A Game-based Learning System for Theory of Computation Using Lego NXT Robot

Mohamed Hamada*, Sayota Sato

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

Abstract

Finite state automata are in the core of theory of computation course and related courses such as discrete mathematics, formal languages, etc. This paper introduces a finite state automata simulator and a robot-based game associated with it for active learning in theory of computation related courses. The simulator is implemented in Java language and the automaton game based robot is built by the Lego NXT Robot set. Learners can build their own automaton graphically in the simulator interface, and then pass it to the robot, which can then simulate the automaton transitions. Learners can learn by observing the robot’s motion.

Keywords: Theory of Computation, Finite State Automata, Simulator, Lego NXT Mindstorms.

1. Introduction

Learning science research indicates that engineering students tend to have active and sensing learning preferences, and engineering related educators are recognizing the need for more active and collaborative learning pedagogy [13].

The concepts of automata theory have important use in designing and analyzing 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, a lecture-driven teaching style is less motivating for them.

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 theory of computation, but also for several other courses such as automata theory, discrete mathematics, computational models, programming languages, compiler design and other related courses. Such courses cover a variety of topics including finite state automata.

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

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 which also 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. [1,2,3,6,11,12]) 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 [6] 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: theory of computations, computational models, discrete mathematics, automata and formal languages, 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, $\Sigma$ is the set of possible inputs to the machine, $\delta$ 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 $\delta$ 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.
- $\lambda$ 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](image-url)
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 (Figure 2) that we used in this research. Lego Mindstorm NXT has the power to work in many environments. As quoted in [7]: "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."

Fig. 2 The robot that was build up from Lego Mindstorm’s NXT

We checked the movement of this simulator in Windows XP and Windows Vista. 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 [8] 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 [9]. This driver is already installed with the standard Lego Mindstorm software.

5. Simulator

Our simulator is a bilingual English/Japanese and easy to use which make it a suitable tool for novice automata learners. Figure 3 shows the user interface of our simulator.
5.1. Initialization

Novice automata learners can easily use the simulator. First 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. The “Dialog Simulator” button allows the user to input and visually simulate any input string. In the mean time it allows the user to define the robot motion and connect it with automaton states, see Figure 4. The “Robot Simulator” button allows the user to start the “guess automaton” game.

If there is a miss in the input automaton, an error window appears when we select the simulator button or the game button. After designing the automaton, users can start using the simulator or playing the automata-guess game as we will explain in Sections 4.2 and 5 respectively. Figure 3 shows the simulator’s interface with an inputted automaton example.

5.2. Using the Simulator

After initialization the simulator with the inputted automaton it is ready to use. The user can then click on the “Dialog Simulator” button; the simulator window (Figure 4) will appear. The following steps show a typical session of using the simulator.

1. Input the string that the user wants to test in the automaton.
2. Click on the "Make File" button.
3. Select a directory to save the Java file
   (The directory must be able to execute command of LeJOS.)
After saving the Java file, there is a need to compile, upload and execute it in Lego Mindstorm NXT robot.

5.3. Compile, Upload, and Execute a file

Assume the automaton file was created by the simulator and saved in a selected directory as explained in the previous subsection under the name MyFirstAutomaton.java. Explanation about how to compile, upload and execute this automaton file with LeJOS is given here.

5.3.1. Compiling a file

We use the "nxjc" command to compile the Java file that was saved in the command prompt. LeJOS uses the standard Sun Java compiler for compiling programs. However, it needs to replace the standard Java library with LeJOS's own version of this - classes.jar. For this reason LeJOS provides a command called nxjc that sets the boot class path to classes.jar. Its parameter are the same as those as javac.

The "nxjc" command can compile one or more java files [10].

Usage:

> nxjc <java-files>

Example:

> nxjc MyFirstAutomaton.java

5.3.2. Linking, Uploading and Executing a file

LeJOS NXJ programs are different from normal Java programs in that they do not support dynamic class loading. Instead all the classes used in the program are collected together and packaged in a binary file with a .nxj extension.
This process is referred to as linking. The linked binary is then uploaded to the NXT. We use the "nxj" command [10] to upload the LeJOS NXJ file in the command prompt. There is a need to connect Lego Mindstorm to a PC when use it.

The “nxj” command is used to link, upload (to the robot) and run the java program that created by the "nxjc" command.

Usage:

> nxj [options] main-class

Example:

> nxj -r MyFirstAutomaton

5.3.3. Executing a program in the robot

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. At the beginning, the Lego robot’s display shows the inputted string as explained in subsection 4.2 (see Figure 4).
2. Pushing any button in the Lego robot will result in the string “start” shown in Lego robot’s display.
3. Pushing any button in Lego robot again the robot will start the motion according to the inputted string and automaton. (Please be sure to put the robot in an empty and wide place to be able to move freely.)
4. When the robot completes its motions, the string "end" will appear in the Lego robot display.
5. 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.

Finally, push any button in Lego robot will terminate the program.

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 players
2. The question master then clicks on the "game" button in the input window. (Then the input window vanishes and the game window (Figure 5) appears.)
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" button.
3. Select the save directory.
4. After saving the file, compile and upload the file to Lego robot.

They can repeat the guess trail as long as they like until success to know the inputted automaton. Otherwise they can click on the “answer” button to see the inputted automaton.

7. 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 initial evaluation, the simulator and the robot game were tested by several students. They expressed their interest and they judged that the simulator and the robot game are convenient as learning tool for automata theory. However we believe that this 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 input their automaton in a graphical editor and it is also better not to use a command prompt to compile the robot file. We also plan to integrate this simulator and the robot game into our comprehensive automata tools given in [5].

References