Fifty Years of Automata Simulation: A Review

• Pinaki Chakraborty • P.C. Saxena • C.P. Katti •

Automata theory is an important subject in computer science and quite consequently, simulation of automata for pedagogical purposes is an important topic in computer science education research. This article reviews the major initiatives in the field of simulation of automata in the last five decades with emphasis on those automata simulators actually used at universities for teaching. A classification of the automata simulators on the basis of their design paradigms has been developed where they have been classified broadly into language based automata simulators and visualization centric automata simulators. Some salient trends in the research on simulation of automata are also identified. The article concludes with an advocacy for continuing research on simulation of automata and integration of automata simulators in teaching.

INTRODUCTION

Automata theory has evolved over the last several decades as an important foundation of computer science. Today, every curriculum on computer science contains at least one course on automata theory. We know automata theory for its formal representations of computing systems and processes. Consequently, the principles of automata theory are routinely applied in various fields of computer science. While research on newer and advanced forms of automata is in progress, the basic forms of automata continue to hold an important place in the computer science curricula.

Educationists were early to understand that it is difficult to teach and learn automata theory. They thought that perhaps the best way to teach and learn automata theory is to take help of pedagogical tools. Since automata theory revolves around abstract machines and processes, automata simulators were conceived as the most common form of pedagogical tools on automata theory. The proliferation of electronic computers in academia helped the cause of the educationists as they started developing automata simulators in the early 1960s [1]. This article reviews major automata simulation projects undertaken in the last five decades to simulate the basic forms of automata, *viz*. finite automata, finite transducers, pushdown automata and Turing machines.

With the emergence of so many automata simulators over the years, their classification has become necessary. Chesñevar, *et al.* [2] have earlier classified simulators and other pedagogical tools on automata theory into tools that support only one form of automata and tools that support multiple forms of automata. We present a more elaborate classification of automata simulators (Figure 1). Automata simulators can be classified based on their design paradigms into language based automata simulators and visualization centric automata simulators. In a language based automata simulator, the definition of an automaton is written in a predefined symbolic language and processed using tools like compilers and interpreters. We can further classify the language based automata simulators into notational language based automata simulators, assembly-like language based automata simulators, procedural language based automata simulators, and descriptive language based automata simulators. Alternatively, a visualization centric automata simulator accepts the specification of an automaton and graphically simulates its working. We can further classify the visualization centric automata simulators into those accepting structured inputs and those accepting diagrammatic inputs.

The rest of this article is organized as follows. Section 2 reviews the language based automata simulators. Section 3 reviews the visualization centric automata simulators. Section 4 tries to identify the trends in research on simulation of automata, and Section 5 concludes the discussion.

LANGUAGE BASED AUTOMATA SIMULATORS

The language-based approach is the older among the two basic approaches of simulating automata. To use a language based automata simulator, the definition of an automaton is written as a program in a symbolic language. There are two methods of processing this program. In the first method, the program is first compiled into an intermediate language. The compiler makes sure that the program is error free. Then an interpreter is used to simulate the working of the automaton for an arbitrary input string. Various tools, as those to display transition diagrams and convert an automaton into another form, may also use the intermediate program. In the second method, the program is simulated directly by the interpreter. The interpreter has to perform lexical and syntax analyses itself, and it is also responsible for error handling. The symbolic languages accepted by the automata simulators are of different types as discussed in subsections 2.1 to 2.4.

2.1 Notational Language Based Automata Simulators

In a notational language, an automaton is defined using formal symbols. The symbols are typically short and there are strict limitations on the choice of the symbols. A rigid structure is enforced on the programs often using space and newline characters. Notational languages characteristically do not provide any construct for data abstraction or flow control. Notational languages are easy to learn and we can use them efficiently to define simple automata. Consequently, several researchers have endorsed the notational language based approach of automata simulation.

In what was perhaps the first study on automata simulators, Coffin *et al.* [1] developed a software tool based on a notational language to simulate Turing machines. In this language, a Turing machine is represented by a series of quintuples each denoting a transition. A translator program, called the builder, transforms the quintuples into their machine language equivalents. Then another program, called the driver, simulates the working of the Turing machine. The tool executed on an SDS 920 machine and it achieved satisfactory performance. The developers observed that the tool is useful for problem solving, algorithm validation and most importantly teaching students the fundamentals of programming.

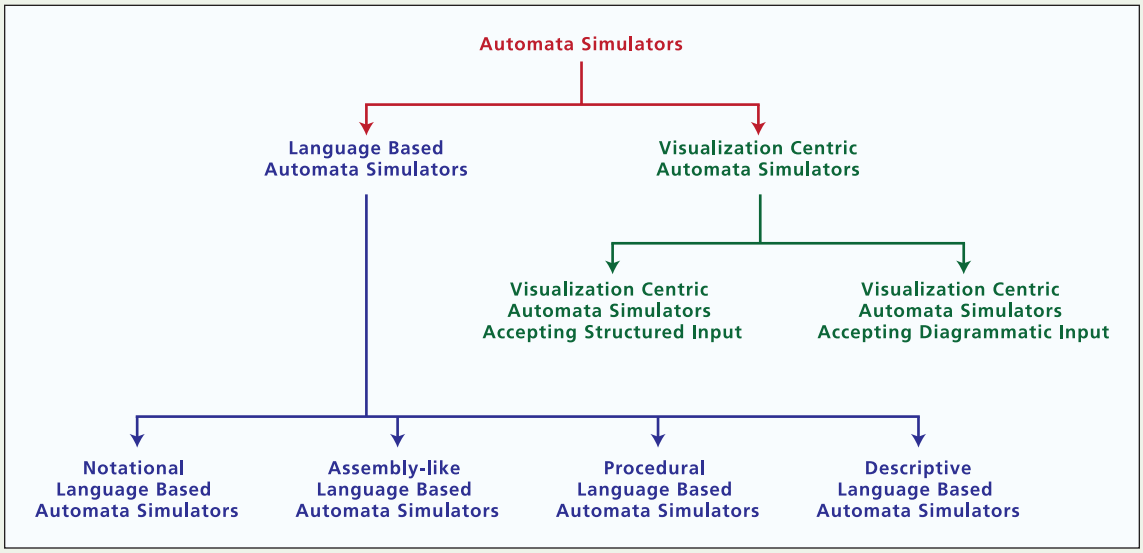

Figure 1: A classification of automata simulators

Head [3] has developed A Simple Simulator for State Transitions. It is a suite comprising of a Finite State Machine Simulator, a Nondeterministic Pushdown Automaton Simulator and a Turing Machine Simulator. The simulators are based on notational languages with rigid formats enforced by space and newline characters. The Finite State Machine Simulator (Figure 2) and the Nondeterministic Pushdown Automaton Simulator support both deterministic and nondeterministic automata. The Turing Machine Simulator supports Turing machines with semi-infinite or infinite tapes, multiple tapes and multi-track tapes. The simulators are available in four different versions, *viz.* graphical, short text only, full text only and course grader's quick to use (batch). The graphical versions are well developed and interactive. The simulation can be performed either stepwise or continuously at different speeds. Moreover, there is also an option for reverse simulation.

```
DFA//Type
DIV5//Title
0 1//Input alphabet
qi q0 q1 q2 q3 q4//States
qi//Initial state
q0//Final state
qi 0 q0//Transitions
qi 1 q1
q0 0 q0
q0 1 q1
q1 0 q2
q1 1 q3
q2 0 q4
q2 1 q0
q3 0 q1
q3 1 q2
q4 0 q3
q4 1 q4
end
```

Figure 2: Definition of a deterministic finite automaton for A Simple Simulator for State Transitions. The deterministic finite automaton accepts all binary numbers whose decimal equivalents are divisible by 5. (The same example has been used throughout this paper to illustrate the designing of deterministic finite automaton by various tools)

Harris [4,5,6] developed a suite of tools to simulate finite automata, pushdown automata and Turing machines based on notational languages. In these languages, automata are represented as a series of tuples each of which represents a transition. The tools are interactive and use high quality graphics. The tool suite strives to teach declarative programming and hence supports both deterministic and nondeterministic automata.

