Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 1466 – 1474

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

Turing Machine and Automata Simulators

Mohamed Hamada

Sofware Engineering Lab,, The University of Aizu, Aizuwakamatsu, Fukushima, 965-8580, Japan

Abstract

This paper introduces a Turing machine and pushdown automata simulators as a virtual environment for learning computational models and automata theory. The twofold contribution of this work is a novel use of modern technology to improve learning and a longitudinal experimental evaluation of its use in context. A preliminary evaluation shows the effectiveness of the simulators in classroom.

Keywords: Finite automata, Pushdown automata, Turing machines, Simulators, Computational science.

1. Introduction

Computational science is an interdisciplinary field in which mathematical models combined with scientific computing methods are used to study systems of real-world problems. In practical use, it is typically the application of computer simulation and other forms of computation to problems in various scientific disciplines [13]. One of such mathematical models is automata theory. The concepts of automata theory, such as Turing machines and pushdown automata, have important use in designing and analysing computational models of several hardware and software applications. These concepts are abstract in nature and hence used to be taught by a traditional lecture-driven style, which is suitable for learners with reflective preferences. Since computer engineering learners tend to have strong active preferences according to learning science research [11], a lecture-driven teaching style is less motivating for them.

Our simulators are designed to tackle this issue and meet the active learning preferences for computer engineering learners. Our approach can be used as a supporting tool for active learning not only for automata theory, but also for several other courses such as theory of computation, discrete mathematics, computational models, programming languages, compiler design and other related courses. Such courses cover a variety of topics including finite state automata, pushdown automata and Turing machines.

* Corresponding author. Tel.: +81-242-37-2617; fax: +81-242-37-2735.
E-mail address: hamada@u-aizu.ac.jp.

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.314

Our simulators are written in Java language which implies that they are portable, machine independent and web-based enabled. This makes them useful tools for interactive and online automata learning.

In designing our simulators, we considered the active construction learning model [4, 12] that has a number of basic design principles which include the following:

1. Teachers act as facilitators not as knowledge transmitters. This means knowledge must be actively constructed by learners, not passively transmitted by teachers.
2. Learning should take place in a collaborative environment.

To show the effectiveness of our simulators as a model of an interactive learning tool, some experiments were carried out. The preliminary results of these experiments showed that using our simulators not only improved the learners' performance but also improved their motivation to actively participate in the learning process of the related subjects and seek more knowledge on their own.

Despite that we focus on the automata theory topics, our research can be considered as a model for a wide range of topics in the undergraduate level.

The paper is organized as follows. Following the introduction, section two introduces related work. Section three gives an overview of the automata topics such as Turing machines and pushdown automata. We will discuss the development of our simulators in section four. The performance evaluation of the environment will be presented in section five. Section six will concludes the paper and discusses future work.

2. Related work

There are a number of finite state automata simulators which have been developed (e.g. [1, 2, 3, 7, 9, 10]) to enhance the learning of automata topics. Most of them suffer from one or more flaws that make them less effective (motivating) as a learning tool, particularly for less advanced students. For example, the tools PetC in [1] lack visual clarity and dynamic capability. When designing an automaton on PetC editor and try to connect two states in both directions, labels on arrows cannot be distinguished. This becomes visually terrible when the automaton is getting bigger. JFLAP [10] is a comprehensive automata tool but it requires skilled learners who already know the basics of automata to make full use of its rich operations. The automata tools in [9] are powerful, but do not provide a convenient mechanism for displaying and visually simulating the finite state machines. The ASSIST automata tools in [7] are difficult to setup and use. Almost all have been designed as tools for advanced learners. These tools assume that the learners have already grasped the fundamental concepts. They lack a clear workflow of learning activities that can guide the new learners how and where to start using the system. This makes it difficult for new students to navigate through the system. They are also dependent on advanced mathematical and idiosyncratic user interactions. On the contrary, our simulators are designed with a clear workflow of learning activities and hence it is easy-to-use and easy-to-learn for new users.

