International Journal of Database Theory and Application
Vol.8, No.4 (2015), pp.241-250
http://dx.doi.org/10.14257/ijdta.2015.8.4.24

Intersection Checking for Regular Expressions Based on Inference System

Jia Liu[1] and Husheng Liao[2]

College of Computer Science, Beijing University of Technology, Beijing, China
[1]jeromeliu2006@gmail.com,[2]liaohs@bjut.edu.cn

Abstract

Decision problem of intersection checking for regular expressions plays an important role in the XML type checking. The typical technique is converted into the problem of automata intersection, which may generate a lot of redundant computing during the conversion. In the present paper, according to the features of XML schema languages, a new intersection checking algorithm based on inference system for regular expressions is proposed. This method is derived directly based on regular expression without the need for any conversion. For general regular expressions that is exponential time algorithm, but without constructing automata and for some special cases, especially for the one-unambiguous regular expressions used in XML type checking, is the polynomial time algorithm. Proofs of the correctness and completeness of the inference rules are given. Experiment results show that our approach are more effective than automatic approach in practical.

Keywords: *XML type checking; regular expression; intersection checking; inference rules.*

1. Introduction

In the past decades, relational database [1] has held the dominant position in the area of information management system by virtues of its rigorous mathematical foundation, simple structured data model and non-procedural query language. However, with the rapid development of web applications such as social networking and e-commerce, traditional relational databases are increasingly difficult to deal with massive semi-structured data on the web. The Extensible Markup Language [2] (XML) proposed by W3C is designed to meet this challenge and has become the universal format for semi-structured data representation of large-scale electronic publishing and information exchange over the Internet. Usually in applications XML data are provided with schemas that the XML data must conform to. A cluster of XML schema languages based on regular tree grammar[3] have been developed, such as the oldest Document Type Definition[2] (DTD), the XML Schema[5] (XSD) described by the W3C as the successor of DTDs, the RELAX NG[8] as an ISO/IEC International Standard, and so on[4,6,7,9]. These schemas are important for improving efficiency in many XML processing tasks such as data query, data integration and type checking. Different extensions of regular expressions are used to describe the content models for modeling the data structures of document elements in the major XML schema languages mentioned above. Therefore, the important decision problems of schema languages which are core operations in the XML type checking, just like inclusion, equivalence, and non-emptiness of intersection, can be easily transformed into the corresponding decision problems of regular expressions.

In the present paper, we investigate the non-emptiness of intersection problem for regular expressions used in XML schema languages. The input to the problem consists of two expressions, and the question is whether there is at least one non-empty string that belongs to both the languages of two input expressions. The classical algorithm starts with

ISSN: 2005-4270 IJDTA
Copyright ⓒ 2015 SERSC

constructing non-deterministic finite automata for each of the expressions, and then constructs DFAs from those NFAs. Then a DFA recognizing the intersection of the languages of the two input expressions is constructed. Finally, the algorithm checks that whether there is at least one final state is reachable in the DFA. In this computation process, super-polynomial blowup occurs when constructing a DFA from the NFA, and the other steps are polynomial time. Usually the regular expressions used in content models of XML schema languages are restricted by a requirement of determinism, which means that a parser recognizing XML document element contents has to be able to decide without look ahead, which content model token to match with the current input token, while processing the document from left to right. Deterministic regular expressions were first formalized and studied by Brüggemann-Klein and Wood[10], and they called determinism more precisely one-unambiguity. Paper[10] also shows a polynomial time construction of DFAs from one-unambiguous regular expressions, therefore, the classical algorithm can be modified to solve the intersection problem in polynomial time when the input expressions are one-unambiguity. No matter whether one-unambiguous regular expressions or not, the classical algorithm need to construct finite automata. For some noone-unambiguous regular expressions, the DFA constructed by classical algorithm is super-polynomial size, and this has been proven theoretically by Myhill and Nerode [12].This paper presents an alternative algorithm for intersection problem of regular expressions. The new algorithm is based on a syntax-directed inference system and does not need to construct finite automata. Another advantage of this new algorithm is that only treats the parts of expressions which are necessary and automatically ignore useless parts of expressions. If the useless parts of expressions correspond exactly super-polynomial size DFAs, the process may therefore just need polynomial-time instead of super-polynomial time.

