in T. |

Initially, ε-closure (s0) is the only state in *Dstates* and it is unmarked;

**while** there is an unmarked state T in *Dsates* **do begin**

mark T;

**for** each input symbol a **do begin**

U: = ε-closure (move (T,a));

**if** U is not in *Dstates* then

add U as an unmarked state to *Dstates* ;

*Dtran* [T,a] := U

**end**

**end**

***Subset construction algorithm***

push all states in T onto *stack*

initialize ε-closure(T) to T;

**while** *stack* is not empty **do begin**

pop t, the top element ,off of stock ;

**for** each state *u* with an edge from *t* to *u* labeled *ε* **do**

**if** *u* is not in *ε*-closure(T) **do begin**

add u to ε-closure(T);

push u onto *stack*

**end**

**end**

***Epsilon-Closure Algorithm***

**3. DFA OPTIMIZATION ALGORITHM**

*Input:* A DFA M with set of states *S*, set of inputs ∑, transitions defined for all states and inputs, start state s0, and set of accepting states F.

*Output:* A DFA *M’* accepting the same language as *M* and having as few states as possible.

*Method:*

1. Construct an initial partition ∏ of the set of states with two groups: the accepting states *F* and the non-accepting states *S -F.*

2. Apply the procedure (\*) described below to ∏ to construct a new partition ∏new.

3. If ∏new= ∏, let ∏final =∏ and thus continue with step (4).Otherwise, repeat step (2) with ∏:= ∏new.

4. Choose one state in each group of the partition ∏final as the *representative* for that group. The representatives will be the states of the reduced DFA *M’*. Let *s* be a representative state, and suppose on input a there is a transition of *M* from *s* to *t*. Let *r* be the representative of *t’s* group (r may be t). Then *M’* has a transition from *s* to *r* on *a*. Let the start table of *M’* be the representative of the group containing the start table s0 of *M*, and let the accepting states of *M’* be the representatives that are in *F*. Each group of ∏final either consists only of states in *F* or has no states in *F*.

5. If *M’* has a dead state, i.e. a state *d* that is not accepting and that has transitions to itself on all input symbols, then remove d from *M’*. Also remove any states not reachable from the start state. Any transitions to *d* from other states become undefined.

The procedure (\*):-

**for** each group *G* of ∏ **do begin**

partition *G* into subgroups such that two states *s* and *t* of *G* are in the same

subgroup if for all input symbols *a*, states *s* and *t* have transitions on

*a* to states in the same group of ∏ ;

/\* at worst, a state will be in a subgroup by itself \*/

replace G in ∏new by the set of all subgroups formed

**end**

***Construction of ∏new***

**COMPONENT DESCRIPTION**

A very brief description of each component has been given below:

**TECHNICAL SPECIFICATION**

Coming to the technologies to be used and the requirements for the development of this project it must be mentioned that it is mainly based on open-source software which means it will certainly be very minimal as far as costing is concerned.

* **Software**

Requirements to run the Simulator:

S/W requirement: - Java (JDK 1.6 Update 4). Appropriate to the platform in which this

application is used.

O/S requirement: - Platform Independent.

* **Hardware**
  + **System requirements**

Processor - Pentium 3 1.2 GHz

RAM - 256 MB RAM

Minimum Disk Space Required- 1 GB

Chapter 2: **Requirements**

The requirements section is one which consists of the following steps:-

1. Acquiring requirements
2. Model analysis
3. Functional requirements
4. Non-functional requirements

**a) Acquiring requirements**

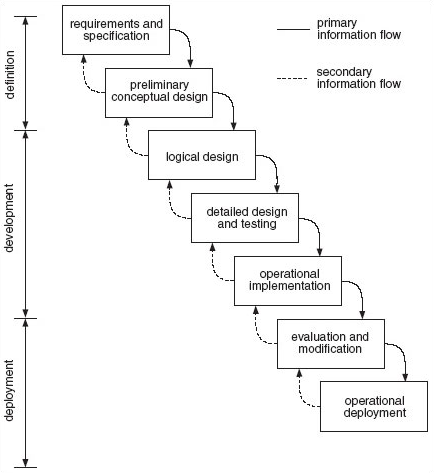
This phase consisted of obtaining what everyone wants to be incorporated in the system. For obvious reasons some parts of the system, which are not included in, the software have been entirely subsided from being any part of this entire project documentation. Henceforth only those entities shall be referred to who have an active participation in the automated system and not otherwise.

