A Recursive Code Generating Algorithm for Automata Control

Monday O. Eze[#1], Shade Kuyoro[#2]

[#1,2]*Senior Lecturer, Department of Computer Science*

Babcock University, Ogun State Nigeria

Abstract—*Automata Theory involves the evolution, study and application of abstract machines to solve computational problems. A number of research domains such as Compiler Construction, Robotics, Logic Programming, and Linguistic Computing make extensive use of automata theory. A key component of the 5-tuple that constitute a finite automata is the set of transition functions, which gives rise to an evolutionary design tool called the transition diagram. A transition diagram models the movement of a machine from one state or location to another. Usually, in cases where the number of states covered by the transiting object is minimal, generating a transition diagram may follow sequentially, from entry till acceptance state through separate invocations. However, as the number of states grow from tens to hundreds or thousands, the need for a computational solution becomes very apparent. Suppose the movement of an automata driven refuse collection robot which covers hundreds of locations per day is modelled using a transition diagram, such that each movement represents a transition from one state to another. Traditionally, this would require an invocation of a separate transition function for every singular transition, giving rise to a number of sequential system calls equivalent to the total number of separate transitions. Such a method could be very tedious, time consuming, and error-prone. The aim of this research is to evolve an algorithm that generates a single line of recursive code that drives an unlimited number of transition moves at once, instead of maintaining separate invocations. A new algorithmic technique termed quantum code blocking was also evolved to test the output, to ensure its correctness.*

Keywords: *Automata Theory, Abstract Machines, Transition Diagram, Recursive Code Generation, Mobile Robots, Algorithms.*

I. Introduction

Computational research into recursive code generation is of key importance for automata control. This is particularly necessary as the size of an automata-driven system grows with the expansion of the number of transition states. The term automata is the plural of automaton. It is an abstract computational model [1] of a digital computer. An automaton is a device that possesses the key features of a digital computer [2]. In other words, it accepts inputs, possesses some storage capability, and produces outputs. It could also make some decisions or take actions in order to produce the output. An automaton could be deterministic or non-deterministic [3]. The focus of this work is on Deterministic Finite Automata (DFA), though with some modifications, it could work for Non-Deterministic Finite Automata (NDFA) [4]. A research [5] defined DFA as composed of 5-tuple as defined in equation (1):

$$DFA = (Q, A, So, T, F) \qquad (1)$$

where Q = Set of states; A=Finite set of symbols known as alphabet; S_0 = Start state, which is an element of Q; T = Transition function; F=Set of acceptance states.

A transition function (T) is defined [6] as a mapping from the Cartesian Product Q × A into the set of states Q. In representing a DFA pictorially as a state diagram [7], a number of standard rules apply, some of which will be outlined. First is that the state diagram should be a directed graph [8], which emanates from the source state, and points towards the next state. Second is that the arcs or directed arrows are labelled with the inputs which are elements of the alphabet A as defined in equation (1). Thus, a transition function T(A, g) = B, will generate a segment of a transition diagram, with a source state A, a destination state B, and an input signal g. Third is that the start state is usually recognized with an external arrow pointing towards it, and the acceptance state is denoted with a double circle [9]. The diagram in Fig. 1 clearly demonstrates the start state S_0 and acceptance state Y respectively.

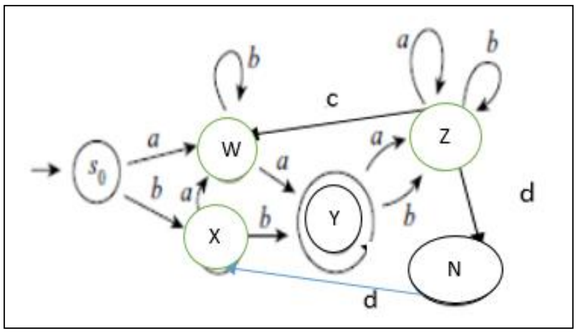


Fig. 1: A Transition Diagram

The theory of abstract machines find application in diverse areas of computing such as Artificial Intelligence [10], Computational Linguistics [11], Compiler Construction [12], Robotics [13], Logic Programming [14] among others. This work is particularly necessary because the new concept and resulting outcome demonstrated by this research will no doubt be relevant in the study, design and control of automata-driven machines such as mobile robots [15], vending machines [16], among others.

