Algorithm to Generate DFA for AND-operator in Regular Expression

Mirzakhmet Syzdykov
Institute of Information and Computing Technologies
125 Pushkin Str., 050010
Almaty, Republic of Kazakhstan

ABSTRACT

For the past time a number of algorithms were presented to produce a deterministic finite automaton (DFA) for the regular expression. These algorithms could be divided into what they used as an initial data from which to produce DFA. The method to produce DFA from non-deterministic finite automaton (NFA) by a subset construction could be generalized for extended regular expressions, including intersection, negation and subtraction of the regular languages. In this article the modified algorithm of subset construction is presented; this algorithm produces a unigram DFA for the regular expression with extensions (specifically AND-operator).

General Terms

Pattern Recognition; Finite Automata; Algorithms.

Keywords

Algorithm; Deterministic; Automaton; Extension; Intersection; Subset Construction.

1. INTRODUCTION

In this paper the common technical and theoretical approach is stated in order to build the finite automaton for extended regular expressions, which include operators like intersection (presented in this article), negation and subtraction. The latter operators (negation and subtraction) will be left for further research and discussion. In this article only *cross-product* of *control vector* for a transitions in NFA is defined and a way to use the vector values in subset construction in order to build a DFA for more effective, practical and technical use. This method could be generalized to the extended operations (intersection, negation, subtraction) over automata for pattern matching and subset construction.

2. REGULAR EXPRESSIONS

The regular expression is a method to describe verbally and syntactically the set of words, which is further defined as a *language*, in more readable, understandable and technically simple way. This expression is described by a grammar which in turn consists of a set of rules.

Let's describe in BNF form the regular expressions with some assumptions:

1) the regular expression describes language (finite or infinite set of words) specified by the following grammar;

2) let's define R and R_i as a regular expression, and A as a set of alphabetic symbols from **a** to **z** ($A = \{a, .., z\}$);

3) let's define $L(R)$ as a language of a regular expression R.

The regular expression then can be defined recursively as (in order of precedence from highest to lowest):

1. $R = \varepsilon$ (an empty word, $L(R) = \{\varepsilon\}$);
2. $R = A$ (a single symbol from alphabet A, $L(R) = \{a: a \in A\}$);
3. $R = R^+$ (an infinite language $L(R) = L(R) \cup L(RR) \cup L(RRR\ldots)$);
4. $R = R^* = \varepsilon \mid R^+$ (an infinite language or Kleene closure: $L(R) = \{\varepsilon\} \cup L(R)$);
5. $R = R?$ (a set of words $L(R) = \{\varepsilon\} \cup L(R)$);
6. $R = R_1R_2$ (a set of words $L(R) = \{ab: \forall a \in L(R_1), \forall b \in L(R_2)\}$);
7. $R = R_1|R_2$ (a set of words $L(R) = L(R_1) \cup L(R_2)$).

2.1 Definition of Extended Regular Expression

The extended regular expression is a regular expression supporting one more operation on languages (AND-operator). The AND-operator can be described in regular expression grammar with an additional definition:

$$R = R_1 \,\&\, R_2 \text{ (a set of words } L(R)\text{: } L(R) = L(R_1) \cap L(R_2)\text{).}$$

This operation is an intersection of languages produced by sub-expressions (conjunction operator). It has the lowest precedence in regular expression.

Extended regular expressions also include subtraction and negation operator which are equal in technical sense, because negation is a subtraction of a *closed language* and an operand:

$$\sim R = A^* - R.$$

The subtraction or MINUS-operator can be defined as follows:

$$R = R_1 - R_2.$$

The language of the subtraction can be also described:

$$L(R) = \{w: w \in L(R_1) \,\&\, w \mathrel{!\in} L(R_2)\}.$$

3. THOMPSON ALGORITHM

In [1] there is a description of algorithm to construct automaton for the regular expression in order to match the word. For each of the element of regular expression starting and final states are recursively created. The algorithm can be illustrated by the diagram on Figure 1. The result of this algorithm is NFA for the input regular expression (an argument).