The main contributions of the paper are as follows: We propose a new algorithm based on inference system for intersection problem of regular expressions. Firstly, this algorithm has a wide range of adaptability and can both deal with one-unambiguous regular expressions or ordinary regular expressions. For ordinary regular expressions, the complexity of this algorithm is super-polynomial time, and for one-unambiguous regular expressions, is polynomial time. For some specific non one-unambiguous regular expressions, this algorithm still terminates in polynomial time. Finally, the new algorithm is derived directly based on regular expression without the need for any conversion of finite automata.

Section 2 introduces preliminary definitions. The inference system is presented in Section 3. Section 4 discusses some properties of inference system. Section 5 describes the new algorithm which based on the inference system described in Section 3. Section 6shows the experiment results, Section7discusses some related works and Section 8contains a conclusion.

2. Definitions

Let Σ be an alphabet of symbols.Assume a , b , and c are members ofΣ.$l, l_1, l_2 \cdots$areused as variables for numbers of Σ.

Definition 1. Regular expression. The regular expression over alphabetΣ are denoted $R_\Sigma$and defined in the following inductive manner:$R_\Sigma \coloneqq R_\Sigma{+}R_\Sigma | R_\Sigma \cdot R_\Sigma | R_\Sigma^* | \Sigma$. Thesemantics of regular expressions is defined in terms of sets of words over the alphabetΣ. The operation union$+$ means if $L_1, L_2 \subseteq \Sigma^*$, then $L_1 + L_2 = \{w_1 \cup w_2 | w_1 \in L_1, w_2 \in L_2\}$. Similarly, operation concatenation $\cdot$ means if $L_1, L_2 \subseteq \Sigma^*$, then $L_1 \cdot L_2 = \{w_1 \cdot w_2 | w_1 \in L_1, w_2 \in L_2\}$, and the star $*$ means the Kleene closure of $L \subseteq \Sigma^*$. Operation$*$ has the highest priority, $\cdot$ followed and $+$ has the lowest priority. To avoid ambiguity, parentheses can be added to the expressions such as $(R_\Sigma + R_\Sigma)$

 Copyright ⓒ 2015 SERSC

and $(R_\Sigma \cdot R_\Sigma)^*$. The concatenation $\cdot$ will often be omitted. We use $r, r_1, r_2, \cdots$ as variables for regular expressions. Notice that we do not allow the empty symbol ϵ in the regular expressions just like the classical definition. That is the most important difference between our definition and others definition. In fact, it is easy to proof that the regular expression without ϵ semantically equivalent to the regular expression with ϵ except to not match an empty string. In this paper, we do not consider the intersection problem with empty string, because there is no practical meaning and will add unnecessary complexity to the algorithm. But we still use the symbol ϵ in the following reference rules, but it just means the termination of regular expression and therefore can only appear in the end of a regular expression.

Definition 2. Language of a regular expression. The language of a regular expression r is denoted $\| r \|$ and is defined by the following inductive rules: $\| r_1 + r_2 \| = \| r_1 \| \cup \| r_2 \|$, $\| r_1 \cdot r_2 \| = \| r_1 \| \cdot \| r_2 \|$, $\| r^* \| = \cup_{0 \le i} \| r \|^i$ and for $a \in \Sigma$, $\| a \| = \{a\}$. The decision problemof non-emptiness of intersection for regular expressions is that whether there is at least one non-empty string$v$, such that$v \in \| r_1 \| \wedge v \in \| r_2 \|$.

Definition 3.1-Unambiguousregular expression. To define the 1-unambiguous regular expression, we first introduce the concept of marked expression. For a regular expression we can mark symbols with subscripts so that in the marked expression each marked symbol occurs only once. For example $(a_1 b_1)^* a_2 b_2 (a_3 + b_3)$ is a marking of the expression$(ab)^* ab(a+b)$. The marking of an expression r is denoted byr_{mark}. We extend the notion for string in the obvious way. A regular expression r is one-unambiguousregular expression if and only if, for all string $uxv, uyv \in \| r_{mark} \|$, where $x, y \in \Sigma$, if $x \neq y$, then $\lambda(x) \neq \lambda(y)$, where the function λ means of unmarkingsubscripts of a string. Examples of one-unambiguousregular expression are $aa^*$and $b^* a(b^* a)^*$, while $a^* a$ and $a^*(b^* a)^* b$ are not one-unambiguous regular expressions.