II. PROBLEM STATEMENT

The necessity for a recursive code generator [17] for automata-driven control as tackled in this research cannot be overemphasized. Supposed a mobile object such as an office assistant robot is modelled using Fig. 1. The machine covers six locations (states) labelled as S_0, W, X, Y, Z and N, where the start state is S_0, and the acceptance state Y. Let us assume that for a particular day, the automata-driven robot is programmed to cover from S_0 to the rest of the other stations, as many times as possible. Suppose also, that for any transition from an original state K_X to a next state K_Y, there is a transition function defined as $K_{X\,(i)} \rightarrow K_Y$, where i is the input to the machine. Let the following transition path consisting of a number of component transition functions be defined for the moving object:So(a)➔W; W(b)➔W; W(a)➔Y; Y(a)➔Z; Z(a)➔Z; Z(b)➔Z; Z(d)➔N; N(d)➔X; X(a)➔W; W(a)➔Y; Y(b)➔Z; Z(c)➔W; W(a)➔Y; Y(b)➔Z; Z(c)➔W; W(a)➔Y; Y(b)➔Z; Z(d)➔N; N(d)➔X; X(b)➔Y.

A look at the movement of the robot [18] shows that it covered a total of 20 transitions from start (S_0) till finish at Y. Going by such a sequential order, there would be a total of 20 separate invocations of the component transition functions. The implication is that if the moving object is to cover a thousand stations, then a total of one thousand separate function calls would be required. Such a crude method will no doubt be very tedious, time-consuming and prone to errors. This research therefore solves this problem by evolving a code generating algorithm, whose main objective is to generate a single recursive code equivalence of the original series of codes. The three parameters required by the code generator are, the Start State, the End State, and a Transition String. The concept of Transition String as used in this work will be further explained and applied at a later section of this paper.

III. RELATED WORKS

A research by Kenneth Rosen [19] presented a mathematical conceptualization of automata transitions model. Two mathematical equations were stated that explain the concept of extending the transition functions. However, the research did not delve into the design of algorithms that could generate the recursive code, and neither was any further work presented on recursive execution. A PhD work [20] discussed the theoretical foundation for generation of sequence diagrams for risk assessment of complex systems. The author however, did not explore any computational algorithm for recursive code generation or execution. A research by [21] focused on the generation of automatic test case in system development. The work treated UML state charts, but was silent on any form of algorithmic techniques for the generation of transition diagrams for automata control. Based on literature reviews, it is obvious that the issue of generating a recursive code for automata control is a gap that needs to be filled, hence the necessity for the current research. Before going into the details of the algorithm, a number of preliminary concepts will be explained.

IV. PRELIMINARY CONCEPTS

A number of new concepts were developed as part of this study. For a better understanding of the subject matter, some of these concepts will be outlined in this section.

A. *Transition String*

A Transition String (TS) is a string whose component characters constitute all the input characters accepted by the transition functions, and needed for a moving object to transit from a particular start state to an end state. In other words, the formation of TS follows a chronological order from the first input character to the last one. Considering Fig. 2, a transition string TS = ‘axhdyf’ will be necessary for an object to transit from state S to H.

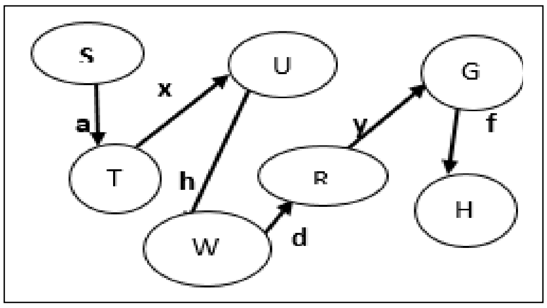


Fig. 2: Demonstrating a Transition String “axhdyf”

It follows that state S accepts input ‘a’ in order to transit to T. Similarly, state T accepts input ‘x’ to transit to U in that order, until state G takes input as ‘f’ to transit to H.