The professionalism of the project lies in the fact that all these requirements have been obtained in lieu of special permission, given and authorized by the teachers of our department, which entitles us to visit every department we wish to enquire into and know (to certain limits) their method of working, the work flow and the internal framework of how basically that department works.

The very motive of making a system automated is that any automated system is basically the ‘best approximation’ towards simulating a manual system, the reason being that a machine can in no way imbibe the immense dynamicity of the human brain and work culture.

**b) Model analysis**

This phase falls more into the abstract portion of the project than the physical one. It deals with the choice of the model, which will aggravate the propagation towards the design and analysis section and shall finally culminate with the implementation or coding of the software. Having said that it must be told that the very roots of the projects lie in this phase, i.e. the choice of the model will actually work as a cornerstone for the project. Coming straight to the point, a close look at the software is enough to understand that the model followed here is the Iterative Waterfall Model.



To follow the Iterative Waterfall Model, we proceeded from one phase to the next in a purely sequential manner. For example, we first completed "requirements specification" — then set in stone the requirements of the software. When the requirements were fully completed, we proceeded to design. The software in question was designed and a

"Blueprint" drawn for implementation (coding) to follow — this design would be a plan for implementing the requirements given. When the design was fully completed, coding made an implementation of that design.

Although this model is not a popular trend any more in the new development industry, the utter simplicity of this project lends itself to be a natural client of the Iterative Waterfall model. Without going into the details of the Waterfall model it must be told that the following advantages were offered by this model in course of time and justifies that the choice of the model was right by all means.

**√** a level by level analysis facility

**√** ability to be developed in short time – as we had very limited time

**√** suits into our decentralized working manner

**√** supports exhaustive simulation

The above points show that the model is almost perfect for the project we have undergone.

**c) Functional requirements**

Here we have defined the internal workings of the [software](http://en.wikipedia.org/wiki/Portal:Software): that is, the workings, technical details, data manipulation and processing and other specific functionality that show how the [users](http://en.wikipedia.org/wiki/Use_case) are to be satisfied. They are supported by non-functional requirements, which impose constraints on the design or implementation (such as performance requirements, or design constraints).

As defined in requirements Engineering, functional requirements specify specific behaviors of a system. This should be contrasted with non-functional requirements, which specify overall characteristics such as cost and reliability. (An alternative view is that functional requirements specify specific behavior while non-functional provides adjectives, which may be used to describe these behaviors.)

Typically, as requirements analysts we have generated functional requirements after acquiring the requirements from faculty members and students. However this might have had several exceptions since software development is an iterative process.

To the best of our knowledge the software requirements have been assumed to be clear, correct, unambiguous, specific, and verified. This brings us to the very point which represents the functional requirements which are as follows:-

* **Identification of Need**

Since it had been thought of that the application will consist of a rich graphical front-end, a need for a programming language which will provide us with the necessary facilities required to develop the said project came into existence. It would have been a very tedious job for us to develop the proposed application using the programming language like C.

Hence, it was decided to use JAVA Technology (J2SE) by which we can design standalone application with rich graphical content and user interaction facilities (like ***Drag-and-Drop***).

J2SE also contains various classes and packages which makes String manipulation less tedious.

* **Feasibility Study**

**d) Non-Functional requirements**

The non-functional requirements consist of assumptions and constraints that are bound with the software. Our application has built to have a very minimal no. of assumptions and constraints. A few of those that are present are as follows:-

* If the length of regular expression becomes very large, then the output graph becomes very congested.
* The nodes of the output graph are printed randomly on the screen and hence may overlap. In this case the user is requested to use the “**Drag-and-Drop**“ facility to re-arrange the nodes of the graph according to his/her preference.
* The nodes of the DFA graphs are labeled with renamed state name, and hence do not show the actual combination of NFA states.

Chapter 3: **Project Design**

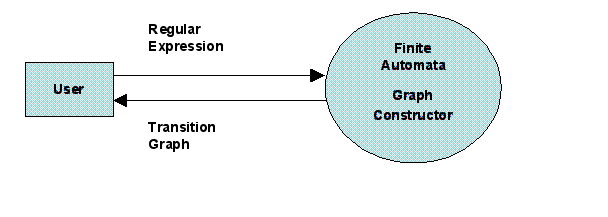
The Data Flow Diagrams show:

* The processes within the system.
* The data stores (files) supporting the system’s operation.
* The information flows within the system.
* The system boundary
* Interactions with external entities.