3. Rules for Intersection

For the convenience of derivation process, we add terminator symbol $\epsilon = \epsilon^* | \epsilon^l$ to the ends of all expressions appeared in all the reference steps. Theterminator is just mark the end of an expression, and does not affect the semantics of regular expressions,that is to say$\| r \cdot \epsilon^* \| = \| r \cdot \epsilon^l \| = \| r \|$.We add the terminator $\epsilon^*$to the input expression r_1 and $r_2$of the derivation process, which creates new input regular expressions $r_1' = r_1 \cdot \epsilon^*$ and$r_2' = r_2 \cdot \epsilon^*$.The function $to\epsilon^l$ defined as $to\epsilon^l(r_1 \cdots r_n \cdot \epsilon) = r_1 \cdots r_n \cdot \epsilon^l$. Actually, $to\epsilon^l$ just replaces the terminator from ϵ^* to ϵ^l, and does also not affect the semantics of regular expressions. The sematic of symbol F appeared in the derivation is that no string s will be find according to the derivation from$r_1$and $r_2$to$F$, such that$s \in \| r_1 \| \wedge s \in \| r_2 \|$. On the contrary, The sematic of Tappearedin the derivation is that at least one string s will be find according to the derivation from$r_1$and r_2 to T, such that $s \in \| r_1 \| \wedge s \in \| r_2 \|$. The seven rules used in the derivation are showed as follows:

Rule $\epsilon 1$: $\frac{\epsilon \cap l \cdot r}{F}[\epsilon = \epsilon^* \; or \; \epsilon^l]$

Rule $\epsilon 2$: $\frac{\epsilon \cap \epsilon}{F}[\epsilon = \epsilon^*]$

Rule $\epsilon 3$: $\frac{\epsilon \cap \epsilon}{T}[\epsilon = \epsilon^l]$

Rule $l1$: $\frac{l_1 \cdot r_1 \cap l_2 \cdot r_2}{F}[l_1 \neq l_2]$

Rule $l2$: $\frac{l_1 \cdot r_1 \cap l_2 \cdot r_2}{to\epsilon^l(r_1) \cap to\epsilon^l(r_2)}[l_1 = l_2]$

Rule $+$: $\frac{r_1 \cap (r_2 + r_3) \cdot r_4}{r_1 \cap r_2 \cdot r_4 \vee r_1 \cap r_3 \cdot r_4}$

Rule $*$: $\frac{r_1 \cap r_2^* r_3}{r_1 \cap r_2 \cdot r_2^* r_3 \rightarrow r_1 \cap r_3}$

We give some necessary explanations for the above inference rules. Each rule consists of a horizontal line with a conclusion below it, and a premise above the line. All rules but rule +and rule∗also have side conditions in square brackets. We only allow rule instances where the side conditions are satisfied. These seven rules can be divided into three groups according to the priority. The first five rules have the highestpriority, the rule + followed and the rule ∗ is the lowest. The symbol∨ in the conclusion of rule + denotes the logic relation *inclusive-or* between the two sub-expressions of the conclusion, that is to say the two expressions are independent of each other. The symbol→ in the conclusion of rule ∗ indicate that the former sub-expression $r_1 \cap r_2 \cdot r_2^* r_3$ must be inferred first, then we infer the lattersub-expression $r_1 \cap r_3$because the pair of conclusion are not independentof each other. More details about this will discuss in section 4. To simplify the inference process, we can extend the inference rules as follows:

Rule ϵ^* : $\frac{\epsilon^* \cap r}{F}$

Rule $l+$: $\frac{l \cdot r_1 \cap (r_2 + r_3) \cdot r_4}{l \cdot r_1 \cap r_2 \cdot r_4 \vee l \cdot r_1 \cap r_3 \cdot r_4}$