B. *Cardinality*

The term cardinality refers to a numeric value that signifies the number or count of items under consideration [22]. For instance in Set Theory, the cardinality of a set refers to the number of elements of the set.

C. *Recursive Transition Format*

The term recursive transition format (RTF) refers to a special format resulting from the code generating algorithm. It is a key deliverables of this research. It is a single logical code that could be recursively executed in line with the movement of a mobile object through the automata transition diagram.

D. *Quantum Code Blocking*

This refers to the breaking of the RTF code into blocks in a specialized manner, beginning from the centre. This technique tests the accuracy of the overall output as will be explained.

V. RECURSIVE CODE GENERATING ALGORITHM

The algorithmic rule for generating the automata control code is made up of eight fundamental steps - labelled as STEP 1, STEP 2 … STEP 8 in the flow diagram in Fig. 3.

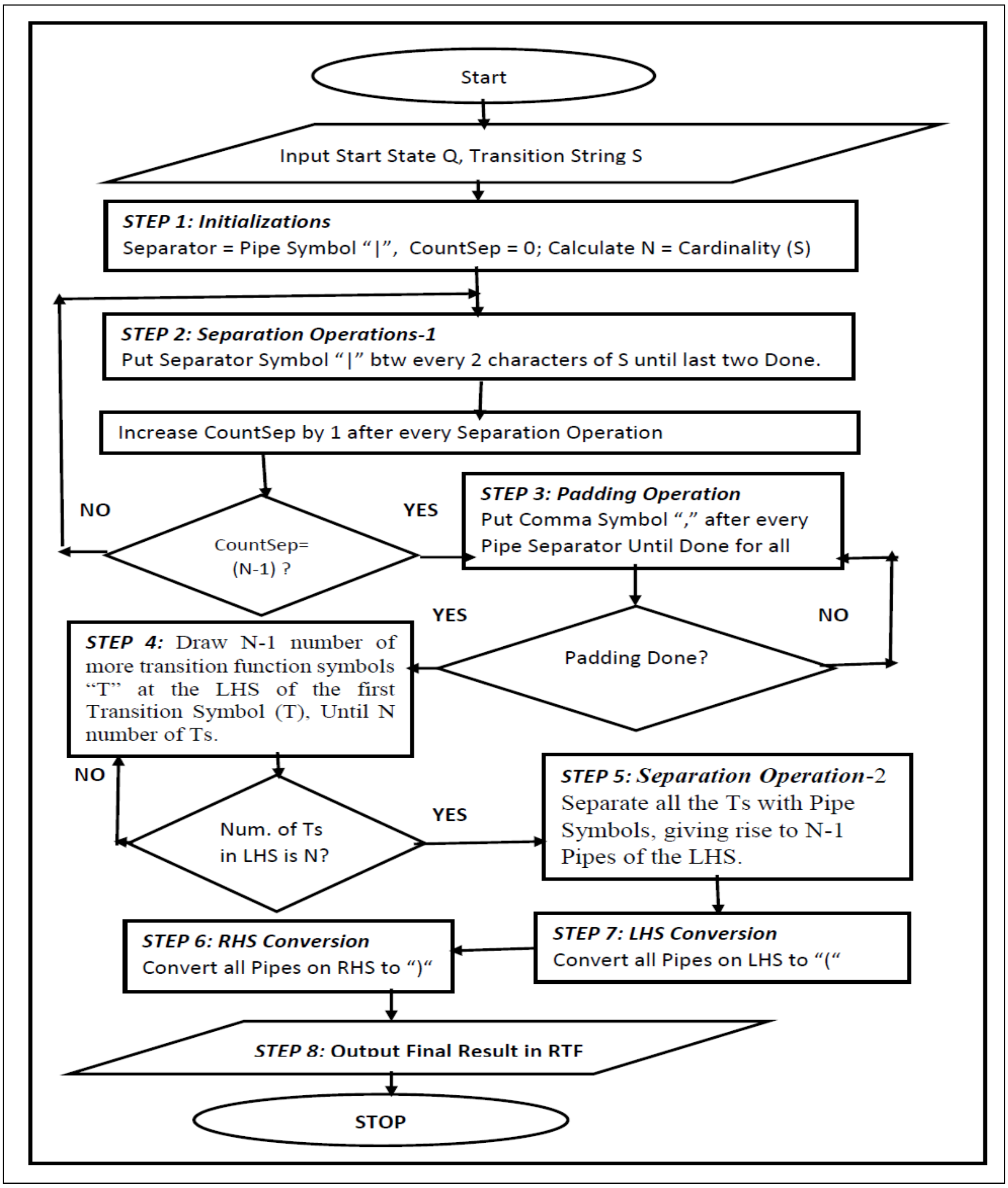

Start
Input Start State Q, Transition String S
STEP 1: Initializations
Separator = Pipe Symbol "|", CountSep = 0; Calculate N = Cardinality (S)
STEP 2: Separation Operations-1
Put Separator Symbol "|" btw every 2 characters of S until last two Done.
Increase CountSep by 1 after every Separation Operation
NO
YES
CountSep= (N-1) ?
STEP 3: Padding Operation
Put Comma Symbol "," after every Pipe Separator Until Done for all
YES
NO
Padding Done?
STEP 4: Draw N-1 number of more transition function symbols "T" at the LHS of the first Transition Symbol (T), Until N number of Ts.
NO
YES
Num. of Ts in LHS is N?
STEP 5: Separation Operation-2
Separate all the Ts with Pipe Symbols, giving rise to N-1 Pipes of the LHS.
STEP 6: RHS Conversion
Convert all Pipes on RHS to ")"
STEP 7: LHS Conversion
Convert all Pipes on LHS to "("
STEP 8: Output Final Result in RTF
STOP

Fig 3: Flow Diagram of the Recursive Code Generating Algorithm

As earlier highlighted, the main aim of the algorithm presented in this study is to generate a single logical code that could be executed recursively to model an automata-controlled moving object. Given a transition function definition as equation (2).

$$T(Q,S) = E \qquad (2)$$

where Q is the start state, S is a transition string, and E is the end state.

