EFFECTIVE IMPLEMENTATION OF WORD-BASED REGULAR EXPRESSIONS NOTATION IN NATURAL LANGUAGE PROCESSING

Denis Postanogov

ScienceSoft, Minsk, Belarus, e-mail: *dpostanogov@scnsoft.com*

Abstract. This paper introduces a technique that allows to build deterministic finite-state automata from word-based regular expressions that are described in [1]. The automata obtained by this technique can be used in automatic analysis of natural language sentences without any loss of algorithmic deterministic automata efficiency. The paper shows the difference between traditional and word-based regular expressions and explains the need for additional transformation of word-based regular grammar in order to build deterministic finite-state automaton.

Introduction

Nowadays apparatus of regular expressions (RE) and finite-state automata (FSA) serve as the most popular and universal tool for solving tasks of automatic text processing [5]. In fact, regular expressions provide sufficiently comfortable technique of developing alphanumerical data processing rules of different types [3]. Finally, rules, grammars and grammar parts written in notation of regular expressions can be transformed to the deterministic finite-state automata (DFSA) of certain type that depends on the purpose of grammar rules. DFSA apparatus allows to efficiently apply grammar rules to alphanumerical data, taking linear O(n) time, where n is the length of the input sequence [2, 4].

However, different stages of solving automatic natural language processing tasks require taking into account more subtle features and criteria rather than just a symbolic representation of text elements. Word-based regular expressions (WRE) apparatus can be used as an appropriate and unified notation for writing grammar rules for processing sentences and other structures that have words or word phrases as their basic elements [1]. This paper introduces a technique that allows to transform word-based regular expressions to deterministic finite-state automata of different types. The automata obtained by this technique can be used for processing natural language sentences (word sequences). On this basis a deeper and more detailed automatic analysis of text data can be accomplished without any loss of DFSA algorithmic efficiency.

Rationale of additional transformation of WRE

Unlike traditional regular expressions that expect character-based input strings, WRE notation uses word-based *tokens*, alphabet symbols that constitute an word-based alphabet. Each of these tokens is a complex structure that doesn't obligatorily show a one-to-one correspondence with the elements of input sequence.

As shown in the examples of Table 1 one WRE token can correspond to more than one words of the input sentence. And vice versa, a single word from the input sentence can correspond to a set of tokens from WRE. This implies that in the general case DFSA for the word-based regular grammar cannot be built without additional transformations.

#	*WRE tokens*	*corresponding word references*
1	*NN*	*a b*
2	*"water"*	*a*
3	*'[a-z]+'*	*a b c*
4	*NN*	*a b c d*

№	*POS-tagged words*	*corresponding token references*
a	water_NN	1 2 3 4
b	gas_NN	1 3 4
c	good_JJ	3 4
d	re-emphasis_VB	4

Table 1 Ambiguity of correspondence between WRE tokens and POS-tagged words as elements of input chain

To prove it, let's consider the following example of WRE:

'[a-z]+' '[0-9]+' | "water" NN (1)

If we try to build DFSA for this expression applying any algorithm of DFSA building used for traditional regular expressions, we'll get the following Rubin-Scott automaton shown in *Fig.*1:

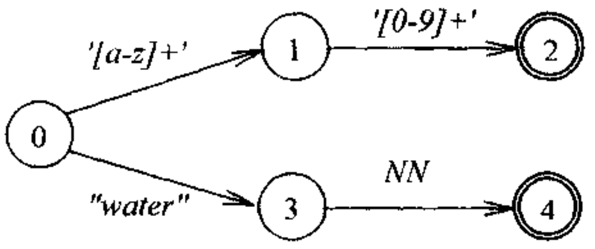


Fig. 1. Automaton built from WRE using traditional technique.

An attempt to apply this automaton to the following input sequence of part of speech-tagged (POS-tagged) words

water_NN pump_NN (2)

shows that such automaton is not *deterministic*. The POS-tagged word 'water_NN' corresponds to the automaton transition from the initial state 0 to the state 1. Then the word 'pump_NN' doesn't have an exit from the state 1. That is, in this case the given word sequence is not admitted by the automaton. At the same time, the word 'water_NN' corresponds to the automaton transition 0→3. This means that for a correct analysis of the input sequence it is necessary to maintain a set of the current automaton states on each iterative step. In other words, it means that the automata built can only be used as non-deterministic which results in a significant loss of text analysis effectiveness.

In the next paragraph we suggest a method that allows to transform the given WRE to the regular language that can be used as the basis of DFSA building.

