Implicit Conversion of Deterministic Finite Automata to Turing Machine

Tulashiram B. Pisal
Department of Computer Science
Sinhgad Institute of Computer Sciences, Pandharpur(MS), India

Dr. Arjun P. Ghatule
Controller of Examinations,
Yashvantrao Chavan Maharashtra Open University, Nashik (MS), India

ABSTRACT: Theory of Computer Science (TCS) is mathematical based subject. We knew that several Computational Science learners face difficulties in designing and understanding of Finite Automata (FA), Push Down Automata (PDA) and the Turing Machine (TM) because the logic has been changing according to examples. FA accepts only regular language while TM machine accepts all types of languages. Literature divulges that JFLAP (Java Formal Languages and Automata Package) is a software tool which is obliging for designing FA, PDA, and TM manually. In JFLAP the designing of DFA, as well as TM by manually, is possible but the conversion of DFA to TM is not possible. In this paper, we have implemented DFA automatically and then it converted designed DFA into TM without modify the symbol being scanned and no movement to every transition. We have represented transition table and the definition of TM for better understanding to new learners of TCS.

Keywords: Input Alphabets, Minimum String, Transition Table, Deterministic Finite Automata and Turing Machine

I. INTRODUCTION

Theory of Computer Science is mathematical based subject in which logic of example is fully dependent on that example. The due dependency of logic on example, it is difficult to implement and understanding of new users. The DFA accept only regular grammar for generating regular language [13-14].

The researchers given have a preference to JFLAP tool to implement FA, PDA and TM [1]. In JFLAP tool implementation is done only manually. Researcher implemented some examples using JFLAP tools for acceptance or rejection of particular string which is a littlebit helpful for understanding. The designing of manually is a difficult task and it does not handle validate for the string.

In JFLAP, it is not possible to draw transition table and represent definition as well as string halting which is very helpful for understanding. The literature reveals that the researcher prefers JFLAP tool is only used for demonstration of acceptance or rejection of a particular constraint. So the designing in JFLAP tool is not able to give a clear idea and better understanding of new learners.

To handle the difficulty in designing and improving the understanding of learners, the researchers are implementing the automatic generation of DFA for starting, ending and substring constraints as well as for divisibility of decimal numbers [7-8]. These researchers are focusing on how to make this subject easier for new learners. Constraints are vital for designing and developing any kind of FA, PDA and TM [6].

Today there is no any tool available for designing the conversion of DFA into TM. The intention of this paper is to give an easier way of implementing DFA which accepts strings having diverse constraints and converted designed DFA to TM. Conversion of DFA to TM is done in two steps. In the implementation phase, several examples to convert DFA to a TM are discussed. The implementation is done by using Microsoft Visual Studio.
II. METHODOLOGY

The conversion of DFA into TM automatically is done in two steps in Theory of Computer Science. In the first step, the designing of DFA is done automatically[3-4]. The definition of DFA is;

Step 1:
Formal Definition: Finite Automata (M) is denoted by 5 tuples i.e.
\[ M = (Q, \Sigma, \delta, q_0, F) \]
Where,
- \( Q \) is a finite set of states
- \( \Sigma \) is a finite set of input symbols
- \( \delta \) is the state-transition function:
  \[ \delta: Q \times \Sigma \rightarrow Q \]  
- \( q_0 \) is the initial state, i.e. \( q_0 \in Q \)
- \( F \) is the set of final states, i.e. \( F \) is a subset of \( Q \).

The DFA is implemented by using digraph. In DFA states are generally labeled by \( q_0, q_1, \ldots, q_n \). Before going to implement a DFA, we have a need to know some basic notions which are used for designing of DFA. The basic notations are given in the following Table 1.

Table 1. Basic notations used for designing of a DFA

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Name of State</th>
<th>Representation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initial or Start</td>
<td><img src="image" alt="q0" /></td>
<td>A ( q_0 ) is initial or starting state and DFA has only one initial state.</td>
</tr>
<tr>
<td>2</td>
<td>Final</td>
<td><img src="image" alt="q2" /></td>
<td>A ( q_2 ) is final state and DFA has more than one final states.</td>
</tr>
<tr>
<td>3</td>
<td>Intermediate</td>
<td><img src="image" alt="q1" /></td>
<td>A ( q_1 ) is intermediate state and DFA has any number of intermediate states.</td>
</tr>
<tr>
<td>4</td>
<td>Loop</td>
<td><img src="image" alt="q3" /></td>
<td>A ( q_3 ) is looping state. Loop indicates any number (zero or more) of times symbol is included for reading. DFA has any number of looping states.</td>
</tr>
<tr>
<td>5</td>
<td>Exception</td>
<td><img src="image" alt="q4" /></td>
<td>A ( q_4 ) is exception state. All input symbols have transitioned from the state itself and DFA has at most one exception state.</td>
</tr>
</tbody>
</table>
The transition between two states is represented using an arrow with the input symbol as a label. The transition between two states \( q_0 \) and \( q_1 \) on input symbol ‘a’ is shown in following Fig 1.