3. Automata Topics

3.1 Finite Automata

Finite state machines or automata (FA) represent a mathematical model for several software and hardware devices. Automata have several applications in both software and hardware. In software design, it can be used in a wide range of modelling from a simple text editor to a more sophisticated compiler. In computer gaming, it can be used to model puzzles, tennis games, and many others. In hardware design, it can be used to model the function of a variety of machines, for example, vending machines, elevators, video players, rice cookers, etc.

Finite state machines are the basics of a variety of courses including: automata theory, formal languages, theory of computations, computational models, discrete mathematics, programming languages, and compiler design. In this section, we will give a brief overview of finite state machines.

Informally, a finite state machine is a machine with a finite number of states and a control unit that can change the current machine state to a new state in response to an external effect (input). It has limited memory capabilities which make it a suitable model for applications that require no information about previous actions.

Depending on the way the machine controller responds to the input, the finite state machine is classified into deterministic (DFA): if the controller can change from one state to another (one) state, nondeterministic (NFA): if it changes from one state to several states, and nondeterministic with empty move (λ-NFA): if (in addition to NFA) it can also change states in response to empty (no) input. Formally, a finite state machine A is defined as a 5-tuple $A= (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states that represent the machine states, Σ is the set of possible inputs to the machine, δ represents the finite state machine controller, $q_0 \in Q$ is the initial (starting) state of the machine, and $F \subseteq Q$ is the set of possible final (accepting) states of the machine. Depending on how the machine controller δ works, machines are classified into DFA, NFA, or λ-NFA.

- If $\delta: Q \times \Sigma \rightarrow Q$ then the machine is a DFA.
- If $\delta: Q \times \Sigma \rightarrow 2^Q$ then the machine is an NFA.
- If $\delta: Q \times (\Sigma \cup \lambda) \rightarrow 2^Q$ then the machine is a λ-NFA.

A sequence of inputs is said to be *accepted* (*recognized*) by the finite state machine if the machine controller, starting from the initial state, scans all the inputs and stops at one of the final states. The class of languages that can be accepted by the finite state machine is called *regular* languages. The three models of finite state machines DFA, NFA, and λ-NFA are equivalent. In other words, given any type of the machine, we can transform it to the other. By definition, we can see that DFA $\subseteq$ NFA $\subseteq$ λ-NFA, but we can transform λ-NFA to NFA and NFA to DFA.

3.2 Pushdown Automata

A pushdown automaton (PDA) is a finite automaton that can make use of a stack containing data. Pushdown automata differ from normal finite state machines in two ways:

1. They can use the top of the stack to decide which transition to take.
2. They can manipulate the stack as part of performing a transition.

A schematic representation of PDA is given in Figure 1.

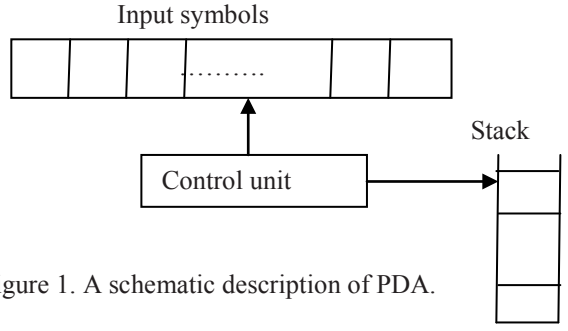

Figure 1. A schematic description of PDA.

The control unit scans the input symbols from left to right starting from the leftmost one. Each move of the control unit reads a symbol from the input, while at the same time changing the contents of the stack through the usual stack operations such as *push* and *pop*. Each move of the control unit is determined by the current input symbol as well as the symbol currently on top of the stack. The result of the move is a new state of the control unit and a change in the top of the stack.

