Methodology to Produce Deterministic Automaton for Extended Operators in Regular Expression

Mirzakhmet Syzdykov, mirzakhmets@gmail.com, Alma-TV LLP

Abstract— Recent and past work for extended operators (AND, MINUS) in regular expression was covered only by Berry-Sethi. However, the estimation of complexity wasn't given (in fact the method produces non-deterministic automaton, NFA). In this paper the methodics known as "overriding" is presented for this task. This methodics uses semantic rules overriding the typical NFA to produce DFA (deterministic finite automaton).

Index Terms— automata theory; pattern recognition; soft computing; subset construction; computational intelligence; extended regular expression; deterministic finite automaton.

1 Introduction

First the regular expressions are studied, then the Thompson [1] method is introduced to produce NFA from DFA.
Similarly, Berry-Sethi algorithm [2] can produce NFA, for which the semantic rules are to be applied in order to build DFA:

NFA == NFA x [Semantic Rules]
→ DFA (for extended operators) (1)

2 Regular Expressions

Regular expressions can be defined as a set of rules to describe the *regular language*. It's known that these languages can be either finite or inifinite. It's also known that regular expressions (R and R[i]) are defined over some alphabet A. This alphabet defined the set of letters from which the words are constructed in the regular language. In [3] the definition of regular expressions is given. Here only the subset of this expressions is studied, in more words the operators AND and MINUS (along with NOT-operator) are described:

R = R[1] & R[2] (a set of words L(R):
L(R) = L(R[1]) L(R[2])) - ***AND****;* (2)

~R = A– R -* ***NOT****;* (3)

R = R[1] – R[2],L(R) = {
w: w L(R[1]) & w ! L(R[2])}) - ***MINUS****.* (4)

3 Known Algorithms

The Thompson algorithm [1] for building NFA from regular expression is well-known and was studied in deep for the past half of century. It has many practical applications and is very simple for understanding and extension. In [4] the alternative automaton (known as a Glushkov automaton) was well-studied and characterized, obviously, Glushkov automaton can be constructed from Thompson's by applying the *follow-rule*:

Glushkov.NFA =
[Follow Semanitcs] (Thompson.NFA) (5)

This case has one more proof of Thompson's method universality and applicability even for large cases as automaton has as fewer states as the input regular expression. For the follow automaton this bound is quadratic due to Kleene-star explosion.
Another interesting approach of constructing automaton from regular expression is based on derivatives [5]. The derivative is an operator to derive from regular expression and, thus, recursively to build an automatonn (either non-deterministic or deterministic). Berry and Sethi in [2] showed that derivatives can be applied to the application of regular expression as a state in automaton. Thus, the whole regular expression products more expressions according to deviation operator. This approach, however, has a low-bound estimation. Also, in [2] the deviation was introduced for extended operators (AND, MINUS and NOT).

4 Subset Construction

The method described in this paper uses subset construction to convert overriden NFA into DFA with respect to the semantic rules due to which the complexity expands. In [6] the theoretical background and a wise-theorem were proven so that the NFA can be converted to DFA. The method described differs only in using the overriden operators to construct the DFA in state-space of NFA. Obviously, the state-space of construction NFA-DFA is regulated by semantic rules.

5 Extended Operators

For intersection operator, as it was proposed in [3], the additional semaphore is introduced which is represented by activators of variable degree. These activators are actually filters

required to model the boolean state space traversal, while the noise is overridden in algorithm traversing this structure during subset construction, when subsets of NFA states are replaced by single representative state in the artificial deterministic-finite automaton (DFA) to be constructed.
This would be better written as:

*Algorithm -> State.Space * Filters.Activators^2* (6)

From programming point of view the activator can be modeled during empty transitions removal or compression as:

```
if (state2.AndCount > 0) --state2.AndCount;
if (state2.AndCount == 0)
{
stack.Push(state2);
}
```

The NFA construction for AND-operator can be better examined below:

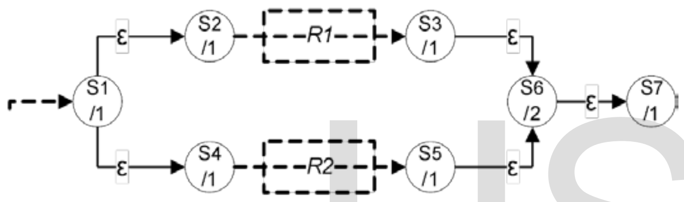

Fig. 1. NFA-construction for the expression “R[1]&R[2]” (counter values after “/”-sign)

The experimental benchmark and estimation data can be examined below. In this cases the regular expression “(((a|b)*a(a|b)*)&((a|b)*b(a|b)*))” is used as a sample which will be repeated by concatenating either by product or AND-operator.

