http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 5 (6): 1093-1097, 2006
ISSN 1812-5638

Modified Turing Machine for Distributed Computing

Saeed ur Rahman Khan and Malik Sikander Hayat Khiyal
Department of Computer Science, International Islamic University, Islamabad, Pakistan

Abstract: Turing machine is model for general-purpose computing device. There are many implementation of Turing Machine. Khan and Khiyal (2006) presented a deterministic Turing model that simulates distributed processing. In current study we enhanced the model for complex tasks and enable to solve equation with several operands on the right hand, to perform nested conditional blocks, nested loop and updated the function execution.

Key words: Turing machine, physical machine, computing model, simulations

INTRODUCTION

Turing Machine is a mathematical model of computing device. Turing Machine, as defined by Alan Turing (1936) in his historical paper, is a 7 tuple as following:

$$TM = \{Q, \Sigma, \Gamma, \delta, S, B, f\}$$

Where, the symbols are sequentially the set of states, the input symbols, the tape symbols, the set of transition functions, the starting state and the blank symbol and finally the set of final states.

There are many simulations of computer using Turing Machine and simulation of Turing Machine using computer to prove them equivalent (Hopscroft *et al*., 2000; Guetta, 2003; Khiyal, 2004). Here, we elaborate them. This study describes a computing model having many addressable identities as basic unit, Turing Machine. Then the complete Turing Machine makes it possible to make communications among these models.

Thus in future, we shall use only TM as the formal representation of what can be computed by any kind of computing device (Khiyal, 2004).

The model developed by Khan and Khiyal (2006) is a compound Turing Machine of many simple Turing Machines. The model does not work for complex tasks. For this purpose we enhance the model discussed in later section. This model enables to solve equation with several operands on the right hand side. Enhance the conditional blocks to perform nested if else, nested loops and updated the function calling to support recursive function.

We now describe systematically the characteristics of our enhanced model.

MODEL IN BRIEF

The proposed model of Khan and Khiyal (2006) is as follows:

$$TM = \{Q^n, \Sigma, \Gamma, \delta, S, B, f\}$$

States: Q^n. The states are compound states of the states of its sub Turing Machines. So we may say states are of the following types:

$$q_1\ q_2\ q_3 \ldots q_n \text{ where } q \in Q.$$

Σ is set of input alphabets.
Γ is set of tape alphabets.

δ is set of moves. The moves are also compound moves of the following formats:

$$\delta\ (p_1, [(a_1, m_1), w_1, x_1])\ (q_1, (b_1, m_1), y_1, z_1], [D, D, D])_1$$
$$\delta\ (p_2, [(a_2, m_2), w_2, x_2])\ (q_2, (b_2, m_2), y_2, z_2], [D, D, D])_2$$
$$\delta\ (p_3, [(a_3, m_3), w_3, x_3])\ (q_3, (b_3, m_3), y_3, z_3], [D, D, D])_3$$
$$\vdots \qquad \vdots \qquad \vdots$$
$$\delta\ (p_n, [(a_n, m_n), w_n, x_n])\ (q_n, (b_n, m_n), y_n, z_n], [D, D, D])_n,$$

where $a, b, m, w, x, y, z \in \Gamma$, $q_1, \ldots q_n, \in Q$, $D \in \{L, R, N\}$

S is the starting state for all sub Turing Machines of the compound Turing Machine.
B is the blank symbol.
f is the final accepting state. But when the entire sub Turing Machines goes to accepting state then the final Turing Machine is assumed to be at the final state. So the final state of the whole Turing Machine is $f_1\ f_2\ f_3 \ldots f_n$. The features and functionalities are discussed in the study of Khan and Khiyal (2006).

Corresponding Author: Malik Sikander Hayat Khiyal, Department of Computer Science, International Islamic University, Islamabad, Pakistan

DESCRIPTION OF OUR TM

Turing Machine described by Khan and Khiyal (2006) consists of components as stated before. This portion describes one component in detail.

Tape: There are three tapes in this TM. They are the follows:

Program tape: This tape contains variable declarations, program and input. When the input instruction is called, the input value is stored in places reserved for input in this tape. This tape has two tracks. One for program and variable and another read only track contains sequence number of boxes of the former track. Point to be noted that the tape has one head with which it read both the track and write only on the first track second one is read only. In another way, we may also describe this tape as only one track having the boxes as tuple, which the TM head read. It writes only on the first part of the tuple, second part of the tuple contains the box number and this part is read only.