A PDA is formally defined as a 7-tuple: $A=(Q, \Sigma, \Gamma, \delta, q_0, Z, F)$ where:

- Q is a finite set of *states*
- Σ is a finite set which is called the *input alphabet*

- Γ is a finite set which is called the *stack alphabet*
- δ: $Q \times \Sigma_\lambda \times \Gamma_\lambda \rightarrow P(Q \times \Gamma_\lambda)$ is the *transition function,*
- q_0 is the *start state*
- Z is the *initial stack symbol*
- $F \subseteq Q$ is the set of *accepting (final) states*

Where $P(S)$ denotes the power set of S, λ is the empty string, $\Sigma_\lambda = \Sigma \cup \{\lambda\}$, and $\Gamma_\lambda = \Gamma_\lambda \cup \{\lambda\}$.
The relevant factors at any time are the triplet (q, w, u), where q is current state of the control unit, w is the unread part of the input string, and u is the current contents of the stack. The triplet (q, w, u) is called *instantaneous description* (ID). A move from one ID to another will be denoted by the symbol $\vdash$; thus:

$$(q_1, aw, bx) \vdash (q_2, w, yx) \text{ is possible if and only if } (q_2, y) \in \delta(q_1, a, b).$$

Moves with arbitrary number of steps will be denoted by $\vdash^*$. There are two classes of pushdown automata: deterministic (DPDA) and nondeterministic (NPDA). Unlike finite automata, DPDA and NDPA are not equivalent. NPDA have more expressive power than DPDA.

3.3 Turing machines

Turing machines (TMs) are the most powerful finite state machines. They can simulate exactly what a digital computer can do. Informally, a TM consists of a finite set of states and a controller that can read or write symbols on an infinite length tape. Unlike DFA, NFA, λ-NFA, and PDA, a TM controller can move in both directions on the tape. The machine starts with an initial state, a finite number of input symbols written on the tape (all other infinite number of tape cells are blank), and the controller is set to the first input symbol from the left. According to the current state and the current scanned symbol on the input tape, the controller takes the next move. It can overwrite the current scanned symbol or leave it untouched, change the current state, then move to either the left or the right, and so on. If no more moves are possible, then the machine *halts*. In some cases, the machine may run forever and never halt.

But more interestingly, Turing machines can be used to compute functions, exactly the same way that modern digital computers can do. In this case, the function arguments are represented as sequences of 1's separated by 0's, and are written on the machine's tape. The function definition is represented as a set of rules suitable for the machine's transition function. Then, the machine works on that input. If it halts, the output symbols left on the tape represent the value of the function application on the arguments.

While teaching these concepts in a traditional lecture-driven style, we noticed that inexperienced (novice) learners have difficulties to fully understand these basic concepts. Moreover learners became less motivated to actively participate in the class.

To deal with this issue, our simulators cover these concepts in a visual and interactive way that is more suitable for engineering students. Using the editors of the simulators, learners can easily build, modify, and simulate a PDA or a Turing machine. Then they can interactively simulate the machine with any desired input to visually see how the machine acts (in distinct steps or as a whole manner) in response to that input.

4. Simulators

4.1 Pushdown automata simulator

The Pushdown Automaton Simulator allows learners to draw an automaton visually and then apply operations on it. During these operations they can observe any change that may happen to the automaton. For example they can check the acceptance/rejection of an input while observing the corresponding changes in the automaton states and in the stack contents. They can also zoom-in and out and do auto outlay to enable more clear view for the automaton. These last operations are particularly useful when the underline automaton is large. The PDA simulator interface is shown in Figure 2.