4. SUBSET CONSTRUCTION

In [2] there is a description of converting NFA to DFA. These conversion is called subset construction. The naming of algorithm is based on fact that algorithm utilizes Kleene closure result for a set of reachable states in corresponding NFA. These sets are thus divided by the DFA-states, which in turn are subsets of NFA states, by the non-empty transitions in NFA. These facts together form a *subset paradigm*.

ALGORITHM 1. SUBSET CONSTRUCTION

```
• NFA can be converted to DFA [2].
• This can be achieved by NFA simulation.
• Functions used in algorithm:
      1. Move $(s_i, a)$ is a set of states reachable from $s_i$ by symbol a;
      2. $\varepsilon - \text{closure}(s_i)$ is a set of states reachable from $s_i$ by empty transition ε.
• Steps of algorithm:
1. Start from the initial state
  $s_0$, $S = \varepsilon - \text{closure}\ (s_0)$;
2. Add DFA-state S to the stack T.
While stack T is not empty perform the following steps:
  Pop element t from the stack T;
    For each symbol $a \in A$:
      Compute set $r_1 = \{\text{Move}(t_i, a) : \forall t_i \in t\}$;
      Compute set $r_2 = \{\varepsilon - \text{closure}(r_i) : \forall r_i \in r_1\}$;
      If DFA-state $r_2$ wasn't encountered before, then add $r_2$ to the stack T.
        Add a DFA-transition $F[t, a] = r_2$.
```

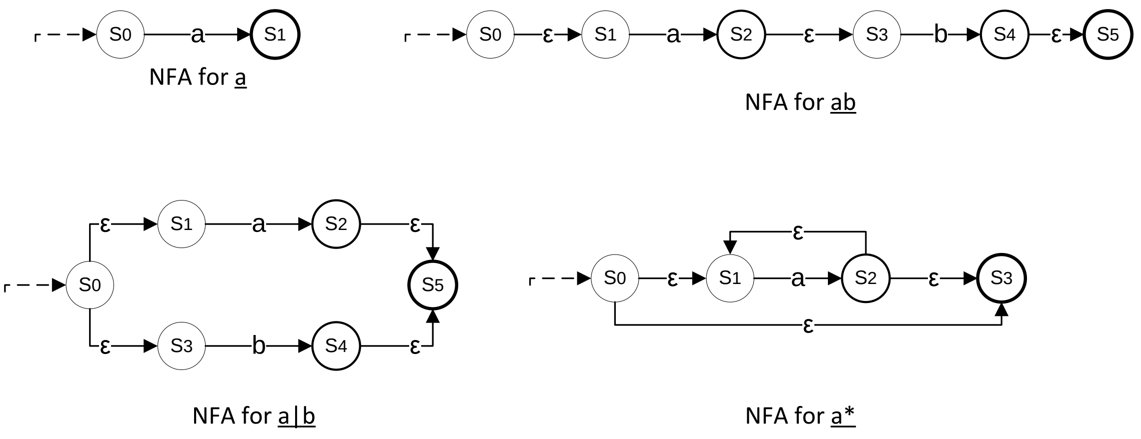

Fig 1: Example of Thomson algorithm result (empty transitions marked with "ε"-sign)

5. ALGORITHM TO PRODUCE DFA FOR AND-OPERATOR

To produce DFA, the subset construction algorithm described in previous section will be parametrized with a modified NFA. This NFA is *extended* as it has counters for each of the state. The counter is a positive integer number representing minimal number of input transitions to the state required to make it active. When the state is active it can be used to compute the next states reachable by empty transitions in function $\varepsilon - closure$.

5.1 NFA Construction for AND-Expression

Let us consider the NFA construction for the expression:

$$R = R_1 \& R_2.$$

Each state in the final NFA has default counter of value one, this means that the only transition is required to make it active. The construction of the NFA for this expression is same as for alternation operator:

$$R = R_1 | R_2.$$

The difference is that the final state of this construction has a counter of value two. Additional new final state is added. This additional state is required to create an *output state* to follow the subset construction from NFA with counters of value two.

