International Journal of Computer Theory and Engineering, Vol.3, No.1, February, 2011
1793-8201

Implementation of Recursively Enumerable Languages in Universal Turing Machine

Sumitha C.H, *Member, ICMLC* and Krupa Ophelia Geddam

***Abstract*—This paper presents the design and working of a Universal Turing Machine (UTM) for the JFLAP platform. Automata play a major role in compiler design and parsing. The class of formal languages that work for the most complex problems belong to the set of Recursively Enumerable Languages (REL). RELs are accepted by the type of automata known as Turing Machines. Turing Machines are the most powerful computational machines and are the theoretical basis for modern computers. Still it is a tedious task to create and maintain Turing Machines for all the problems. To solve this Universal Turing Machine (UTM) is designed in this paper. The UTM works for all classes of languages including regular languages, Context Free Languages as well as Recursively Enumerable Languages. A UTM simulates any other TM, thus providing a single model and solution for all the computational problems. The creation of UTM is very tedious because of the underlying complexities. Also many of the existing tools do not support the creation of UTM which makes the task very difficult to accomplish. Hence a Universal Turing Machine is developed for the JFLAP platform. JFLAP is most successful and widely used tool for visualizing and simulating all types of automata.**

***Index Terms*—CFG, CFL, delta rule, DFA, FSA, PDA, JFLAP, REL, transitions, UTM**

I. Introduction

An automaton is a mathematical model for a finite state machine (FSM). A FSM is a machine that has a set of input symbols and transitions and jumps through a series of states according to a transition function . Automata play a major role in compiler design and parsing. Turing Machines are the most powerful computational machines. They possess an infinite memory in the form of a tape, and a head which can read and change the tape, and move in either direction along the tape or remain stationary. Turing Machines are equivalent to algorithms, and

are the theoretical basis for modern computers. Still it is a very complex task to create and maintain Turing Machines for all problems. This will consume large amount of memory space. Also the creation of TMs for multiple tasks is very complex. The solution to this problem is a UTM. A Turing Machine that is able to simulate any other Turing Machine is called a Universal Turing Machine (UTM, or simply a universal machine) [4].

A UTM is the abstract model for all computational models. A UTM T_U is an automaton that, given as input the description of any Turing Machine T_M and a string w, can simulate the computation of M on w [6]. JFLAP represents a Turing Machine as a directed graph. JFLAP is extremely useful in constructing UTMs as the Turing Machine transducers can run with multiple inputs. Complex Turing machines can also be built by using other Turing Machines as components or building blocks for the same [2].

Manuscript received April 20, 2010.
Sumitha C.H, Department of Computer Science and Engineering, Karunya University, Coimbatore, India (e-mail: sumithach.1986@gmail . com)
Krupa Ophelia Geddam, Department of Computer Science and Engineering, Karunya University, Coimbatore, India, (e-mail: ophelia@karunya.edu.in)

II. Universal Turing Machine In JFLAP

Turing Machines are the most powerful computational machines. The Turing Machine (TM) is the solution for the halting problem and all other problems that exist in the domain of computer science. Still it is a tedious task to create and maintain TMs for all the problems. The Universal Turing Machine (UTM) is a solution to this problem. A UTM simulates any other TM, thus providing a single model and solution for all the computational problems.

A. Context Free Languages

The set of all languages that can be accepted by Deterministic Finite Automata (DFA) fall into the category of regular languages. Regular languages are effective in describing certain simple patterns. But these cannot represent many of the complexities that are actually found in programming languages. In order to cover this, the family of languages is enlarged to include more complicated features. This led to the concept of Context Free Languages (CFL).

The regular grammar has two restrictions for its productions. The left side of the production must be a single variable and the right side takes a different and special form. A Grammar G= (V, T, S, P) is said to be a CFG if all the productions in P of G have the form

$$A \rightarrow x \tag{1}$$

Where
$A \in V$ and $x \in (V \cup T)^*$,
V is a finite set of symbols called variables,
T is a finite set of symbols called terminals.