The analysis and design shall consist of the various design levels in the form of DFDs. They are as follows:-

**CONTEXT FLOW DIAGRAM**:

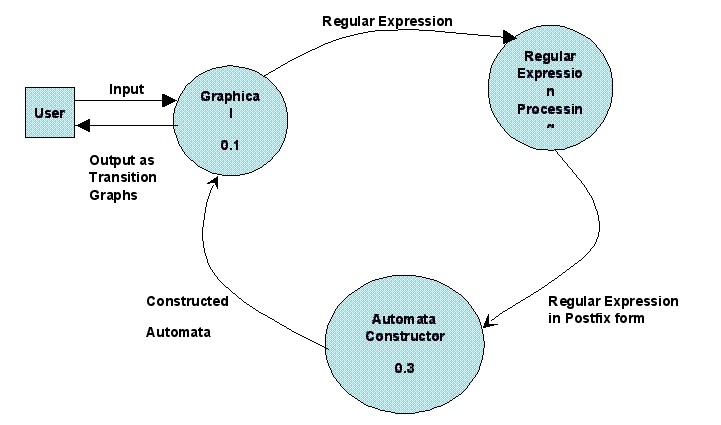
The proposed context diagram (showing user-system interaction) is given below:



**CONTEXT FLOW DIAGRAM (Level 0 DFD)**

**Level 1 DFD:**

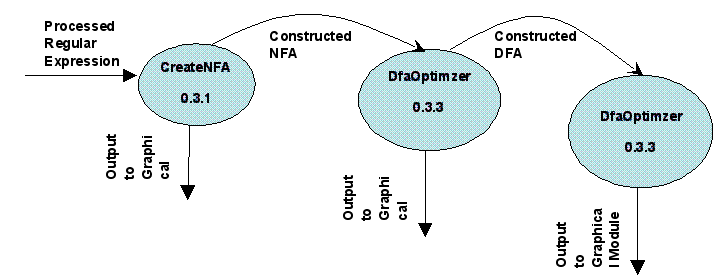
The following level 1 DFD shows the main functional modules of the system along with interaction and data flow among them.

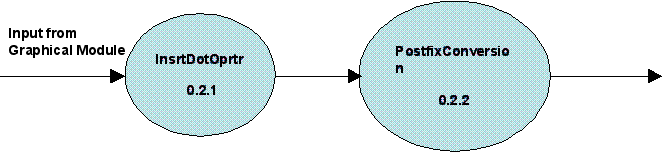


**Level 1 DFD**

**LEVEL 2 DFD:**

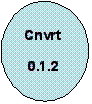
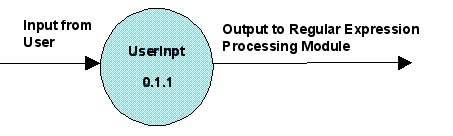
The following diagram is the level 2 DFD of the system with only expansion of those modules that are not fully expanded in level 1 DFD for clarity and simplicity of the Diagram.



