Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 9 (2012) 1763 – 1772

Procedia Computer Science

International Conference on Computational Science, ICCS 2012

A Learning System for a Computational Science Related Topic

Mohamed Hamada*, Sayota Sato†

Graduate School of Computer Science, The University of Aizu, Aizuwakamatsu, Fukushima, 965-8580, Japan

Abstract

Computational science is an interdisciplinary field in which mathematical models combined with scientific computing methods are used to study systems of real-world problems. One of such mathematical models is automata theory. This paper introduces a learning system for automata theory. The learning system is a combination of java and robots technologies. Learners can build their own automaton graphically in the systems' interface, and then pass it to the robot, which can then simulate the automaton transitions. Learners can learn by observing the robot's motion. A preliminary evaluation shows the effectiveness of the system in classroom.

Keywords: Computational Science, Finite State Automata, Simulator, Lego NXT Mindstorms.

1. Introduction

Computational science (or scientific computing) is the field of study concerned with constructing mathematical models and quantitative analysis techniques and using computers to analyse and solve scientific problems. In practical use, it is typically the application of computer simulation and other forms of computation to problems in various scientific disciplines [15]. One of such mathematical models is automata theory. The concepts of automata theory 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 [13], a lecture-driven teaching style is less motivating for them.

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

doi:10.1016/j.procs.2012.04.194

Our robot-based game and simulator 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.

Our simulator and robot-based game 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 simulator and robot-based game, we considered the active construction learning model [4, 14] 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 integrated environment tools as a model of interactive online collaborative learning tools, some experiments were carried out. The preliminary results of these experiments showed that using our tools 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. This work enhances in several ways our previous work [6].

The paper is organized as follows. Following the introduction, section two introduces related work. Section three gives an overview of the automata topics. The Lego Mindstorm robot that we use will be described in section four. We also will discuss the development of our simulator in section five. Our automata robot-based game will be introduced in section six. The performance evaluation of the environment will be presented in section seven. Section 8 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, 11, 12]) 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 [12] 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 [11] 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 tools are designed with a clear workflow of learning activities and hence it is easy-to-use and easy-to-learn for new users. In addition our tools are the first (up to our knowledge) to integrate robots into automata learning process, which attracts many of the less motivated students.

3. Finite State Automata

Finite state machines or automata 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 (see Figure 1).

Just as finite automata are used to recognize patterns of strings, regular expressions are used to generate patterns of strings. A regular expression is an algebraic formula whose value is a pattern consisting of a set of strings, called the language of the expression. Operands in a regular expression can be any of the following:

- Characters from the alphabet over which the regular expression is defined.
- Variables whose values are any pattern defined by a regular expression.
- λ which denotes the empty string containing no characters.
- Null which denotes the empty set of strings.

Regular expressions and finite state machines have equivalent expressive power:

- For every regular expression R, there is a corresponding finite state machine that accepts the set of strings generated by R.
- For every finite state machine A, there is a corresponding regular expression that generates the set of inputs accepted by A.

Figure 1 show the cycle of transformation between finite state machines and regular expressions.

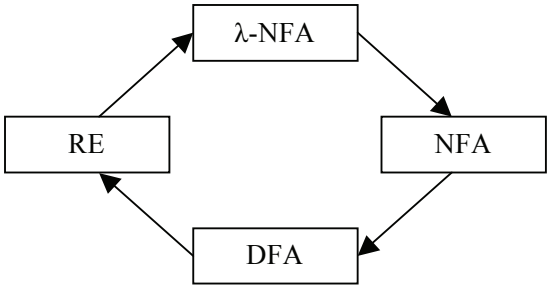


Fig. 1 Transformation between finite state machines and regular expressions

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 simulator covers these concepts in a visual and interactive way that is more suitable for engineering students. Using the editor of the simulator, learners can easily build, modify, and simulate a finite state 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.

After building the machine and simulating it with the desired inputs, learners can now start the robot automata game. They first can connect the robot with the computer via the USB cable. Then transform the automaton to the robot memory. At that stage the robot is ready to act according to the transition function of the underlying automaton that was previously build in the simulator's editor.

4. Robot

We used Lego Mindstorm NXT set to build the robot that we used in this research. Lego Mindstorm NXT has the power to work in many environments. As quoted in [8]: " With Lego Mindstorms you can build and program robots that do what you want! With the contents in the set you get everything you need to build and program your very own intelligent LEGO robot, and make it perform loads of different operations". See Fig 2 for different robot designs build up from Lego Mindstorm NXT.