Shelburne [7] has developed a Nondeterministic Pushdown Automata Simulator and a Turing Machine Simulator for teaching a course on the theory of computation at the University of Wittenberg. Both simulators are based on notational languages. The description of a pushdown automata or a Turing machine consists of a series of quintuples each denoting a transition (Figure 3). Both simulators are available with suitably developed integrated development environments, which allow creation, modification and simulation of automata. One can perform a simulation stepwise or instantaneously. While simulating a nondeterministic pushdown automaton, the user needs to specify which transition to use whenever there are multiple options. While simulating a Turing machine, the R/W head can move to a position left or right, or stay at the same position during a transition. The stack used for simulating pushdown automata is restricted to 75 characters and the tape used to simulate Turing machines is restricted to 1000 characters on either side of the central

```
Q0,a,A,Q1,R)
(Q1,a,a,Q1,R)
(Q1,B,B,Q1,R)
(Q1,b,B,Q2,R)
(Q2,b,b,Q2,R)
(Q2,C,C,Q2,R)
(Q2,c,C,Q3,L)
(Q3,a,a,Q3,L)
(Q3,b,b,Q3,L)
(Q3,c,c,Q3,L)
(Q3,A,A,Q3,L)
(Q3,B,B,Q3,L)
(Q3,C,C,Q3,L)
(Q3, , ,Q4,R)
(Q4,a,A,Q1,R)
(Q4,A,A,Q4,R)
(Q4,B,B,Q4,R)
(Q4,C,C,Q4,R)
(Q4, , ,Q5,R)
```

Figure 3: Definition of a Turing machine for Turing Machine Simulator. The Turing machine accepts the language $\{a^n b^n c^n \mid n>0\}$. (The same example has been used throughout this paper to illustrate the designing of Turing machine by various tools)

position. The simulators have been developed for the MS DOS platform and feature professional grade interactive graphics.

Scott [8] developed a Turing machine simulator and used it in an introductory course on computer science at the University of Northern Colorado. The simulator takes as input the description of a Turing machine in a notational language and simulates its workings. The description of a Turing machine comprises of a series of tuples each representing a transition.

In a recent study, Erlacher [9] developed a Pushdown Automata Simulator. The simulator takes as input the description of a pushdown automaton in a notational language. The programs follow a rigid format enforced using commas and semicolons. A lexer and a parser verify the correctness of the program. The simulator has a graphics based interactive interface for creating, modifying and simulating pushdown automata. During simulation, options for moving a single step forward or backward, resetting and completing instantaneously are available. The simulator supports both deterministic and nondeterministic automata. While simulating a nondeterministic automaton, options to select transitions manually or automatically are available. The simulator also supports conversions between pushdown automata that accept by empty stack and pushdown automata that accept by final states.

2.2 Assembly-like Language Based Automata Simulators

The behavior of an automaton can be expressed as a program in the assembly language of a hypothetical machine. Such a program consists of simple instructions like those of a typical assembly language. Flow of execution is controlled using conditional and unconditional jump, subroutine call and return instructions. However, there is no construct for data abstraction. Assembly languages are more efficient than notational languages in defining large and complicated automata.

In an early study, Curtis [10] developed a Turing Machine Simulator, which was used as a tool for teaching and research at the Wesleyan University. The simulator is based on an assembly-like language that provides a basic set of instructions to read and write the tape contents, move left and right on the tape, jump conditionally and unconditionally, and invoke subroutines. The simulator

consists of a two-pass loader, one pass to set up symbol tables and another to load the instructions, a module to load the tape and a module to interpret the instructions. The simulator executed on an IBM 1620 machine. The results obtained from experiments on universal Turing machines have been encouraging.

In another study, Rose *et al.* [11] defined an assembly-like language to model automata. The language consists of twenty-six primitive instructions to read and write, manipulate registers, push and pop, and jump conditionally and unconditionally. One can use the language to model finite automata, pushdown automata, and other abstract machines. An interpreter of this language, called Automata, consisting of a parser and an executer was developed. The parser first reads and recognizes the definition of an abstract machine. Then the executer is invoked to simulate the working of the abstract machine. The language and its interpreter have been used as a teaching aid at the Pennsylvania State University. Students and professors have appreciated this approach to teaching. We also observed that the use of this approach results in better insight into the operational nature of the abstract machine among the students.

Pierce *et al.* [12] presented a Turing machine simulator called Tutor. The simulator takes as input the definition of a Turing machine in an assembly-like language. The language has seventeen predefined basic machines that combine to obtain Turing machines of any size and complexity. The simulator executed on an IBM System/360 machine and found to be quite fast with the basic machines requiring 8 to 50 μs to execute.

2.3 Procedural Language Based Automata Simulators

Suitably designed procedural languages can also be used to define automata. Such a procedural language provides various high-level language features for data abstraction and flow control. Procedural languages are inherently more powerful than assembly languages and can be best used to define automata of very large size and complexity.

Knuth and Bigelow [13] demonstrated that the techniques used in programming real computers could also apply to construct programs for automata. They defined a procedural language for modeling stack automata, a generalized form of pushdown automata; they also showed how it could be easily adapt for other forms of automata. The definition of an automaton in this language consists of different types of statements such as conditional statements, jump statements, procedure statements, and accept statements. The programs in this language manually translate into an assembly language called XMAP, assembled and simulated.

2.4 Descriptive Language Based Automata Simulators

Descriptive languages allow defining automata in formal ways similar to those used in textbooks. Unlike notational, assembly-like and procedural languages that need some effort to learn, we can readily use descriptive languages.