![Fig 1: Transition between two states](image1)

The transitions between states are represented by the transition function \( \delta \). The DFA has one rule for the transition from each and every state due to its deterministic nature.

Transaction rule of DFA: “Each and every input symbol has exactly one transition from each and every state”.

The string is accepted iff the string reading is started from the initial state and after reading the last symbol it reaches to the final state. If \( w \) is a string, \( q_0 \) is the initial state and \( q_2 \) is a final state, then ‘\( w \)’ string acceptance is shown in following Fig 2.

\[
\delta(q_0, w)^* \Rightarrow (q_2)
\]

![Fig 2: String acceptance](image2)

The definition of DFA is represented by using tuple format which is helpful for better understanding.

**Step 2:**

In step two we converted DFA into TM with no stack operation to every transition. The TM was introduced by Alan Turing in 1936[2,5] which is used for computation. The TM is core part of Artificial Intelligence which works as a human being[9-12]. The TM accepts all types of grammar and generates languages for the grammar. The Chomsky provides for four types of grammar which are known as Chomsky hierarchy. The following Table 2 shows Chomsky hierarchy as;

<table>
<thead>
<tr>
<th>Sr.No.</th>
<th>Type of Grammar</th>
<th>Type of Language</th>
<th>Production Rule</th>
<th>Tools or Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Type-0 Grammar or Unrestricted Grammar</td>
<td>Type-0 Language or Unrestricted Language or Recursively Enumerable Language</td>
<td>( \alpha \rightarrow \beta ), ( \alpha \in (VUT)^+ ), ( \beta \in (VUT)^* )</td>
<td>Turing Machine</td>
</tr>
<tr>
<td>2</td>
<td>Type-1 Grammar or Context Sensitive Grammar</td>
<td>Type-1 Language or Context Sensitive Language</td>
<td>( \alpha \rightarrow \beta ), ( \alpha \in (VUT)^+ ), ( \beta \in (VUT)^* )</td>
<td>Linear Bounde dAutomata</td>
</tr>
<tr>
<td>3</td>
<td>Type-2 Grammar or Context-Free Grammar</td>
<td>Type-2 Language or ContextFreeLanguage</td>
<td>A is single Variable or Non-terminal, ( \beta \in (VUT)^+ )</td>
<td>Push Down Automata</td>
</tr>
<tr>
<td>4</td>
<td>Type-3 Grammar or RegularGrammar</td>
<td>Type-3 Language or RegularLanguage</td>
<td>A is single variable, ( \beta \in (VUT)^+ ), ( x \in T^* )</td>
<td>Finite State Machine</td>
</tr>
</tbody>
</table>

TM is more powerful than other machines[15]. The following Fig.3 shows the power of various machines:

\[
FA \leq PDA \leq TM
\]

(2)
Turing Machine accepts all type of languages. The following figure Fig 3 shows the language accepted by TM.

![Fig 3: The language accepted by TM](image)

The definition of TM is:

**Formal Definition:** Turing Machine \( M \) is denoted by 7 tuples i.e.

\[
M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)
\]

Where
- \( Q \) is a finite set of states
- \( \Sigma \) is a finite set of input alphabet not containing \( B \)
- \( \Gamma \) is a finite set of tape symbols including \( B \)
- \( q_0 \) is the initial state, i.e. \( q_0 \in Q \)
- \( B \) is special initial stack symbol
- \( F \) is the set of final states, final states are also known as halting states i.e. \( F \) is a subset of \( Q \)

\( \delta \) is the state-transition function:

\[
\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R,N\}
\]

where
- \( L \rightarrow \) move to left cell
- \( R \rightarrow \) move to right cell
- \( N \rightarrow \) No movement

The state transition function (\( \delta \)) depends on the current state, next input symbol including epsilon, stack topmost symbol and performs stack operations. Stack operation includes PUSH, POP and No Operation (NOP). The following Fig 3 shows the transition of TM;

![Fig 4: Transition of TM](image)

The conversions of DFA to TM by without modify symbol being scanned and no movement. It reading symbol being scanned current input symbol, modify symbol without any movement.
The String Acceptance: The string reading starts from the initial state and after reading the last symbol of the string it reaches to the halting state then string is accepted. If \( w \) is a string, \( q_0 \) is the initial state and \( q_2 \) is a halting state, then string acceptance by TM is shown in following Fig 4.

\[
\delta(q_0, a) \xrightarrow{w} (q_2, B, \lambda)
\]

**Fig 5: String acceptance by TM**

### III. ALGORITHM

The conversion of DFA to TM is implemented by using the following algorithm:

**Step1:** Start

**Step2:** Accept the input alphabet, string, and constraints.

**Step3:** Validate the entered input string and constraints with alphabets.

**Step4:** Find out the minimum string.

**Step5:** Draw a transition diagram for minimum string and apply the rule of DFA.

**Step6:** Without modify symbol and no movement to every transition of DFA.

**Step7:** Construct a transition table for TM.

**Step8:** Using transition diagram and transition table represent \( M \).

**Step9:** Check the halting condition.

**Step10:** Stop.

### IV. IMPLEMENTATION

The conversion of DFA to TM is implemented by using the above algorithm. The implementation is done step by step as follows:

**Step-1:** It accepts the input alphabet and validates input alphabets which contain only the sequence of alphabets or numbers. It cannot accept combinations of both alphabets and numbers as well as it does not accept the special characters and symbols in it.

