International Journal of scientific research and management (IJSRM)
\||Volume||3||Issue||5||Pages|| 2748-2751||2015||
Website: www.ijsrm.in ISSN (e): 2321-3418

Conversion of Regular Expression in To Finite Automata

Neha, Abhishek Sharma

M.Tech, Assistant Professor

Department of Cse, Shri Balwant College of Engineering &Technology

Dcrust University

***Abstract* - Regular expressions are used to represent certain set of string in algebraic manner. This paper describes a method for constructing a minima deterministic finite automaton (DFA) from a regular expression. It is based on a set of graph grammar rules for combining many graphs (DFA) to obtain another desired graph (DFA). The graph grammar rules are presented in the form of a parsing algorithm that converts a regular expression R into a minimal deterministic finite automaton M such that the language accepted by DFA M is same as the language described by regular expression R. The proposed algorithm removes the dependency over the necessity of lengthy chain of conversion, that is, regular expression → NFA with ε-transitions → NFA without ε- transitions → DFA → minimal DFA. Therefore the main advantage of our minimal DFA construction algorithm is its minimal intermediate memory requirements and hence, the reduced time complexity. The proposed algorithm converts a regular expression of size n in to its minimal equivalent DFA in O(n.log2n) time. In addition to the above, the time complexity is further shortened to O (n.logen) for n ≥ 75.**

Keywords: **DFA, NDFA, RE, Automata, Finite Machine, state, Function**

I. INTRODUCTION

The derivative of a set of strings S with respect to a symbol a is the set of strings generated by stripping the leading a from the strings in S that start with a. For regular sets of strings, i.e., sets defined by regular expressions (REs), the derivative is also a regular set. In a 1964 paper, Janusz Brzozowski presented an elegant method for directly constructing a recognizer from a regular expression based on regular-expression derivatives (Brzozowski, 1964). His approach is elegant and easily supports extended regular expressions; i.e., REs extended with Boolean operations such as complement. Unfortunately, RE derivatives have been lost in the sands of time, and few computer scientists are aware of them.1 Recently, we independently developed two scanner generators, one for PLT Scheme and one for Standard ML, using RE derivatives. Our experiences with this approach have been quite positive: the implementation techniques are simple, the generated scanners are usually optimal in size, and the extended RE language allows for more compact scanner specifications. Of special interest is that the implementation techniques are well-suited to functional languages that provide good support for symbolic term manipulation (e.g., inductive data types and pattern matching). The purpose of this paper is largely educational. Our positive experience with RE derivatives leads us to believe that they deserve the attention of the current generation of functional programmers, especially

those implementing RE recognizers. We begin with a review of background material in Section 2, introducing the notation and definitions of regular expressions and their recognizers. Section 3 gives a fresh presentation of Brzozowski's work, including DFA construction with RE derivatives.

A. Preliminaries

We assume a finite alphabet Σ of symbols and use $\Sigma *$ to denote the set of all finite strings over Σ. We use a, b, c, etc., to represent symbols and u, v, w to represent strings. The empty string is denoted by ε. A language of Σ is a (possibly infinite) set of finite strings $L \subseteq \Sigma *$.

II. Regular expressions

Our syntax for regular expressions includes the usual operations: concatenation, Kleene closure, and alternation. In addition, we include the empty set (∅) and the Boolean operations "and" and "complement."2

Definition 2.1
The abstract syntax of a regular expression over an alphabet Σ is given by the following grammar: r, s ::= ∅ empty set | ε empty string | a a ∈ Σ | r · s concatenation | r ∗ Kleene-closure | r + s logical or (alternation) | r & s logical and | ¬r complement These expressions are often called extended regular expressions, but since the extensions are conservative (i.e., regular languages are closed under Boolean operations (Rabin & Scott, 1959)), we refer to them as regular expressions. Adding boolean operations to the syntax of regular expressions greatly enhances their expressiveness, as we demonstrate in Section 5.1. We use juxtaposition for concatenation and we add parentheses, as necessary, to resolve ambiguities. The regular languages are those languages that can be described by regular expressions according to the following definition.

A. Finite state machines

Finite state machines (or finite automata) provide a computational model for implementing recognizers for regular languages. For this paper, we are interested in deterministic automata, which are defined as follows:

Definition 2.2

A deterministic finite automaton (DFA) over an alphabet Σ is 4-tuple hQ, q0, F, δi, where Q is a finite set of states, q0 ∈ Q is the distinguised start state, F ⊆ Q is a set of final (or accepting) states, and δ : Q × Σ → Q is a partial function called the state transition function. We can extend the transition function δ to strings of symbols ˆδ(q, ε) = q ˆδ(q, au) = ˆδ(q 0 , u) when q 0 = δ(q, a) is defined The language accepted by a DFA is defined to be the set of strings {u | ˆδ(q0, u) ∈ F}

B. Regular expression derivatives

In this section, we introduce RE derivatives and show how they can be used to construct DFAs directly from Res . 3.1 Derivatives The notion of a derivative applies to any language. Intuitively, the derivative of a language L ⊆ Σ ∗ with respect to a symbol a ∈ Σ is the language that includes only those suffixes of strings with a leading symbol a in L.

