International Journal of Latest Trends in Engineering and Technology
Special Issue SACAIM 2017, pp. 354-361
e-ISSN:2278-621X

A SURVEY OF PATTERN MATCHING ALGORITHMS IN REGULAR EXPRESSIONS

Jesslee Velmon Crasto[1] & Soujanya[2]

Abstract- A regular expression or regex is a sequence of characters that define a search pattern. This pattern is then used by string searching algorithms for performing finding and replacing operations on strings. This concept was invented by American mathematician Stephen Cole Kleene in 1950's when he formalized the description of a regular language. A regex processor translates a regular expression into an internal representation which can be executed and matched against a string representing the text being searched in. One possible approach is the Thompson's construction algorithm to construct a nondeterministic finite automaton (NFA), which is then made deterministic and the resulting deterministic finite automaton (DFA) is run on the target text string to recognize substrings that match the regular expression. This paper shows the conversion of Regular expression into NFA to match strings using Thompsons Construction and DFA to Regular Expression using Kleene's algorithm.
Keywords – DFA, NFA, Regular Expression, Thompsons Construction, Kleene's Algorithm

1. INTRODUCTION

Regular Expressions is nothing but a sequence of characters. These expressions say for example [0-9] means that the expression should contain numbers. Regular expressions are used in many situation in computer programming. Majorly in search pattern matching, parsing, filtering of results and so on. You may see this websites where you are only forced to enter only numbers or only characters or minimum 8 characters and so on. These are controlled by regular expression behind the screens. The DFA or Deterministic Finite Automata is a finite state machine that rejects or accepts strings of symbols and only produces computation which is unique for each input string of the automaton. NFA (Nondeterministic Finite Automata) does not obey the restrictions of the DFA. NFA takes in a string of input symbols. Until all input symbols have been consumed, each input symbol transitions to a new state. Thompsons Construction transforms regular expression to NFA to match strings against regular expressions. Kleene's Algorithm transforms a DFA into a regular expression.
The rest of the paper is organized as follows. Thompsons Construction algorithm and Kleene's Algorithm are explained in section II. Experimental results are presented in section III. Concluding remarks are given in section IV.

2. PROPOSED ALGORITHM

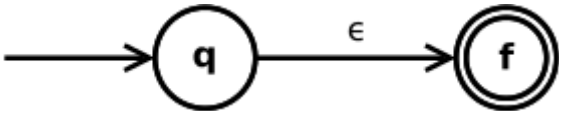
2.1 Thompson's Construction algorithm –
Thompson's Construction algorithm works by recursive splitting of an expression into subexpression, from which the NFA will be constructed using a set of rules.[1] More precisely, from a regular expression E, the obtained automaton A with the transition function δ respects the following properties:

- A has exactly one initial state q_0, which is not accessible from any other state. That is, for any state q and any letter a, does not contain q_0.

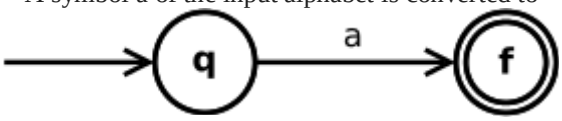
- A has exactly one final state q_f, which is not co-accessible from any other state. That is, for any letter a, .
- Let c be the number of concatenation of the regular expression E and let s be the number of symbols apart from parentheses — that is, |, *, a and ε. Then, the number of states of A is 2s − c (linear in the size of E).
- The number of transitions leaving any states is at the most two.
- Since NFA of m states and e transitions from every state can match a string of length n in time O(emn), a Thompson NFA can match pattern in linear time, assuming a fixed-size alphabet.

Rules
The following rules are depicted according to Aho et al. (1986),[3] p. 122. N(s) and N(t) is the NFA of the subexpression s and t, respectively.
The empty-expression ε is converted to

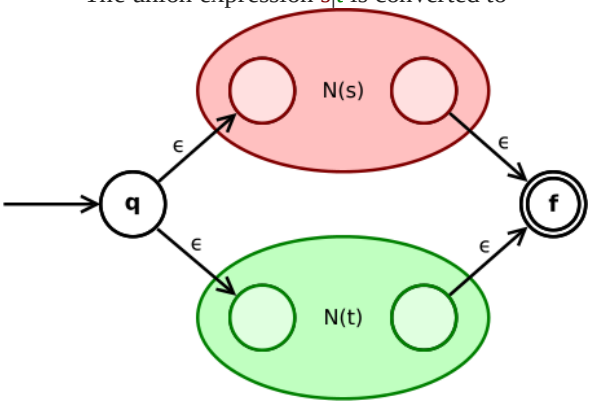
[1] Department of Master of Computer Applications, St. Aloysius Institute of Management and Information Technology (AIMIT), Mangalore, Karnataka, India
[2] Department of Master of Computer Applications, St. Aloysius Institute of Management and Information Technology (AIMIT), Mangalore, Karnataka, India