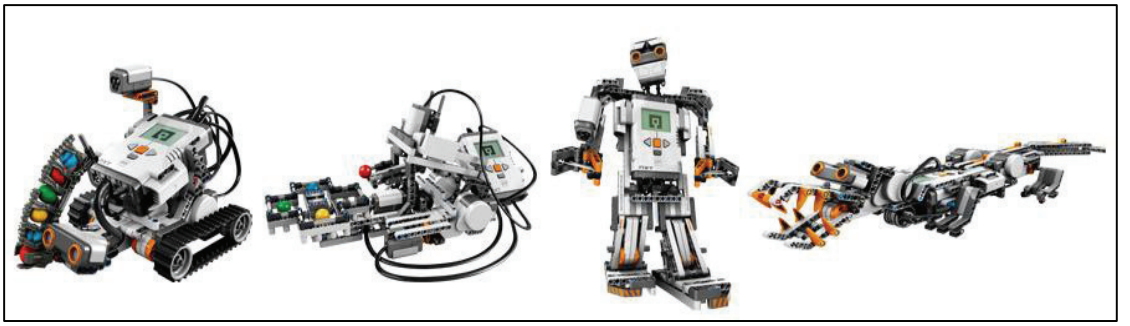


Fig. 2 Different robot designs build up from Lego Mindstorm's NXT

We designed the robot shown in Fig. 3. This robot will operate with the simulator (explained in next section) that is built up with Java language. Hence the robot must be able to communicate with the simulator in a Java environment. For this purpose we use the LeJOS NXJ [9] which is a Java programming environment for the Lego Mindstorms NXT. It allows us to program LEGO robots in Java. To run LeJOS NXJ on Microsoft Windows we need a suitable USB driver on the PC [10]. This driver is already installed with the standard Lego Mindstorm software. We checked the movement of this simulator in Windows XP and Windows Vista.

Fig. 3 Our designed robot that was build up from Lego Mindstorm's NXT

5. Simulator

Our simulator is a bilingual English/Japanese and easy to use which make it a suitable tool for novice automata learners. Figure 4 shows the user interface of our simulator.

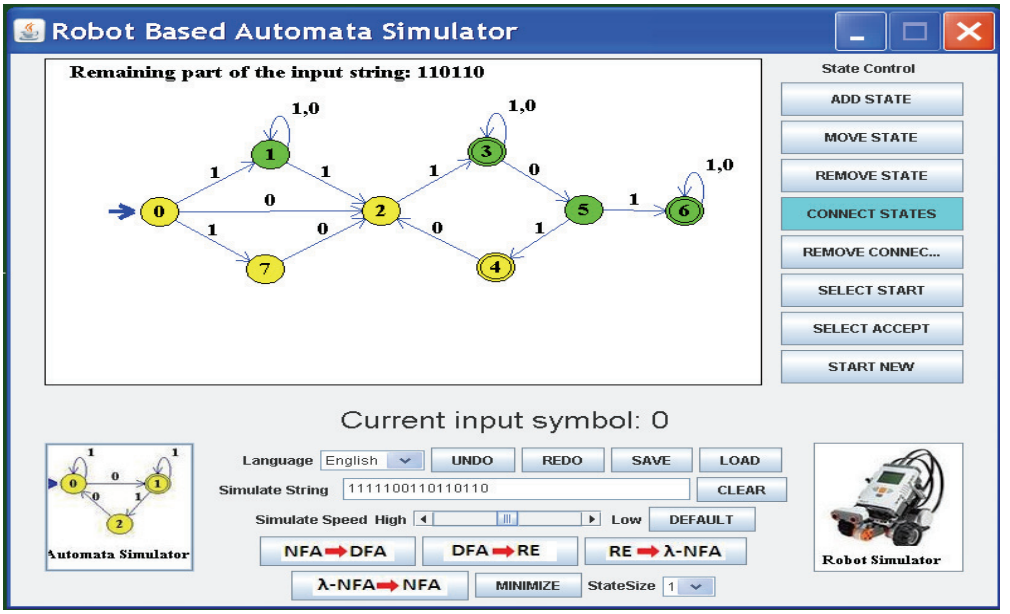


Fig. 4 The main simulator's interface.

5.1. Initialization

Novice automata learners can easily use the simulator. First the learner can select the desired language from the pop-down menu. Currently the system supports English and Japanese languages, but more languages can be added in future versions. Next the learner selects the "Add State" button to insert any number of states in the simulator editor. Then he/she can select "Connect" button to connect the states with the desired input symbols. The "Select Start" and "Select Accept" buttons set the starting and accepting states of the automaton. The "Move State" and "Remove State" buttons can move and remove states from the automaton. The "Start New" button removes the current automaton and starts a new one. "Undo", "Redo", "Save", and "Load" buttons can facilitate the editing process, saving and loading the automaton.