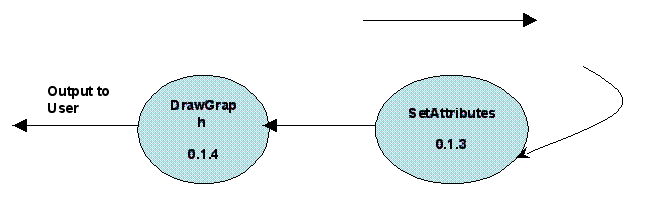


**Well formed Regular Expression**

**Output to Automata Constructor**

****

**Output from Automata** **Constructor Module**

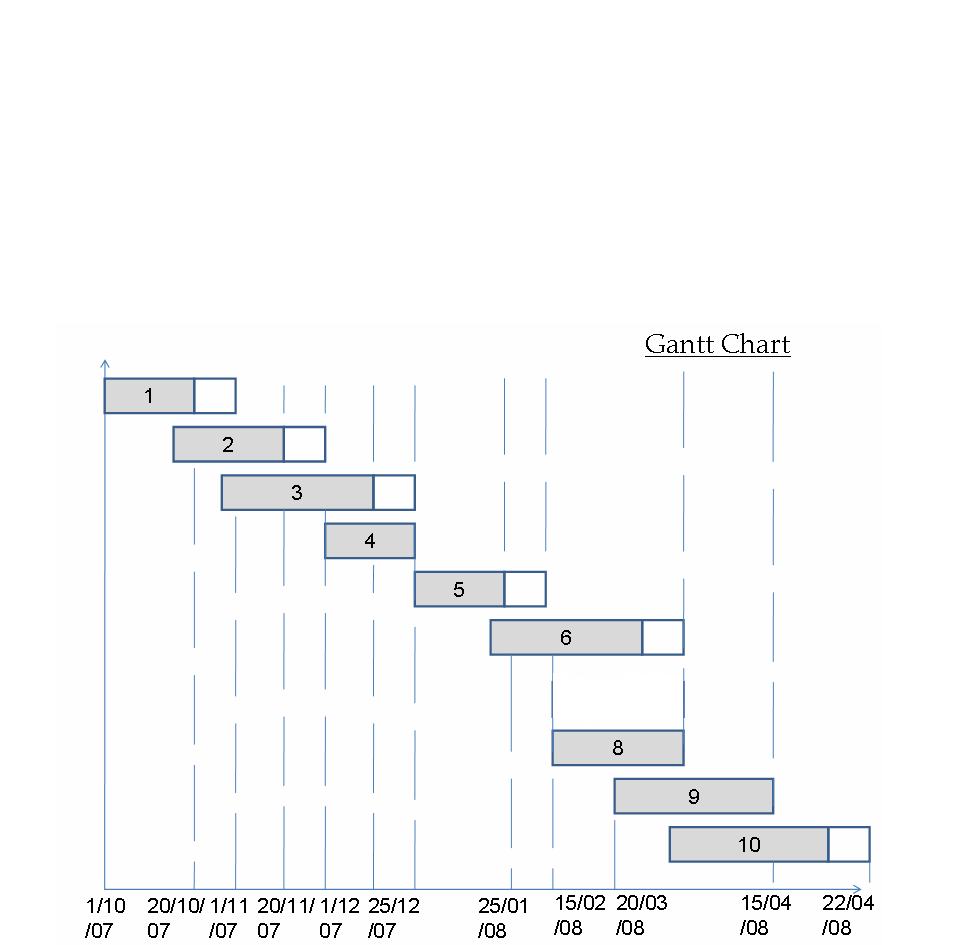
****

****

**Level 2 DFD**

|  |  |  |
| --- | --- | --- |
| **Modules** | **Submodules** | **Functions** |
| **Graphical**  **(0.1)** | **UserInpt**  **(0.1.1)** | **Generates an input window for accepting a Regular Expression from the user and sends the input data to the “Regular Expression Processing” module.** |
|  | **Cnvrt**  **(0.1.2)** | **Copy the contents of HashMap (State Table) into an array.** |
|  | **SetAttribute**  **(0.1.3)** | **Attaches a set of attributes to each Automata states.These attributes define the behaviour of a particular state i.e. the nature of the state (whether it is a start state,final state,start as well as final state or none of these) and also the location of the state where it will be displayed (x coordinate & y coordinate value).** |
|  | **DrawGraph**  **(0.1.4)** | **Accepts input from “SetAttribute” module and generates the Transition graphs as output.** |
| **Regular**  **Expression**  **Processing**  **(0.2)** | **InsrtDotOprtr**  **(0.2.1)** | **Accepts the Regular Expression, inserts the ‘.’ operator where required and constructs a well formed Regular Expression.** |
|  | **PostfixConversion**  **(0.2.2)** | **Accepts the Regular Expression and converts into it’s equivalent postfix form.** |
| **Automata**  **Constructor**  **(0.3)** | **CreateNFA**  **(0.3.1)** | **Implementation of Thompson’s Construction algorithm.**  **Accepts the postfix Regular Expression and constructs the corresponding NFA.** |
|  | **SubsetConst**  **(0.3.2)** | **Implementation of Subset Construction & Computation of Epsilon-Closure algorithm.**  **Accepts an NFA and converts it to it’s equivalent DFA.** |
|  | **DfaOptimzer**  **(0.3.3)** | **Implementation of Optimization/Minimization algorithm.**  **Accepts a DFA and converts it to it’s equivalent minimized DFA.** |

**Module Details :-**

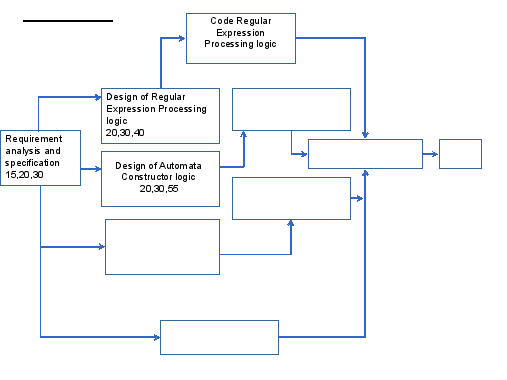
****

**DESCRIPTION OF GANTT CHART**

* **X-axis – Time**
* **Y-axis – Different Modules**

**Different modules are as follows:-**

1. Requirement Analysis and Specification.
2. Design of Regular Expression Processing logic
3. Design of Automata Constructor logic
4. Design GUI and Graph Drawing logic
5. Code Regular Expression Processing logic.
6. Code Automata Constructor logic.
7. Code GUI and Graph Drawing logic.
8. Integration and testing.
9. Writing user manual and documentation.