Chakraborty *et al.* [14] defined a Finite Automaton Description Language for modeling deterministic and nondeterministic finite automata. In this language, the description of a finite automaton is written in a formal textbook-like way (Figure 4). A finite automaton modeled in this language is compiled by using a fast single-pass compiler. The compiler consists of four phases, *viz.* lexical analyzer, syntax analyzer, semantic analyzer and code generator. Then a suitably designed interpreter simulates the working of the compiled finite automaton.

```
FA=({qi,q0,q1,q2,q3,q4},
{0,1},D,qi,{q0});
D(qi,0)=q0,
D(qi,1)=q1,
D(q0,0)=q0,
D(q0,1)=q1,
D(q1,0)=q2,
D(q1,1)=q3,
D(q2,0)=q4,
D(q2,1)=q0,
D(q3,0)=q1,
D(q3,1)=q2,
D(q4,0)=q3,
D(q4,1)=q4;
```

Figure 4: Definition of a deterministic finite automaton in Finite Automaton Description Language

A tool to display the transition diagram of the compiled finite automaton is also available (Figure 5). Additionally, tools to convert a nondeterministic finite automaton into a deterministic finite automaton and a deterministic finite automaton into a Turing machine are available. A two-pass optimizing compiler is also available but only for deterministic finite automata. It minimizes the number of states in the deterministic finite automata and it also uses other techniques to make the object programs shorter and faster. The optimizing compiler has a code optimizing phase apart from the four phases of its non-optimizing counterpart.

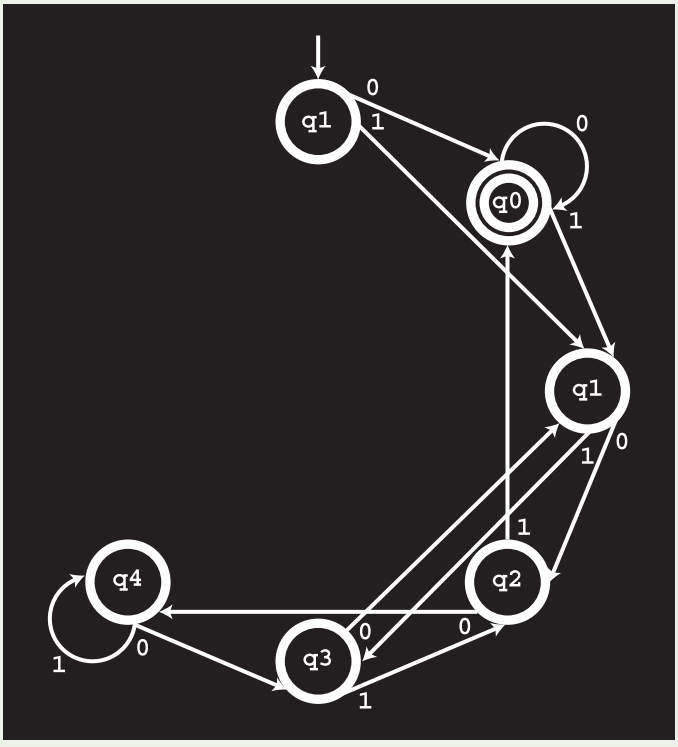

Figure 5: Transition diagram of a deterministic finite automaton as shown by a tool associated with Finite Automaton Description Language

In a similar study, Chakraborty [15] defined a Turing Machine Description Language (Figure 6). The properties and the processing of this language are similar to those of the Finite Automaton Description Language. A single-pass fast compiler, a two-pass optimizing compiler [16, 17], and an interpreter have been developed. The interpreter uses a dynamically allocated array to simulate the tape. The dynamically allocated array is resized and rearranged as per the requirements of the simulation process. The tools have been well accepted by the students of Jawaharlal Nehru University and G. G. S. Indraprastha University; they felt that the tools helped them learn better [18].

```
TM=({q0,q1,q2,q3,q4,q5},
{a,b,c},{a,b,c,A,B,C,_},
D,q0,_,{q5});
D(q0,a)=(q1,A,R),
D(q1,a)=(q1,a,R),
D(q1,B)=(q1,B,R),
D(q1,b)=(q2,B,R),
D(q2,b)=(q2,b,R),
D(q2,C)=(q2,C,R),
D(q2,c)=(q3,C,L),
D(q3,a)=(q3,a,L),
D(q3,b)=(q3,b,L),
D(q3,c)=(q3,c,L),
D(q3,A)=(q3,A,L),
D(q3,B)=(q3,B,L),
D(q3,C)=(q3,C,L),
D(q3,_)=(q4,_,R),
D(q4,a)=(q1,A,R),
D(q4,A)=(q4,A,R),
D(q4,B)=(q4,B,R),
D(q4,C)=(q4,C,R),
D(q4,_)=(q5,_,R);
```

Figure 6: Definition of a Turing machine in Turing Machine Description Language

VISUALIZATION CENTRIC AUTOMATA SIMULATORS

Visualization centric automata simulators try to demonstrate the working of automata using high quality graphics often augmented with animation. They accept, as input, specifications of automata either in predefined structured forms or in diagrammatic forms as discussed in subsections 3.1 and 3.2, respectively. After entering the specification of an automaton, we can often simulate its workings in adjustable speeds or in a stepwise mode. Care is taken to make the simulations informative. Simple tools, as those to convert one form of automata into another, are often attachments to the simulators.

3.1 Visualization Centric Automata Simulators Accepting Structured Input

Some visualization centric automata simulators accept specifications of automata in predefined structured formats. Such a format often comprises of a table to store the transition function. The user fills in a form providing the necessary details of an automaton and promptly starts the simulation process. The visualization centric automata simulators that accept structured inputs are known for their ease of use. As a result, quite a few of them have been developed.

In a unique study, Gilbert and Cohen [19] developed a hardware model of Turing machine. The developers observed that the reasoning needed to program Turing machines to solve simple problems is similar to that needed to program real computers. Consequently, instructors used the tool to teach the fundamentals of programming at the Brandeis University. The tool consists of a tape unit containing the tape and the state registers, a programming board unit containing the coding and decoding circuitry, and an input console unit through which the user can alter the content of the tape and the state registers. In this tool, the tape of a Turing machine was implemented as a circular array with 72 locations. A three-pole 72-position electrically driven rotary switch moved the read/write head over the tape. The content of the tape, state registers, position of the read/write head and direction of movement are visually stored using NE-2 neon lamps. The tool could be used in a stepwise mode or in an automatic mode with approximately two transitions per second. They also developed a language for teaching Turing machine programming. A cross compiler, that runs on a real computer, was used to obtain optimized Turing machine code that can be executed by this tool.

Jagielski [20] developed a tool for graphical simulation of deterministic finite automata called MACH0. The tool first interactively accepts the definition of a deterministic finite automaton. Then the transition diagram of the deterministic finite automaton is displayed. Then the working of the deterministic finite automaton is simulated for any string entered by the user. The tool uses colorful animations to enhance pedagogy; it also allows the user to control the graphics to some extent. It was used as a teaching tool at the Swinburne Institute of Technology, now called the Swinburne University of Technology.

Lee [21] developed an Abstract Machine Simulator to assist students to learn about finite automata, Mealy machines and Moore machines at the Chinese University of Hong Kong. The simulator accepts the definition of an automaton in a tabular format. Two simulation modules, one text based and another graphical, are available to simulate the working of the automaton. The graphical simulator module uses an animated transition diagram to demonstrate the processing of a string by the automaton. A module to generate words recognized by the automaton is also available.