Rule $l*$: $\frac{l \cdot r_1 \cap r_2^* \cdot r_3}{l \cdot r_1 \cap r_2 \cdot r_2^* \cdot r_3 \vee l \cdot r_1 \cap r_3}$

Rule $++$: $\frac{(r_1 + r_2) \cdot r_3 \cap (r_4 + r_5) \cdot r_6}{r_1 \cdot r_3 \cap r_4 \cdot r_6 \vee r_2 \cdot r_3 \cap r_4 \cdot r_6 \vee r_1 \cdot r_3 \cap r_5 \cdot r_6 \vee r_2 \cdot r_3 \cap r_5 \cdot r_6}$

4. Properties of the Rules

Theorem 1. The rules are sound.

Proof. For rule $\epsilon 1$. Because the *firstset* of $l \cdot r$ is$l \in \Sigma$, so the beginning characterfor any string $s \in \| l \cdot r \|$must be$l$. Without loss of generality, we assume $r_1 = r_1' \cdot \epsilon$ and $r_2 = r_2' \cdot l \cdot r + r_2''$. Then for any $s' \in \| r_1' \| \wedge s' \in \| r_2' \|$, we have $s' \cdot s \in \| r_2' \cdot l \cdot r \| \wedge s' \cdot s \notin \| r_1' \cdot \epsilon \|$. So the conclusion of rule $\epsilon 1$ is F.

For rule $\epsilon 2$. The premise of this rule is $\epsilon^* \cap \epsilon^*$. Let $r_1 = r_1' r_2' \cdots r_n' \cdot \epsilon^*$ and $r_2 = r_1'' r_2'' \cdots r_m'' \cdot \epsilon^*$. The rule $l2$ must not be used in the process of derivation from $r_1 \cap r_2$ to $\epsilon^* \cap \epsilon^*$, becausethe end symbol ϵ^* of a string can be changed into $\epsilon^l$by the function $to\epsilon^l$which only used in rule $l2$. Therefore only rule $*$ can be used in the process of eliminating $r_1', r_2', \cdots, r_n'$ and $r_1'', r_2'', \cdots, r_m''$. So we cannot find a sting s, such that $s \in \| r_1 \| \wedge s \in \| r_2 \|$ in the derivation from expression r_1,$r_2$to $\epsilon^* \cap \epsilon^*$. So the conclusion of rule $\epsilon 2$ is F.

For rule $\epsilon 3$. The premise of this rule is $\epsilon^l \cap \epsilon^l$. Let regular expressions $r_1 = r_1' r_2' \cdots r_n' \cdot \epsilon^l$ and $r_2 = r_1'' r_2'' \cdots r_m'' \cdot \epsilon^l$. Then at least once rule $l2$ used in the derivation from $r_1 \cap r_2$ to $\epsilon^l \cap \epsilon^l$, because the end symbol ϵ^* of a string can be changed into $\epsilon^l$by the function $to\epsilon^l$which only used in rule $l2$. Therefore at least exist one string ulv, such that $ulv \in \| r_1 \| \wedge ulv \in \| r_2 \|$,where$l \in \Sigma$ is the eliminated character applied by rule $l2$ and $u, v \in \Sigma^*$ are any other characters. So the conclusion of rule $\epsilon 3$ is T.

For rule $l1$. The premise is $l_1 \cdot r_1 \cap l_2 \cdot r_2$, $l_1 \neq l_2$. We can proof the rule bycontradiction. We assume the conclusion of the rule is T. Then there isat least onestring lv, $v \in \Sigma^*$, such that $lv \in \| l_1 \cdot r_1 \| \wedge lv \in \| l_2 \cdot r_2 \|$, so $l = l_1 = l_2$. That contradicts with the premise$l_1 \neq l_2$. So we can get F from $l_1 \cdot r_1 \cap l_2 \cdot r_2$, $l_1 \neq l_2$.