**Enter the Constraints:** It accepts a valid string which is derivative from input alphabets. If the user does not enter valid string according to constraints, then an appropriate error message is displayed and the string is not accepted. How to handle constraints is shown in following Fig 5.

![Validation for alphabets](image)

**Fig 6: Validation for alphabets**

**Step 2:** After checking the validation of input alphabets, check validation of string constants. The string constraint is the only combination of given alphabets. The following Fig 6 shows how to handle validation for the string.
Step 3: After checking validation of input alphabets and string the minimum string is finding for drawing a DFA. The following Fig 8 shows how to find the minimum string.

Step 4: The DFA is drawn by using minimum string and then apply the rule of DFA to draw complete DFA. Complete DFA is shown in the following Fig 9.
**Step 5:** The complete DFA is converted into TM by without modifying input symbol and without any movement. The following Fig 10 shows transition diagram for TM;

![Transition diagram of TM](image)

**Step 6:** After conversion of DFA into TM, we had drawn transition rules. The following Fig 11 shows transition rules for above TM as;

![Transition rules for TM](image)
Fig 11: The transition rules for TM

Step 7: The definition of TM is represented by using seven tuples. The following Fig 12 shows the definition of TM as:

Fig 12: Definition of TM in tuple format
Step 8: The string halting of TM is shown in the following Fig 13 as:

**Fig 13:** The halting of the string by TM

Step 9: The string rejection of TM is shown in the following Fig 14 as:

**Fig 14:** The rejection of string by TM

V. RESULT
The result of the implicit conversion of DFA to TM is compared with the JFLAP tools result. The comparison is summarized in the following Table 3.

**TABLE 3: The comparison of implicit conversion of DFA to TM and JFLAP tool**

<table>
<thead>
<tr>
<th>Sr.No.</th>
<th>Features</th>
<th>Implicit Conversion</th>
<th>JFLAP Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Validation for input alphabets</td>
<td>It is possible.</td>
<td>It does not possible.</td>
</tr>
<tr>
<td>2</td>
<td>Validation for the input string</td>
<td>Validation is done.</td>
<td>Validation is not possible.</td>
</tr>
<tr>
<td>3</td>
<td>Finding minimum string</td>
<td>Minimum String can be found.</td>
<td>Such facility is not available.</td>
</tr>
<tr>
<td>4</td>
<td>Drawing DFA</td>
<td>DFA is drawn implicitly.</td>
<td>DFA is drawn manually.</td>
</tr>
<tr>
<td>5</td>
<td>Conversion of DFA to TM</td>
<td>DFA can be easily converted into TM implicitly.</td>
<td>Such functionality is not available.</td>
</tr>
<tr>
<td>6</td>
<td>Representation of transition table</td>
<td>The transition table is drawn automatically.</td>
<td>Drawing of transition table is not possible.</td>
</tr>
<tr>
<td>7</td>
<td>Representation of definition</td>
<td>The definition can be represented using tuple format.</td>
<td>Such representation is not available in JFLAP.</td>
</tr>
<tr>
<td>8</td>
<td>String halting</td>
<td>String halting is implemented.</td>
<td>Such implementation is not possible.</td>
</tr>
<tr>
<td>9</td>
<td>Understanding</td>
<td>It is very helpful for the new learners of the theory of Computer Science.</td>
<td>It is quite difficult for the new learners of the theory of Computer Science.</td>
</tr>
</tbody>
</table>

VI. CONCLUSION

The result of the implicit conversion of DFA to TM is compared with the JFLAP tools result. In JFLAP tools DFA is drawn manually and we can’t give the transition table as well as representation of definition. Design of DFA implicitly validates all constraints which are essential for better understanding to new learners. The conversion of DFA to TM is implemented by using a transition graph, transition table and the definition using tuple format. It also gives halting of string. The result reveals that performance of implicit conversion of DFA to TM gives better performance than JFLAP tools result. Still, there are few options for future works such as the undecidability problem of TM.

ACKNOWLEDGMENT

This research paper is a part of my research entitled “Sequential Pattern Mining using Turing Machine. This paper is partially supported by Sinhgad Institute of Computer Sciences, Korti-Pandharpur of Solapur University, Solapur (MS), India.

REFERENCES