Hannay [22] developed a suite of tools to simulate finite automata, pushdown automata and Turing machines. In these tools, the specifications of the automata are entered directly in the state transition tables. Then the workings of these automata can be readily simulated either continuously or stepwise. A decade later, Hannay [23] developed a more sophisticated suite of tools to simulate finite automata, pushdown automata and Turing machines. The tools accept the specifications of the automata through interactive interfaces. The working of the automata can be simulated stepwise or instantaneously. Both the tool suites have been used for teaching at the Union College in Schenectady where the students used them to write intricate automata programs.

Vieira *et al.* [24] developed a suite of tools called Language Emulator to simulate finite automata, Mealy machines and Moore machines. The tools accept the specifications of the automata through an interactive interface and simulate their behavior. Tools are also available for converting a nondeterministic finite automaton into a deterministic finite automaton, minimizing a finite automaton, conversions between a Mealy machine and a Moore machine, and displaying the transition diagram of an automaton. The

tool suite is available in English and Portuguese. It was used as a teaching tool at the Universidade Federal de Minas Gerais where a high majority of students found it helpful.

Hamada [25] developed a Turing Machine Simulator. In this tool, the specification of a Turing machine in a notational language and the initial content of the tape are entered by the user. The tool simulates the working of the Turing machine stepwise. During the simulation process, the tool displays short informative comments about Turing machines.

Dominguez [26] developed a simple tool, called Automata en Java, to simulate deterministic finite automata. The tool accepts the specification of a deterministic finite automaton interactively and simulates it stepwise (Figure 7). The tool is available in Spanish.

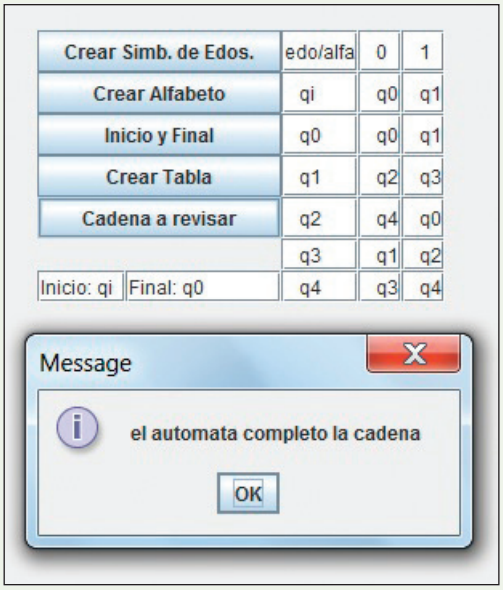

Figure 7: Simulation of a deterministic finite automaton using Automata en Java

3.2 Visualization Centric Automata Simulators Accepting Diagrammatic Input

Some visualization centric automata simulators need the user to draw the transition diagrams of the automata. Such a tool typically provides a canvas where states and transitions are added and positioned by clicking and dragging the mouse. This gives the user a feel of drawing an automaton on paper. Such automata simulators typically boast good graphics and animation. They also have various tools for processing automata integrated with them. As a result, students favor such tools and a number of them have been developed.

Barwise and Etchemendy [27, 28] developed a tool, called Turing's World, to visually design and simulate Turing machines. The tool supports the use of sub-machines to design more complicated Turing machines. Animation has been used to make the process of simulation more pedagogic. The tool runs on Macintosh systems and has been used for teaching at the Stanford University.

Rodger and co-researchers [29-31] developed a Formal Language and Automata Package to design and simulate finite automata, pushdown automata and Turing machines. The tool supports several deterministic and nondeterministic variants of these automata. Using this tool, the user can easily draw an automaton by clicking and dragging the mouse. The properties of the states and the transitions of the automaton can be set interactively. After an automaton has been drawn, its working can be simulated in either a fast simulation mode or a slower step-by-step simulation mode. Animation has been used in this tool to enhance pedagogy. The tool has been used successfully for teaching at the Duke University.

Luce and Rodger [32] developed a tool called Turing Building Blocks for designing and simulating Turing machines. The tool supports modular design of Turing machines. Using this tool, a Turing machine is defined in terms of previously defined modules, called building blocks, which can be stored in a library. The tool provides a graphical editor using which a Turing machine can be designed visually and interactively. A Turing machine thus designed can be simulated at varying speeds.

McFall and Dershem [33] developed a Turing Machine Simulator to design and simulate finite automata and Turing machines. The tool allows the user to design an automaton visually by clicking and dragging the mouse. The tool supports both deterministic and nondeterministic automata. The tool also allows the use of sub-machines, which can be saved separately, in defining an automaton. After designing an automaton, its working can be simulated at different speeds with or without the intervention of the user. While simulating the working of an automaton, the user can choose whether to simulate the working of its sub-machines visually or in the background. The tool has been used to teach an introductory course in computer science at the Hope College in Holland, Michigan, where it was found to be enhancing the understandings of the students.

Rodger and co-researchers have developed a Java Formal Languages and Automata Package [34-44]. It has actually evolved from its predecessor Formal Languages and Automata Package. In the last two decades, the tool has been under continuous enhancement and new features have been added regularly. The effort put into developing this tool is unparalleled in the field of simulation of automata. As a result, today it is the most sophisticated tool for simulating automata. It now covers a large number of topics on automata and related fields. The tool is also the best documented among the tools for simulation of automata. The tool supports several deterministic and nondeterministic variants of finite automata, pushdown automata, and Turing machines as well as Mealy machines and Moore machines. In this tool, the user has to visually design an automaton (Figures 8 and 9).

The working of this automaton can then be simulated either in a step-by-state mode, in a fast run mode, or in a multiple run mode. The tool can be used to convert a nondeterministic finite automaton into a deterministic finite automaton, minimize a finite automaton and perform other important operations on the automata. The tool uses state of the art graphics and is one of the easiest to use. The tool is undoubtedly the most widely used tool for simulation of automata developed to date. Thousands of students have used it at numerous universities in more than a hundred countries. It has received outstanding responses at various universities where it has been used [44], especially at Duke University [35-40, 43] and at Colgate University [45].

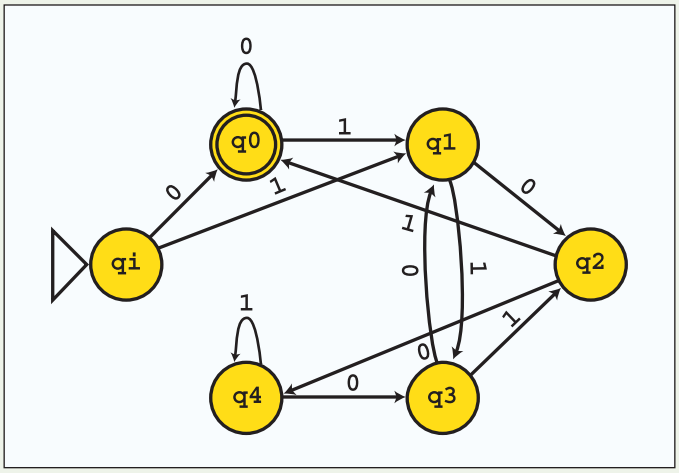

Figure 8: Designing a deterministic finite automaton using Java Formal Languages and Automata Package

