Membership Problem in Non-deterministic Finite Automata for Extended Regular Expressions in Linear Polynomial Time

Mirzakhmet Syzdykov

Institute of Problems in Informatics and Control,Almaty, Kazakhstan

mspmail598@gmail.com

Abstract. The recent work on implementing regular expression (RE) matching or membership problem solution for extended operators (ERE) like intersection, complement and subtraction gave exponential results on the size of input. As our previous result was focused on deterministic finite automata (DFA) for extended regular expressions, we present the new linear algorithm for ERE matching on non-deterministic finite automata (NFA) with the help of De Morgan's law to re-write the expression in exceptional order so that the conditional matching satisfies the correctness of the matching algorithm on NFA. We also prove that the proposed methodology is correct within the extensions of NFA like logical states as it was done before for DFA in the prior scientific work. We will also show how to implement these logical states in NFA using typical constructions like Thompson's. The proof of linear working time is also given which is obvious due to the re-writing rules for correct implementation. The important role of concatenation operator over extended constructions is also shown.

Keywords: regular expression, membership problem, extended operators, linear algorithm.

Introduction. Regular expressions denoted by R are the grammar rules which are used to express the languages L(R). This is widely used and powerful tool for working with textual data and parsing. It's commonly used in such programming products like Java, Perl or C#, etc.

We will define the languages and regular expressions as follows with respect to the supported operators:

$L(R) = \{ a: a \text{ in } A \}$, where A is the alphabet and a is a single symbol in it;

$L(R_1 \mid R_2) = \{ L(R_1) + L(R_2) \}$ - union of two languages;

$L(R_1 * R_2) = \{ L(R_1) * L(R_2) \}$ - concatenation of two languages;

$L(R^*) = \{ \text{eps}, L(R), L(R * R) \ldots L(R * R * \ldots * R) \}$ - Kleene closure or star operator, where "eps" stands for an empty word.

For the extended regular expressions we define the additional operators which give the possibility to operate on regular languages as if they would be the sets:

$L(R_1 \& R_2) = \{ L(R_1) \& L(R_2) \}$ - intersection of two languages defined by the expressions;

$L(R_1 - R_2) = \{ L(R_1) - L(R_2) \}$ - subtraction of two languages;

$L(\sim R) = \{ \sim L(R) \}$ - complement operator which defines all the strings in dictionary which aren't matched by regular expression R.

For our purposes we use the re-writing for complement operator which can be defined by subtraction operator as:

$$L(\sim R) = A^* - L(R). \quad (1)$$

In Combinatorics for logical and set operations there's De Morgan's law which is defined as follows:

$$A \mid B = \sim(\sim A \,\&\, \sim B),$$
$$A \,\&\, B = \sim(\sim A \mid \sim B). \quad (2)$$

The law (2) is true even for sets which can be defined by regular expressions and thus can be applied for the purpose of re-writing gramatically the regular expressions. Further we will show that it's required in special cases when concatenation operator is used.

In the next section the review of the previous work is given with respect to the actually presented optimal results in this work.

Before we will define the membership problem for the word w and regular expression R as follows:

$$w \text{ in } L(R). \quad (3)$$

Past work. Rosu gives the exponential algorithm for extended operators – this is the bes known result to the present time [1]. Our algorithm differs from the prior version for DFA with the same constructions [2]. Thompson constructions are linear to the size of regular expression and are used in our algorithm [3], this mainly doesn't limit the type of algorithm for NFA synthesis as we will prove further. Rabin and Scott provide us with the subset construction algorithm of converting NFA directly to DFA [4] – this is as stated before the previous result proposed by an author. Gelade and Neven give the exponential estimation of the size of the size, in our case, of the states for extended operators [5]. The last work focuses on the state explosion during subset construction when we were building DFA from the NFA[6] – this is the main fact why there is the reason of building the linear NFA within the linear matching time for the membership problem.

The prior works [7, 8] were focused on the building of semantic rules for subset construction of DFA supporting the extended operators – this is necessary to note that in our NFA the same logical rules are applied.

The work [9] is important to use in the sense of past published work with respect to the extended operators.