Expansion of word-based regular expression alphabet

Let's assume that $\Sigma = T_1, T_2, \ldots, T_n$ is the alphabet of tokens of the given WRE.

Let's denote by *Words*(T) the set of tagged words that corresponds to the token T. In a diagram each token T can be represented as a circle that is in fact a set of words *Words(T)*. Tokens *NN* and *"water"* will meet the correlation shown in the diagram (see *Fig.*2).

The intersection of the circles in the above diagram corresponds to the word 'water' that is tagged as NN, i.e. it shows the token *"water"/NN*.

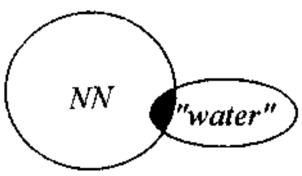


Fig. 2. Intersection of word sets corresponging to the tokens NN and "water".

Let's consider the token T that describes a set of tagged words *Words*(T) = *Words*(T_i) ∩ *Words*(T_j). This token will be called *intersection* of tokens T_i and T_j. While building the intersection one should take into account that the intersection of certain pairs of tokens may result in an empty set of words. For instance, for the simple word tokens "water" and "gas" or POS-tag tokens NN and JJ the intersection results in an empty set. In this case we will say that the intersection of tokens doesn't exist.

Let Σ' be an alphabet that consists of all possible intersections of tokens from the alphabet Σ. This alphabet Σ' will be called the *expanded alphabet* of WRE. We can say that the expanded alphabet describes all the sets that correspond to the intersections in a diagram. Here the tokens of the alphabet Σ form the intersections with themselves so they are included in the resulting expanded alphabet. We should also notice that the expanded alphabet will consist of $2^n - 1$ tokens at most, where n is the number of tokens of the source alphabet Σ.

Let's define on the expanded alphabet of WRE the ordering relationship that meets the following rule: any token from the expanded alphabet that is results from the intersection of tokens T_i and T_j, has the index k, where $k \leq j$ и $k \leq i$. The expanded alphabet Σ" having such linear ordering relationship is called *ordered expanded alphabet* of WRE.

We propose to use the following algorithm in order to build the ordered expanded alphabet Σ" from the given source WRE alphabet Σ.

Let's assume that target expanded alphabet Σ" is an ordered list structure that supports the operation of element insertion. Then we can use the following procedure.

i. Put special token ∀ that is called "any word" into Σ". We should notice that intersection of any token T with ∀ will gives T.

ii. If there are no unprocessed tokens left in Σ, stop. The ordered set Σ" can be obtained by numbering the list elements in the order they occur.

iii. Otherwise, choose any unprocessed token T from Σ.

iv. Try to get the intersection of token T with all existing tokens T_j from Σ" one by one starting from the lowest j. If the intersection T ∩ T_j exists then it is inserted in the list of tokens of Σ" before the token T_j.

v. Go to step ii.

To illustrate the algorithm, let's consider the alphabet of WRE tokens for rule (1):

Σ = { *'[a-z]+'*, *'[0-9]+'*, *"water"*, *NN* }.

0). Σ" = { ∀ }.

1). Choose token *'[a-z]+'* from Σ. Try to intersect it with the first and single Σ" token ∀. The result of intersection is equal to *'[a-z]+'*. Insert it into Σ" before ∀.

Σ" = { *'[a-z]+'*, ∀ }.

2). Choose next unprocessed token *'[0-9]+'* from Σ. Intersect it with the tokens from Σ" step by step:

'[0-9]+' $\cap$ *'[a-z]+'* $= \varnothing$ (intersection doesn't exist);
'[0-9]+' $\cap \ \forall =$ *'[0-9]+'*;
$\Sigma' = \{$ *'[a-z]+'*, *'[0-9]+'*, $\forall \}$.
3).*"water"* $\cap$ *'[a-z]+'* $=$ *"water"*;

Since we got "water" as a result of intersection we don't need to proceed with the intersection process for other tokens from Σ'' because tokens that are more narrow can not be inserted after the token "water".

As a result, $\Sigma'' = \{$ *"water"*, *'[a-z]+'*, *'[0-9]+'*, $\forall \}$.

4) Choose the last token NN from alphabet Σ. Its intersections are:
NN $\cap$ *"water"* $=$ *"water"/NN*;
NN $\cap$ *'[a-z]+'* $=$ *'[a-z]+'/NN*;
NN $\cap$ *'[0-9]+'* $=$ *'[0-9]+'/NN*;
NN $\cap \ \forall =$ *NN*;
$\Sigma'' = \{$ *"water"/NN*, *"water"*, *'[a-z]+'/NN*, *'[a-z]+'*, *'[0-9]+'/NN*, *'[0-9]+'*, *NN*, $\forall \}$.