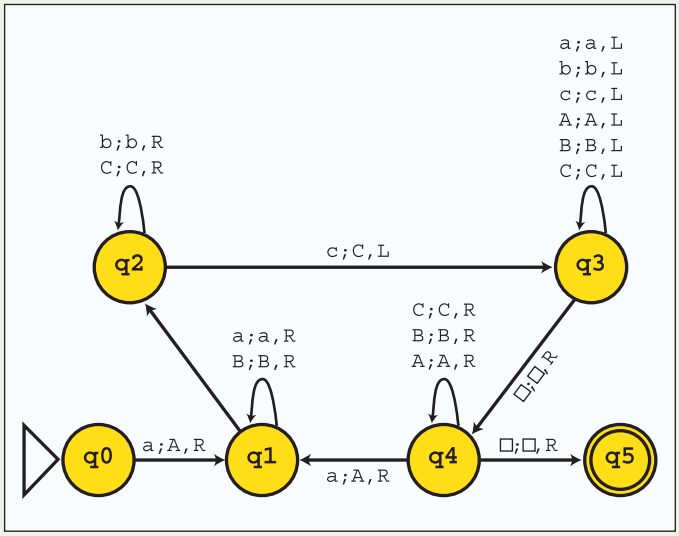

Figure 9: Designing a Turing machine using Java Formal Languages and Automata Package

Robinson [46] and Robinson *et al.* [47] developed a Java Computability Toolkit to design and simulate graphically finite automata and Turing machines. The toolkit supports both deterministic and nondeterministic finite automata. One can use it to convert a deterministic finite automaton into a nondeterministic finite automaton, minimize a finite automaton, and perform other important operations on finite automata. In this toolkit, a Turing machine is designed using simpler sub-machines. The toolkit provides five basic sub-machines, *viz.* move left, move right, move left until a specified symbol, move right until a specified symbol and write a specified symbol, to design other sub-machines and Turing machines. The toolkit uses quality graphics. It has been used for teaching at the State University of New York Institute of Technology and has received positive feedback.

Bergström [48] developed a tool, called PetC, for visually designing and simulating deterministic and nondeterministic finite automata. In this tool, one can simulate a finite automaton in either of three modes, *viz.* a normal animated mode, a fast mode, and a stepwise mode. The tool supports important operations like converting a nondeterministic finite automaton to a deterministic finite automaton and minimizing a finite automaton. In fact, the tool allows the user to choose from the three available algorithms for minimizing a finite automaton. In this tool, both English and Swedish letters can be used in the input alphabet of a finite automaton. The tool uses high quality graphics and animation.

Burch [49] developed an Automaton Simulator to visually design and simulate finite automata, pushdown automata and Turing machines. In this tool, an automaton can be simulated either stepwise or instantaneously. An option for rewinding the simulation process is also available. The tool uses high quality graphics.

McDonald [50] developed an Interactive Pushdown Automata Animation that allows the user to graphically design and simulate pushdown automata. Using this tool, a pushdown automaton can be simulated either stepwise or instantaneously. The tool also explains to the user the activities that take place in each step of the simulation process. The tool uses high quality graphics and animation to enhance pedagogy.

Grinder and co-researchers [51-54] developed a Finite State Automata Simulator to allow students to experiment with finite automata. The tool allows the user to design graphically both deterministic and nondeterministic finite automata. Thus designed, one can simulate an automaton either stepwise or instantaneously. The tool uses state of the art animation to enhance pedagogy. The tool has been used for teaching at the Montana State University at Bozeman.

Chesñevar and co-researchers [2, 55, 56] developed a Transducer Automata Graphical Simulator for visually designing and simulating Mealy machines and Moore machines. After an automaton has been visually drawn using this tool, it can be simulated either stepwise or continuously at adjustable speeds. Option for rewinding the simulation process is also available. The tool can also be used for minimizing an automaton, converting a Mealy machine into a Moore machine and vice versa. The tool provides an interesting utility of analyzing state properties by the virtue of which the mouse can be pointed to a state in the transition diagram to display important properties of the state like whether it is reachable from the starting state and which states are reachable from that state. The tool is available in English and Spanish. It has been used for teaching at the Universidad Nacional del Sur.

Hamada and Shiina [57] and Hamada [25] developed a Finite State Machine Simulator to design and simulate visually deterministic and nondeterministic finite automata. The tool supports important operations like conversion of a nondeterministic finite automaton to a deterministic finite automaton and minimization of a finite automaton. The tool has been used for teaching at the University of Aizu.

Bovet [58] developed a Visual Automata Simulator for visually designing and simulating finite automata and Turing machines (Figure 10). In this tool, a Turing machine is designed using primitive commands like those to write a specified symbol on the tape, move left, move right, move left until a specified symbol, move right until a specified symbol, move left until not a specified symbol, move right until not a specified symbol, call a sub-machine, accept and reject. The tool also provides support for batch simulation.

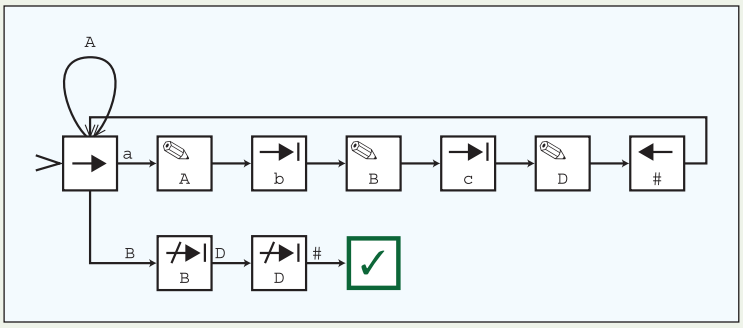

Figure 10: Designing a Turing machine using Visual Automata Simulator

White and Way [59] developed a Java Finite Automata Simulation Tool to design and simulate visually finite automata, pushdown automata, Turing machines, and other types of automata. The tool supports both deterministic and nondeterministic machines. The tool allows designing complex automata by integrating simpler sub-machines. This tool has been used for teaching at less advanced levels and encouraging results have been observed.

García-Osorio *et al.* [60] developed a tool called Thoth for teaching and learning automata theory. The tool allows visual designing and simulation of finite automata, pushdown automata and several variants of Turing machines. The tool supports both deterministic and nondeterministic automata. The tool is available in Spanish.

Čerňanský *et al.* [61] and Chudá and Rodina [62] developed an integrated simulation tool, called SimStudio, for finite automata, pushdown automata, Turing machines and other types of abstract machines. The tool supports both deterministic and nondeterministic machines. The tool attempts to merge the language based and the visualization centric approaches of simulation of automata. The tool can accept the input in two ways. It allows the user to design visually an automaton. Alternatively, the specification of an automaton can be entered in a descriptive language. The working of the automaton can be simulated stepwise or in a free run mode. The tool is available in Slovak. It has been used for teaching fundamentals of theoretical computer science at the Slovak University of Technology in Bratislava and good results have been observed.

TRENDS IN AUTOMATA SIMULATION RESEARCH

Automata simulators are being developed and used for academic purposes around the globe for the last five decades. The automata simulation research initiatives show some interesting trends, depicted as follows.