Postfix tape/Buffer tape: The arithmetic expressions are converted into postfix form before execution. The postfix equation is stored in this tape. At the beginning, this tape is empty. During the functionality, this tape can also be used for other purpose as well such as buffer while sending a buffer over a network. After finishing work, it is empty.

Stack tape: This tape acts as stack for this TM. At the beginning, this tape is empty. The functionality is same like the other tape but it is called stack tape as it is used as stack. Its way of functionality varies in the following way.

- When a value is pushed the tape head moves right and write the value or sometimes it writes and then moves when the current block is already empty.
- When a value is popped, the head erases the current value and moves left or it moves left and then erases the value if the current box is empty.
- At any time to read the top value just read the current box under the head or sometimes when the current box is empty it moves left and read the value.
- Even then, this is a tape so it can be used for other purpose as tape. Sometimes the push pop is merged with other moves.

As default the second and third tape head is kept over a blank box beside a filled box. When there is a value on either of these tapes that to be read constantly then that tapes head is kept on that box containing value.

Moves: Moves of a single component are of the following form: δ (q_1, [(a, m), w, x]) (q_2, (b, m), y, z], [D, D, D])

where

a, b, m, w, x, y, z $\in \Gamma$, q_1, q_2, $\in$ Q, D $\in$ {L, R, N}

Tape symbols: There are tape symbols for each instruction and each reserve words and each value with which the program is written on the tape. No defined name such as variable name or function name can be repeated. So at same time a name cannot be used as variable name and function. Let alone overriding/overloading etc. Tape symbols can be categorized in the following categories:

- Value
- Reserve words/key words
- Instruction symbols/state symbols: Note that there is a tape symbol for some of the states if not for all.
- Brackets of different types e.g. {, }, (,).
- Operators: + - * / = > < ! :=

WORKING OF TM AND MOVES

The workings of the following points are same as Khan and Khiyal (2006).

- Variable definition/storage retrieval.
- Variable definition at function parameter or variable as reference to another variable.
- Variable reading.
- Variable updating
- Assignment statement
- Read variable from user/write output.
- Send receive expression to another Turing Machine
- Converting expression to postfix expression

In variable reading part two moves are changed as follows:
The move:

δ (v_n, [(δ, m), B, x]) ($v_{r\text{-}n}$, (δ, m), B, B], [R, N, R])
is replaced by the move,
δ (v_n, [(δ, m), B, x]) ($v_{r\text{-}n}$, (δ, m), B, x], [R, N, R])
The move:
δ (v_{egr}, [(a, m), B, x]) (v_{egr}, (a, m), B, x], [R, R, N])
is replaced by the move.
δ (v_{egr}, [(a, m), B, x]) (v_{egr}, (a, m), B, x], [R, N, N])
In send to other Turing Machine part the move:
δ (s, [(Λ, m), B, B]) (e, (Λ, m), B, R], [R, N, R])
is replaced by the move.
δ (s, [(Λ, m), B, B]) (e, (Λ, m), B, B], [R, N, N])
The moves in the following points:

- Evaluating postfix
- Conditional block.

- Loops.
- Function definition/calling

Are mostly changed and complete new moves are as below.

Evaluating postfix equation: We have enhanced the evaluating postfix equation part to evaluate long equations. Previous moves could only perform binary operations. After modifying the machine it can now perform equation evaluation of any length.

To evaluate a postfix expression, we will again need to use the stack. However, the stack will contain numeric values instead of operators.

When a numeric value is encounter, the value is pushed on the stack.

When an operator (other than ;) is encountered, two values are popped from the stack the operation is performed on these two values and the result is pushed onto the stack.

When the no operand is encountered, there should only be one value on the stack above the ';'. This value is the result of the expression.