TABLE 1
BENCHMARK DATA SET

	Concatenated		Concatenated by AND-operator	
Repeat Count	Running Time (sec.)	Number of States	Running Time (sec.)	Number of States
1	0,013	9	0,05	9
2	0,024	15	0,22	9
4	0,148	27	0,068	9
8	1,082	51	0,319	9
16	7,924	99	1,268	9

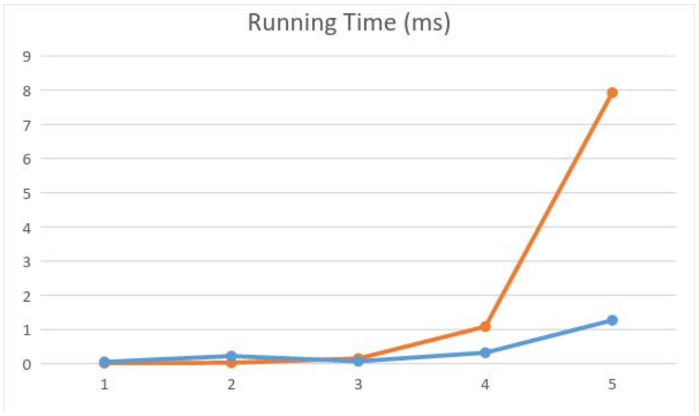

Fig. 2. Running Time plot

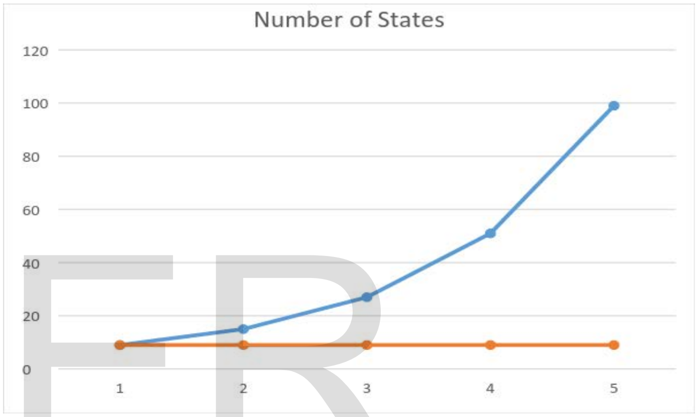

Fig. 3. States count plot

Obviously, the results converge to the almost linear model.
In [7] the methodics for MINUS- and NOT-operator was presented. For this purpose the event-driven model is simply used. Thus, the minus operator can be refactored to two events:

word in L(R[1]) AND word not in L(R[2]) (7)

It would be better understood from programming point of view:

```
if (state2.VisitIndex != Tuple.VisitIndex)
{
state2.VisitIndex = Tuple.VisitIndex;
if (state2.Type == C_NFA_State_Type.AND)
  state2.AndCount = 2 - 1;
else if (state2.Type == C_NFA_State_Type.MINUS)
{
  if (!state1.out_edges[i].IsNegatedPart)
  {
    stack.Push(state2);
  }
}
else
{
  stack.Push(state2);
}
}
else
{
 if (state2.Type == C_NFA_State_Type.AND)
 {
  if (state2.AndCount > 0) --state2.AndCount;
  if (state2.AndCount == 0)
  {
   stack.Push(state2);
  }
 }
}
```

Let's study the example outputs of presented algorithms:

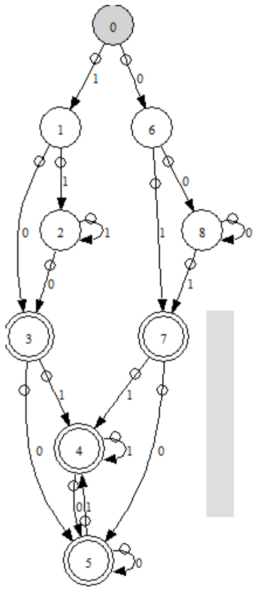

Fig.4.DFA for expression "(((0|1)*0(0|1)*)&((0|1)*1(0|1)*))"