Hsieh gives the product algorithm for construction of certain types of operator extensions in ERE [10] – it was shown that these product constructions are of exponential nature [7]. For practical reasons of product construction algorithm on intersection, complement and subtraction we use Møller's software: for certain types of expressions like "(0*1*… | …) [&, &~] (1*0*… | …)" it demonstrates the exponential explosion of the number of states and transitions as well as DFA state explosion for expressions in form "(0|1)1(0|1)(0|1) … (0|1)".

Preliminaries. For the preliminary section we will define the NFA as a tuple:

$$<A, S, s_0, T, F>, \quad (4)$$

where A is an alphabet, S is a set of states, s_0 is a starting state, T stands for transitions and F is a set of final states.

In the past work [2] we defined the special states by which we extended the definition of NFA (4) with the intersection and subtraction operator constructions as follows:

$$<A, S, s_0, T, F, B(S)>, \quad (5)$$

where B(S) is the set of logical conditions to be satisfied during the matching process or subset construction.

We define the set B(S) for intersection and subtraction while the complement is given by re-writing rule (1):

$$B(S) = \{ (1 \mid 2): L(R_1) \,\&\, L(R_2); (1 \,\&\, \sim 2) : L(R_1) - L(R_2) \}, \quad (6)$$

where pair of number one and two stands for the flag conditions in the final implementation in order to the state become active and, thus, is to be included further in the stack during the matching process in general as opposed to the subset construction [2].

As we are finished with introduction, review of the past work and preliminary information it's time to present the algorithm with a proof.

Re-writing algorithm. Our algorithm for NFA construction is based upon the previous result [2], while the DFA construction is omitted as mainly DFA are exponentially large for the predefined set of expressions due to the Kleene star closure. Thus, we use the same non-deterministic constructions.

Before the construction we have to give the notion to the re-writing rules which are used for concatenation operator of two regular expressions with the different operators like union and intersection: other operators like complement and subtraction are omitted because there is no need to re-write the rules according to De Morgan's law so that the matching process is correct within the typical matching algorithm which is linearly defined by the upper bound:

$$O(NFA\ Matching) = O(N*M), \quad (7)$$

where N is the size of the regular expression and M is the limit of input. We will show in proof section of the algorithm that this is not required.

We state that in order of concatenation the following conditions are to be met in order to satisfy the correctness of the construction of NFA for solving membership problem:

$$NFA((R_1 \mid R_2) * (R_3 \,\&\, R_4)) =$$
$$NFA(\sim(\sim R_1 \,\&\, \sim R_2) * (R_3 \,\&\, R_4)) =$$
$$NFA((R_1 \mid R_2) * \sim(\sim R_3 \mid \sim R_4)), \quad (8)$$

where NFA(~R) is defined according to re-writing rule (1).

The same is true for subtraction and complement:

$$NFA((R_1 \mid R_2) * (R_3 - R_4)) = NFA(\sim(\sim R_1 \,\&\, \sim R_2) * (R_3 - R_4)),$$
$$NFA((R_1 \mid R_2) * \sim R_3) = NFA(\sim(\sim R_1 \,\&\, \sim R_2) * \sim R_3). \quad (9)$$

And finally for the last combination like intersection:

$$NFA((R_1 \,\&\, R_2) * (R_3 - R_4)) = NFA(\sim(\sim R_1 \mid \sim R_2) * (R_3 - R_4)),$$
$$NFA((R_1 \,\&\, R_2) * \sim R_3) = NFA(\sim(\sim R_1 \mid \sim R_2) * \sim R_3). \quad (10)$$

Thus, we use the re-writings in (8)-(10) according to De Morgan's law – the linear size of complement from regular expression here plays an important role and gives us the possibility to safe the size of the built NFA which is linear to the size of regular expression. The NFA by itself is constructed according to Thompson's rules [3].

It's necessary to note that the methodology as to the previous works [2, 7, 8] is also applicable to the general case for NFA as the preliminary automaton to be converted to deterministic is of the same modeling approach as it's used in the typical matching algorithm for the solution of membership problem.

In the next section we will prove the correctness of re-writing algorithm for certain types of intersection operator conditions when the operators like union and intersection are put in the right order.