A language L is said to be a Context Free Language CFL if and only if there is a CFG G such that L=L (G). The class of automata that can be associated with Context Free Languages is called Push Down Automata (PDA). PDAs have stack as the storage mechanism to store and retrieve symbols in the reverse order.

B. *Recursively Enumerable Languages*

Regular languages form a proper subset of Context Free Languages. So PDAs are more powerful than finite automata. But CFLs are limited in scope because many of the simple languages like $a^n b^n c^n$ are not context free. So to incorporate the set of all languages that are not accepted by PDAs and hence that are not context free, more powerful language families has been formed. This creates the class of Recursively Enumerable Languages (REL).

The finite automaton has no mechanism for storage. The PDA is more powerful than FAs as they have the stack as the temporary storage. The new class of languages REL is accepted by a new type of automaton that has the most powerful storage as well as the computation mechanisms. This created the Turing Machine (TM). A TM's storage has an infinite tape, extendable in both directions. The tape is divided into cells, each cell capable of holding one symbol. The information can be read as well as changed in any order. The TM has the capability of holding unlimited amount of information.

Turing Machines work for regular languages, CFLs as well as RELs. A language L is said to be recursively enumerable if there exists a TM that accepts it.

C. *Universal Turing Machine*

A UTM simulates any other TM, thus providing a single model and solution for all the computational problems. A UTM T_U is an automaton that, given as input the description of any Turing Machine T_M and a string w, can simulate the computation of M on w. It reduces the memory usage when compared to using multiple TMs.

The transition function is the core part of a UTM. The UTM works on the basis of the rules defined in it. The transition function δ for a UTM with single tape is defined as

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\} \quad (2)$$

The transition function δ is a partial function on Q X Γ and its interpretation gives the principle by which a Turing Machine operates. The arguments of δ are the current state of the control unit and the current tape symbol being scanned. The result is a new state of the control unit, a new tape symbol which replaces the old one and a move symbol L or R [6].

The UTM is represented as

$$T_U = (Q, \sum, \Gamma, \delta, q_0, \quad , F) \quad (3)$$

Where

Q is the set of all internal states,
∑ is the input alphabet,
Γ is a finite set of symbols called the tape alphabet,
δ is the transition function,
$q_0 \in Q$ is the initial state,
$\in \Gamma$ is a special symbol called the blank,
$F \in Q$ is the set of all final states.

A UTM can accept regular languages, CFGs as well as RELs. A UTM can solve any problem that can be solved using a FSA, PDA or even a standard Turing Machine. The UTM designed in this paper supports a restricted alphabet of {a, b, c, x, y, z, }. It does not support non-determinism. Any standard TM with a maximum of ten states can be simulated using this UTM. It has over 1000 states to simulate a standard TM. A Universal Turing Machine can be represented as in Fig. 1.

1) *Working of the UTM*

The UTM has an infinite tape extendable in both directions to hold the input and perform the computation. It also has a read-write head to position the input symbol. The UTM has infinite memory. There are two other tapes also that are used for the processing. The first tape holds the description of the original Turing Machine T_M and the other tape to hold the internal state of T_M [3].

The input to the UTM T_U is given in the form of $< T_{M,} w>$ where T_M is the Turing Machine that has to be manipulated and w is an input string for T_M. The execution of the Turing Machine is specified by transition rules or delta rules.
Each transition is of the form

$$\delta (q_i , a) = (q_j , b, R) \quad (4)$$

Where

q_i is the current state,
a is the current read symbol,
q_j is the next state or destination state,
b is the write symbol and
R is the direction to which the tape head has to move.

The tape head of the Turing Machine as well as the UTM can move in either direction, left specified by L or move right specified by R. There is an option to stay in the current position also which is specified by S. The encoded input is given to the UTM. The tape head scans the contents of the tape and reads the current input symbol and the current internal state. It checks the transition rules stored in the description tape and performs the operation as specified in the transition rule of the original T_M. When all the input symbols have been scanned, the T_U enters the final state check section and performs the same as T_M [1].