 Copyright ⓒ 2015 SERSC

For rule $l2$. The premise is $l_1 \cdot r_1 \cap l_2 \cdot r_2, l_1 = l_2$. That is $l \cdot r_1 \cap l \cdot r_2$. If $l \cdot r_1 \cap l \cdot r_2$is not empty, then there is at least one string lv, $v \in \Sigma^*$, such that $lv \in \| l \cdot r_1 \| \wedge lv \in \| l \cdot r_2 \|$. Therefore we have $v \in \| r_1 \| \wedge v \in \| r_2 \|$, so $r_1 \cap r_2 \neq \emptyset$. And according to the definition of function ϵ^l:$\| to\epsilon^l(r) \| = \| r \|$, so we have $to\epsilon^l(r_1) \cap to\epsilon^l(r_2) \neq \emptyset$. Thenrule $l2$ is sound.

For rule $+$. The premise is $r_1 \cap (r_2 + r_3) \cdot r_4$. If $r_1 = \epsilon$, clearly the conclusion is T. If $r_1 \neq \epsilon$, we assume the conclusion is T, then there is at least on string lv, $v \in \Sigma^*$, such that $v \in \| r_1 \| \wedge lv \in \| (r_2 + r_3) \cdot r_4 \|$. According to the definition 2, we have $v \in \| r_1 \| \wedge lv \in (\| r_2 \cdot r_4 \| \cup \| r_3 \cdot r_4 \|)$, so we have $lv \in \| r_1 \| \wedge lv \in \| r_2 \cdot r_4 \| \vee lv \in \| r_1 \| \wedge lv \in \| r_3 \cdot r_4 \|$. Thenrule $+$is sound.

For rule $*$. The premise is $r_1 \cap r_2^* r_3$. If $r_1 = \epsilon$, clearly the conclusion is T. If $r_1 \neq \epsilon$, there are three cases for the expressions derived from premise. The first case that at least one of the conclusionsis$T$; the second case is that the conclusions are all F and there are no premise$r_1 \cap r_2^* r_3$expressions; the third case is that the conclusions are all F but there is at least onepremise$r_1 \cap r_2^* r_3$ expressions. For the first case, the is at least one string lv , $v \in \Sigma^*$, such that $lv \in \| r_1 \| \wedge lv \in \| r_2^* r_3 \|$, Thereforewe have $lv \in (\| r_1 \| \wedge lv \in \| r_2 \cdot r_2^* r_3 \| \vee \| r_1 \| \wedge lv \in \| r_3 \|)$;For the second case, clearly the conclusion is F; for the third case, if the end of premise$r_1 \cap r_2^* r_3$ is ϵ^l, then the end of derived expression $r_1 \cap r_2^* r_3$ may be ϵ^l. Because the next setp of $r_1 \cap r_2^* r_3$ is $r_1 \cap r_2 \cdot r_2^* r_3$, if$r_2$ is not the regular language with ϵ, then the only next step is rule $l2$, and the $l2$ will replace ϵ^* to ϵ^l. Then all the end of expressions in conclusions will become ϵ^l.Thereforerule $*$ is sound.

Theorem 2. The rules are complete.

Proof. For the completeness of rules, we can proof that for any regular expressions $r_1$and r_2, there is at least one rule satisfying $r_1$and r_2. According to the definition of regular expression, each expression must belong to one of the following four cases: ϵ , $l \cdot r$, $(r_1 + r_2) \cdot r$, $r_1^* \cdot r$ where $l \in \Sigma$, $r_i \in R_\Sigma$ and the ϵonly appears at the end of the regular expressions. We combine two by two of the four cases described above, then the following 10 kinds of cases can be obtained: $\epsilon \cap \epsilon$; $\epsilon \cap l \cdot r$; $\epsilon \cap (r_1 + r_2) \cdot r_3$; $\epsilon \cap r_1^* \cdot r_2$; $l_1 \cdot r_1 \cap l_2 \cdot r_2$; $l \cdot r_1 \cap (r_2 + r_3) \cdot r_4$; $l \cdot r_1 \cap r_2^* \cdot r_3$; $(r_1 + r_2) \cdot r_3 \cap (r_4 + r_5) \cdot r_6$; $(r_1 + r_2) \cdot r_3 \cap r_4^* \cdot r_5$; $r_1^* \cdot r_2 \cap r_3^* \cdot r_4$. Now we will discuss each kind of those cases.