•

δ (e_v, [(; , m), n, B])] (e_v, [(; , m), B, n], [N, R, R])
δ (e_v, [(; , m), o, B])](e_{vo}, [(; , m), B, B], [N, N, L])
oϵ {+,-, *, /, <, >, = ,!}n = numeric value
δ (e_{vo}, [(;, m), B, n])] (e_{vo-e}, [(; , m), n, B], [N, N, L])
δ (e_{vo-e}, [(;, m), n, n_1])] (e_v, [(; , m), B,$n_1$on], [N, R, R])
δ (e_v, [(;, m), B, B])] (e_{v-1}, [(; , m), B, B], [R, N, L])
δ (e_{v-1}, [(a, m), B, n])] (e_{v-2}, [(a, m), n, m], [L, R, R])
δ (e_{v-2}, [(a, m), B, B])] (e_{v-3}, [(a, m), B,W], [L, N, R])
δ (e_{v-3}, [(a, m), B, B])] (e_{v-3}, [(a, m), B, B], [L, N, N])
δ (e_{v-3}, [(;, m), B, B])] (e_{v-4}, [(;, m), B, B], [L, N, N])
δ (e_{v-4}, [(η, m), B, B])] (v_{pn}, [(η, m), B,B], [L, N, N])

The next moves from here are discussed in the variable part. For every expression we first push ';' as we told before, then convert the expression into postfix then evaluate, then pop';'.

Conditional blocks: We have enhanced the conditional blocks part to perform nested if else. It was not supported in model developed by Khan and Khiyal (2006).

Conditional blocks are of two types

Block like if else statements in C: when it reads an α, it goes to expression state. In expression if it reads a β then it evaluates and the Boolean result is in the third tape (stack) it also goes to the condition state. If the condition is false it goes to false state and do nothing until it goes to a normal state ('s') in false state it only moves right but when reads an α it just push an α and when it finds ε it just pops the α. If it finds ϵ in false state and the stack-top is false then it pops the false and goes to normal state. If it reads χ then it goes to else part so if before it was false (there was false at the top) then it goes to true state else it goes to false state. The moves are as follows:

δ (s, [(α, m), B, B])] (e, [(α, m), B,;], [R, N, R])
δ (e, [(β, m), B, B])](e_{v-p}, [(β, m), B, B], [N, N, L])
δ (e_v, [(β, m), B, B])] (c, [(β, m), B, B], [R, N, L])
δ (c, [(a, m), B, F])] (s_f, [(a, m), B, F], [N, N, N])
δ (s_f, [(a, m), B, w])] (s_f, [(a, m), B, w], [R, N, N])
wε {F,α}
δ (s_f, [(a, m), B, F])] (s_{fa}, [(α, m), B, F], [R, N, R])
δ (s_{fa}, [(a, m), B, B])] (s_f, [(a, m), B,α], [R, N, N])
δ (s_f, [(ε, m), B,α])] (s_f, [(ε, m), B, B], [R, N, L])
δ (s_f, [(ε, m), B,F])] (s, [(ε, m), B, B], [R, N, N])
δ (c, [(a, m), B,T])] (s, [(a, m), B, B], [N, N, N])
δ (s, [(χ, m), B, B])] (s_f, [(χ, m), B, F], [R, N, N])
δ (s_f, [(χ, m), B, F])] (s, [(χ, m), B, B], [R, N, N])
δ (s, [(ε, m), B, B])] (s, [(ε, m), B, B], [R, N, N])

Block like switch statements in C: For some simplicity reason of this design this part is not covered.

Loops: We have enhanced the loops part to perform nested loop. It was not supported in model developed by Khan and Khiyal (2006).

When there is an instruction symbol for loop. It push the program tape number as return point. Then it checks the condition. Depending on the result of the check it goes to either true or false state and remains there till it gets end loop symbol. At end if it gets the end symbol while false state it pops the top (return point) else if it was in true state while reading the end loop symbol it goes back to the old point to check the condition again. The moves are as follows:

