Experimental Design of Formal Language and Automata Course

Pan Qinghe
School of Computer and Information Engineering, Harbin University of Commerce, Harbin 150028, China
[a]panqinghe200@sina.com

Keywords: Formal languages, Automata, F#, Graphviz

Abstract: Formal languages and automata is a major theoretical course. This course studies the theoretical problems of automata and formal languages with a high degree of abstraction. In order to enable students to better understand and grasp the contents of the course, it needs to design the experimental courses. In this paper, a design scheme based on a functional programming language F# and a drawing tool named Graphviz is proposed. By using F#, the theoretical content is converted to the actual running code, then the code is run and the results are displayed with Graphviz. The practice proves that the scheme is feasible and can enable students to deepen their understanding of what they have learned, improve the ability to solve problems.

1. Introduction

Formal languages and automata is an important theoretical course in computer science and technology. This course mainly introduces all kinds of automata models, formal language theories, and the relationship between them. It focuses on theoretical analysis and research and is the basis of subsequent computation theory. Although the course focuses on formal definitions and proofs of theorems, as a computer professional course the practice part should not be ignored. The experiment can help students to deepen the understanding of classroom learning, grasp the abstract concepts of formalized language, and improve the ability to use related theories to solve various problems on program designs.

2. A Brief Introduction to the Course Content

According to the requirements of the course and the limit of the teaching hours in our school, the course mainly includes the following contents:

Definitions of deterministic finite state automaton (DFA), nondeterministic finite state automaton (NFA), and non-deterministic finite state automaton with epsilon transitions (ε-NFA), and their mutual transformations. The course introduces ways to convert between automata and regular expression (RE).

Two forms of pushdown automata (PDA) are given: acceptance by final states and acceptance by empty stack. The methods of mutual transformation between them are given. The instantaneous description (ID) of the pushdown automaton is defined. Pushdown automata can be used to recognize context free grammars (CFG). The course studies the transformations between PDA and CFG.

The Turing machine (TM) is studied, the definition of the deterministic Turing machine is given, and the instantaneous description (ID) is also defined.

3. Construction of Experimental Environment

In the process of convert theoretical content into code implementation, there are three important factors to consider: the choice of programming language, visualization tool and development platform.

3.1 Programming Languages.

Many kinds of programming languages, such as C++, Java, C, Python, Ruby and JavaScript, can all be used as experimental languages. But in view of the content of the experiment, it is a good choice to use the functional programming languages, because the description and definition of many theoretical problems can be expressed clearly and succinctly by the expression pattern provided by the functional programming languages. At present, functional programming languages [1] in various fields are becoming more and more important. The ideas and patterns of solving problems in the traditional Lisp/Scheme functional programming languages are being absorbed by other languages, such as the introduction of Lambda expressions in both C++ 11[2] and Java8[3].

The F# function programming language[4] is used in our experiment. F# is a functional programming language developed by Microsoft. It supports functional, imperative and object-oriented programming styles. It can execute code through command line interaction or compile source code into executable file. Although it requires students to master an additional programming language in the experiment, it turns out that this is not a difficult problem. In order to improve the efficiency and effectiveness of learning F#, we have write a lecture [5] to introduce the basic knowledge of F# language required in the experiment.

3.2 Visualization Tools.

Visualization of computation results is also an important part of experiment. Visualization can reflect the results directly and vividly. The practice shows that if the theory of the book is converted into code and the results of the calculation are displayed, it can not only verify the correctness of the theory, but also strengthen students' accomplishment and establish the confidence to further solve the other problems. For drawing, if the corresponding data input is provided, the tool will arrange the layout rationally and output the graphic result according to its layout algorithms. The drawing tool Graphviz[6] can meet these requirements, so it is chosen as a visualization tool in the experiment. The tool is cross platform, free, and has good drawing quality. The method of using it will be given later.

3.3 The Choice of Operating Systems.

The Windows 7 and Ubuntu operating systems are currently deployed in the laboratory. For Windows system to install Visual Studio (2010 and later version), you can use F# to program. Because F# is a cross-platform programming language, it can also be used under Linux. The laboratory has installed the F# environment in the Ubuntu environment. For students they may use various systems when they go back to complete their work. In order to unify the experimental environment, a virtual machine can be installed to deploy the same software environment as the laboratory.

4. Design of experimental content

Experimental design. The experimental content included in this course is shown in Table 1. It has 9 experiments.