Case 1. $\epsilon \cap \epsilon$. According to the definition of ϵ, the case 1 can be divided into the following three subcases: $\epsilon^* \cap \epsilon^*$, $\epsilon^l \cap \epsilon^l$and $\epsilon^* \cap \epsilon^l$. For $\epsilon^* \cap \epsilon^*$and $\epsilon^l \cap \epsilon^l$, we can directly apply the rule$\epsilon 2$and $\epsilon 3$. Forthepair$\epsilon^* \cap \epsilon^l$, itcannot be appearedin the derived process. Because on the one hand, if ϵ^lappeared, then in the derivation, rule $l2$ must apply at least once. The conclusion of $l2$ is $to\epsilon^l(r_1) \cap to\epsilon^l(r_2)$, that both expressions in the conclusion must apply function $to\epsilon^l$, therefore if the one side of $\epsilon \cap \epsilon$ is ϵ^l, then the other side also must beϵ^l. On the other hand, because there are no rules that can replace the end of expression from ϵ^* to ϵ^l, therefore it is impossible to get $\epsilon^* \cap \epsilon^l$ from $\epsilon^* \cap \epsilon^*$. So thepair$\epsilon^* \cap \epsilon^l$cannot be appearedin the derived process.

Case 2. For the case$\epsilon \cap l \cdot r$, rule $\epsilon 1$ match this pair, then we can get the conclusion F.

Case 3. For the case $\epsilon \cap (r_1 + r_2) \cdot r_3$, rule $+$ match this pair, then we can get the conclusion $\epsilon \cap r_1 \cdot r_3 \vee \epsilon \cap r_2 \cdot r_3$.

Case4.Forthe case $\epsilon \cap r_1^* \cdot r_2$, rule$*$ match this pair ,then we can get the conclusion $\epsilon \cap r_1 \cdot r_1^* r_2 \vee \epsilon \cap r_2$.

Case5. For the case $l_1 \cdot r_1 \cap l_2 \cdot r_2$, if $l_1 \neq l_2$, then rule $l1$ match this pair, and we can get the conclusion F; if $l_1 = l_2$, then the rule $l2$ match this pair, and we can get the conclusion $to\epsilon^l(r_1) \cap to\epsilon^l(r_2)$.

Case 6. For the case $l \cdot r_1 \cap (r_2 + r_3) \cdot r_4$, the rule $+$ match this pair, then we can get the conclusion $l \cdot r_1 \cap r_2 \cdot r_4 \vee l \cdot r_1 \cap r_3 \cdot r_4$.

Case 7. For the case $l \cdot r_1 \cap r_2^* \cdot r_3$, the rule$*$ match this pair, then we can get the conclusion $l \cdot r_1 \cap r_2 \cdot r_2^* \cdot r_3 \vee l \cdot r_1 \cap r_3$.

Case 8. For the case $(r_1 + r_2) \cdot r_3 \cap (r_4 + r_5) \cdot r_6$,we can apply the rule + twice and will get the conclusion $r_1 \cdot r_3 \cap r_4 \cdot r_6 \vee r_2 \cdot r_3 \cap r_4 \cdot r_6 \vee r_1 \cdot r_3 \cap r_5 \cdot r_6 \vee r_2 \cdot r_3 \cap r_5 \cdot r_6$.

Case 9. For the case $(r_1 + r_2) \cdot r_3 \cap r_4^* \cdot r_5$, firstly, we can apply the rule + and get the conclusion$r_1 \cdot r_3 \cap r_4^* \cdot r_5 \vee r_2 \cdot r_3 \cap r_4^* \cdot r_5$, then apply the rule * and get the conclusion $r_1 \cdot r_3 \cap r_4 \cdot r_4^* r_5 \vee r_1 \cdot r_3 \cap r_6 \vee r_2 \cdot r_3 \cap r_4 \cdot r_4^* r_5 \vee r_2 \cdot r_3 \cap r_5$.

Case 10. The case $r_1^* \cdot r_2 \cap r_3^* \cdot r_4$similar with case 8, then we can apply rule * twice.