If there is a miss in the input automaton, an error window appears when we select the automata simulator button or the robot simulator button. After designing the automaton, users can start using the simulator or playing the robot based automata-guess game as we will explain in Sections 5.3 and 6 respectively. Figure 4 shows the simulator's interface with an inputted automaton example.

5.2. 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 4 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.

5.3. Using the Robot Simulator

The "Robot Simulator" button allows the user to define the robot motion and connect it with automaton states, see Figures 4 and 5. 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. See Fig. 5 for an example of a four-state automaton.

After initializing the robot simulator with the inputted automaton it is ready to use. The following steps show a typical session of using the robot simulator.

1. Initialize the simulator with a four-state automaton (Figure 5 shows an example of such automaton).
2. Click on the "Robot Simulator" button (located at the down-right corner of the main simulator interface), the Robot Simulator interface window will appear (see Figure 6).
3. Input the string that the user wants to test with the robot motion in the "Simulate String" text area.
4. Select the robot motion direction associated with each automaton state: forward, backward, left, or right.
5. Click on the "Make file for robot and send" button. This step will automatically create java source file and its representing classes. Through the appearing window you will be asked to name the file and select a directory to save it. Then the named java source file will be saved in the selected directory and its associated classes will be uploaded into the robot's memory. These classes represent the underlying automaton and the inputted string.
6. Execute the file in the robot (the execution method will be described in section 5.4)

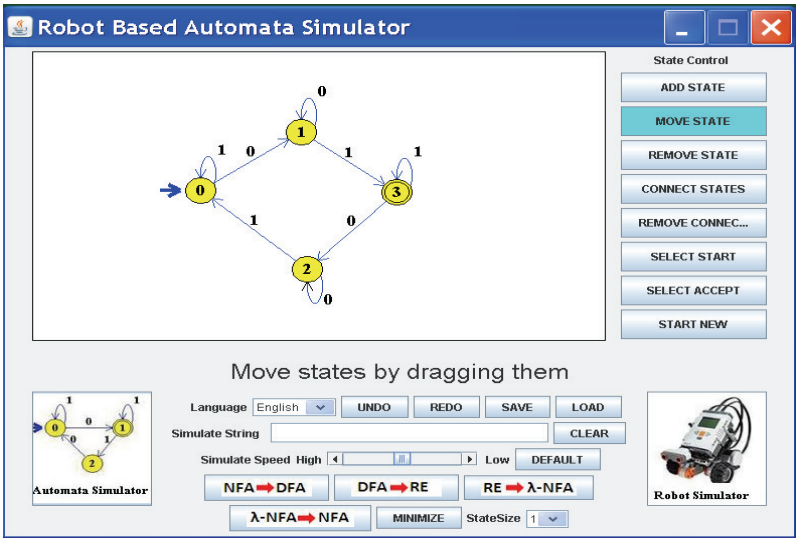


Fig. 5 A four-state automaton example

By now the automaton and the input string are uploaded to the robot's memory and ready to go to the next step for execution and running.

5.4. Executing the automaton files into the Robot set

After uploading the automaton program into the Lego NXT robot as explained in the previous subsection, we can start to execute the program and check the automaton behavior (with respect to the given input) by observing the robot motion and action. This process can be summarized in the following steps.

1. Select “Files” option from the Lego robot’s display main menu. See Figure 7(a).
2. Select the file name that was uploaded to the robot’s memory during the initialization process (see Figure 7(b) for a test file example).
3. At the beginning, the Lego robot’s display shows the inputted string and the automaton information. See Figure 7 (c) and (d) respectively.
4. Pressing any button in the Lego robot will result in the options: “Execute”, “Set as Default”, and “Delete file” appearing in Lego robot’s display. See Figure 7(e).
5. Pressing on the “Execute” option, the robot will start the motion according to the inputted string and automaton. (Be sure to place the robot in an empty and wide area to be able to move freely.)
6. When the robot completes its motions, the string "end" will appear in the Lego robot display and a beep sound will be heard from the robot.
7. At this stage, pushing any button in the Lego robot will result in one of the strings “accepting” or “rejecting” on the robot’s display showing whether the underlying automaton accepting or rejecting the given input string. See Figure 7(f).
8. Finally, pushing any button in Lego robot will terminate the program.

