THE INTERNATIONAL JOURNAL OF SCIENCE & TECHNOLEDGE

Regular Expression Simulator: Regular Expression to String Generation

Praharsh Jha
3rd Year Computer Engineering, St. Francis Institute of Technology, Mumbai, Maharashtra, India
Shravan Jaiswal
3rd Year Computer Engineering, St. Francis Institute of Technology, Mumbai, Maharashtra, India

Abstract:
Automata Validation is commonly done using a simulator on computer. However due to user giving input for the same, the chances of error is very high. We are therefore proposing an algorithm which accepts regular expression and returns possible strings generated by regular expression. These strings are then passed to simulator and accordingly verified by simulator. This reduces error by checking all possible conditions and improves simulator efficiency.

Keywords: *Theory of Computation, Automata, Regular Expression*

1. Introduction

The regular practice for verifying automata is to first design an automata, then to give string input for the same and verify the automata for verification. This is done in many small applications such as pattern matching, syntax analysis and software verification to some of major integrated applications in field of computers and information technology which has increased the need for uniform algorithm based design and verification methods to cope with emerging technical needs such as network security, mobile intelligent devices and high performance computing. So, it is very much essential for user to test the automata quickly and efficiently.

However manual input of string by user causes some cases to be neglected and also very tedious for user. The probability of a wrong automata being verified is very high in this case, because of some complex cases remaining unnoticed.

We are therefore, through this paper proposing a concept wherein the user after designing automata does not directly give string as an input, instead user will be prompted for the regular expression from which the automata is created for, and all possible strings from desired regular expression will be checked on the automata finally verifying to user whether automata is valid or not.

In the next section, problem definition is explained. Section III describes the existing solution and Section IV explains our proposed solution.

2. Problem Definition

At present the simulator available, it lets the user first design an automata based on the regular expression for a particular condition, then various strings are given as input by user and finally the automata is validated based on these strings.

Under such circumstances there are chances, the user might not be able to cover all the conditions or strings that the automata must accept or reject. This is the part of human error. So, it is very much possible that for a complex regular expression, the created automata might not be verified properly since not all the strings can be given as an input by user and also it takes time for user to enter and verify all the strings, thus making the process is time consuming.

3. Existing Solution

Currently for a regular expression, user creates automata and continues to give various strings so that all variations for the said condition is covered. All the strings are checked one by one and verified with the automata created for a particular regular expression finally informing to user if string is accepted or rejected for created automata.

However the string input given might not be able to cover all condition. Thus this approach of giving manual input is not a foolproof method to validate automata.

4. Proposed Solutions

Instead of giving strings of the required regular expression as input, the regular expression itself is given as input. Based on this regular expression, a possible set of strings are generated by the program which is then supplied so as to validate the automata.

As strings are generated using desired regular expression, chances of missing a particular condition are minimized. Even for very complex problem, maximum of the strings will be checked which might be difficult or rather tedious for any particular user to check and thus increasing efficiency of the simulator and saving time of the user.

Following are steps in which the entire approach should be working:

4.1. Finite Automata

An automata with a set of states, and its "control" moves from state to state in response to external "inputs" is called a finite automaton.

A finite automaton, FA, provides the simplest model of a computing device. It has a central processor of finite capacity and it is based on the concept of state. It can also be given a formal mathematical definition. Finite automata are used for pattern matching in text editors, for compiler lexical analysis.

Another useful notion is the notion of nondeterministic automaton.

We can prove that deterministic finite automata, DFA, recognize the same class of languages as NDFA, i.e., they are equivalent formalisms.

It is also possible to prove that given a language L there exists a unique (up to isomorphism) minimum finite state automaton that accepts it, i.e. an automaton with a minimum set of states.

The automata in the examples are deterministic, that is, once their state and input are given, their evolution is uniquely determined.

4.2. Regular Expression

Basically, a regular expression is a pattern describing a certain amount of text. Their name comes from the mathematical theory on which they are based. But we will not dig into that. You will usually find the name abbreviated to "regex" or "regexp". This tutorial uses "regex", because it is easy to pronounce the plural "regexes".

