IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 2, March 2022
Impact Factor: 6.252

# Turning Machine Multiplication and Division

**Abhishek S Harsoor, Gudugunti Anirudh, Ijaz Ibrahim, Gulam Rabbani**
Students, Department of Computer Science and Engineering
Alva's Institute of Engineering and Technology, Mijar, Mangalore, Karnataka, India
abhishekharsoor@gmail.com,anirudhg2308@gmail.com, ijazibrahim818@gmail.com, g.rabbani4me07@gmail.com

**Abstract:** *This paper introduces a Turing machine, pushdown automata simulators, multiplication, and division using The Turing machine as a virtual environment for learning computational models and automata theory. On a Turing machine, the multiplication and division problems are solved. The simulators' effectiveness in the classroom has been demonstrated in a preliminary study.*

**Keywords:** Simulators, Finite Automata, Turing Machines, Computational Science, Pushdown Automata

## I. INTRODUCTION

Computational science is an interdisciplinary field that studies systems of real-world problems using mathematical models and scientific computing methods. In practice, it refers to the use of computer simulation and other forms of computation to solve issues in a variety of scientific fields [3]. Automata theory is one of these mathematical models. Turing machines and pushdown automata are examples of automata theory concepts that are useful in building and assessing computational models for a variety of hardware and software applications. These topics are abstract, thus they're usually taught in a typical lecture-driven format that's good for introspective learners. According to learning science research [4], Students in computer engineering have strong active preferences; therefore, a lecture-based teaching method is less stimulating for them.

Our multiplication solution is more optimized than one found on the internet, while our division solution is brand new and not found on the internet. Our simulators are built to address this problem and cater to computer engineering students' active learning preferences. Not just for automata theory, but also for a variety of other courses such as the theory of computing, discrete mathematics, computational models, programming languages, compiler design, and other relevant courses, our approach can be used as a supporting tool for active learning. Finite-state automata, pushdown automata, and Turing machines are among the subjects covered in such courses.

Our simulators are written in the Java programming language, making them portable, machine-independent, and web-based. As a result, they're useful for interactive and online automata training.

We used the active construction learning paradigm [7, 2] to develop our simulators, which comprises several basic design concepts, including the following:

1. Teachers serve as facilitators rather than transmitters of knowledge. This means that rather than being passively communicated by teachers, knowledge must be actively generated by learners.
2. Learning should be carried out in a collaborative setting.

Experiments were conducted to demonstrate the efficiency of our simulators as a model of an interactive learning aid. The preliminary findings of these trials revealed that utilizing our simulators boosted learners' motivation to actively participate in the learning process of relevant subjects and seek more knowledge on their own.

Even though we focus on automata theory, our study can be used as a model for a wide range of undergraduate topics. The following is the breakdown of a paper's structure. The second portion, which follows the introduction, introduces related work. The third section provides an overview of automata subjects including Turing machines and pushdown automata. In section four, we'll talk about how we built our simulators. The evaluation of the environment's performance will be presented in section five. The paper will be concluded in section six, which will discuss future work.

## II. RELATED WORK

Several finite-state automata simulators have been developed (e.g. [8, 9, 10, 6, 5, 1]) to reinforce the training of automata topics. Most of them suffer from one or more flaws that make them less effective (motivating) as a learning tool, particularly for fewer advanced students. For example, the tools PetC in lack visual clarity and dynamic capability. labels on arrows

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 2, March 2022
Impact Factor: 6.252

cannot be distinguished When designing the automaton on PetC editor and trying to connect two states in both directions. This becomes visually terrible while the automaton is getting bigger. JFLAP [1] is a comprehensive automata tool but it requires skilled learners who already know the fundamentals of automata to form full use of its rich operations. The automata tools in [5] are powerful but do not provide a convenient mechanism for displaying and visually stimulating the finite state machines. The ASSIST automata tools in [6] are difficult to set up and use. Almost all are designed as tools for advanced learners. These tools assume that the learners have already grasped the elemental concepts. They lack a clear workflow of learning activities that can guide the new learners on how and where to start using the system. This makes it difficult for brand spanking new students to navigate through the system. They are also hooked into advanced mathematical and idiosyncratic user interactions. On the contrary, our simulators are designed with a transparent workflow of learning activities and hence it's easy to use and straightforward to find out for brand spanking new users.

## III. AUTOMATA TOPICS

### 3.1 Finite Automata

Finite state machines or automata (FA) are mathematical models for a variety of software and hardware devices. Automata have a wide range of software and hardware applications. It can be used to mimic anything from a simple text editor to a more complex compiler in software design. It can be used to model puzzles, tennis games, and a variety of other games in computer gaming. It can be used to mimic the function of a range of machines in hardware design, such as vending machines, elevators, video players, rice cookers, and so on.