As earlier stated, a transition string S is a concatenation of the characters accepted by each of the component transition functions. It is the chronological batch of inputs necessary for the object to successfully transit from state Q to state E as shown in equation (3):

$$S = S_0 S_2 S_3 S_4 \dots\dots\ S_N \qquad (3)$$

where N is the cardinality of S and S_K is the Kth character in S.

The algorithmic steps are outlined as follows.

A. Initialization Operation

This is the first algorithmic step and involves the initialization of important system parameters. The first initialization is the Separator variable which is initialized to pipe symbol “|”. The other ones are the variable CountSep which is a counter, used to keep track of the number of separators, and the variable N which captures the cardinality of the transition string S. Accordingly, CountSep is initialized to 0 while N is set to the cardinality of S.

B. Separation Operation 1

This is the second algorithmic step. The major action taken here is to separate all the characters that make up the transition String S with the Separator symbol “|”. Every two adjacent characters of S are separated accordingly, followed by an increment of CountSep by one until the total counter value of N-1 is reached at completion.

C. Padding Operation

This is the third algorithmic step. It involves annexing the comma symbol “,” after every of the separator symbols already inserted during the latest step.

D. T-Batching Operation

This is the fourth algorithmic step. The T-Batching involves drawing N-1 number of more transition function symbols “T” at the Left Hand Side (LHS) of the first Transition Symbol (T), making a total of N number of Ts. During this operation, system also keeps count of the number of “T’s drawn, to ensure completion.

E. Separation Operation 2

This is the second round of separation operation. This algorithmic step takes place at the LHS of the Transition Equation. Here, the “T”s are separated with the pipe separator symbol “|”. A total count of N-1 separator symbols are inserted at completion.

F. LHS Conversion Operation

This algorithmic operation takes place in the LHS of the evolving Recursive Transition Format code. At this point, all pipes symbols “|” located at the LHS are converted to left bracket symbol “(“.