1. Although a number of tools already exist for automata simulation, the scientific community continues to welcome new ones. Each tool comes with its own principles and concepts, and often provides new utilities.
2. The automata simulators have been highly influenced by the software development tools available at the time of their development. For example, the invention of Java helped to develop automata simulators with good graphics support.
3. Developers typically publish details of the design, implementation and working of their tools in various forms of refereed literature. This allows the tools to be discovered by others in the archives even at a much later date. Other researchers may use the tools to teach and learn automata theory, or use their experience while developing new tools.
4. Some researchers make theirs tools available on the web. This facilitates widespread dissemination of the tools. However, if the web address changes and there is no refereed literature available about the tool, then it becomes difficult to trace.
5. State of the art graphics and animation techniques are often used to enhance the pedagogical nature of the automata simulators.
6. Since an automata simulator is primarily a pedagogical tool, feedback from students are accepted, reported and used for improvements.
7. Although most automata simulators are available in English, some are available in other languages. This allows the proliferation of the tools among a larger user community.

Appendix A provides an executive summary of the review.

CONCLUSION

This article tries to preserve the historicity of automata simulators. Automata theory is, and will continue to be, an important field in computer science. Automata simulators are helpful tools to teach and learn automata theory. Therefore, there is a need for continuity of research on automata simulators and their integration in teaching. Realizing the importance of automata simulators, most researchers who have worked on simulation of automata [8, 11, 13-15, 19, 31, 33-35, 37, 39, 47, 50, 51, 59, 60] have expressed their interest in continuing to work in the field and/or asked others to join them. **Ir**

Acknowledgments

The authors are thankful to a number of scientists who sent reprints of their papers and provided miscellaneous information about the tools they have developed.

References

[1] Coffin, R. W., Goheen, H. E. and Stahl, W. R. 1963. Simulation of a Turing machine on a digital computer. *Proceedings of the Fall Joint Computer Conference*, pp. 35-43.
[2] Chesñevar, C. I., Cobo, M. L. and Yurcik, W. 2003. Using theoretical computer simulators for formal languages and automata theory. *inroads – ACM SIGCSE Bulletin*, 35(2): 33-37.
[3] Head, E. F. S. 1997. *ASSIST: A Simple SImulator for State Transition*. http://www.cs.binghamton.edu/~software/ASSIST.html.
[4] Harris, J. 1998. YATS - yet another Turing machine simulator. *Journal of Computing in Small Colleges*, 13(3): 31-35.
[5] Harris, J. 1999. Programming a universal pushdown automaton. *Proceedings of the Annual Southeast Regional Conference*, article no. 27.
[6] Harris, J. 2002. Programming nondeterministically using automata simulators. *Journal of Computing in Small Colleges*, 18(2): 237-245.
[7] Shelburne, B. J. 2002. *Software Projects*. http://www4.wittenberg.edu/academics/mathcomp/bjsdir/software.shtml.
[8] Scott, T. A. 2006. Turing machine simulation used in a breadth first computer science course. *Journal of Computing in Small Colleges*, 22(1): 240-245.
[9] Erlacher, F. 2009. *Pushdown Automata Simulator*. B.Sc. dissertation, Institut für Informatik, Universität Innsbruck.