C

**Entity Relationship Diagram (contd …)**

hapter 4: **Implementation**

The implementation is by far the only stage where any physical involvement of any sort of computer hardware is involved. Prior to this every piece of advancement was based mainly on paperwork or thoughtful simulation of some entity which is directly or indirectly associated in any way with the system. Nevertheless it is those very pieces of exhaustive designs and works patters which enable us to finally and successfully implement the ‘thought’ into a working system. Although our system or project is void of any sort of cost considerations we did have little more freedom than professional developers in terms of time and support. This section will cover two of the most vital portions among those that deal with the implementation of the project. They are,

* Architecture followed in developing the system
* Technology used to perform the actual coding of the project

The third and final part of any implementation, i.e. the actual code shall be submitted as a separate document which shall consist of several hundred pages of well structured, optimized, and properly commented code that, when compiled and tried, will show us perfectly how the system works in effect.

**Architecture**

An integral part of hassle-free implementation of any software is to follow some sort of framework that suits the developers as well as serves the purpose to the best of approximations, if not perfect it. It is here that the innovativeness can make a world of difference both to the developers and to those who are judging the piece of work. At the beginning of this session (a ‘session’ being the total time given for designing and implementing the project) we literally let our imagination fly and from our technical knowledge accompanied with whatever past experience all of us have had, we converged upon a self-thought framework which is quite modern by current standards and suits our technical finesse by all measures. Henceforth we could proudly say that this framework is our brainchild and has several salient features, which are not only unique but also developer-friendly and well structured from head to toe. Without any more exaggeration we hereby present below the very model we’re talking about.

Chapter 5: **Future Prospects and Stability**

The future prospects and stability is an insight into the factors that measure the deviation of the performance of the designed Software from a set benchmark level. It is also an assurance on our part towards betterment of the Software in terms of numbers and quality of services to be offered.

This simulation software can act as a basis for a monumental work on Simulation Programming which can provide a complete package for Simulation based on Automata Theory.

Other popular algorithms like Pumping Lemma, Membership algorithms and various other Automatons like PDA, Turing Machine, etc. can be added to this existing software to increase it’s area of application.

Adding simulation features for various parsers like LR Parser can make this simulation package more popular in the field of Language Processor.

To make the long story short this application served as both a working prototype of an automated system, and a necessary part of our course’s Final year project. This brings us to the very limitation under which the project thrives:-

1. Time – An apparent 4 months and an actually allotted 2 months of design has greatly affected the multiplicity and efficiency of the design of the system. Had we been given more time towards the design of the project then the system would have been more robust and service-heavy.
2. Environment – The environment provided fell short of being adequate enough to lend itself to be converted to a fully automated system. In such cases as ours the least that the developer expects is a fully networked structure waiting for automation, something that was available only in a laboratory atmosphere.
3. Professionalism – Considering the above factors a certain lack in professionalism on our part was evident by all means, and which has probably not brought out the best from within us. This was unprecedented and thus has indirectly acted as a limitation.

Nevertheless, as the popular saying goes, - *“when the going gets tough, the tough gets going”*, we have tried our very best to overcome any limitations whatsoever and have designed and implemented what will be known as the very first prototype of a totally automated Regular Finite Automata Simulator in Future Institute of Engineering and Management, and we think very humbly that we do deserve a little bit of appreciation, if not be proud of it.

All said and done, even a very quick look at the application will enable anyone to notice that it can be easily implemented. The bigger picture might present a situation in which this may also be uploaded onto the web (the Internet) and made accessible to everyone throughout the world.

Finally we, the developers, have to say that although we do not know what the future holds for us, we can make the best conjecture and work towards making this world a much faster and better place to live in, - an initiative to which we have made our little contribution by developing the very system that we have been talking about for so long. Thank you.

**Bibliography**

* Alfred V. Aho, Ravi Sethi, Jeffrey D.Ullman, **Compilers-Principles, Techniques, and Tools,** 2006 Pearson Education.
* Peter Linz, **An Introduction to Formal Language and Automata,** Narosa Publication.
* Herbert Schildt, **The Complete Reference-Java J2SE 5 Edition**, Tata McGraw Hill
* Sun Java Tutorial, downloaded from [www.sun.com](http://www.sun.com/)