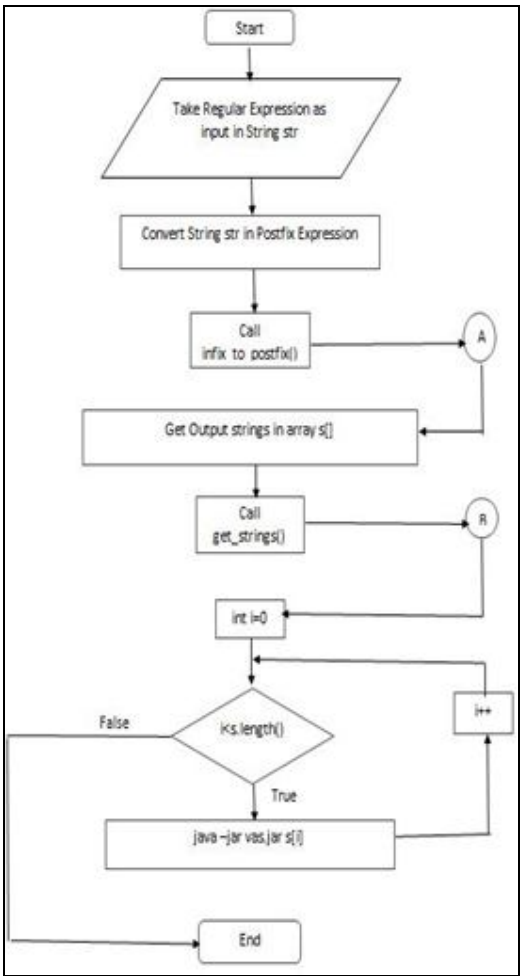
\b[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\b is a more complex pattern. It describes a series of letters, digits, dots, underscores, percentage signs and hyphens, followed by an at sign, followed by another series of letters, digits and hyphens, finally followed by a single dot and between two and four letters. In other words: this pattern describes an email address.

With the above regular expression pattern, you can search through a text file to find email addresses, or verify if a given string looks like an email address. This uses the term "string" to indicate the text that the regular expression is applied to. The term "string" or "character string" is used by programmers to indicate a sequence of characters. In practice, you can use regular expressions with whatever data you can access using the application or programming language you are working with.

4.3. Regular Expresion to String Generation

The following are a few examples of regular expressions and the languages generated using these operators:

- a+b+c = {a, b, c}
- abc = {abc}
- (!+a)bc = {bc, abc}
- ab* = {a, ab, abb, abbb, ...}
- (ab)* = (λ, ab, abab, ababab, ...)
- (a+b)* = (λ, a, b, aa, ab, ba, bb, aaa, ...)
- a+b* = (a, λ, b, bb, bbb, ...)
- a+!* = (a, λ)
- (a+!)* = (λ, a, aa, aaa, aaaa, ...)



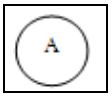
4.4. Passing Strings Generated to the Automata

The strings thus generated according to regular expression are passed to the automata and verified with automata, thus covering all the cases.

4.5. Algorithm

- Start
- Create automata as per condition.
- Accept Regular Expression in str
- Convert str in postfix expression, call infix_to_postfix()
- Get output strings in s[], call get_strings()
- Pass strings to Visual Automata Simulator for loop from i=0 to i<s.length() java –jar vas.jar s[i]
- Stop

4.6. Flow Chart



4.7. Algorithm for Infix to Postfix

- Define a stack
- Go through each character in the string
- If it is between 0 to 9, append it to output string.
- If it is left brace push to stack
- If it is operator *+-/ then
- If the stack is empty push it to the stack
- If the stack is not empty then start a loop:

- If the top of the stack has higher precedence
- Then pop and append to output string
- Else break
- Push to the stack
- If it is right brace then
- While stack not empty and top not equal to left brace
- Pop from stack and append to output string
- Finally pop out the left brace.
- If there is any input in the stack pop and append to the output string.