[10] Curtis, M. W. 1965. A Turing machine simulator. *Journal of the Association of Computing Machinery*, 12(1): 1-13.
[11] Rose, L. L., Jones, N. D. and Barnes, B. H. 1971. Automata: a teaching aid for mathematical machines. *ACM SIGCSE Bulletin*, 3(1): 12-20.
[12] Pierce, J. C., Singletary, W. E. and Vander Mey, J. E. 1973. Tutor – a Turing machine simulator. *Information Sciences*, 5: 265-278.
[13] Knuth, D. E. and Bigelow, R. H. 1967. Programming languages for automata. *Journal of the Association of Computing Machinery*, 14(4): 615-635.
[14] Chakraborty, P., Saxena, P. C. and Katti, C. P. 201x. A compiler-based toolkit to teach and learn finite automata. *Computer Applications in Engineering Education*, in press.
[15] Chakraborty, P. 2007. A language for easy and efficient modeling of Turing machines. *Progress in Natural Science*, 17(7): 867-871.
[16] Nayar, A., Jha, A., Malik, A. and Anand, N. 2010. *Optimizing Compiler for the Turing Machine Language*. B.Tech. dissertation, G. T. B. Institute of Technology, G. G. S. Indraprastha University.
[17] Chakraborty, P., Taneja, S., Anand, N., Jha, A., Malik, D. and Nayar, A. 2011. An optimizing compiler for Turing machine description language. *The IUP Journal of Computer Sciences*, in press.
[18] Chakraborty, P., Taneja, S., Saxena, P. C. and Katti, C. P. 2011. Teaching purpose compilers – an exercise and its feedback. *ACM Inroads*, 2(2): 47-51.
[19] Gilbert, I. and Cohen, J. 1972. A simple hardware model of a Turing machine: its educational use. *Proceedings of the ACM Annual Conference*, pp. 324-329.
[20] Jagielski, R. 1988. Visual simulation of finite state machines. *ACM SIGCSE Bulletin*, 20(4): 38-40.
[21] Lee, M. C. 1990. An abstract machine simulator. *Lecture Notes in Computer Science*, 438: 129-141.
[22] Hannay, D. G. 1992. Hypercard automata simulation: finite-state, pushdown and Turing machines. *ACM SIGCSE Bulletin*, 24(2): 55-58.
[23] Hannay, D. G. 2002. Interactive tools for computation theory. *inroads – ACM SIGCSE Bulletin*, 34(4): 68-70.
[24] Vieira, L. F. M., Vieira, M. A. M. and Vieira, N. J. 2004. Language emulator, a helpful toolkit in the learning process of computer theory. *inroads – ACM SIGCSE Bulletin*, 36(1): 135-139.
[25] Hamada, M. 2008. Supporting materials for active e-learning in computational models. *Lecture Notes in Computer Science*, 5102: 678-686.
[26] Dominguez, A. E. O. 2009. Automata. http://torturo.com/wp-content/uploads/Automata.jar.
[27] Barwise, J. and Etchemendy, J. 1986. *Turing's World: An Introduction to Computability*, Academic Courseware Exchange.
[28] Barwise, J. and Etchemendy, J. 1998. Computers, visualization, and the nature of reasoning. In Bynum, T. W. and Moor, J. H. (Eds.) *The Digital Phoenix: How Computers are Changing Philosophy*, Blackwell, pp. 93-116.
[29] LoSacco, M. and Rodger, S. H. 1993. FLAP: a tool for drawing and simulating automata. *Proceeding of the World Conference on Educational Multimedia and Hypermedia*, pp. 310-317.
[30] Caugherty, D. and Rodger, S. H. 1994. NPDA: a tool for visualizing and simulating nondeterministic pushdown automata. In Dean, N. and Shannon, G. E. (Eds.) *Computational Support for Discrete Mathematics*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 15, American Mathematical Society, pp. 365-377.
[31] Rodger, S. H. 1997. Integrating hands-on work into the formal languages course via tools and programming. *Lecture Notes in Computer Science*, 1260: 132-148.
[32] Luce, E. and Rodger, S. H. 1993. A visual programming environment for Turing machines. *Proceedings of the IEEE Symposium on Visual Languages*, pp. 231-236.
[33] McFall, R. and Dershem, H. L. 1994. Finite state machine simulation in an introductory lab. *ACM SIGCSE Bulletin*, 26(1): 126-130.
[34] Procopiuc, M., Procopiuc, O. and Rodger, S. H. 1996. Visualization and interaction in the computer science formal languages course with JFLAP. *Proceedings of the Frontiers in Education Conference*, pp. 121-125.
[35] Bilska, A. O., Leider, K. H., Procopiuc, M., Procopiuc, O., Rodger, S. H., Salemme, J. R. and Tsang, E. 1997. A collection of tools for making automata theory and formal languages come alive. *ACM SIGCSE Bulletin*, 29(1): 15-19.
[36] Rodger, S. H. and Gramond, E. 1998. JFLAP: an aid to studying theorems in automata theory. *inroads – ACM SIGCSE Bulletin*, 30(3): 302.
[37] Gramond, E. and Rodger, S. H. 1999. Using JFLAP to interact with theorems in automata theory. *inroads – ACM SIGCSE Bulletin*, 31(1): 336-340.
[38] Hung, T. and Rodger, S. H. 2000. Increasing visualization and interaction in the automata theory course. *inroads – ACM SIGCSE Bulletin*, 32(1): 6-10.
[39] Rodger, S. H. 2002. Using hands-on visualizations to teach computer science from beginning courses to advanced courses. *Proceeding of the Program Visualization Workshop*, pp. 104-113.
[40] Cavalcante, R., Finley, T. and Rodger, S. H. 2004. A visual and interactive automata theory course with JFLAP 4.0. *inroads – ACM SIGCSE Bulletin*, 36(1): 140-144.
[41] Rodger, S. H. 2006. Learning automata and formal languages interactively with JFLAP. *inroads – ACM SIGCSE Bulletin*, 38(3): 360.
[42] Rodger, S. H., Bressler, B., Finley, T. and Reading, S. 2006. Turning automata theory into a hands-on course. *inroads – ACM SIGCSE Bulletin*, 38(1): 379-383.
[43] Rodger, S. H., Lim, J. and Reading, S. 2007. Increasing interaction and support in the formal languages and automata theory course. *inroads – ACM SIGCSE Bulletin*, 39(3): 58-62.
[44] Rodger, S. H., Wiebe, E., Lee, K. M., Morgan, C., Omar, K. and Su, J. 2009. Increasing engagement in automata theory with JFLAP. *inroads – ACM SIGCSE Bulletin*, 41(1): 403-407.
[45] Sanchis, L. A. 2001. Computer laboratories for the theory of computing course. *Journal of Computing Sciences in Colleges*, 16(4): 262-269.
[46] Robinson, M. B. 1998. *A Java-based Tool for Models of Computation*. M.S. dissertation, State University of New York Institute of Technology.
[47] Robinson, M. B., Hamshar, J. A., Novillo, J. E. and Duchowski, A. T. 1999. A Java-based tool for reasoning about models of computation through simulating finite automata and Turing machines. *inroads – ACM SIGCSE Bulletin*, 31(1): 105-109.
[48] Bergström, H. 1998. *Applications, Minimisation, and Visualisation of Finite State Machines*. M.Sc. dissertation, Royal Institute of Technology, Stockholm University.
[49] Burch, C. 2001. Automaton Simulator. http://ozark.hendrix.edu/~burch/proj/autosim/index.html.
[50] McDonald, J. 2002. Interactive pushdown automata animation. *inroads – ACM SIGCSE Bulletin*, 34(1): 376-380.
[51] Grinder, M. T. 2002. Animating automata: a cross-platform program for teaching finite automata. *inroads – ACM SIGCSE Bulletin*, 34(1): 63-67.
[52] Grinder, M. T., Kim, S. B., Lutey, T. L., Ross, R. J. and Walsh, K. F. 2002. Loving to learn theory: active learning modules for the theory of computing. *inroads – ACM SIGCSE Bulletin*, 34(1): 371-375.
[53] Grinder, M. T. 2003. A preliminary empirical evaluation of the effectiveness of a finite state automaton animator. *inroads – ACM SIGCSE Bulletin*, 35(1): 157-161.
[54] Cogliati, J. J., Goosey, F. W., Grinder, M. T., Pascoe, B. A., Ross, R. J. and Williams, C. J. 2005. Realizing the promise of visualization in the theory of computing. *ACM Journal of Educational Resources in Computing*, 5(2): article no. 5.
[55] Esmoris, A. and Chesñevar, C. I. 2003. Una herramienta para la simulación de autómatas traductores en la enseñanza de teoría de la computación. *Proceedings of the Argentinean Congress in Computer Science*, pp. 287-295.
[56] Esmoris, A., Chesñevar, C. I. and González, M. P. 2005. TAGS: a software tool for simulating transducer automata. *International Journal of Electrical Engineering Education*, 42(4): 338-349.
[57] Hamada, M. and Shii*na, K. 2004. A classroom experiment for teaching automata*. inroads – ACM SIGCSE Bulletin, 36(3): 261.
[58] Bovet, J. 2004. *Visual Automata Simulator*. http://www.cs.usfca.edu/~jbovet/vas.html.
[59] White, T. M. and Way, T. P. 2006. jFAST: a Java finite automata simulator. *inroads – ACM SIGCSE Bulletin*, 38(1): 384-388.
[60] García-Osorio, C., Mediavilla-Sáiz, I., Jimeno-Visitación, J. and García-Pedrajas, N. 2008. Teaching pushdown automata and Turing machines. *inroads – ACM SIGCSE Bulletin*, 40(3): 316.
[61] Čerňanský, M., Nehéz, M., Chudá, D. and Polický, I. 2008. On using of Turing machine simulators in teaching of theoretical computer science. *Journal of Applied Mathematics*, 1(2): 301-312.
[62] Chudá, D. and Rodina, D. 2010. Automata simulator. *Proceedings of the International Conference on Computer Systems and Technologies*, pp. 394-399.