G. RHS Conversion Operation

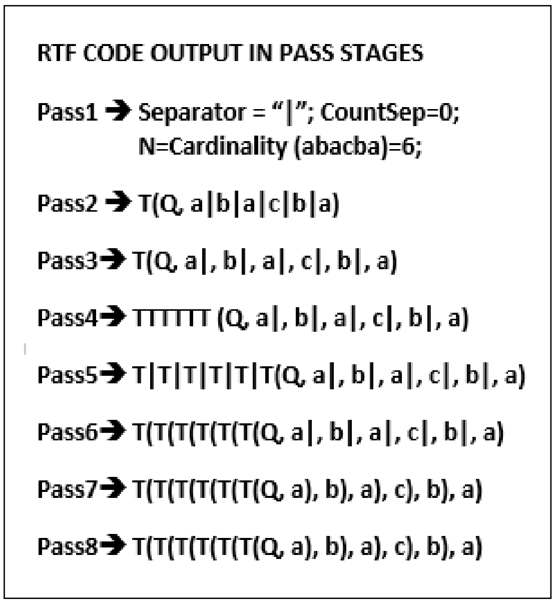
This is the seventh algorithmic operation, and it takes place in the RHS of the evolving Recursive Transition Format code. The major action taken is to convert all pipes symbols “|” located at the RHS to right bracket symbol “)“.

H. Final Result

This is the final step, and culminates the algorithm with the generation of the recursive transition format (RTF). It is important to state that these algorithmic steps itemized were implemented sequentially through a series of procedural formatting using a simple word pad.

VI. RESEARCH RESULT

The final output of this research is a recursive transition format (RTF) generated to model the movement of a transiting object through an automata transition diagram. A sample output will be demonstrated in this section. Given a Transition String S= ‘abacba’, the initial problem is set up as T(Q,’abacba’). The generating algorithm follows sequentially to arrive at the RTF given by T(Q,’abacba’) = T(T(T(T(T(T(Q, a), b), a), c), b), a) . A total of eight passes were involved to arrive at the final result as detailed in Fig. 4.



RTF CODE OUTPUT IN PASS STAGES

Pass1 ➔ Separator = “|”; CountSep=0;
N=Cardinality (abacba)=6;

Pass2 ➔ T(Q, a|b|a|c|b|a)

Pass3➔ T(Q, a|, b|, a|, c|, b|, a)

Pass4➔ TTTTTT (Q, a|, b|, a|, c|, b|, a)

Pass5➔ T|T|T|T|T|T(Q, a|, b|, a|, c|, b|, a)

Pass6➔ T(T(T(T(T(T(Q, a|, b|, a|, c|, b|, a)

Pass7➔ T(T(T(T(T(T(Q, a), b), a), c), b), a)

Pass8➔ T(T(T(T(T(T(Q, a), b), a), c), b), a)

Fig. 4: Generating a Sample RTF Code in 8 Passes

The transition diagram that accommodates the RTF, and models the movement of the transiting object is shown in Fig. 5. As shown in the diagram, the paths through which the moving object traverses are marked with red dots.