δ (s, [(Π, m), B, B])] (e_{loop}, [(Π, m), B, m], [R, N, R])
δ (e_{loop}, [(a, m), B, B])] (e, [(a, m), B, ;], [N, N, R])
δ (e, [($\prod$, m), B, B])] (e_{v-p}, [($\prod$, m), B, B], [N, N, L])
δ (e_v, [(Π, m), B, B])] (c_l, [(Π, m), B, B], [R, N, L])
δ (c_l, [(a, m), B, F])] (s_{lf}, [(a, m), B, F], [N, N, N])
δ (s_{lf}, [(a, m), B, x])] (s_{lf}, [(a, m), B, x], [R, N, N])
δ (s_{lf}, [(Ω, m), B, F])] (s_{lf3}, [(Ω, m), B, B], [R, N, L])
δ (s_{lf3}, [(a, m), B, x])] (s, [(a, m), B, B], [N, N, N])
δ (s_{lf}, [($\prod$, m), B, x])] (s_{lf}, [($\prod$, m), B, x], [R, N, R])
δ (s_{lf}, [(a, m), B, B])] (s_{lf}, [(a, m), B, $\prod$], [R, N, N])
δ (s_{lf}, [(Ω, m), B, $\prod$])] (s_{lf2}, [(Ω, m), B, B], [R, N, L])
δ (s_{lf2}, [(a, m), B, x])] (s_{lf}, [(a, m), B, B], [N, N, L])
δ (c_l, [(a, m), B, T])] (s, [(a, m), B, B], [N, N, N])
δ (s, [(Ω, m), B, B])] (s_r, [(Ω, m), B, B], [L, N, L])
δ (s_r, [(a, m), B, x])] (s_r, [(a, m), B, x], [L, N, N])
δ (s_r, [(a, m), B, m]) (s_r, [(a, m), B, B], [L, N, N])

Function definition/calling: We have enhanced the function calling part to support recursive function. It was not supported in model developed by Khan and Khiyal (2006).

Function is defined before the main program starts. Unlike the variable definition, there is no move for function definition area. It is covered at S state before reading # and going to s state. Function is defined as follows:

$$\gamma\ F_n \Phi\ \Delta\ V_n \Phi\ \delta V_m\ \eta\ \psi, \Delta\ V_n \Phi\ \delta V_m\ \eta\ \psi,..\ \theta\ \{\text{Body block again coded as the main program is coded. }\}\xi$$

The parameter variables are all by reference whether going in or out. That is there format is $\Delta\ V_n \Phi\ \delta V_m\ \eta\ \psi$ Where

- γ, the left end symbol of definition,
- F_n is a function name,
- Φ symbol used as separator between function name and parameters,
- $\Delta\ V_n \Phi\ \delta V_m \eta\ \psi$ is a variable referenced by value details of which is already discussed at variable part,
- θ, the left point of definition before the body.
- Then comes the body of function as sequences of code.
- ξ, the last point of body.

Each occupies one block of tape, as all of these are tokens.

Function is used as follows:

$$\lambda\ F_n (\delta\ V_n\ \eta, \delta\ V_n\ \eta \ldots.)\ \sigma$$

Where

- λ, the left end symbol of definition,
- F_n is a function name,
- $\delta V_n\ \eta$ is the variable call that is already discussed at variable part.
- σ, the left point of function call.

When TM read a Function call it goes to function call state and then it reach to the end of the call point 'σ', then push the current program point (block number of program tape), the box address as return point then it start moving left and push the parameters from the left side, at last push the Function name then it starts moving left going to state finding function. When it finds it, read a Φ followed by the matched name. Point to be noted that when it read Φ it comes to know that there is a function name at left, so it check the left box and if left box match the top-stack the definition is found). It starts moving right two boxes and then parameters are starting there. It first then push # then goes to the last part of definition and start coming back and pushing the whole function body until the starting of function. Then it goes to the end of the tape, starts popping the whole function. When it pop # it assumes the next is parameters so it places parameters in appropriate places. Then it goes to the s state and start executing body. At the end (when it read Ω) it again goes to the return point from where it was called.

We have told in the variable part that global variables using inside a function body must be defined before the function definition so there is no problem in global variable retrieval process.