Table 1. The experimental content in this course

Automata	Formal languages	Experiments
DFA, NFA, ε-NFA	RE，RL	(1) NFA↔DFA (2) ε-NFA↔DFA (3) DFA→RE (4) RE→ε-NFA (5) Minimize DFA
PWD (acceptance by empty stack or final states)	CFG，CFL	(1) PDA ID (2) PDA→CFG (3) CFG→PDA
TM (Turing Machine)	Phrase Structure Grammar	(1) TM ID

Although the purpose and content of these experiments are not the same, the language used in each experiment is F# and the drawing tool is Graphviz. The following experiments show how to use them to solve various problems. The examples don't focus on the theory and principles but they mainly reflect the simplicity and powerful expressiveness of F#, and the easy use of Graphviz.

5. Examples

There are three examples. All of them are from the textbook [7] used in class.

Example 1 gives the F# code to describe the Turing machine. Through this example, we can see that F# can clearly describe the automaton using its data structure.

Example 2 uses the F# recursive code to describe the inductive definition of the $\hat{\delta}$ function in DFA. We can see that using F# can easily and clearly describe the definition of inductive form using pattern matching and recursion.

Example 3 shows an example of drawing an automaton graph using Graphviz. The example is based on the experiment ε-NFA↔DFA in Table 1, which converts ε-NFA to DFA or vice versa. But here we will only focus on the Graphviz drawing in ε-NFA→DFA direction. Use F# to convert a automaton structure into a text file that conforms to the Graphviz language, and then plot it with graphviz.

5.1 Example1 1. The Definition of Turing Machine.

This example is Example 1.1 of the textbook[7] used in class. The example is a Turing machine that can recognize the language $\{0^n1^n|n \geq 1\}$. The Fig. 1 shows the 7-tuple description of the Turing machine, the state transition diagram, and the F# code describing the 7-tuple structure.

In Fig. 1, the top-left part gives the Turing machine's 7-tuple description and gives the form of the delta function in the form of a table. The lower left part is the corresponding state transition diagram of the Turing machine. The right side of Fig. 1 is the F# code description corresponding to the Turing machine. The arrows clearly show the correspondence between the tuple description and the code description.

In this example, you can see the data structure and language characteristics of lists, tuples, strings, and pattern matching in F#, which can be conveniently used to describe the structure of the automaton.

5.2 Example 2. The Definition of $\hat{\delta}$.

The example is the definition 1.2 of the textbook [7] used in class. It gives the definition of $\hat{\delta}$, the extended form of the δ function of DFA. The definition of $\hat{\delta}$ is given by inductive definition which mainly includes two parts: basis and induction. The definition of $\hat{\delta}$ is as follows:

Basis: $\hat{\delta}(q, \epsilon) = q$. Where ε represents an empty string. This means that when in state q, the automaton is still in state q if no input is read.

Induction: $\hat{\delta}(q, w) = \delta\big(\hat{\delta}(q, x), a\big)$. That is, if the length of the string is not 0, $|w| \geq 1$, then w

can be divided into two parts as *w=xa*, where *a* is the last character, and *x* is the remaining string in *w* after removing *a*.

The Fig. 2 below shows the F# code implementation of this definition. The left side of the figure is an inductive definition. The right side of the figure is the F# code implementation. It also uses arrows to indicate the correspondence between the basic and inductive parts of the definition and the F# code. Because of the need to process strings, two functions, get_a and get_x, are defined.

It can be clearly seen that the use of pattern matching and recursive functions can succinctly express the definition of the induction form.