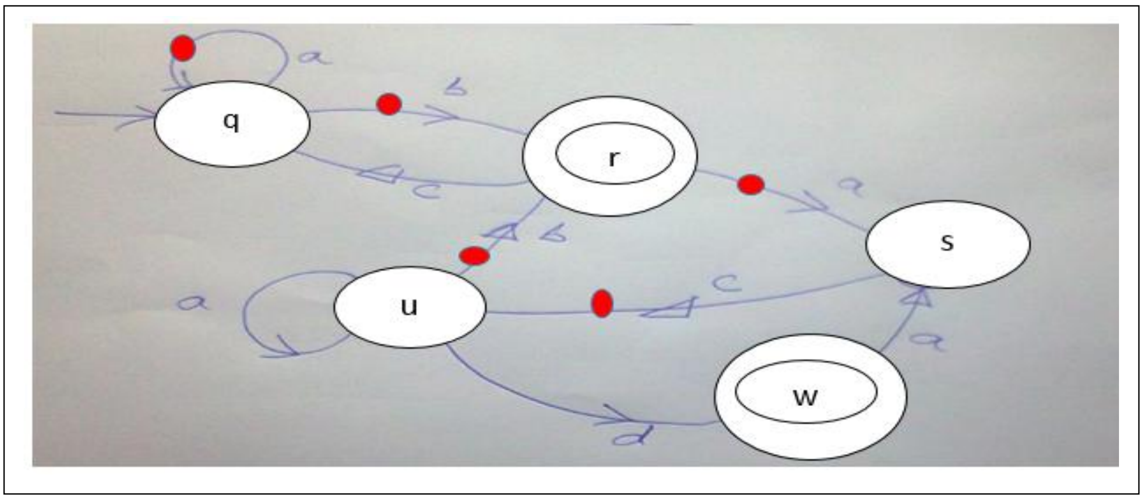


Fig. 5: Transition Diagram Modelled by RTF Code

VII. RESULT CHECKS

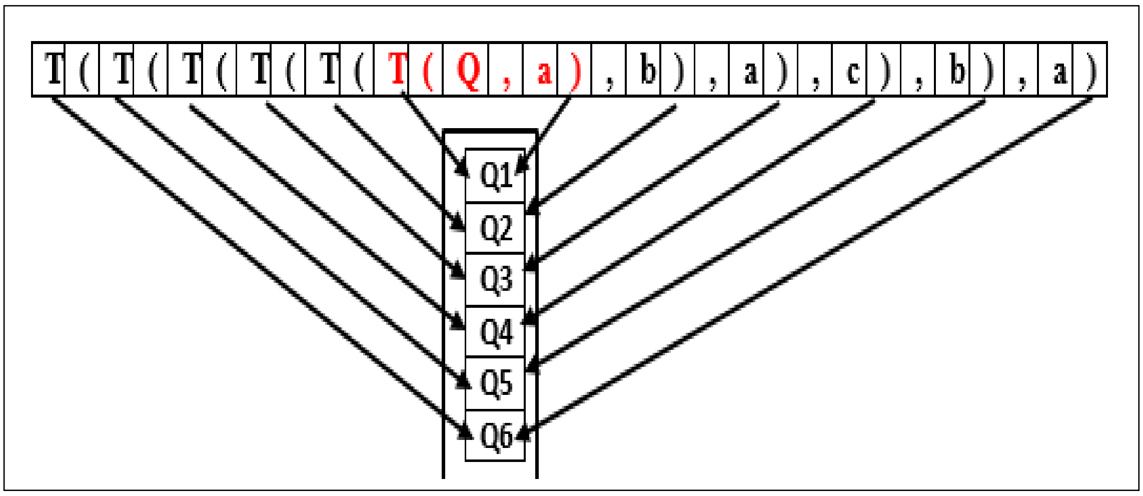
A procedural technique known as quantum code blocking was developed for testing the final output. This technique is a process that executes the whole RTF code through a series of breakout and execute operations. The process emanates from the middle of the RTF code.

Each chunk of the code is executed, and the preliminary result resubmitted into the original code as shown in Fig. 6. This series of breaking, executing, re-submitting of results takes place until recursive execution of the whole code is completed. This is one way of testing the accuracy of the generated code.

As shown in Fig. 6, the first block of code "T(Q,a)" is at the center of the RTF code. It is shown with two arrows as the block of code that starts from the rightmost transition symbol "T" up to the next closing bracket ")". After executing the first chunk, the generic result designated as Q1 is resubmitted, giving rise to T(Q1,b) which forms the next code block. The recursive execution continues in that manner until completion. The last code block T(Q5,a) leads to the final transition state Q6, thus confirming that the final RTF code is syntactically accurate.

VIII. CONCLUSION

This work has demonstrated how to generate a single recursive code T(T(T(T(T(T(Q, a), b), a), c), b), a) that simulates the object movement across a transition diagram. The resulting recursive code was also successfully tested through the simple technique of quantum code blocking. While the test case was purposely chosen to be simplistic, the importance of this work will be realized as the size of transitionsystem expands into hundreds or thousands of states or functions. It is no doubt that the result and overall techniques developed in this research will find relevance in diverse areas of automata control, especially in the study of the mobility of robots. Oneof the proposed future extension of this work is in the area of integration to robotics movement.