Moves

δ (s, [(λ, m), B, B)] (f_{unc}, [(λ, m), B, B], [R, N, N])
δ (f_{unc}, [(a, m), B, B)] (f_{unc}, [(a, m), B, B], [R, N, N])
δ (f_{unc}, [(σ, m), B, B)] (f_{pr}, [(σ, m), B, B], [R, N, N])
δ (f_{pr}, [(a, m), B, B)] (f_{pp}, [(a, m), B, m], [L, N, R])
δ (f_{pp}, [(a, m), B, B)] (f_{pp}, [(a, m), B, α], [L, N, R])
δ (f_{pp}, [(x, m), B, B)] (f_{pp}, [(x, m), B, B], [L, N, N])
xϵ[δ, η, σ]
δ (f_{pp}, [(λ, m), B, B)] (f_{ff}, [(λ, m), B, B], [L, N, L])
δ (f_{ff}, [(a, m), B, x)] (f_{ff}, [(a, m), B, x], [L, N, N])
δ (f_{ff}, [(ϕ, m), B, x)] (f_{ff}, [(ϕ, m), B, x], [L, N, N])
δ (f_{fn}, [(a, m), B, x)] (f_{ff}, [(a, m), B, x], [L, N, N])
δ (f_{fn}, [(x, m), B, x)] (f_{n}, [(x, m), B, #], [R, N, R])
δ (f_{n}, [(a, m), B, B)] (f_{n}, [(a, m), B, B], [R, N, N])
δ (f_{n}, [(ξ, m), B, B) (f_{pf}, [(ξ, m), B, B], [N, N, N])
δ (f_{pf}, [(a, m), B, B)] (f_{pf}, [(a, m), B, a], [L, N, R])
δ (f_{pf}, [(γ, m), B, B)] (f_{ge}, [(γ, m), B,γ], [R, N, N])
δ (f_{ge}, [(a, m), B, x) (f_{ge}, [(a, m), B, x], [R, N, N])
δ (f_{ge}, [(B, m), B, x)] (f_{sf}, [(x, m), B, B], [R, N, L])
δ (f_{sf}, [(B, m), B, x)] (f_{sf}, [(x, m), B, B], [R, N, L])
δ (f_{sf}, [(B, m), B, #)] (f_{sps}, [(B, m), B, B], [L, N, L])
δ (f_{sps}, [(a, m), B, x)] (f_{sps}, [(a, m), B, x], [L, N, N])
δ (f_{sps}, [(γ, m), B, x)] (f_{sp}, [(γ, m), B, x], [R, N, N])
δ (f_{sp}, [(a, m), B, x)] (f_{sp}, [(a, m), B, x], [R, N, N])
δ (f_{sp}, [(ϕ, m), B, x)] (f_{sp}, [(#, m), B, x], [R, N, N])
δ (f_{sp}, [(Δ, m), B, x)] (f_{sp-1}, [(Δ, m), B, x], [R, N, N])
δ (f_{sp-1}, [(a, m), B, x)] (f_{sp-1}, [(a, m), B, x], [R, N, N])
δ (f_{sp-1}, [(δ, m), B, x)] (f_{sp-2}, [(δ , m), B, x], [R, N, N])
δ (f_{sp-2}, [(a, m), B, x)] (f_{sp-1}, [(x , m), B, B], [R, N, L])
δ (f_{sp-1}, [(θ, m), B, x)] (s, [(θ, m), B, x], [R, N, R])
δ (f_{sp}, [(θ, m), B, x)] (s, [(θ, m), B, x], [R, N, R])
δ (s, [(ξ, m), B, B)] (f_{d}, [(B, m), B, B], [L, N, N])
δ (f_{d}, [(a, m), B, B)] (f_{d}, [(B, m), B, B], [L, N, N])
δ (f_{d}, [(γ, m), B, B)] (s_{r}, [(B, m), B, B], [L, N, L])

CONCLUSIONS

Khan and Khiyal (2006) have described one component of our whole Turing machine such components communicate

among themselves by the moves described in the send receive section. This way the whole Turing Machine works as a model of distributed computing.

In this study we have enhanced the above mentioned model and enabled it to solve equation with several operands on the right hand, also to perform nested conditional blocks, nested loop and updated moves of function execution. The new model has been tested using MS visual C++.

As future work we may include more complex tasks such as object oriented structure etc.

REFERENCES

Guetta, D., 2003. Turing Machine development environment. 19 Church Mount, London, N2 0RW, England.

Hopscroft, J.E., R. Motwani and J.D. Ullman, 2000. Introduction to Automata Theory, Languages and Computation. 2nd Edn., Addison Wesley CO, USA.

Khan, S.R. and M.S.H. Khiyal, 2006. Turing Model for distributed computing. Information Technol. J., 5: 305-313.

Khiyal, M.S.H., 2004. Theory of Automata and Computation, National Book Foundation, Islamabad, Pakistan.

Turing, A.M., 1936. On Computable numbers with an application to the Entscheidung sproblem (decision problem). Proceed. London Mathematical Soc., 42: 230-265.