Automata theory, formal languages, theory of computations, computational models, discrete mathematics, programming languages, and compiler design are all topics that cover finite state machines. We'll give a quick review of finite state machines in this section.

A finite state machine is a machine having a finite number of states and a control unit that can alter the current state of the machine to a new state in response to an external impact (input). It has limited memory capacity, making it a good fit for applications that don't need to know what happened before.

The finite state machine is classed as deterministic (DFA) if the controller can shift from one state to another (one) state, or nondeterministic (NFA) if the controller cannot change from one state to another (one) state.

-NFA): if it can change states in response to empty (no) input in addition to NFA. Formally, a finite state machine A is defined as a 5-tuple A= ( 0, F), where Q may be a finite set of states that represent the machine states, is the set of possible inputs to the machine, represents the finite state machine controller, q0 Q is that the initial (starting) state of the machine and F and Q is that the set of possible final (accepting) states of the machine. Depending on how the machine controller -NFA is configured.

- If $\delta$: Q x $\Sigma \rightarrow$ Q then the machine is a Deterministic finite automata.
- If $\delta$: Q x $\Sigma \rightarrow 2^Q$ then the machine is an Non deterministic finite automata.
- If $\delta$: Q x ($\Sigma$U $\lambda$) $\rightarrow 2^Q$ -NFA.

If the machine controller scans all of the inputs starting from the initial state and stops at one of the final states, the sequence of inputs is said to be accepted (recognized) by the finite state machine. Regular languages are a type of language that can be accepted by a finite state machine. The three finite state machine -NFA models are interchangeable. To put it another way, we may convert any sort of machine into the other. By definition, we can see DFA $\subseteq$ NFA $\subseteq$ $\lambda$ NFA, but we can transform NFA to NFA and NFA to DFA

### 3.2 Pushdown Automata

A pushdown automaton (PDA) is a finite automaton that can work with a data stack. In two aspects, pushdown automata differ from traditional finite state machines:

1. They can choose which transition to make by looking at the top of the stack.
2. They can manipulate the stack as part of a transition.

Figure 1- shows a schematic illustration of PDA.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 2, March 2022
Impact Factor: 6.252

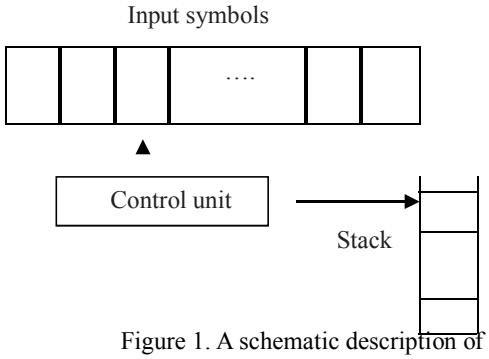


Figure 1. A schematic description of Pushdown automata.

Starting with the leftmost input symbol, the control unit scans the input symbols from left to right. Each control unit movement receives a symbol from the input while also modifying the contents of the stack using standard stack operations like push and pop. The current input symbol, as well as the symbol currently on top of the stack, dictates each movement of the control unit. The move results in a new state for the control unit as well as a shift in the stack's top.

A personal digital assistant (PDA) is formally described as a 7-0 Z, F) where:

- Q is a set of states with a finite number of possibilities.
- Σis a finite set which is called the input alphabet
- ⌈ is a finite set which is called stack alphabet
- δ: Q X Σλ X ⌈λ → P (Q X ⌈λ) is the transition function
- q0 is the start state
- Z is the initial stack symbol
- F ⊆ Q is the set of accepting (final) states

Where P(S) is power set of S, and λ is the empty string, Σλ = Σ U {λ} and ⌈λ = ⌈λ U { λ}

The relevant factors at any time are the triplet (q,w,u ), where q is the current state of the control unit, is the unread part of the input string, and is the current contents of the stack. The triplet (q,w,u) is called instantaneous description (ID). A moves from one ID to another will be denoted by the symbol |-; thus

(q1,aw,bx) |- (q2, yx) is possible if and only if (q2 y) (q,a,b).

Symbolized moves with an arbitrary number of step Pushdown automata are divided into two types: deterministic (DPDA) and non-deterministic (NDPA) (NPDA). DPDA and NDPA are not the same as finite automata. NPDA has a greater expressive range than DPDA.