D. *Simulation of Turing Machines in the UTM*

The universal Turing Machine, T_U, requires as input a string that contains both the encoding of some arbitrary Turing Machine, T_M, and an input string w for T_M. The UTM processes this input and performs the same operation that would be performed by T_M. When carrying out the emulation of T_M, T_U will need to keep track of several things. T_U maintains the current contents of the memory tape of T_M, the current state of T_M, and the current location of T_M's read-head. The enhanced UTM consists of a single tape to store the internal states, tape contents as well as the description of the standard Turing machine T_M. The enhanced UTM can be represented as in Fig. 2.

1) Creation of input for the UTM

T_U uses the # character to represent a blank symbol on the tape of TM. The portion of T_U's tape that represents the tape of TM will both start and end with a cell containing #. The input is encoded such that it contains three sections: (1) a list of the final states of T_M, (2) the transitions of T_M, (3) the tape contents of T_M just prior to the start of execution.

</FinalState List/> : </Delta Rule/>[;</Delta Rule/>]* : #</Initial StateID/></Input String w/>#

The transition rules of T_U are written in an order reversed from that of the standard Turing Machine quintuple. From left-to-right, the five characters are: the read-head shift direction, the write character, the destination state, the read character, and the source state.

The final section represents the initial configuration of the tape just prior to the start of execution. Both the first and final character of this section will be # [7].

2) Manipulation of Turing Machine in the UTM

T_U keeps the current state-id-number that corresponds to the current state of T_M in the tape cell to the left of the cell where the read-head of T_M is currently aligned. Thus using only a single memory cell, T_U is able to keep track of both the current state and the current read-head position of the T_M that is being simulated. One step of T_M will correspond to several steps in T_U, since the location of the current-state-id digit might need to be swapped with one of its neighbors to execute the transition rules.

The T_U will never alter the portion of its memory that contains the list of final states of T_M as well as the list-of-transitions of T_M. The tape contents of T_U in the last section will evolve over time exactly in conformance with the simulated execution of T_M. As T_U carries out the execution of T_M, the changing state of T_M's tape will be duplicated on this portion of T_U's tape which actually holds the tape configuration. The other two sections are different because they do not change with time as they contain the list of final states as well as the transition rules of the T_M being simulated. This has to be kept consistent till the execution is over.

T_U repeatedly carries out the classic von Neumann fetch-and-execute process. The fetch process determines which delta-rule of T_M to be emulated next. T_U begins from the initial configuration section of its tape. It begins by reading the current state-id digit and the alphabet character that lies one cell to its right in its copy of the tape of T_M which is present in the last section of the tape. T_U will then locate the correct transition rule of T_M from the list-of-rules in the portion of its tape enclosed by the two colon symbols, which is the middle section of the tape. If no matching transition rule is found, T_U will crash similar to what T_M would have done [5].

When the appropriate transition of T_M is found, T_U enters the execution phase. In order to execute a transition rule, the destination state identifier, write character, and T_M read-head shift direction must be stored. After the simulation of each delta-rule, T_U enters the final state check section to determine if T_M has entered a final state or not.

When the final state check starts, T_U has its read-head positioned over the current state identifier of T_M. T_U will branch into different states depending on the state of T_M as well as the total number of states in the T_M. T_U steps through the final state list and halts if the branch state identifier is equal to the state-id under the read head. If T_M has not halted, then T_U will repeat the fetch-and-execute process till it halts [1].

III. Test Results

The working of a UTM for a recursively enumerable language can be explained with an example $a^nb^nc^n$. that is solved using a standard Turing Machine T_M and the UTM T_U. The language $a^nb^nc^n$ is a REL which cannot be implemented using a FA as well as a PDA. The standard Turing machine T_M for the language $a^nb^nc^n$ is given in Fig. 3. The same problem can be solved with the UTM also. The difference lies in the way the UTM branches into states and transitions as a single move of T_M corresponds to multiple moves of T_U. For a deterministic Turing machine with m symbols in the alphabet such that | ∑ | = m and total number of states n, m X n transitions are possible.