This can be better illustrated by the diagram in Figure 2.

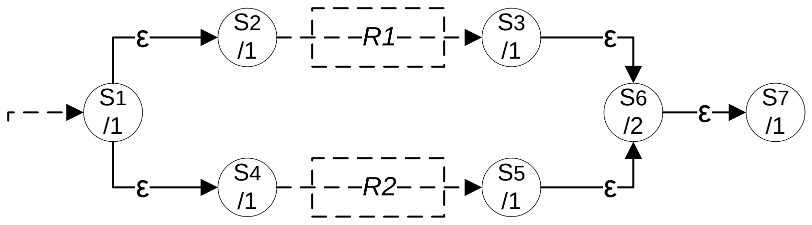

Fig 2: NFA construction for the expression "R1&R2" (counter values after "/"-sign)

Let us write the definition of the typical NFA as a tuple –

$<S, s_{initial}, F, E, A>$,

Where: S is a set of states,

$s_{initial}$ is an initial state ($s_{initial}$ is an element of S),

F is a set of final (accepting) states,

E is a set of edges in NFA:

$(e \in E: e \in S \times S \times (A \cup \{\varepsilon\}))$, and

A is a set of alphabet.

The edge thus is a triple (s_a, s_b, c) where c is a symbol mark on the edge accepting the alphabet element and s_a and s_b are initial and final states of the corresponding edge.

The definition of the extended NFA is same except that there is additional set C (a set of counter values for each of the state in the non-extended NFA):

$$C = \{c \in C: c \in S \times \{1,2\};\ \forall s \in S\}.$$

Please note, this NFA construction could be applied to two automata by replacing the R_1 and R_2 on Figure 2 with the corresponding graphs.

5.2 DFA Construction for Extended NFA

The algorithm is based on subset construction (Algorithm 1). The difference is in computing the reachable states in function $\varepsilon - closure$.

This algorithm is the same as for a typical NFA with respect to the counter value of the reached state:

1) At each step of computing the closure the counters are reinitialized to their default values;
2) Each time the state is reached its counter value is decreased by one;
3) The state can be considered active if its counter value equals zero.

5.3 Example of DFA for Extended Regular Expression

Let's consider the following regular expression:

$$(a^*|b^+)^+\&(a^+|b^*)^*.$$

The resulting DFA (initial state is grayed and the final states are marked with double-circle) can be viewed on Figure 3.

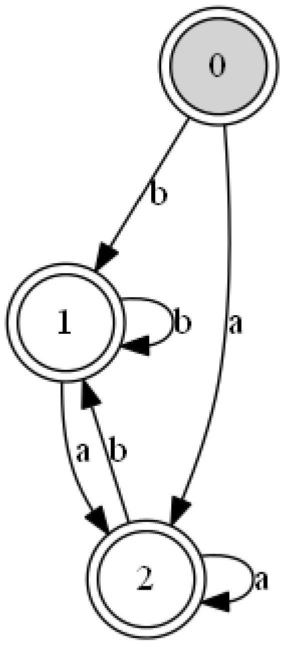

Fig 3: Example of DFA for AND-operator in regular expression

6. COMPARATIVE STUDY

The algorithm can be compared to the method of product construction for the DFA – D_1 and D_2. The resulting product DFA D_3 has the following property as a graph structure:

$$N(D_3) = N(D_1) * N(D_2),$$

Where: N is the number of states.

In [3] algorithm to build the minimal product of regular expressions for the intersection of their languages is described, it's shown that this method gives fewer states than product construction. This is based on the possibility of product automaton – the resulting DFA of product operation – to include the excess edges and states which have no *deterministic meaning* and thus have no path, or in more complex case – an access, to the final states.

Experimental study shows that algorithm, presented in this article, can produce not a feasible DFA. This means that the resulting DFA has states and edges that has no possible path to the final states (*deterministic meaning*). Thus, this DFA is to be optimized by the inverse edges from the final states. The states, which cannot be reached, are excluded.