### 3.3 Turing Machines

The most powerful finite state machines are Turing machines (TMs). They are capable of simulating the functions of a digital computer. Turing machine is made up of a finite number of states and a controller that can read or write symbols on infinite length tape. NFA is not the same as -NFA, DFA, and PDA, On the tape, a TM controller can travel in both directions. The machine starts with an initial state, a finite number of input symbols written on the tape (the remainder of the tape is blank), and therefore the controller is about to the first input symbol from the left. The controller makes the next step based on the current state and the current scanned symbol on the input tape. It can either replace or leave the current scanned symbol alone, change the current state, then move to the left or right, and so on. The machine will come to a standstill if no more moves are possible. In some situations, the machine may operate indefinitely without stopping. Turing machines, on the other hand, can be used to compute functions in the same manner that modern digital computers can. The function arguments are represented as sequences of 1s separated by 0s and written to the machine's tape in this example. A collection of rules suited for the machine's transition function is used to define the function. The machine then processes those data. The value of the function application on the arguments is represented by the output symbols left on the tape if it halts. While teaching these concepts in a traditional lecture-driven style, we noticed that inexperienced (novice) learners have difficulties fully understanding these basic concepts. learners became less motivated to actively participate in the class.

To address this problem, we created a simulator that visualizes and interacts with these concepts in a way that is more appropriate for engineering students. Learners may simply design, modify, and replicate a PDA or a Turing machine using

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 2, March 2022
Impact Factor: 6.252

the simulator's editor. Then students can interactively simulate the machine with any desired input to see examine how it responds to that input (in discrete steps or as a whole).

## IV. SIMULATORS

### 4.1 Pushdown Automata Simulator

The Pushdown Automaton Simulator allows students to graphically sketch an automaton and then operate on it. During these procedures, they can watch for any changes in the automaton. They can, for example, check whether an input is accepted or rejected while observing the accompanying changes in the automata states and stack contents. They can also zoom in and out and do auto outlay to better view the automaton. When the underline automaton is enormous, these last procedures are really important. Figure 2 depicts the PDA simulator interface.

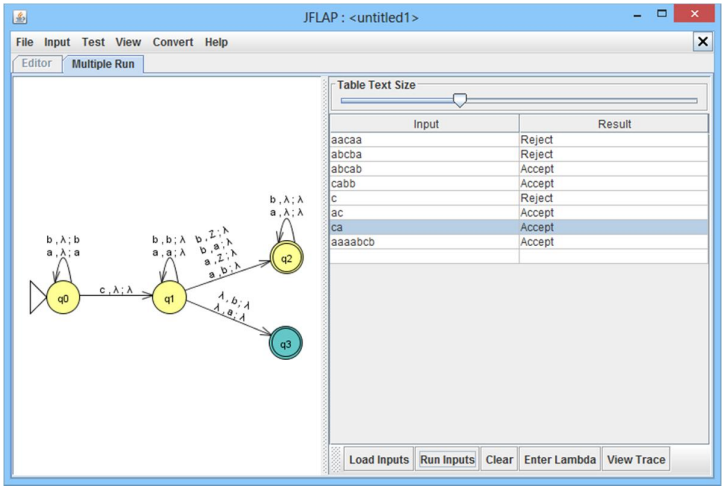


**Figure 2.** Pushdown automata simulator

### 4.2 Turing Machine Simulator

As previously stated, the normal lecture-driven approach to teaching and learning Turing machines is time-consuming and difficult for average students to understand. A Turing machine simulator component is implemented into the environment to aid the teaching-learning of basic Turing machine ideas for typical engineering students. Learners can easily construct their Turing machine and follow along as it works on any given input in a step-by-step manner. They can create machines that can act as language recognizers as well as function computers. It includes a user-friendly UI and some useful features.

**A. Multiplication:**

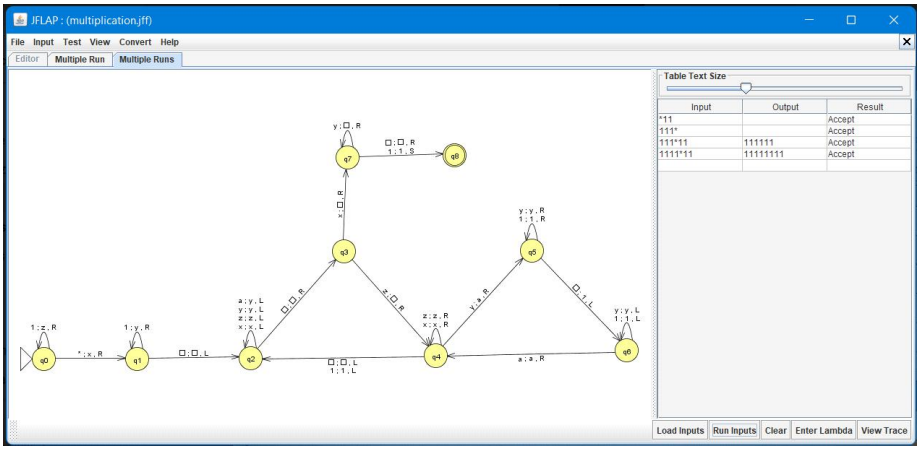


**Figure 3.** Multiplication using Turing machine

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 2, March 2022
Impact Factor: 6.252