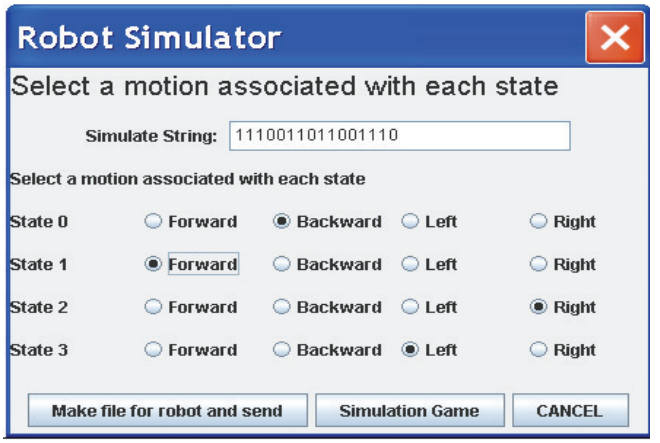


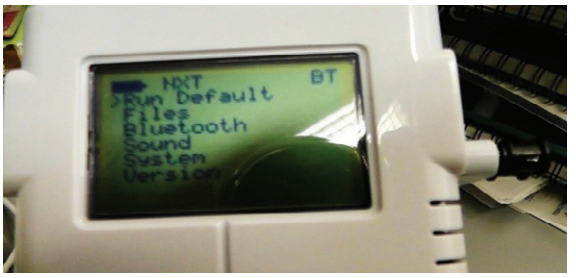
Fig. 6 The Robot Simulator Interface.

6. Robot-based Automaton Game

This is a guess automaton game. The purpose of this game is to motivate students to learn automata with fun while they are playing. The game can be played by two or more players. Some players act as question masters. Other players are answerers. A player can be either a question master or an answerer but cannot be both at the same time. A typical scenario of the game is as follows.

First, it is the question master's turn.

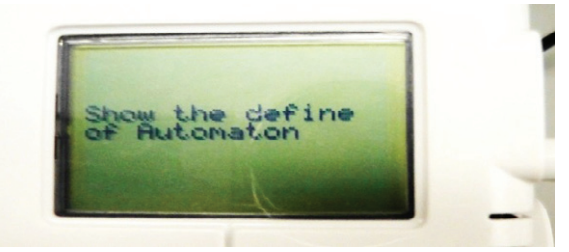
1. Using the simulator editor, the question master input an automaton of his/her choice. At this stage the automaton is hidden from the answerers.
2. From the main simulator interface click the “Robot Simulator” button (then Robot simulator interface window will appear, see Figure 6)
3. The question master then clicks on the "Game Simulation" button in the robot simulator interface window. Then the robot simulator interface window will vanish and the “Game Simulator” interface window will appear, see Figure 8.



(a) The robot display main menu.

(b) The Robot display showing the saved file name.

(c) The Robot display showing the inputted string.



(d) The display shows the automaton information.

(e) The Robot display showing the file options.

(f) The Robot display showing the final result.

Fig. 7. The robot processing steps.

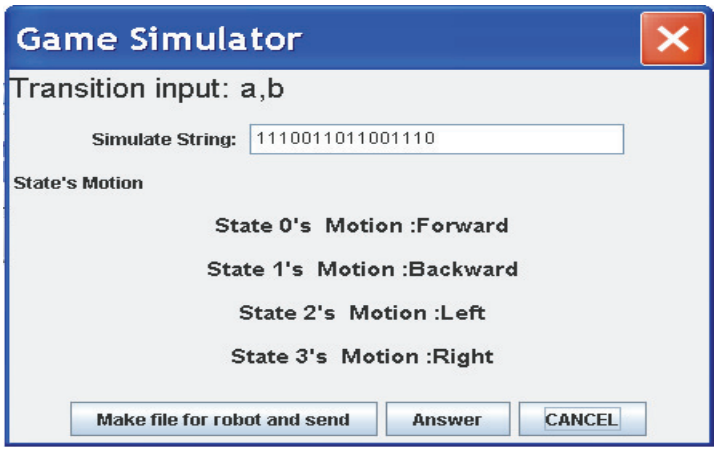


Fig. 8. Game Simulator Interface

Next, it is the answerer's turn after the question masters select the "game" button, the answerers should try to guess the inputted automaton by trying different inputs and observing the robot's motion. The procedure is as follows.

1. Input an arbitrary (guessed) string to test in the automaton.
2. Click on the "Make File for Robot and Send" button.
3. Select the save directory.
4. After saving the file, execute a program, and guess the automaton that is inputted by question master from the motion of the robot.