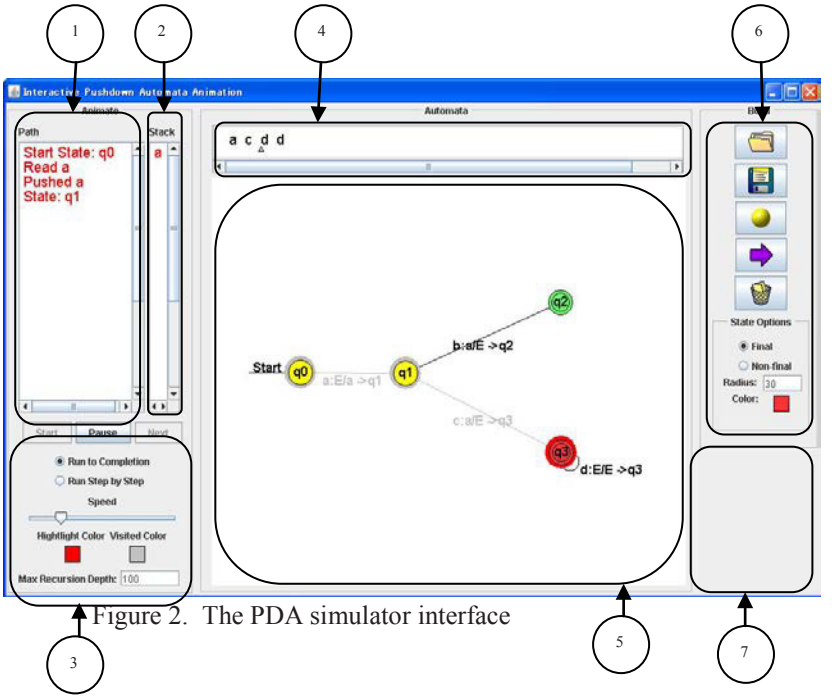

Figure 2. The PDA simulator interface

In Figure 3 the numbers from 1 to 7 are explained as follows.

1. Number 1 represents the action view area. In this area the user can watch the PDA state changes in every step during the PDA operating in a given input.
2. Number 2 represents the stack view area. In this area the user can watch the stack changes in every step while the PDA operating in a given input.
3. Number 3 represents the animation control panel. In this control panel the user can control the PDA animation speed (fast/slow/step-by-step), zoom-in and out, and layout the edited PDA.
4. Number 4 represents the input edit/display area. In this area the user can insert any valid string as an input for the PDA. This input string will be scanned and processed by the PDA from left to right.
5. Number 5 represents the PDA edit/display area. In this area the user can create/modify the PDA state by state and link by link, define the start and final states, connect the states, move the states, change the state color, etc.
6. Number 6 represents the editing control area. In this area the user can save/load PDAs, chose action to perform on the edited PDA, chose editing color, etc.
7. Number 7 represents a reserved area for possible future extensions.

After designing the automaton, we can start to simulate with input strings. The input window is displayed after we push the "start" button. Input strings in the editing window and click "OK" button. Then PDA Simulator judge whether that automaton accepts or rejects those input strings. Users can see the simulation in a step-by-step manner or as a whole.

4.2 Turing machine simulator

As stated before, traditional lecture-driven style for teaching-learning Turing machines is time consuming and difficult for average students to grasp its basic concepts. In order to facilitate the teaching-learning of basic Turing machine concepts for average engineering students, a Turing machine simulator component is integrated into the environment. Learners can easily build their own Turing machine and follow in a step-by-step manner how the Turing machine works on any given input. They can build machines that behave as language recognizers in addition to building machines that can behave as function computers. It has a friendly user interface with some

animation and sound effects that enhance the component and make learning more attractive, active and interesting to the learner.