Proof. In this section we will provide the reader with the proof of the correctness of the re-writing rules for the concatenation of extended operator of intersection and union-operator which is far more typical.

Let's define the function T(R) as the degree of possibility of matching the regular expression R in NFA – this function gives us the possibility to observe the number of situations during which the concatenation matching is resumed.

Thus, T(R) is defined as follows for the specific case like concatenation of union and intersection which, in turn, is the only special case to be considered:

$$T(R_1 \mid R_2) = 2,$$
$$T(R_1 \,\&\, R_2) = 1. \quad (11)$$

For the concatenation operator we define the correctness of the matching in membership problem as the state when logical conditions are satisfied for union and intersection as well – this can be written as:

$$T(R_1 * R_2): T(R_1) <= T(R_2). \quad (12)$$

As per our re-writing rules the necessary condition (12) is satisfied, thus the correctness of algorithm is proved. In general we can use empty string function [9] to compute the next node in abstract syntax tree when parsing regular expression, however, this can be omitted as we can simply rewrite the union operator according to De Morgan's law.

Another proof is to be made for the logical states in NFA: as per non-deterministic case they are remained without changes and are to be addressed for the static matching process without closure evaluation [2].

Conclusion. Thus, we obtained the linear polynomial results by applying De Morgan's law in re-writing rules. This gives us opportunity to build NFA-based engines for extended regular expressions as they work optimally and aren't of perfect fit for other problems which are not

omitted in the presented algorithm – we gain all the power of NFA in membership problem by matching.

The author lefts not opened questions upon the effective linear matching of extended operators in RE, however, there could be other open problems like implementation the specific features in the same time.

For the practical purposes we have also developed the Java version of regular expression engine based on the algorithm described in this article. This program can be obtained upon the request.

Acknowledgements

The author expresses gratitude to Dr. Steven Kearns for co-operation on finishing this project and giving important and valuable notions – without his support and interest it wouldn't be possible to obtain the highest results described in this work.

Funding

This work was partially supported by an educational grant of the Ministry of Education and Sciences of Republic Kazakhstan during author's work in 2006-2009 at the Institute of Problems in Informatics and Control.

References

1. Roşu, Grigore. "An effective algorithm for the membership problem for extended regular expressions." *International Conference on Foundations of Software Science and Computational Structures.* Springer, Berlin, Heidelberg, 2007.

2. Syzdykov, Mirzakhmet. "Deterministic automata for extended regular expressions." *Open Computer Science* 7.1 (2017): 24-28

3. Thompson, Ken. "Programming techniques: Regular expression search algorithm." *Communications of the ACM* 11.6 (1968): 419-422.

4. Rabin, Michael O., and Dana Scott. "Finite automata and their decision problems." *IBM journal of research and development* 3.2 (1959): 114-125.

5. Gelade, Wouter, and Frank Neven. "Succinctness of the complement and intersection of regular expressions." *ACM Transactions on Computational Logic* (TOCL) 13.1 (2012): 1-19.

6. Yang, Yi-Hua E., and Viktor K. Prasanna. "Space-time tradeoff in regular expression matching with semi-deterministic finite automata." *2011 Proceedings IEEE INFOCOM*. IEEE, 2011.

7. Syzdykov, Mirzakhmet. "Algorithm to Generate DFA for AND-operator in Regular Expression." *International Journal of Computer Applications* 975 (2015): 8887.

8. Syzdykov, Mirzakhmet. "Methodology to Produce Deterministic Automaton for Extended Operators in Regular Expression." *International Journal of Scientific & Engineering Research* 8 (2017): 1497-1500.

9. Berry, Gerard, and Ravi Sethi. "From regular expressions to deterministic automata." *Theoretical computer science* 48 (1986): 117-126.

10. Hsieh, Samuel. “Product Construction of Finite-State Machines.” *Proceedings of the World Congress on Engineering and Computation Science*, Vol. I (2010): 141-143.

11. Møller, Anders. “dk.brics.automaton – finite-state automata and regular expressions for Java,” 2021. http://www.brics.dk/automaton/ (accessed August 22, 2021).