A UTM with n states, | ∑ | = m and p possible directions branches into m X n X p states for execution. A problem that can be solved with a multi tape Turing machine with m tapes in O (n) moves can be done with a UTM in O (n^m) moves. For Fig. 4 shows the UTM for the context free language $a^nb^nc^n$

IV. Conclusion And Future Work

Turing machines are the most powerful computational machines. A Universal Turing Machine simulates any other Turing Machine, thus providing a single model and solution for all the computational problems. The Turing Machines provide an abstract model to all the problems. This paper describes the working of a Turing Machine as well as a Universal Turing Machine for Recursively Enumerable Languages.

The Turing Machines differ from all other automata as it can work with Recursively Enumerable Languages. The language $a^nb^nc^n$ is a recursively enumerable language which cannot be implemented using a Finite Automata or a PDA but can be done using a T.M. This requires more storage than for Context Free Languages and hence the Turing Machines with the infinite tapes, extendable in both directions are used for this.

The UTM designed in this paper has the following features.

1) It supports an alphabet of {a, b, c, x, y, z, }.
2) It simulates standard Turing Machines with a maximum of ten states.
3) It does not support non-determinism.
4) Does not support JFLAP's pattern-like "~" (any) and "!" (not) characters.

The future work includes enhancing the concept of universality by including more symbols in the input alphabet as well as in the tape alphabet.

Acknowledgment

We thank Mr. Jonathan Jarvis and Mr. Joan M. Lucas for the work on Universal Turing Machine. We also thank Dr. Susan Rodger of Duke University for the meritorious work on the same.

References

[1] Jonathan Jarvis and Joan M.Lucas, “Understanding the Universal Turing Machine: An implementation in JFLAP”, ACM Portal Volume 23, Issue 5, Pages 180-188, 2008.
[2] Eric Gramond and Susan H.Rodger, “Using JFLAP to interact with theorems in automata theory”, ACM Portal Proc. in SIGCSE, Pages 336-340, 1999.
[3] Susan H.Rodger, Eric Wiebe, Kyung Min Lee, Chris Morgan, Kareem Omar and Jonathan Su, “Increasing engagement in automata theory with JFLAP”, ACM Transactions, Pages 403-407, 2009.
[4] Alan Rosen and David Rosen, “The Turing Machine Revisited”, MCon Inc., 2007.
[5] [Online] http://www.jflap.org/tutorial/JFLAP tool and tutorials.

[6] J.Hopcroft, R.Motwani and J.Ullmann, Introduction to Automata Theory, Languages and Computation, 3rd Edition, Addison-Wesley, 2006.

[7] Susan H. Rodger and Thomas W. Finley, JFLAP: An Interactive Formal Languages and Automata Package, ISBN 0763738344, Jones & Bartlett Publishers, 2006.