The Turing machine simulator is integrated into the environment as well. The TM simulator is based on the work of [8]. Learners can write their machine in the input window, and then write the input of the machine on the (infinite) tape. After that, they can start to operate the machine on the input and observe how it works. For example, to add two positive integers *m* and *n*, the function *add*(*m*, *n*) = *m*+*n*, is represented by the Turing machine rules shown in Figure 4. A rule in the form *"a b c >"* means that if the current state is *a* and the current input tape symbol is *b*, then the controller changes the current state to *c* and moves one step to the right (right is represented by "*>*" and left by "*<*"). A rule in the form "*a b c d*" means that if the current state is *a* and the current input tape symbol is *b*, then the controller changes the current state to *c* and the current input tape symbol to *d*. If the learner wants to add, for example, 2 and 3, i.e. compute the function *add*(2,3)=2+3, then he/she must write 2 as 11 and 3 as 111 separated by 0 on the input tape, which means the input string will be 110111. Running the machine on this input by clicking the run button will result in the machine halting with the output string 11111 written on the tape, which means 5. Learners can see the machine running on the input symbol in a step-by-step manner which can help the learner to see how the Turing machine acts on input and how it can compute functions. All operations of the Turing machines can be simulated by this simulator. The component interface showing the Turing machine simulator is shown in Figure 3. While the Turing machine operates on its input, a number of short comments also appear on the editor to give the learners more information about the theory of Turing machines. The Turing machine simulator also includes sound effects to make learning more fun and interesting to learners.

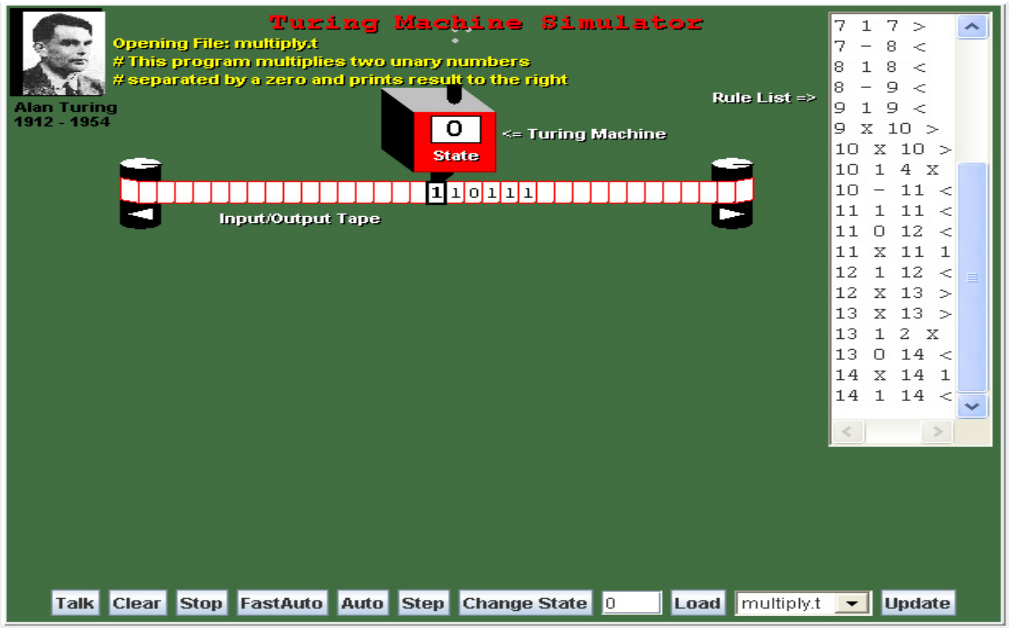

Figure 3. The Turing machine simulator interface in the IE.

```
0 1 0 >
0 0 1 1
1 1 1 >
1 - 2 <
2 1 3 -
3 - 4 <
4 1 4 <
4 - 5 >
```

Figure 4. A Turing machine example to add two positive integers.

4.3 Finite automata simulator

The pushdown automata and the Turing machine simulators discussed in the previous sections are a complementary work for our previous finite automata simulator [6] which we briefly discuss it here.

Our finite automata simulator is a bilingual English/Japanese and easy to use which make it a suitable tool for novice automata learners. Figure 5 shows the user interface of our simulator.