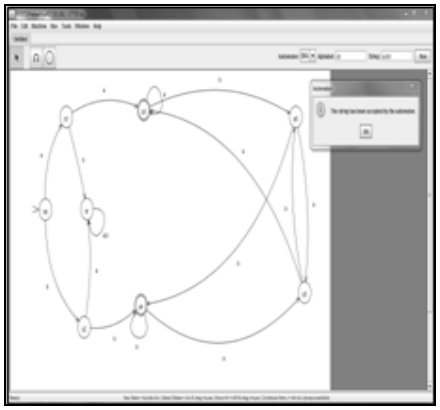


Figure 1: Output of String accepted

5. Results

- Automata drawn in Simulator
- Output for getting strings according to Regular Expression
  C:\user\student\desktop>javac RE.java
  C:\user\student\desktop>java RE
  Enter Regular Expression: aa(a+b)*bb + bb(a+b)*aa
  Basic Strings: aa,bb,aabb,bbaa
- Accepted Strings: aaabb, bbabb, bbababb, aababababb, bbaabbaa…
- The String "aabb" getting accepted

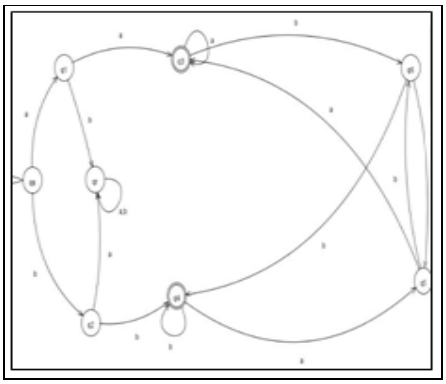


Figure 2: Automata starting and ending with double letter

6. Constraints

Limitation for this idea is it cannot give output for automata or regular string which has infinite input strings as computer memory cannot store unlimited data. E.g. if we take Language that has condition "ends with a" in {a,b}, then it can produce unlimited strings which ends with a i.e. ba,bba,baa,bbababa,bababababbbbaaa…

7. Conclusion

We have been able to achieve a new approach for validation. Thus the human error as a result of manual input and validation is minimized. Looking forward we believe that similar approach can also be used to validate various java, c++ codes. Instead of giving manual input we must develop mechanism whereby computer itself covers all instances and validates the code.

8. Acknowledgment
We would like to thank our TCS Professors Ms.Sridhari Iyer and Ms. Prachi for their patience and guiding us on the said topic.

9. References

1. Nils Klarlund and Michael I. Schwartzbach ," A Domain-Specific Language for Regular Sets of Strings and Trees",Proc. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 25, NO. 3, MAY/JUNE 1999
2. Trevor Martin, Member, IEEE, Yun Shen, and Ben Azvine," Incremental Evolution of Fuzzy Grammar Fragments to Enhance Instance Matching and Text Mining",Proc. IEEE TRANSACTIONS ON FUZZY SYSTEMS, VOL. 16, NO. 6, DECEMBER 2008
3. Tsern-Huei Lee, Senior Member, IEEE," Hardware Architecture for High-Performance Regular Expression Matching",Proc. IEEE TRANSACTIONS ON COMPUTERS, VOL. 58, NO. 7, JULY 2009
4. Tips for Writing Technical Papers, http://infolab.stanford.edu/~widom/paper-writing.html
5. Writing a technical paper, http://homes.cs.washington.edu/~mernst/advice/write-technical-paper.html
6. Regular Expressions Reference, http://www.regular-expressions.info/reference.html
7. Regular Expressions, http://www.grymoire.com/Unix/Regular.html
8. Regular Expression, http://wiki.tcl.tk/461
9. Details of Regular Expression Behavior, http://msdn.microsoft.com/en-us/library/e347654k.aspx
10. Regular Expressions and Converting to a NFhttp://www.jflap.org/tutorial/regular/A
11. Algorithm for infix to postfix conversion, http://c4learn.com/tutorials/data-structure/stack/algorithm-for-infix-to-postfix-conversion-using-stack/
12. Finite Automata, http://www.mec.ac.in/resources/notes/notes/automata/dfa.htm
13. Deterministic Finite Automata (DFAs) http://lambda.uta.edu/cse5317/notes/node8.html