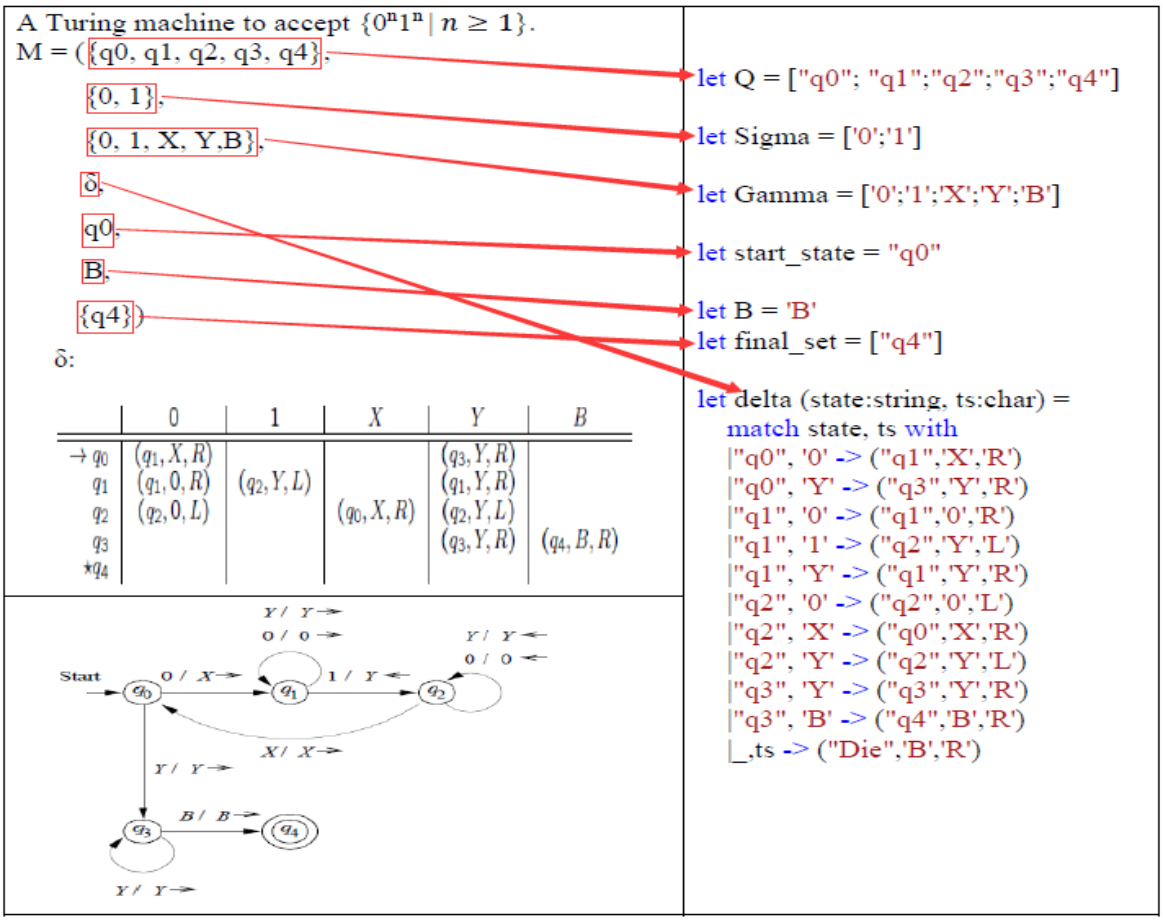

Figure 1. The Turning machine that recognizes the language $\{0^n1^n|n \geq 1\}$ and F# code

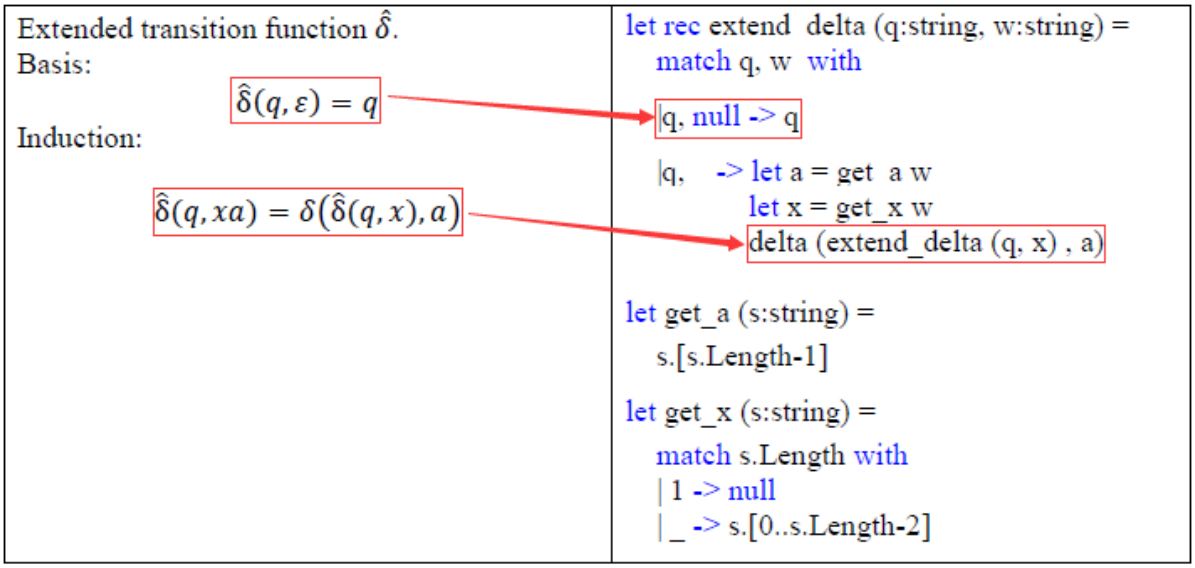

Figure 2. The definition of $\hat{\delta}$ and F# code

5.3 Example 3. The Use of Graphviz.

This example is Example 1.3 of the textbook [7] used in class. This example uses the extended subset construction method to convert ε-NFA to DFA. The state transition diagram of ε-NFA is shown in the Fig. 3.