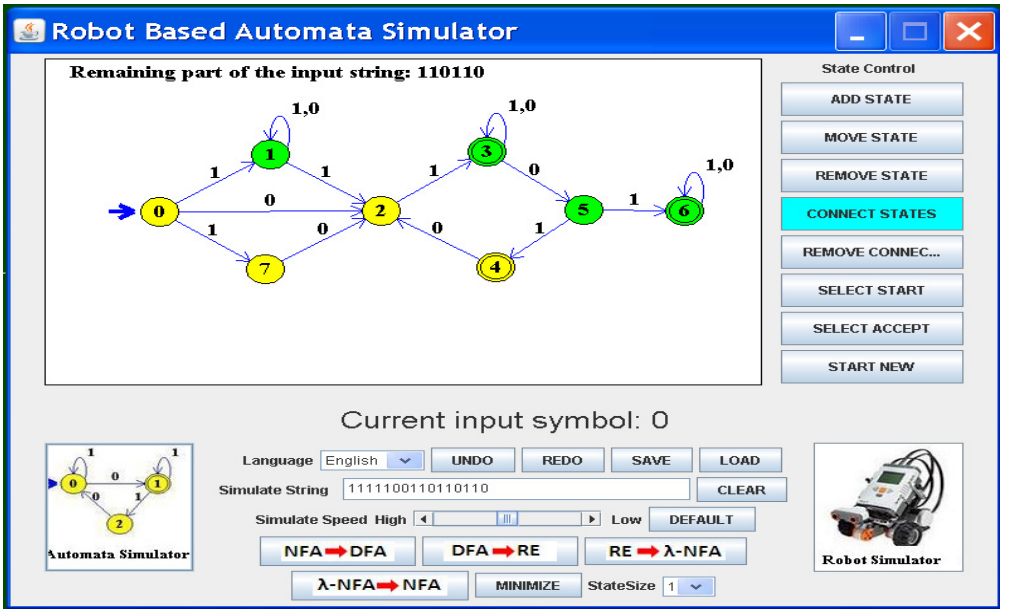

Fig. 5 The main simulator's interface.

This automata simulator integrates robot based game for enhancing automata learning.

4.3.1 Using the Automata Simulator

Once the automaton editing process is completed, the learner can then study many automata algorithms through the given operations: NFA to DFA conversion, DFA to RE conversion, RE to λ-NFA conversion, λ-NFA to NFA conversion, and automata minimization algorithms. Learners also can simulate any input string with the given automaton in an animated style and speed control. The animated style simulation is useful for enhancing visual clarity and the speed control simulation is useful for giving the students chance to reflect on the underlying automaton properties during the simulation process. The "Automata Simulator" button (located at the down-left corner of the main simulator interface) allows the user to input and visually simulate any input string. Figure 5 shows an edited automaton and the simulation of the input string "1111100110110110". The green coloured states represent the current state(s) during the simulation process. The remaining part of the input string and the current input symbol are also displayed to enhance the visual clarity of the simulation process.

4.3.2 Using the Robot Simulator

The "Robot Simulator" button allows the user to define the robot motion and connect it with automaton states. It allows the user to start the "guess automaton" game.

In the case of the robot simulator, the underlying (inputted) automaton is limited to four states that represent the four directions of the robot motion: forward, backward, left and right. In future versions will extend the robot motion to allow motion in different angles and hence allow automata with more than four states.

5. Evaluation

To test the effectiveness of our automata simulators as an active learning tool for automata theory, it was integrated into a one semester automata course. In an evaluation sample of 60 automata students, the students were divided into four groups, each group containing 15 students. A set of 40 selected exercises was distributed among the groups, 10 for each group. Each group members could collaborate inside their group but not with any other group members. No group could see the exercises of other group. Two groups were asked to answer their assigned exercises using the simulators and the other two groups without using it. An equal time period was provided to all the groups. The result showed a better performance for the two groups using the simulators. Then, the experiment was repeated by redistributing the exercises among the four groups. Again, the two groups using the simulators showed a better performance.