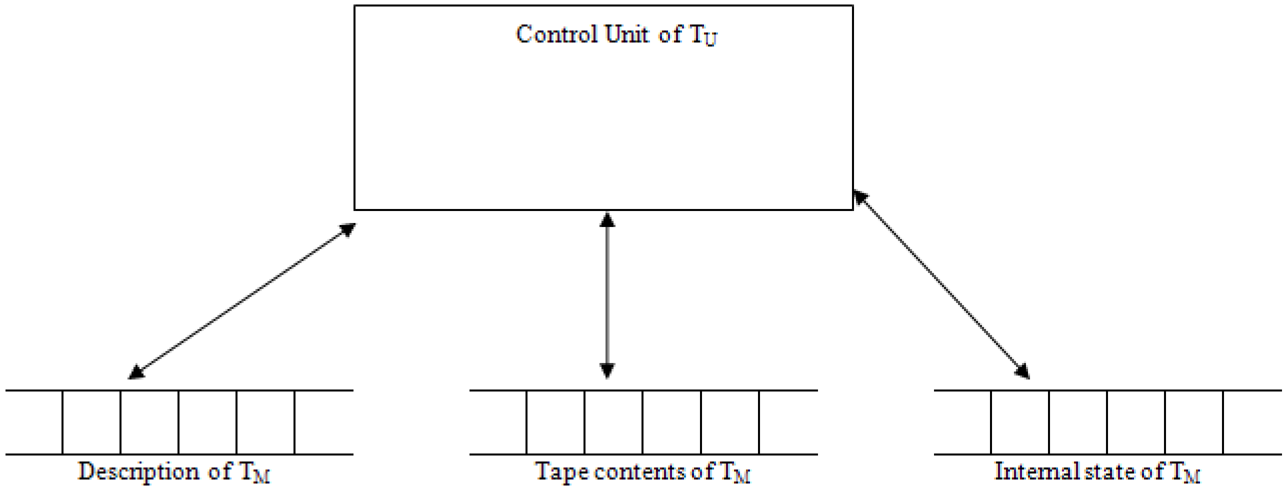
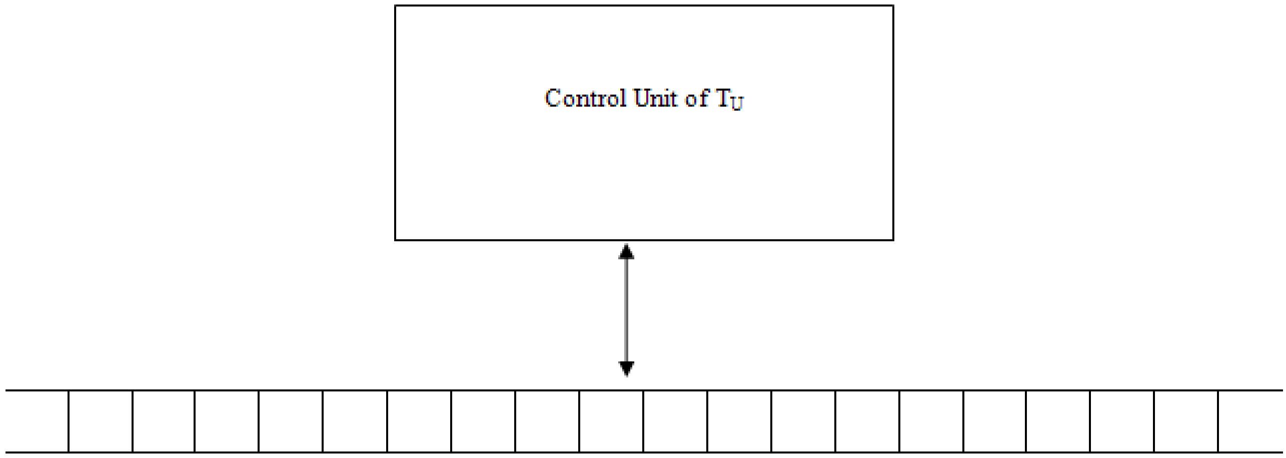


Figure 1. Internal representation of Universal Turing Machine



Tape storing the internal states, tape contents and description of the standard Turing Machine T_M