**Algorithm of Multiplication:**

step 1: initial state.
# Loop if '1' appears replace it with 'z' go to the right.
# if '*' appear replace it with 'x' go to the right and to step 2.

step 2: # Loop if '1' appears to replace with 'y' go to the right.
# if blank appears ignore go to the left and to step 3.

step 3: # loop if 'a','y','z','x' appear ignore go to left.
# if blank appears ignore go to the right and to step 4.

step 4: # if 'x' appears replace it with blank go to right and to step 8.
# if 'z' appears replace it with blank go to right and to step 5.

step 5: # Loop if 'x','z' appear ignore go to right.
# if 'y' appear replace with 'a' go to the right and to step 6.
# if blank,'1' appears ignore go to the left and to step 3.

step 6: # Loop if '1','y' appear ignore go to right.
# if blank appears replace with '1' go to the left and to step 7.

step 7: # Loop if '1','y' appear ignore go to left.
# if 'a' appear ignore go to the right and to step 5.

step 8: # Loop if 'y' appear to replace with blank go to right.
# if blank appears ignore go to the right and to step 9.
# if '1' appears ignore be in the same place and to step 9.

step 9: # accepting state.

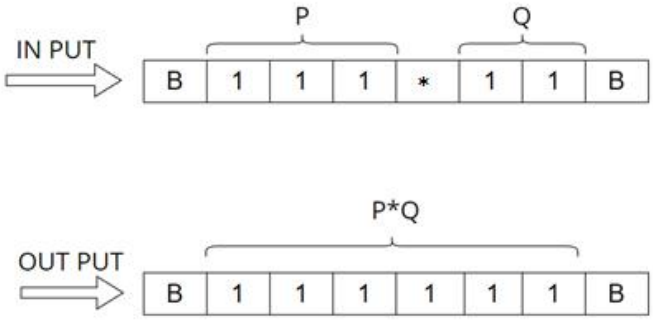


Figure 4. Input and Output sample

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 2, March 2022
Impact Factor: 6.252

**B. Division:**

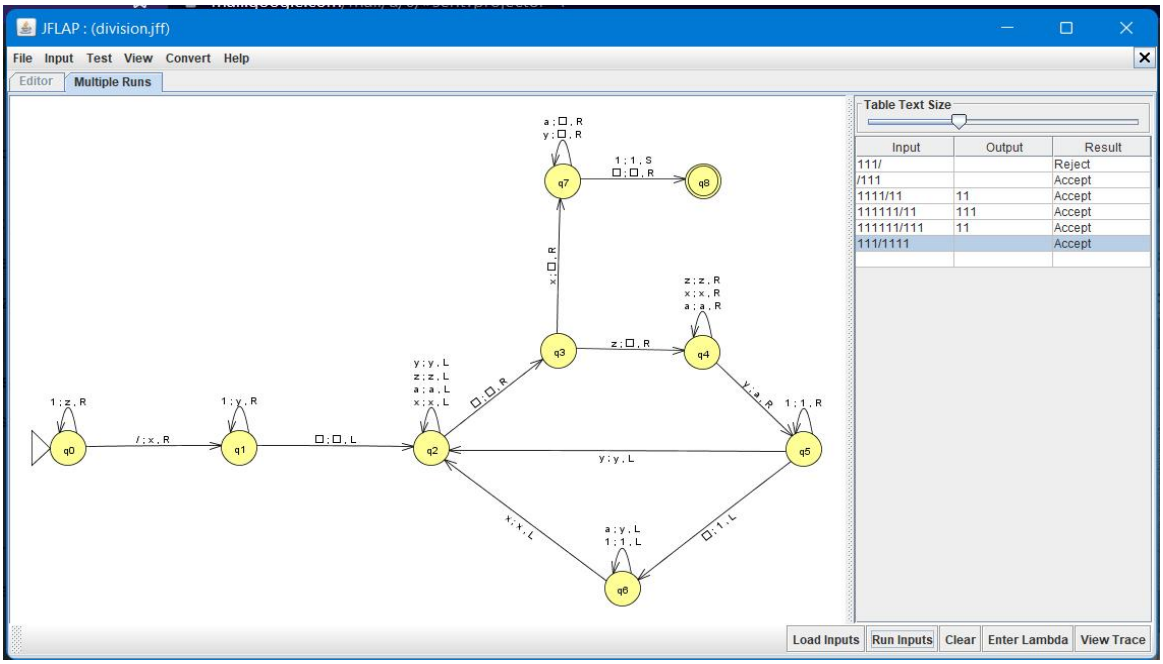


**Figure 5.** Division using Turing machine

**Algorithm of Division:**

step 1: initial state.
# Loop if '1' appears replace it with 'z' go to the right.
# if '/' appears, replace it with 'x' go to the right and step 2.