At the end of the course the effectiveness of our automata simulators was evaluated by students. The evaluation was carried out according to the following five items: design, user friendly, functionality, automata simulators' usability, and how it can help their automata learning process. Students were asked to evaluate the simulators based on the above items with a five scale evaluation measure ranges from 1 (worst) to 5 (best). Table 1 shows the average of evaluation points for each evaluation item.

According to the evaluation data our simulator was highly evaluated in many evaluation items. In their feedback students also mentioned that they wish to see an interacting dialog during the simulation process. We will consider this observation in future versions of the simulators.

Table.1 Evaluation result of our robot simulator tools

Evaluation Items	*Average Evaluation Point*
Design	4.4
User Friendly	4.1
Functionality	3.9
Automata simulator usability	3.8
How it can help your automata learning process	4.0

6. Conclusion

In this work we introduced pushdown automata and Turing machine simulators. The purpose was to introduce active learning simulators for novice learners of automata theory and related topics. As an evaluation step, the simulators were tested by students in the automata classroom. A preliminary study shows that the simulators can improve the learning process of computer engineering students who study automata theory and related courses. Students expressed their interest and they judged that the simulators are convenient learning tools for automata theory.

Based on the classroom experience and students' feedback, we believe that the simulators can be improved in several ways which we will consider in the future work. For example it will be more convenient for students to run a dialog during the simulation process. We also plan to integrate these simulators into our comprehensive automata tools given in [5].

References

1 H. Bergstrom, Applications, Minimization, and Visualization of Finite State Machines. Master Thesis. Stockholm University, 1998. Related website at: http://www.dsv.su.se/~henrikbe/petc/.

2 J. Bovet, Visual Automata Simulator, a tool for simulating automata and Turing machines. University of San Francisco. Available at: http://www.cs.usfca.edu/~jbovet/vas.html , 2004.

3 N. Christin, DFApplet, a deterministic finite automata simulator. Available at: http://www.sims.berkeley.edu/~christin/dfa/. 1998.
4 S. Hadjerrouit, Toward a constructivist approach to e-learning in software engineering. Proc. E-Learn-World Conf. E-Learning Corporate, Government, Healthcare, Higher Education, Phoenix, AZ, pp. 507-514, 2003.
5 M. Hamada, An Integrated Virtual Environment for Active and Collaborative e-Learning in Theory of Computation. IEEE Transactions on Learning Technologies, Vol. 1, No. 2, pp. 1-14, 2008.
6 M. Hamada and Sayota Sato, A Learning System for a Computational Science Related Topic. Elsevier Procedia Computer Science Vol. 9, 2012, pp. 1763-1772, 2012.
7 E. Head, ASSIST: A Simple Simulator for State Transitions. Master Thesis. State Univesity of New York at Binghamton. 1998. Related website at: http://www.cs.binghamton.edu/~software/.
8 Java Team, Buena Vista Univ., http://sunsite.utk.edu/winners_circle/education/EDUHM01H/applet.html, 2008.
9 M. Mohri, F. Pereria, and M. Riley, AT&T FSM Library. Software tools. 2003. Available at: http://www.research.att.com/sw/tools/fsm/.
10 S. Rodger, Visual and Interactive tools. Website of Automata Theory tools at Duke University, http://www.cs.duke.edu/~rodger/tools/. 2006.
11 Transforming undergraduate education in science, mathematics, engineering, and technology. In "Committee on Undergraduate Science Education", Center for Science, Mathematics, and Engineering Education. National Research Council ed. Washington, DC: National Academy Press, 1999.
12 G. Wilson, Ed., Constructivist Learning Environments: Case Studies in Instructional Design. Englewood Cliffs, NJ: Educational Technology, 1998.
13 Wikipedia: http://en.wikipedia.org/wiki/Computational_science. Accessed on Feb. 2012.