Now we are mainly interested in the graph drawing part by Graphviz, so it is assumed that the program ε-NFA→DFA in Table 1 will transfer the ε-NFA shown in Fig. 3 to the corresponding DFA. The Fig4. shows the use of the F# language to generate the Graphviz file based on the structure of the converted DFA. The code mainly uses F#'s file reading and writing methods, uses the for-loop structure to output each component of the DFA as text and saves it as a file that can be processed by Graphviz.

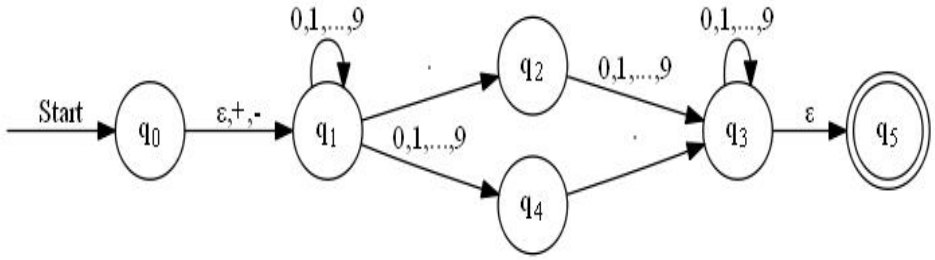

Figure 3. ϵ-NFA that accepts decimal

```
open System.IO
open System
Environment.CurrentDirectory <- @"d:\\"
let pwd = Environment.CurrentDirectory
let gen_dot () =
    File.Delete(pwd
  +"epsilonNFA_to_DFA.dot")
    use wr = new
  StreamWriter(pwd+"epsilonNFA_to_DF
  A.dot", true)
  wr.WriteLine("digraph DFA {")
  wr.WriteLine("rankdir=LR;")
  wr.WriteLine("size=\"18,18\";")
  wr.WriteLine("
  Start [shape = circle, style = invis];")
  let qD = ϵ_closure "q0"
  let mutable acc_set =[qD]
```

```
let Qd_ =accesible_set (qD, &acc_set)
let final_s = get_Fd (qD,Qn,Fn)
wr.WriteLine("Start"+"->"+"\""+
                (get_state_str qD)
+"\""+" [ label = \"Start\" ];")
for f_state in final_s do
   let fs = get_state_str f_state
   wr.WriteLine("\""+fs+"\""+
          " [shape = doublecircle];")
for s in Qd_ do
 for a in Σn do
   let left = get_state_str s
   let right = get_state_str (δd (s,a))
   let label = a.ToString()
   wr.WriteLine("\""+left+"\""+"->"+"\""+right
   +"\""+" ["+"label = "+"\""+label+"\""+" ];")
wr.WriteLine("}")
```

Figure 4. F# code that generating Graphviz file

Running gen_dot () will generate the "epsilonNFA_to_DFA.dot" file in the "D:\" directory. The file contents are as follows:

```
digraph DFA {
   rankdir=LR;
   size="18,18";
   Start [shape = circle, style = invis];
   Start->"{q0,q1}" [ label = "Start" ];
   "{q2,q3,q5}" [shape = doublecircle];
   "{q3,q5}" [shape = doublecircle];
   "{q5}" [shape = doublecircle];
```

```
    "{q0,q1}"->"{q1,q4}" [label = "0" ];
    "{q0,q1}"->"{q1,q4}" [label = "1" ];
    ……
}
```

Execute the following command at the command line which also is shown in Fig. 5.

dot -Tjpg epsilonNFA_to_DFA.dot -o epsilonNFA_to_DFA.jpg

```
D:\>dot -Tjpg epsilonNFA_to_DFA.dot -o epsilonNFA_to_DFA.jpg

D:\>
```

Figure 5. Command in CMD in Windows system

Open the resulting image epsilonNFA_to_DFA.jpg as shown in Fig. 6.