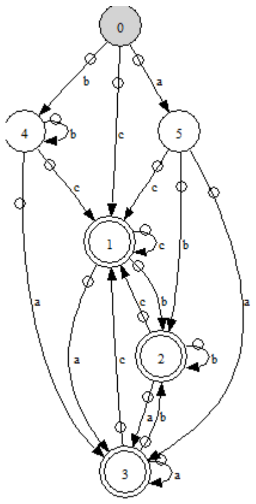

Fig.5. DFA for expression "(a|b|c)*-(a|b*)"

6 Comparative Study

In [7] the experimental study was proposed for extended regular expressions. Algorithm differs from that presented in this paper in fact that it uses derivative trees and optimization technique for rewriting the regular expression in some canonical form. Our algorithm produces similar results for sample expression "~(0~01)1", shown on Figure 6.

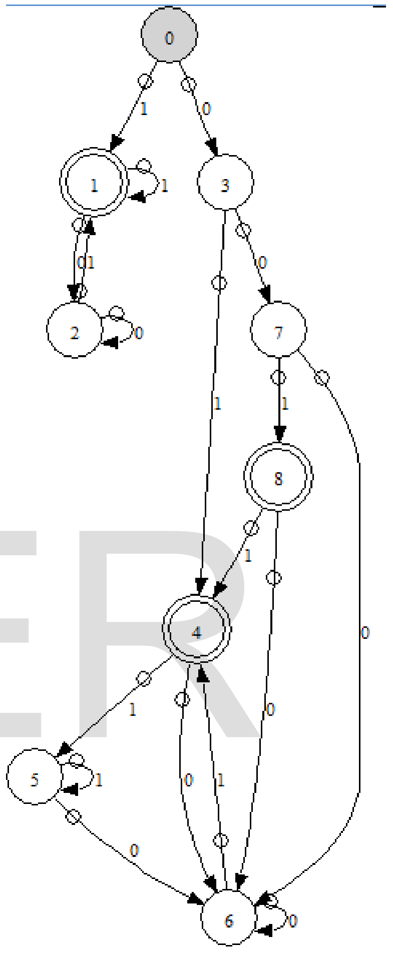

Fig.6. Sample result

6 Conclusion and Further Work

In [3] the conclusion ends as:
"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 opera-

tor, including the methods to use them in order to build the final DFA. ”
In this work the results of the theoretical continuation of [8] are presented:

1. The activators and events can override the NFA;
2. The semantic rules can be used for DFA construction by subset-methodics;
3. The semantic ruels can be extended for specific case to traverse only selected region of state-space.
4. State-space is, by definition, a Boolean which can be studied from the noise paradigm [8], where the sought state-space region is reduced by implementing control vectors production operator.

The further work consists of generalization of experimentally obtained results for better studying of noise in computer science.

ACKNOWLEDGMENT

The author of this article expresses gratitude to the members of Institute of Informational and Computational Technologies (Almaty, Republic of Kazakhstan, http://www.ipic.kz): Mamyrbayev Orken Zhumazhanovich, Kapalova Nursulu Aldazharovna and Yunicheva Nadiya Rafkatovna for their work in organizing the conference, devoted to the 25th anniversary of the establishment of the institute

REFERENCES

[1] K. Thompson, “Regular expression search algorithm,” Comm. ACM, 11 (6), pp. 419–422, 1968.
[2] G. Berry, R. Sethi, “From regular expressions to deterministic automata,” Theoretical Computer Science, 48, pp. 117-126, 1986.
[3] M. Syzdykov, “Algorithm to Generate DFA for AND-operator in Regular Expression,” International Journal of Computer Applications (0975-8887), Volume 124 - No. 8, pp. 31-34, 2015.
[4] Pascal Caron, Djelloul Ziadi, “Characterization of Glushkov automata,” Theoretical Computer Science, 233, pp. 75-90, 2000.
[5] Brzozowski, Janusz A, “Derivatives of regular expressions,” Journal of the ACM, 11(4), pp. 481– 494, 1964.
[6] M.O. Rabin, D. Scott, “Finite automata and their decision problems,” IBM J. Res. Develop., 3 (2), pp. 114– 125, 1959.
[7] Koushik Sen, Grigore Rosu, “Generating optimal monitors for extended regular expressions,” Electronic Notes in Theoretical Computer Science 89 No. 2, pp. 226 – 245, 2003.
[8] M. Syzdykov, “Theory of Noise with Applications: A Practical and Theoretical Guide,” Lambert AP, pp. 46-54, 2016.