A symbol a of the input alphabet is converted to

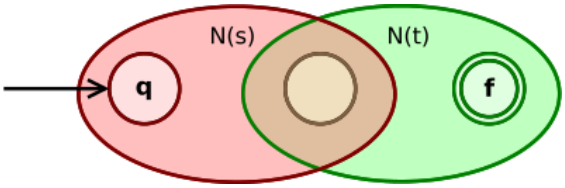


The union expression s|t is converted to



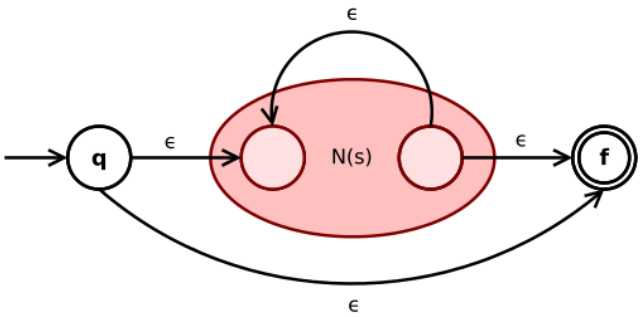
State q goes via ε to the initial state of N(s) or N(t). The final states become intermediate states of the whole NFA and merge via two ε-transitions into the final state of the NFA.

The concatenation expression st is converted to



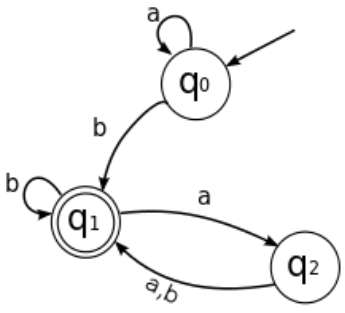
The initial state of N(s) is the initial state of the whole NFA. The final state of N(s) becomes the initial state of N(t). The final state of N(t) is the final state of the whole NFA.

The Kleene star expression s^* is converted to



An ε-transition connects final and initial state of the NFA with the sub-NFA N(s) in between. Another ε-transition from the inner final to the inner initial state of N(s) allows for repetition of expression s according to the star operator.

2.2 Kleene's algorithm –



In computer science, Kleene's algorithm transforms a given (DFA) deterministic finite automaton into a regular expression. Together with other conversion algorithms, it establishes the equivalence of several description formats for regular languages.
Algorithm description
The algorithm can be traced back to Kleene (1956). According to Gross and Yellen (2004),
This description follows Hopcroft and Ullman (1979). Given a deterministic finite automaton M = (Q, Σ, δ, q, F), with Q = { q,...,q } its set of states, the algorithm computes
Here, "going through a state" means entering and leaving it, so both i and j may be higher than k, but no intermediate state may. Each set Rkij is represented by regular expressions. The algorithm computes step by step for k = -1, 0, ..., n. Since there is no state numbered higher than n, the regular expression Rn0jrepresents the set of all strings that take M from its start state q to q. If F = { q,...,q } is the set of accept states, the regular expression Rn01 | ... | Rn0f represents the language accepted by M.
The initial regular expressions, for k = -1, are computed as

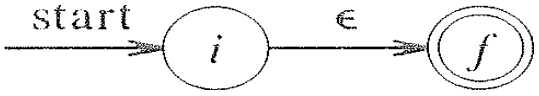
R−1ij = a | ... | a | ε, if i=j, where δ(q,a) = ... = δ(q,a) = q
In each step the expressions Rkij are computed from the previous ones by

3. EXPERIMENT AND RESULT

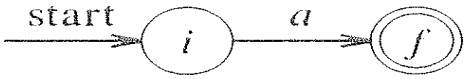
3.1 Thompsons Construction

Construction of an NFA from a Regular Expression
INPUT: A regular expression r over alphabet C. OUTPUT: An NFA N accepting L(r)

Begin by parsing r into its subexpressions. The rules for constructing an NFA contains of basis rules for handling subexpressions with no operators, and inductive rules for constructing larger NFA's from the NFA's for the immediate subexpressions of a given expression.
BASIS: For expression e construct the NFA