C. Regular Expression

A regular expression (RE) is a pattern that describes some set of strings. Regular expression over a language can be defined as:

1) Regular expression for each alphabet will be represented by itself. The empty string (ϵ) and null language (ϕ) are regular expression denoting the language {ϵ} and {ϕ} respectively.

2) If E and F are regular expressions denoting the languages L (E) and L (F) respectively, then following rules can be applied recursively.

    a. Union of E and F will be denoted by regular expression E+F and representing language L (E) U L (F).

    b. Concatenation of E and F denoted by EF and representing language L (E*F) = L (E) * L (F).

    c. Kleene closure will be denoted by E* and represent language (L (E))*.

3) Any regular expression can be formed using 1-2 rules only.

III. Conversion of RE to FA

It turns out that every Regular Expression has an equivalent NFA and vice versa. There are multiple ways to translate RE into equivalent NFA's but there are two main and most popular approaches. The first approach and the one that will be used during this thesis is the Thompson algorithm and the other one is McNaughton and Yamada's algorithm.

A. Thompson's algorithm

Thompson algorithm was first described by Thompson in his CACM paper in 1968. Thompson's algorithm parse the input string (RE) using the bottom-up method, and construct the equivalent NFA. The final NFA is built from partial NFA's, it means that the RE is divided in several sub expressions, in our case every regular expression is shown by a common tree, and every subexpression is a sub tree in the main common tree. Based on the operator the sub tree is constructed differently which results on a different partial NFA construction.

B. McNaughton and Yamada Algorithm

The idea of the McNaughton and Yamada algorithm is that it makes diagrams for sub expressions in a recursive way and then puts them together. According to Store and Chang [9] the McNaughton and Yamada's NFA has a distinct state for every character in RE except the initial state. We can say that McNaughton and Yamada's automaton can also be viewed as a NFA transformed from Thompson's NFA.

IV. Problem Statement

Regular expressions, also known as regex, are commonplace in computing. They are generally used for matching or replacing strings of text, such as when searching documents for words, filtering email for spam, or searching the web. They feature in most major programming languages and are central to the working of parsers such as Lex. More formally, they can define a class of languages known as regular languages. For regular expressions to be usable by computers they are converted to various types of finite automata. It is this process of conversion that we will investigate here. The types of finite automata we will deal with are:

- *DFA (deterministic finite automata)*
- *NFA (nondeterministic finite automata)*
- *GNFA (general nondeterministic finite automata)*

The steps we will perform to create them are:

- *Conversion of regular expression to NFA*
- *Conversion of NFA to DFA*
- *Conversion of DFA to minimal DFA*
- *Conversion of DFA to regular expressions*

This last step is to demonstrate that our implementation of the algorithms is correct – if the final regular expression describes the same language as the first in each test instance it will be a strong indicator of success.

It became clear early in the investigation that the most sensible method to reach our goal was to use the following:

▪ Thompson's Algorithm

▪ Subset Construction

▪ DFA minimisation by removal of dead, inaccessible and redundant states from DFA

▪ Conversion of DFA to GNFA and removal of states from GNFA

V. Implementation

The source code for a Java applet that, upon the user entering a regular expression can perform the following:

▪ *Convert regular expression to NFA*

▪ *Convert NFA to DFA*

▪ *Minimise DFA*

▪ *Convert DFA back to regular expression via GNFA*

A. Classes

The applet consists of six classes. Mostly they were straightforward to implement.

1. RegexAutomaton.class (The main class)

It creates the GUI, deals with input and creates any Regex objects as required which it then adds to its database.

2. Regex.class (An individual regex)

It contains the terms for the original regular expression, trees of Node objects for the NFA, DFA and minimised DFA, a two dimensional array representing the GNFA, and the terms for the final regular expression. It also contains the algorithms for converting one to another.

3. Node.class (An abstract parent class of the NfaNode and DfaNode classes)

It contains the basic commonality of Nodes – node number, indicators of initial and final states.

4. DfaNode.class (A subclass of the Node class)

It implements a DFA node. In addition to the sections inherited from its parent class it contains a vector for DfaNodes moved to from this DfaNode and functions to iterate through DfaNodes.

5**. NfaNode.class (A subclass of the Node class)**

It is similar to the DfaNode class but with the different transition table according to NFA requirements.

6. IllegalRegexTermsException.class

An exception class thrown when illegal regular expression terms are provided to the application, e.g. two pipes in a row, a regex that starts with a closing bracket, and so on.