7. EXPERIMENTAL RESULTS

The experimental results are provided for the regular expression "(((a|b)*a(a|b)*)&((a|b)*b(a|b)*))" (more practical example of similar expression can be found in [4]). This expression in test cases was repeated with direct concatenation and AND-operator. In Table 1 the benchmarks were collected, including the number of times the expression was repeated ("Repeat Count"), "Running Time" and "Number of States" in the produced DFA. The Figure 4 gives a graphical plot of the performance and memory allocation, depending on the number of states. As you can see the concatenation requires more running time and memory.

Table 1. Benchmarks of subset construction

	Concatenated		Concatenated by AND-operator	
Repeat Count	**Running Time (sec.)**	**Number of States**	**Running Time (sec.)**	**Number of States**
1	0,098	9	0,102	9
2	0,297	17	0,22	9
4	1,269	45	0,643	9
8	8,648	149	2,341	9
16	94,816	549	9,175	9

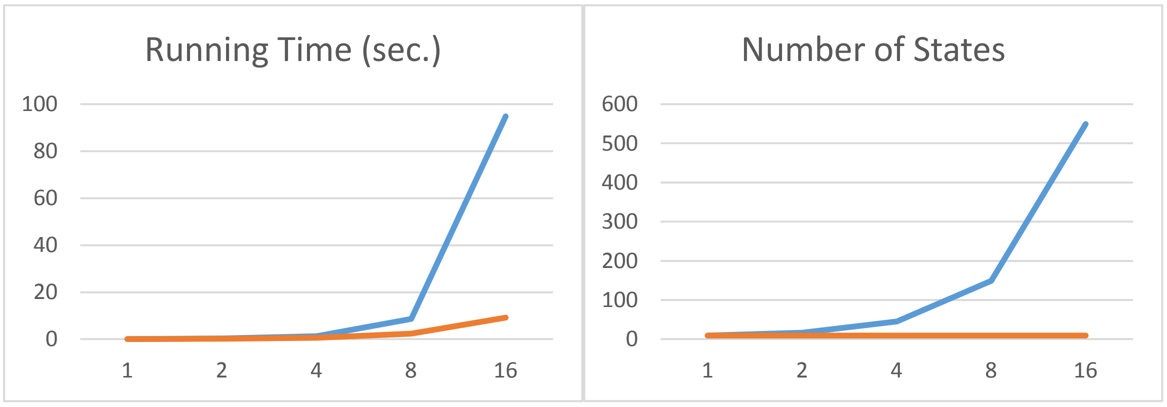

Fig 4: Graphical representation of data in Table 1

8. CONCLUSION AND FURTHER WORK

The algorithm in overall can be represented as a *cross-product* of NFA and *control vector* (see Section 1) which forms a *hierarchy*. The hierarchy can be defined as a set of semantic rules due to which the complexity expands. This hierarchy is bounded by AND-operator counters. The hierarchies *(a, b)* and *(a, b, c)* are equivalent over step of subset construction if in parameterized NFA the states *a* and *b* are met earlier than state *c*. These states are assigned the counter of value two, as they are states for the AND-operator construction.

The further work is to describe the theoretical continuation of cross-over product transition options for negation and subtraction operator, including the methods to use them in order to build the final DFA.

9. ACKNOWLEDGMENTS

It's necessary to acknowledge the previous work in the field of studying the subjects of finite automata, pattern matchings and applied algorithms. It's also required to acknowledge the Editorial Support Team of the International Journal of Computer Applications (Foundation of Computer Science, New York, USA) for the quick response on the request to publish the article.

10. REFERENCES

[1] K. Thompson. Regular expression search algorithm. Comm. ACM, 11 (6) (1968), pp. 419–422.

[2] M.O. Rabin, D. Scott. Finite automata and their decision problems. IBM J. Res. Develop., 3 (2) (1959), pp. 114–125.

[3] Samuel C. Hsieh. Product Construction of Finite-State Machines. Proceedings of the World Congress on Engineering and Computation Science, Vol. I (2010), pp. 141-143.

[4] Kai Wang, Jun Li. Towards Fast Regular Expression Matching in Practice. Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM (2013), pp. 531-532.