PINAKI CHAKRABORTY, P.C. SAXENA, AND C.P. KATTI
School of Computer and Systems Sciences
Jawaharlal Nehru University
New Delhi 110067, India
pinaki_chakraborty_163@yahoo.com
pcsaxena@mail.jnu.ac.in
cpkatti@mail.jnu.ac.in

Categories and Subject Descriptors: A.1 [General Literature]: Introductory and Survey; D.3.4 [Programming Languages]: Processors—Compilers; interpreters; F.1.1 [Computation by Abstract Devices]: Models of Computation—Automata; bounded-action devices; I.6.8 [Simulation and Modeling]: Types of Simulation—Visual; K.3.2 [Computers and Education]: Computer and Information Science Education—Computer science education
General terms: Experimentation, Languages, Theory
Keywords: Automata simulator, pedagogical tool

DOI: 10.1145/2038876.2038893

APPENDIX A – Summary of Survey

Year[1]	Name of Automata Simulator	Reference	Language in which available	University where used for teaching purpose[4]	Types of automata supported				Support for non-determinism	Support for sub-machines	Language of implementation[4]	Category according to classification by Chesñevar *et al.* [2][5]	Category according to classification proposed in this paper[6]
					Finite automata	Pushdown automata	Turing machines	Finite transducers					
1963	Simulation of Turing machine on a digital computer[3]	Coffin *et al.* [1]	English				✓					SCA	LB-NL
1965	Turing Machine Simulator	Curtis [10]	English	Wesleyan University			✓			✓	IBM 1620 assembly language	SCA	LB-AL
1967	Tool suite based on programming languages for automata[3]	Knuth and Bigelow [13]	English			✓				✓		SCA	LB-PL
1971	Automata	Rose *et al.* [11]	English	Pennsylvania State University	✓	✓					Fortran IV	MCA	LB-AL
1972	A simple hardware model of Turing Machine[3]	Gilbert and Cohen [19]	English	Brandeis University			✓					SCA	VC-SI
1973	Tutor – A Turing Machine Simulator	Pierce *et al.* [12]	English				✓			✓	IBM System/360 assembly language	SCA	LB-AL
1986	Turing's World	Barwise and Etchemendy [27, 28]	English	Stanford University			✓					SCA	VC-DI
1988	MACH0	Jagielski [20]	English	Swinburne University of Technology	✓						C	SCA	VC-SI
1990	Abstract Machine Simulator	Lee [21]	English	Chinese University of Hong Kong	✓			✓			C (graphical simulator) and Prolog (otherwise)	MCA	VC-SI
1992	Hypercard Automata Simulation	Hannay [22]	English	Union College	✓	✓	✓				HyperTalk	MCA	VC-SI
1993	Formal Language and Automata Package	LoSacco and Rodger [29] and others [30, 31]	English	Duke University	✓	✓	✓		✓		C++	MCA	VC-DI
1993	Turing Building Blocks	Luce and Rodger [32]	English				✓			✓	C++	SCA	VC-DI
1994	Turing Machine Simulator	McFall and Dershem [33]	English	Hope College	✓		✓		✓	✓		MCA	VC-DI

APPENDIX A – Summary of Survey *(continued)*

Year[1]	Name of Automata Simulator	Reference	Language in which available	University where used for teaching purpose[4]	Types of automata supported				Support for non-determinism	Support for sub-machines	Language of implementation[4]	Category according to classification by Chesñevar *et al.* [2][5]	Category according to classification proposed in this paper[6]
					Finite automata	Pushdown automata	Turing machines	Finite transducers					
1996	Java Formal Languages and Automata Package	Procopiuc *et al.* [34] and others [35-45]	English	Duke University and several others	✓	✓	✓	✓	✓	✓	Java	MCA	VC-DI
1997[2]	A Simple Simulator for State Transitions	Head [3]	English		✓	✓	✓		✓		Java	MCA	LB-NL
1998	Automata simulators[3]	Harris [4-6]	English		✓	✓	✓		✓		Visual Basic	MCA	LB-NL
1998	Java Computability Toolkit	Robinson [46] and Robinson *et al.* [47]	English	State University of New York Institute of Technology	✓		✓		✓	✓	Java	MCA	VC-DI
1998	PetC	Bergström [48]	English, plus Swedish letters in input alphabet		✓				✓		Delphi	SCA	VC-DI
2001[2]	Automaton Simulator	Burch [49]	English		✓	✓	✓				Java	MCA	VC-DI
2002[2]	Nondeterministic Pushdown Automata Simulator	Shelburne [7]	English	University of Wittenberg		✓			✓		Pascal	SCA	LB-NL
2002[2]	Turing Machine Simulator	Shelburne [7]	English	University of Wittenberg			✓				Pascal	SCA	LB-NL
2002	Interactive tools for computation theory[3]	Hannay [23]	English	Union College	✓	✓	✓				JavaScript	MCA	VC-SI
2002	Interactive Pushdown Automata Animation	McDonald [50]	English			✓					Java	SCA	VC-DI
2002	Finite State Automata Simulator	Grinder [51] and others [52-54]	English	Montana State University at Bozeman	✓				✓		Java	SCA	VC-DI
2003	Transducer Automata Graphical Simulator	Chesñevar *et al.* [2] and others [55, 56]	English and Spanish	Universidad Nacional del Sur				✓				SCA	VC-DI
2004	Language Emulator	Vieira *et al.* [24]	English and Portuguese	Universidade Federal de Minas Gerais	✓			✓	✓		Java	MCA	VC-SI

APPENDIX A – Summary of Survey *(continued)*

Year[1]	Name of Automata Simulator	Reference	Language in which available	University where used for teaching purpose[4]	Types of automata supported				Support for non-determinism	Support for sub-machines	Language of implementation[4]	Category according to classification by Chesñevar *et al.* [2][5]	Category according to classification proposed in this paper[6]
					Finite automata	Pushdown automata	Turing machines	Finite transducers					
2004	Finite State Machine Simulator	Hamada and Shiina [57] and Hamada [25]	English	University of Aizu	✓				✓		Java	SCA	VC-DI
2004[2]	Visual Automata Simulator	Bovet [58]	English		✓		✓		✓	✓	Java	MCA	VC-DI
2006	A Turing machine simulation[3]	Scott [8]	English	University of Northern Colorado			✓				Python	SCA	LB-NL
2006	Java Finite Automata Simulation Tool	White and Way [59]	English		✓	✓	✓		✓	✓	Java	MCA	VC-DI
2007	Tool suite based on Turing Machine Description Language[3]	Chakraborty [15] and others [16-18]	English	Jawaharlal Nehru University and G. G. S. Indraprastha University			✓				C++	SCA	LB-DL
2008	Turing Machine Simulator	Hamada [25]	English				✓				Java	SCA	VC-SI
2008	Thoth	García-Osorio *et al.* [60]	Spanish		✓	✓	✓		✓			MCA	VC-DI
2008	SimStudio	Čerňanský *et al.* [61] and Chudá and Rodina [62]	Slovak	Slovak University of Technology in Bratislava	✓	✓	✓		✓		C#	MCA	VC-DI and LB-DL
2009	Pushdown Automata Simulator	Erlacher [9]	English			✓			✓		Java	SCA	LB-NL
2009[2]	Automata en Java	Dominguez [26]	Spanish		✓						Java	SCA	VC-SI
2011	Tool suite based on Finite Automaton Description Language[3]	Chakraborty *et al.* [14, 18]	English	Jawaharlal Nehru University and G. G. S. Indraprastha University	✓				✓		C++	SCA	LB-DL

[1] Year of first publication

[2] Estimated from website

[3] Name given in this paper since no name is mentioned in original literature

[4] Information not available for some tools

[5] SCA: Tool for a specific class of automata, MCA: Tool for multiple classes of automata

[6] LB-NL: Notational language based automata simulator, LB-AL: Assembly-like language based automata simulator, LB-PL: Procedural language based automata simulator, LB-DL: Descriptive language based automata simulator, VC-SI: Visualization centric automata simulator accepting structured input, VC-DI: Visualization centric automata simulator accepting diagrammatic input