T (T (T (T (T (T (Q , a) , b) , a) , c) , b) , a)
Q1
Q2
Q3
Q4
Q5
Q6

Fig. 6: Demonstrating the use of Code Blocking on the output RTF code

REFERENCES

[1] J. Starzyk (2011). "A Computational Model of Machine Consciousness", International Journal of Machine Consciousness, Vol. 3, Issue 02, pp255-281

[2] P. Linz. (2012). "An Introduction to Formal Languages and Automata", 5th Ed, Jones & Bartlett Learning, Ontario Canada.

[3] B. Melnikov (2002). "A New Algorithm of Constructing the Basis Finite Automaton", Informatica, Vol. 13, No. 3, pp299–310.

[4] M. John (2011). "Introduction to Languages and the Theory of Computation", 4th Ed, McGraw-Hill, New York..

[5] A. James (2006). "Automata Theory with Modern Applications", Cambridge University Press, Cambridge, UK.

[6] A. Maheshwari & M. Smid (2014). "Introduction to Theory of Computation", School of Computer Science, Carleton University, Ottawa, Canada.

[7] A. Alrehily, R. Fallatah and V. Thayananthan (2015). "Design of Vending Machine using Finite State Machine and Visual Automata Simulator", International Journal of Computer Appl., Vol. 115, No. 18,pp37-42

[8] D. Harel & Y. Koren (2002). "A Fast Multi-Scale Method for Drawing Large Graphs,Journal of Graph Algorithms and Applications Vol. 6, No. 3, pp. 179-202

[9] A. Clark & F. Thollard (2004). "PAC-learnability of Probabilistic Deterministic Finite State Automata", Journal of Machine Learning Research 5, pp473-497.

[10] S. Singh &S. Sukhvinder (2010). "Artificial Intelligence", International Journal of Computer Applications, Volume 6– No.6, pp 21-23

[11] M. van Zaanen & C. de la Higuera (2009). "Grammatical Inference and Computational Linguistics", Proceedings of the EACL 2009 Workshop on Computational Linguistic Aspects of Grammatical Inference, Athens, Greece, pp1-4

[12] S. Aastha, S. Sinha, and A. Priyadarshi (2013). "Compiler Construction", International Journal of Scientific and Research Publications, Volume 3, Issue 4,pp1-6

[13] P. Sapaty (2015). "Military Robotics: Latest Trends and Spatial Grasp Solutions", Int. Journal of Advanced Research in Artificial Intelligence, Vol. 4, No.4, pp1-10

[14] A. F. Abbas (2014). "Comparison Between Programming Languages Prolog , C ++ , Pascal", Mathematical Theory and Modeling", Vol.4, No.14,pp27-40

[15] S. Nurmaini, and A. Primanita (2012). "Modeling of Mobile Robot System with Control Strategy Based on Type-2 Fuzzy Logic", International Journal of Information and Communication Technology Research, Volume 2 No. 3, pp235-242

[16] R. M. Varkey and J. M. Sunny (2014). "Design and Implementation of Multi Select Smart Vending Machine", International Journal of Computer Networks and Wireless Communications, Vol.4, No1, pp42-45

[17] H. Li (2016). "Binary Tree's Recursion Traversal Algorithm and Its Improvement", Journal of Computer and Communications, Vol 4, pp42-47

[18] J. Iovine(2004). "PIC Robotics A Beginner's Guide to Robotics Projects Using the PICmicro", McGraw-Hill, New York

[19] K. Rosen (2007). "Discrete Mathematics and Its Applications", 7th Ed, McGraw-Hill, NewYork, NY

[20] S. Hamed (2007). "Automatic Generation of Generalised Event Sequence Diagrams for Guiding Simulation-Based Dynamic Probalitlistic Risk Assessment of ComplexSystems", A PhD Dissertation submitted to the Faculty of the Graduate School of the University of Maryland, USA,

[21] R. Swain1, P. Kumar, and D. Prasad (2012). "Automatic Test case Generation From UML State Chart Diagram", International Journal of Computer Applications, Vol. 42, No. 7,pp26-36.

[22] M. Dhar (2013). Cardinality of Fuzzy Sets: An Overview", International Journal of Energy, Information and Communications, Vol. 4, Issue 1, February, pp15-22