As a result we have the following ordered expanded alphabet:

$\Sigma'' = \{$ **1**. *"water"/NN*, **2**. *"water"*, **3**. *'[a-z]+'/NN*, **4**. *'[a-z]+'*, **5**. *'[0-9]+'/NN*, **6**. *'[0-9]+'*, **7**. *NN*, **8**. $\forall \}$. (3)

Let T be a token from the source WRE alphabet Σ. Token T_k from the ordered alphabet Σ'' is called *child* of the token T if it either corresponds to the token T itself or it is obtained as a result of the intersection of any child of T with some other token (recursive definition). The diagram shows that the children of T agree with the intersections that completely belong to *Words*(T) set.

Let's denote by *Children*(T) a set of indices of children of T from the ordered alphabet Σ''. Table 2 shows the values of function *Children*() for the tokens from (3).

T	*Children*(T)
'[a-z]+'	1, 2, 3, 4
'[0-9]+'	5, 6
"water"	1, 2
NN	1, 3, 5, 7

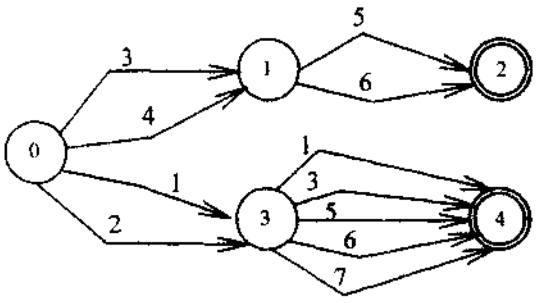
Table 2 Values of children() function for tokens from (1)

In order to transform the original WRE to the regular language for building DFSA, we replace each token T with disjunction (OR) of numbers from Children(T). In such a way WRE from example (1) will be transformed to the following regular expression:

$$(1 \mid 2 \mid 3 \mid 4) (5 \mid 6) \mid (1 \mid 2) (1 \mid 3 \mid 5 \mid 7).$$

At further steps this regular expression will be transformed to the following DFSA (see *Fig.3*).

DFSA built can be used to parse sentences on the basis of a set of source WRE rules according to the following technique. Each POS-tagged word W from the sentence being processed should be converted to the smallest number j of the token T_j from the expanded alphabet Σ'', where T_j: $W \in \mathrm{Words}(T_j)$.



(4)

Fig. 3. Deterministic finite-state automaton of transformed WRE.

In order to instantiate the parsing of sentence with the automaton built, let's consider phrase (2). The smallest token that admits the POS-tagged word *water_NN* from (3) is a token *"water"/NN* number 1. The word *pump_NN* doesn't admit the first and second tokens, but admits token *'[a-z]+'/NN* number 3. Thus, the word sequence (2) will be transformed into the input sequence of weight "1, 3" of the automaton (4). As can be seen, this type of input is admitted by the automaton.

Conclusion

This paper describes the techniques that are used to expand the WRE alphabet as well as to transform the source WRE. The techniques mentioned allow to apply the DFSA apparatus to the analysis of tagged sentences using WRE notation that considers various complex characteristics of sentence words (i.e. dictionary information, spelling and others). In this case it is possible to achieve a complexity of sentence analysis that is estimated as $O(n\cdot(l+d))$, where n is the number of words in the sentence, l is the average length (number of letters) of the word and d is the average number of tags in the dictionary for one word. The above mentioned technique of WRE alphabet transformation and expansion can also be applied to the processing of other regular structures that present an ambiguity with respect to the correlation of input sequence elements with the terminal symbols.

References

[1] A. Cheusov. Word-based regular expressions in NLP tasks. *Proceedings of the I International conference INFORMATIONAL SYSTEMS AND TECHNOLOGIES'02*, Minsk 2002.

[2] A. Gill. Introduction to the Theory of Finite-state Machines, *McGraw-Hill Book Company*, New York, 1962

[3] M. Gross and A. Lentin. Notion sur les Grammaires Formelles. *Gauthier-Villars*, Paris, 2-ème edition, 1970.

[4] J.E. Hopcroft and J.D. Ullman. Introduction to Automata Theory, Languages and Computation. *Addison Wesley, Reading*, Massachusetts, 1979.

[5] Jurafsky D., Martin J.H. (2000). Speech And Language Processing. – *Prentice Hall*, Upper Saddle River, New Jersey 07458.