step 2: # Loop if '1' appears to replace with 'y' go to the right.
# if blank appears ignore go to the left and to step 3.

step 3: # loop if 'a','y','z','x' appear ignore go to left.
# if blank appears ignore go to the right and to step 4.

step 4: # if 'x' appears replace it with blank go to right and to step 8.
# if 'z' appears replace it with blank go to right and to step 5.

step 5: # loop if 'a','z','x' appear ignore go to right.
# if 'y' appear replace with 'a' go to the right and to step 6.

step 6: # loop if '1' appears ignore go to the right.
# if blank appears replace with '1' go to the left and to step 7.
# if 'y' appear ignore go to the left and to step 3.

step 7: # loop if '1' appears ignore go to left.
# Loop if 'a' appear to replace with 'y' go to left.
# if 'x' appears ignore go to the left and to step 3.

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 2, Issue 2, March 2022
Impact Factor: 6.252

step 8: # Loop if 'a','y' appear to replace with blank go to right.
# if blank appears ignore go to the right and to step 9.
# if '1' appears ignore be in the same place and to step 9.

step 9: # accepting state

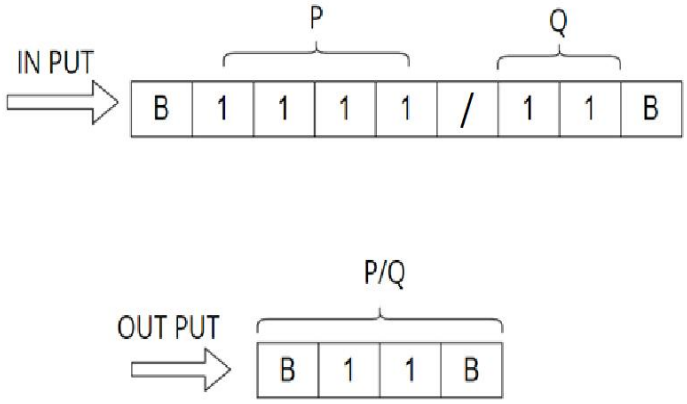


Figure 6. Input and Output sample

## REFERENCES

**[1].** S. Rodger, Visual and Interactive tools. Website of Automata Theory tools at Duke University, http://www.cs.duke.edu/~rodger/tools/. 2006.
**[2].** G. Wilson, Ed. Constructivist Learning Environment: Case Studies on Instructional Design. NJ: Educational Technology, Englewood Cliffs,1998.
**[3].** Wikipedia: http://en.wikipedia.org/wiki/Computational_science. Accessed on Feb. 2012.
**[4].** mathematics, engineering, Transforming undergraduate education in science, and technology. In Center for Science, Mathematics, "Committee on Undergraduate Science Education", and Engineering.
**[5].** M. Mohri, F. Pereria, and M. Riley, AT&T FSM Library. Software tools. 2003. Available at: http://www.research.att.com/sw/tools/fsm/.
**[6].** E. Head, ASSIST A Simple Simulator for State Transitions. Master Thesis. The State University of New York at Binghamton. 1998. Related website at: http://www.cs.binghamton.edu/~software/.
**[7].** S. Hadjerrouit, Proc. E-Learning Corporate, E-Learn-World Conf, Toward a constructivist approach to e-learning in software engineering. Higher Education, Phoenix, Government, Healthcare, AZ, pp. 507-514, 2003.
**[8].** H. Bergstrom, Minimization, Applications and Visualization of FSM. Master Thesis. Stockholm University, 1998. Related website at: http://www.dsv.su.se/~henrikbe/petc/.
**[9].** J. Bovet, a tool for simulating automata, Visual Automata Simulator, and Turing machines. The University of San Francisco. Available at: http://www.cs.usfca.edu/~jbovet/vas.html, 2004.
**[10].** N. Christin, DFApplet, a deterministic finite automata simulator. Available at: http://www.sims.berkeley.edu/~christin/dfa/. 1998.