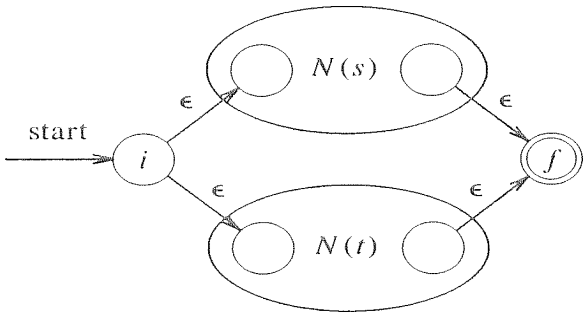


i is a new state, the start state of this NFA, and f is other new state, the accepting state for the NFA.
For any subexpression a in C, construct the NFA

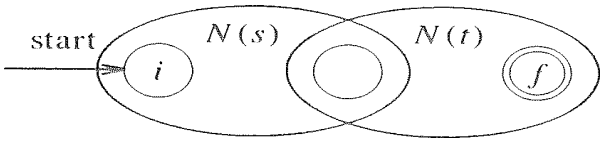


INDUCTION: Suppose N(s) and N (t) are NFA's for regular expressions s and t, respectively.

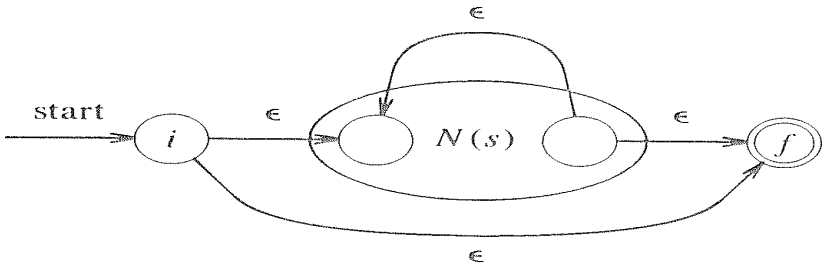
a) For the regular expression s|t,



b) For the regular expression st,



c) For the regular expression S*,



d) Finally, suppose r = (s). Then L(r) = L(s), and we can use the NFA, N(s) as N(r).

a. N(r) has at most twice as many states as there are operators and operands in r. This bound follows from the fact that each step of the algorithm creates at most two new states.

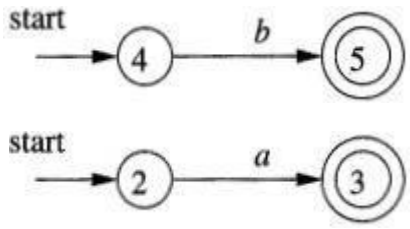
b. N(r) has one start state and one accepting state. The accepted state has no outgoing transitions, and the state which starts has no incoming transitions.

c. Each state of N (r) other than the accepting state has either one outgoing transition on a symbol in C or two outgoing transitions, both on E**.**