Figure 2. Internal representation of Enhanced Universal Turing Machine

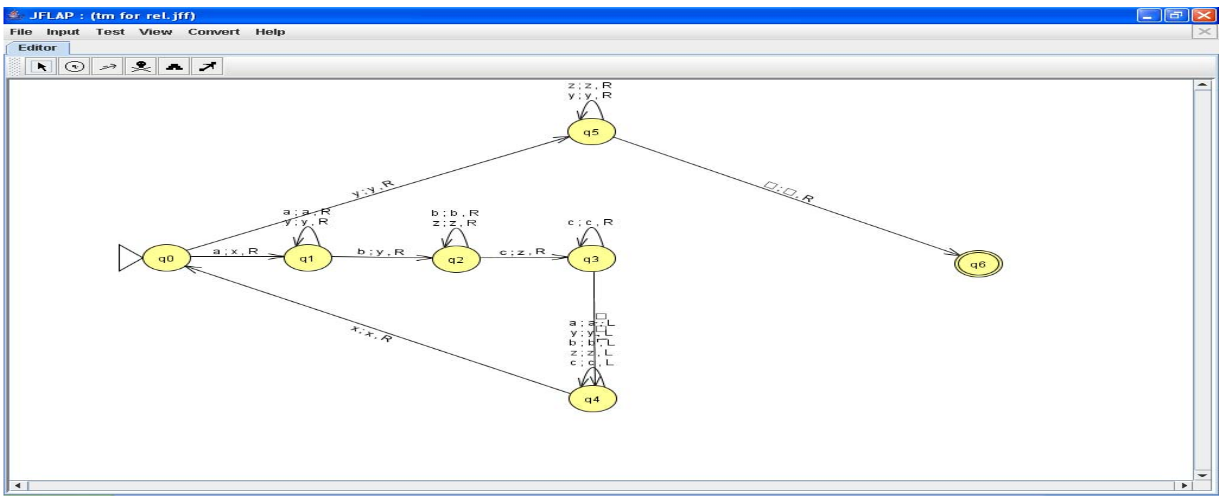


Figure 3. Standard Turing Machine T_M for $a^n b^n c^n$

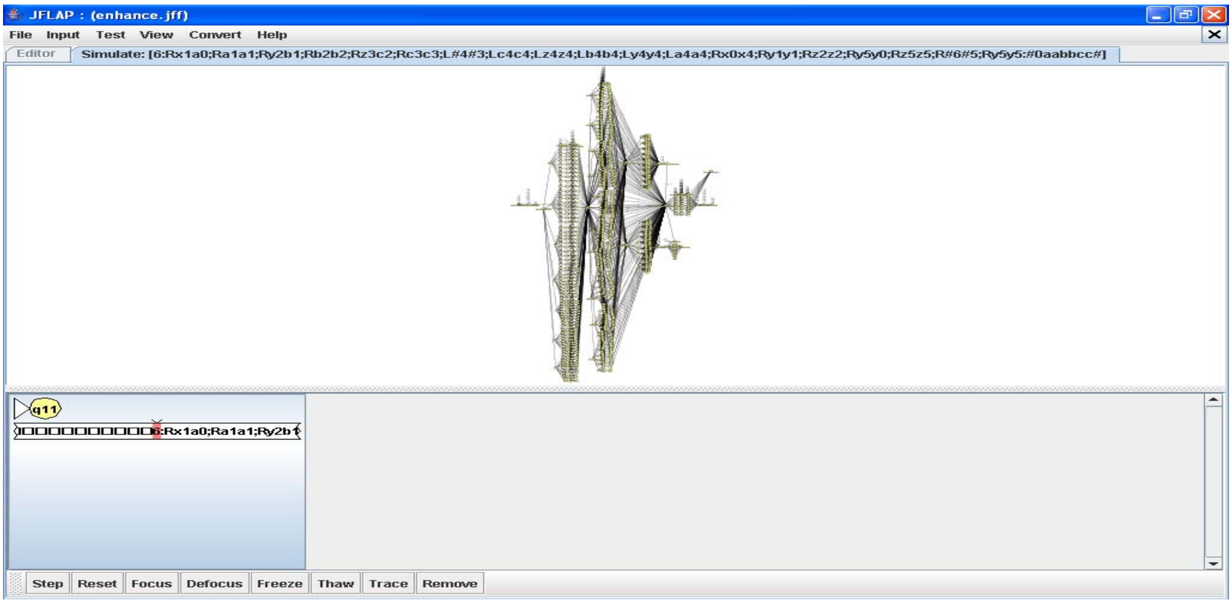


Figure 4. Universal Turing Machine for $a^n b^n c^n$