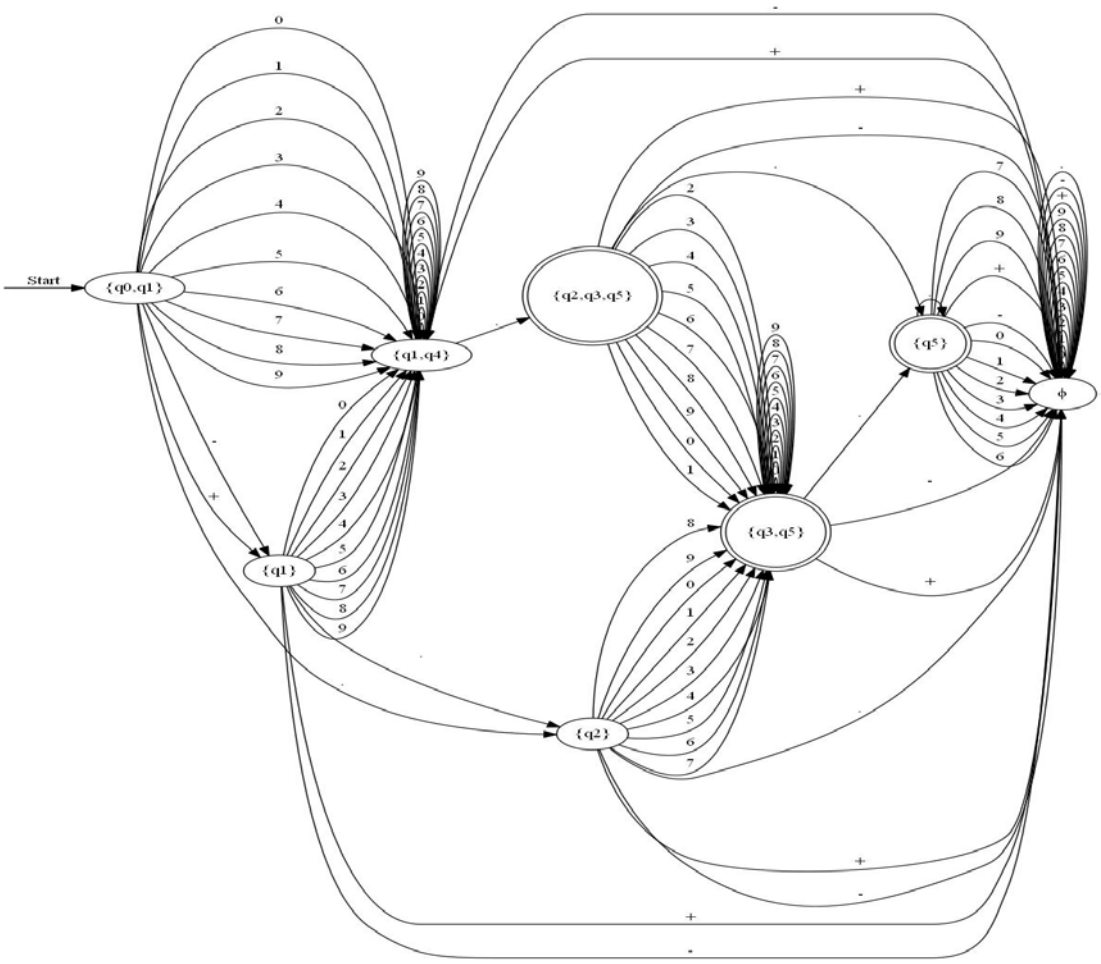

Figure 6. The graph generated by Graphviz

It can be seen that very good visual effects have been achieved.

6. Summary

Formal language and automata are relatively abstract theoretical courses. This article studies how to design the experiment content of this course, and describes how to choose programming language, development platform and visualization tool. Through three examples, it is shown that F# and Graphviz proposed in this paper can be used to design the experimental content of this course conveniently, effectively and clearly.

References

[1] Khanfor A, Yang Y. An Overview of Practical Impacts of Functional Programming//Software Engineering Conference Workshops (APSECW), 2017 24th Asia-Pacific. IEEE, 2017: 50-54.

[2] Nowakowski G. Effective use of lambda expressions in the new C++ 11 standard. Czasopismo Techniczne, 2015.

[3] Urma R G, Fusco M, Mycroft A. Java 8 in action: lambdas, streams, and functional-style programming. Manning Publications Co., 2014.

[4] Smith C. Programming F#: A comprehensive guide for writing simple code to solve complex problems. O'Reilly Media, Inc. 2009.

[5] Pan Qinghe, Sun Huadong. F# programing languages and its applications in automata theory. Publishing House of Electronics Industry, 2016.

[6] Ellson J, Gansner E, Koutsofios L, et al. Graphviz-open source graph drawing tools[C]//International Symposium on Graph Drawing. Springer, Berlin, Heidelberg, 2001: 483-484.

[7] John E. Hopcroft, Rajeev Motwani and Jeffrey D. Ullman. Introduction to Automata Theory, Languages, and Computation (3rd Edition). Pearson Education India, 2008.