B. Functions

The key functions used in above classes are explained below:

1. GetStatement () – This function takes care of most of the parsing. It checks for symbols or sequences of symbols. The various combinations of symbols it looks for are clearly commented in the code. Each term is converted to an NfaNode object, or a number of NfaNode objects, and is linked to the NfaNode objects created so far.

2. DoUnion () - If getStatement () function encounters a union it parses the left hand part itself and then calls this function to deal with the right hand side of the union.

3. ConvertNFAtoDFA () – It converts the graph of NfaNode objects to a graph of DfaNode objects.

4. MinimiseDFA () – It copies the graph of DfaNode objects and minimises it.

5. ConvertDFAtoRegex () – It converts the minimised graph of DfaNode objects back to a regex by creating and minimizing a GNFA in the form of a two dimensional array of String objects.

6. MakeCell () – applies Arden's rule to the GNFA matrix cells to minimize the GNFA

VI. Result

Regular expression to deterministic finite automata and vice versa using heuristics proposed by various researchers. Software for conversion has been developed in java. After execution of software the GUI screen of java applet is displayed. The screen consists of multiple elements with which users can interact with the software as shown in figure 6.1. The components that are displayed on the user interface are listed below:

▪ **JLabel:** A Label is a graphical control that is used to display any specific text on the applet window. The text (caption) displayed by the label control is not directly modifiable. Normally label control is placed before input and output box for representing I-O message.

▪ **JTextField:** Like label, it is also a Graphical control that is used to display any specific text on the form window. But we can perform modification of this text directly. Therefore, Text box control can be used for accepting input & displaying output.

▪ **JButton:** It is most widely used control on the user interface. A user can initiate action on this control by clicking on it. A caption is specified on button to represent type of action to be performed.

▪ **JScrollPane:** A scroll pane is a component that presents a rectangular area in which a component may be viewed. Horizontal & Vertical scroll bars may be provided if necessary. The scroll panes are implemented by JScrollPane class.

▪ **JTextArea:** The JTextArea component is used for displaying or entering multiple lines text. The user can enter any number of lines of text, using the enter key to separate them

Conclusion

Researching this paper has shown that the conversion of regular expressions to DFA and back again are processes that are well understood and are implementable without any great difficulty. The most time-consuming part of the project was coding the parser for the regular expression. This is because while regular expressions define regular languages, they themselves are not regular and must be described by context-free grammars.

References

[1] Alfred V. Aho, "Constructing a Regular Expression from a DFA", Lecture notes in Computer Science Theory, September 27, 2010, Available at http://www.cs.columbia.edu/~aho/cs3261/lectures.

[2] Ding-Shu Du and Ker-I Ko, "Problem Solving in Automata, Languages, and Complexity", John Wiley & Sons, New York, NY, 2001.

[3] Gelade, W., Neven, F., "Succinctness of the complement and intersection of regular expressions", Symposium on Theoretical Aspects of Computer Science. Dagstuhl Seminar Proceedings, vol. 08001, pages 325–336. IBFI (2008).

[4] Gruber H. and Gulan, S. (2009), "Simplifying regular expressions: A quantitative perspective", IFIG Research Report 0904.

[5] Gruber H. and Holzer, M., "Provably shorter regular expressions from deterministic finite automata", LNCS, vol. 5257, pages 383–395. Springer, Heidelberg (2008).

[6] Gulan, S. and Fernau H., "Local elimination-strategies in automata for shorter regular expressions", In Proceedings of SOFSEM 2008, pages 46–57 (2008).

[7] H. Gruber and M. Holzer, "Finite automata, digraph connectivity, and regular expression size", In Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Iceland, July 2008. Springer.

[8] H. Gruber and J. Johannsen, "Optimal lower bounds on regular expression size using communication complexity", In Proceedings of the 11th International Conference Foundations of Software Science and Computation Structures, volume 4962 of LNCS, pages 273–286, Budapest, Hungary, March–April 2008. Springer.

[9] J. J. Morais, N. Moreira, and R. Reis, "Acyclic automata with easy-to-find short regular expressions", In 10th Conference on Implementation and Application of Automata, volume 3845 of LNCS, pages 349–350, France, June 2005. Springer.

[10] K. Ellul, B. Krawetz, J. Shallit, and M.Wang, "Regular expressions: New results and open problems", Journal of Automata, Languages and Combinatorics, 10(4):pages 407– 437, 2005.

[11] Larkin, H., "Object oriented regular expressions", 8th IEEE International Conference on Computer and Information Technology , vol., no., pages 491-496,8-11 July,2008

[12] Peter Linz, *Formal Languages and Automata (Fourth Edition)*, Jones and Bartlett Publishers, 2006

[13] Michael Sipser, *Introduction to the Theory of Computation*, Thomson Course Technology, 2006