They can repeat the guess trail as long as they like until success to know the inputted automaton. The winner is the player who will be first to guess the automaton. Otherwise they can click on the "Answer" button to see the inputted automaton or "Cancel" button to try another automaton.

7. Evaluation

To test the effectiveness of our robot-based automata simulator as an active learning tool for automata theory, it was integrated into a one semester automata course. Figure 9 shows a group of students while using the robot-based simulator in an automata learning session. At the end of the course the effectiveness of our robot-based automata simulator was evaluated by students. The evaluation was carried out according to the following six items: design, user friendly, functionality, automata simulator usability, robot-based simulator suitability, and how it can help their automata learning process. Students was asked to evaluate the simulator based on the above items with a five scale evaluation measure ranges from 0 (worst) to 4 (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. The robot-based simulator was highly appreciated by students as a new active learning method of an abstract topic like automata theory. In their feedback students also mentioned that they wish to see the simulation process in a step-by-step manner with a dialog box. We will consider this observation in future versions of the simulator.

Table.1 Evaluation result of our robot simulator tools

Evaluation Items	Average Evaluation Point
Design	4.0
User Friendly	3.3
Functionality	3.3
Automata simulator usability	3.6
Robot-based simulator suitability	3.6
How it can help your automata learning process	3.8

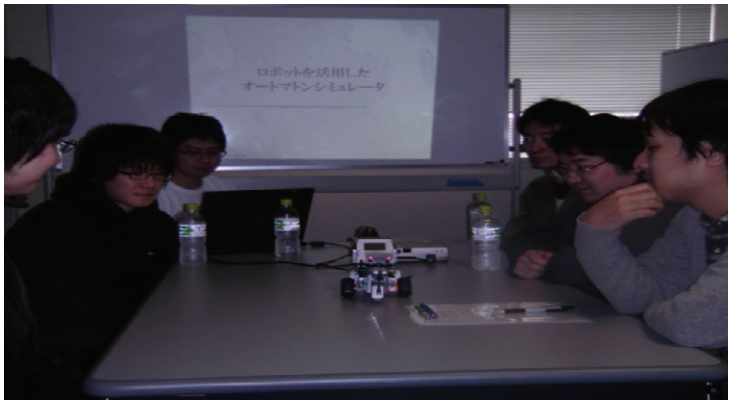


Fig. 9. A Group of students during an automata learning session with our robot-based simulator

8. Conclusion

In this work we introduced a Finite State Machine Simulator. The simulator can communicate with a robot designed for this purpose from the Lego NXT set. The simulator robot communication is achieved via Java environment and LeJOS software. The purpose was to introduce a simulator and robot-based automaton game for novice learners of automata theory and related topics. As an evaluation step, the simulator and the robot game were tested by students in the automata classroom. They expressed their interest and they judged that the simulator and the robot game are convenient as learning tool for automata theory.

Based on the classroom experience and students' feedback, we believe that the simulator 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 the simulation in a step-by-step manner.

In the current version of our simulator the robot can only move in four directions: forward, backward, left and right. This limits the underlying automaton to only for states corresponding to the four directions of robot motion. In future version we plan to allow the robot to move in different angles which will result in the ability to use larger automata (more than four states).

We also plan to integrate this simulator and the robot game 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 S. Sato, A Game-based Learning System for Theory of Computation Using Lego NXT Robot, ICCS 2011, Procedia Computer Science 4 (2011), pp. 1944-1952, 2011.
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. LEGO.com MINDSTORMS : What is in the box? [Online] Available: http://mindstorms.lego.com/en-us/history/default.aspx.
9. LEJOS Java for LEGO Mindstorms, Available: http://lejos.sourceforge.net/nxt/nxj/tutorial/Preliminaries/Intro.htm.
10. LEJOS Java for LEGO Mindstorms, Available: http://lejos.sourceforge.net/nxt/nxj/tutorial/Preliminaries/GettingStartedWindows.htm
11. M. Mohri, F. Pereria, and M. Riley, AT&T FSM Library. Software tools. 2003. Available at: http://www.research.att.com/sw/tools/fsm/.
12. S. Rodger, Visual and Interactive tools. Website of Automata Theory tools at Duke University, http://www.cs.duke.edu/~rodger/tools/. 2006.
13. 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.
14. G. Wilson, Ed., Constructivist Learning Environments: Case Studies in Instructional Design. Englewood Cliffs, NJ: Educational Technology, 1998.
15. Wikipedia: http://en.wikipedia.org/wiki/Computational_science. Accessed on Feb. 2012.