construct an NFA for r

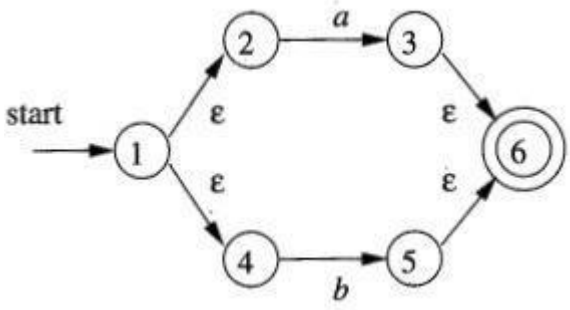
= (a|b)*a For r1 = a,

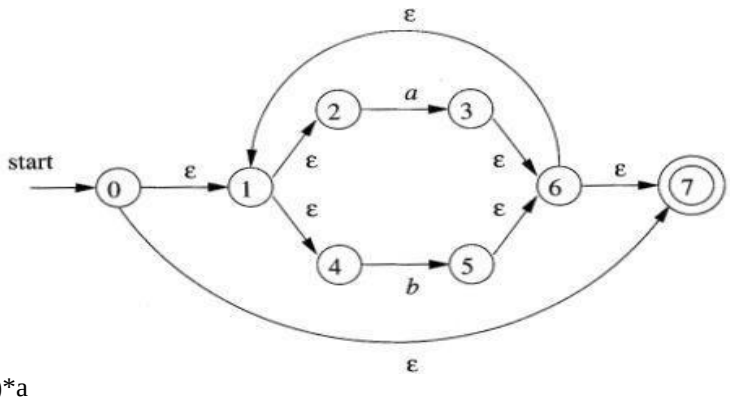
For r1 = b,



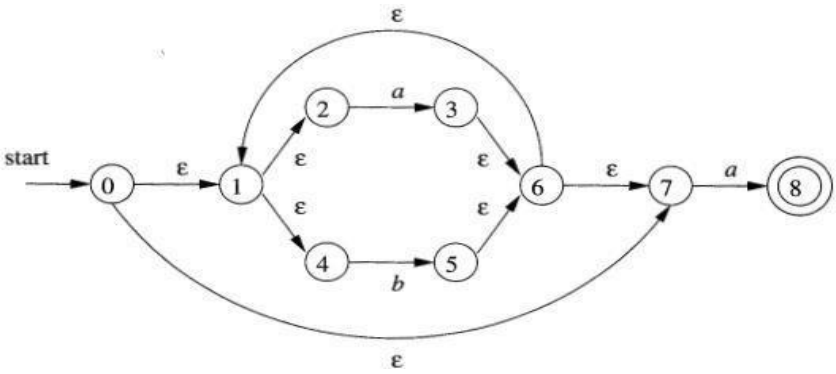
For r3 = a| b

The NFA for r5 = (r3) is the same as r3. The NFA for r6 = (r3)*





Finally NFA for r = (a|b)*a



3.2 Kleene's Algorithm

The automaton shown in the picture can be described as M = (Q, Σ, δ, q_0, F) with

- the set of states Q = { q_0, q_1, q_2 },
- the input alphabet Σ = { a, b },
- the transition function δ with $\delta(q_0,a)=q_0$, $\delta(q_0,b)=q_1$, $\delta(q_1,a)=q_2$, $\delta(q_1,b)=q_1$, $\delta(q_2,a)=q_1$, and $\delta(q_2,b)=q_1$,
- the start state q_0, and
- set of states accepted F = { q_1 }.

Kleene's algorithm computes the regular expressions as

R^{-1}_{00} = a | ε

R^{-1}_{01} = b

R^{-1}_{02} = ∅

R^{-1}_{10} = ∅

R^{-1}_{11} = b | ε

R^{-1}_{12} = a

R^{-1}_{20} = ∅

R^{-1}_{21} = a | b

R^{-1}_{22} = ε

After that, the Rk
ij are computed from the Rk-1
ij step by step for k = 0, 1, 2. Kleene algebra equalities are used to simplify the regular expressions as much as possible.
Step 0:

R0 00	= R−1 00 (R−1 00)* R−1 00 | R−1 00	= (a | ε) (a | ε)* (a | ε) | a | ε		= a*
R0 01	= R−1 00 (R−1 00)* R−1 01 | R−1 01	= (a | ε) (a | ε)* b	| b	= a* b
R0 02	= R−1 00 (R−1 00)* R−1 02 | R−1 02	= (a | ε) (a | ε)* ∅	| ∅	= ∅
R0 10	= R−1 10 (R−1 00)* R−1 00 | R−1 10	= ∅	(a | ε)* (a | ε) | ∅	= ∅
R0 11	= R−1 10 (R−1 00)* R−1 01 | R−1 11	= ∅	(a | ε)* b	| b | ε = b | ε
R0 12	= R−1 10 (R−1 00)* R−1 02 | R−1 12	= ∅	(a | ε)* ∅	| a = a
R0 20	= R−1 20 (R−1 00)* R−1 00 | R−1 20	= ∅	(a | ε)* (a | ε) | ∅	= ∅
R0 21	= R−1 20 (R−1 00)* R−1 01 | R−1 21	= ∅	(a | ε)* b	| a | b = a | b
R0 22	= R−1 20 (R−1 00)* R−1 02 | R−1 22	= ∅	(a | ε)* ∅	| ε = ε

Step 1:

R1 00	= R0 01 (R0 11)* R0 10 | R0 00	= a*b	(b | ε)* ∅	| a* = a*

R1 01 = R0 01 (R0 11)* R0 = a^*b (b | ε)* (b | ε) | a^* b = a^* b^* b
11 | R0 01

R1 02 = R0 01 (R0 11)* R0 = a^*b (b | ε)* a | ∅ = a^* b^* ba
12 | R0 02

R1 10 = R0 11 (R0 11)* R0 = (b | ε) (b | ε)* ∅ | ∅ = ∅
10 | R0 10

R1 11 = R0 11 (R0 11)* R0 = (b | ε) (b | ε)* (b | ε) | b | ε = b^*
11 | R0 11

R1 12 = R0 11 (R0 11)* R0 = (b | ε) (b | ε)* a | a = b^* a
12 | R0 12

R1 20 = R0 21 (R0 11)* R0 = (a | b) (b | ε)* ∅ | ∅ = ∅
10 | R0 20

R1 21 = R0 21 (R0 11)* R0 = (a | b) (b | ε)* (b | ε) | a | b = (a | b) b^*
11 | R0 21

R1 22 = R0 21 (R0 11)* R0 = (a | b) (b | ε)* a | ε = (a | b) b^* a | ε
12 | R0 22

Step 2:

R2 00 = R1 02 (R1 22)* R1 = a^*b^*ba ((a|b)b^*a | ε)* ∅ | a^* = a^*
20 | R1 00

R2 01 = R1 02 (R1 22)* R1 = a^*b^*ba ((a|b)b^*a | ε)* (a|b)b^* | a^* b^* b =
21 | R1 01

R2 02 = R1 02 (R1 22)* R1 = a^*b^*ba ((a|b)b^*a | ε)* ((a|b)b^*a | ε) | a^* b^* ba =
22 | R1 02

R2 10	= R1 12 (R1 22)* R1 20 | R1 10	= b* a	((a|b)b*a | ε)* ∅	| ∅	= ∅
R2 11	= R1 12 (R1 22)* R1 21 | R1 11	= b* a	((a|b)b*a | ε)* (a|b)b*	| b*	=
R2 12	= R1 12 (R1 22)* R1 22 | R1 12	= b* a	((a|b)b*a | ε)* ((a|b)b*a | ε)	| b* a	=
R2 20	= R1 22 (R1 22)* R1 20 | R1 20	= ((a|b)b*a | ε)	((a|b)b*a | ε)* ∅	| ∅	= ∅
R2 21	= R1 22 (R1 22)* R1 21 | R1 21	= ((a|b)b*a | ε)	((a|b)b*a | ε)* (a|b)b*	| (a | b) b*	=
R2 22	= R1 22 (R1 22)* R1 22 | R1 22	= ((a|b)b*a | ε)	((a|b)b*a | ε)* ((a|b)b*a | ε)	| (a | b) b* a | ε	=

((step 2 simplification to be completed))

Since q_0 is the start state and q_1 is the only accept state, the regular expression R2 01 denotes the set of all strings accepted by the automaton.

4. CONCLUSION

The proof of Kleene's theorem implicitly defines a algorithm that can be used to construct the regular expression. It also suggests a recursive function that can be used to construct the expression. The most efficient solution, however, would be a dynamic programming solution, combining the simple and inefficient table-driven approach with the recursive solution.

5. REFERENCES

[1] Ken Thompson (Jun 1968). "Programming Techniques: Regular expression search algorithm". Communications of the ACM. 11 (6): 419–422. doi:10.1145/363347.363387.

[2] ^ Xing, Guangming. "Minimized Thompson NFA" (PDF).

[3] ^ Alfred V. Aho, Ravi Sethi, Jeffrey Ullman: Compilers: Principles, Techniques and Tools. Addison Wesley, 1986

[4] ^ Watson, Bruce W. (1995). A taxonomy of finite automata construction algorithms (PDF) (Technical report). Eindhoven University of Technology. Computing Science Report 93/43.

[5] ^ R. McNaughton, H. Yamada (Mar 1960). "Regular Expressions and State Graphs for Automata". IEEE Transactions on Electronic Computers. 9 (1): 39–47. doi:10.1109/TEC.1960.5221603.

[6] Jonathan L. Gross and Jay Yellen, ed. (2004). Handbook of Graph Theory. Discrete Mathematics and it Applications. CRC Press. ISBN 1-58488-090-2. Here: sect.2.1, remark R13 on p.65

[7] ^ Kleene, Stephen C. (1956). "Representation of Events in Nerve Nets and Finite Automate" (PDF). Automata Studies, Annals of Math. Studies. Princeton Univ. Press. 34.

[8] ^ John E. Hopcroft, Jeffrey D. Ullman (1979). Introduction to Automata Theory, Languages, and Computation. Addison-Wesley. ISBN 0-201-02988-X. Here: Theorem 2.4, p.33-34