5. Algorithm and Examples

Based on the inference rules, we develop an intersection checking algorithm for regular expressions. The input of the algorithm is two regular expressions. To simplify the process of the inference, add symbolϵ^* at the end of input regular expressions. The algorithm is depth-first searching based on inference system, finding the first conclusion$T$ which indicates that the algorithm already find a non-empty string s conforming regular expression$r_1$andr_2, which means $r_1$and $r_2$have intersection, thus Yes returned and the process is over. When the conclusion of the inference is F, it indicates that cannot find non-empty string s conforming both $r_1$and r_2 inthe current inference branch, thus go back to the closest branch point and search along another inference path. If there is not conclusion after traversal all the searching space, then $r_1$disjoints r_2 and No returned.

According to paper[11], we know that decision problem of intersection checking for regular expressions is PSPACE-complete. For the general regular expressions, the algorithm is exponential, but when both of the regular expressions are one-unambiguous regular expressions, the algorithm is polynomial [10]. Figure2 is about decision problem of intersection checking for one-unambiguous regular expressions. As we will mention in section7, compared with other researchers' works[19], our algorithm has a much wider applicability, especially for certain ambiguity regular expressions, the algorithm can give a result in polynomial time, see figure3 for more detailed information.

INPUT: regular expression $r_1$and r_2
OUPUT: "Yes" or "No"
Initialize stack T and set S empty;
push(r_1, r_2) on T;
While T is not empty do
    pop(r_3, r_4) from T;
    if(r_3, r_4)$\notin$S
        find an inference rule matching (r_3, r_4);
        if its conclusion is T; Return "Yes";
        if its conclusion is ($r_5 \cap r_6$) $\vee \cdots \vee$($r_m \cap r_n$); push(r_5, r_6) ,⋯,(r_m, r_n) on T;
        if the conclusion is ($r_5 \cap r_6$)→(r_5, r_7); push (r_5, r_7) , ($r_5 \cap r_6$) on T;
        add(r_3, r_4) to S;
    end
    if(r_3, r_4)$\in$S
        if the end of (r_3, r_4) is ϵ^l, then find a regular expression pair (r_8 , r_9)in T which is equivalent to (r_3, r_4), if the end of (r_8 , r_9)isϵ^*, replace the end of all the regular expressions upon (r_8 , r_9) in T with ϵ^l;
    end
end
return "No";

Figure 1. Algorithm of Intersection Checking

 Copyright ⓒ 2015 SERSC

Figure2 and 3 are two cases about decision problem of intersection checking for regular expressions. Figure2is$a^*b^*\cap(a+b)^*$, as each of the input regular expressions don't has unambiguous symbols, thus they are all one-unambiguous regular expressions. Notice that for convenience, we add ϵ^*at the end of r_1 and r_2 at the beginning of the inference. When we calculate to expression 8, result of the inference is same with expression 1, thus replace the end of expression 1 and all the regular expressions below it withϵ^l. In figure2, we use "…" to omit some simple inference steps. When we calculate to expression 19, the result is "T"by using rule$l1$, then return "Yes".

The right side of figure3 is a non-deterministic regular expression$\big(a+(b+c)^*c(b+c)(b+c)(b+c)(b+c)\big)b$, among which the non-deterministic regular expression is $(b+c)^*c(b+c)(b+c)(b+c)(b+c)$, indicating string whose fifth-last symbol is c. The deterministic automata for regular expression $(b+c)^*c(b+c)^n$ has the number of states proportional to $O(2^n)$, that is to say its states exponentially grow. In process of inference in figure3, the non-deterministic sub expression in the right side is ignored, saving much computing time. Thus as for regular expressions like$(r_1+r_2)r_3$, in which $r_1$and r_3 are one-unambiguous regular expressions,r_2 is an ambiguityregular expression. When we get "F" from the inference ofr_1r_3, our algorithm can automatic ignorer_2, this makes time complexity of the whole inference still keeps polynomial time.

$1: a^*b^*\epsilon^l\cap(a+b)^*\epsilon^l$

$(*)2{:}a^*b^*\epsilon^l\cap(a+b)(a+b)^*\epsilon^l \rightarrow 3: a^*b^*\epsilon^*\cap\epsilon^*$

$(+)4{:}a^*b^*\epsilon^l\cap a(a+b)^*\epsilon^l \vee 11$

$(*)6{:}aa^*b^*\epsilon^l\cap a(a+b)^*\epsilon^l \rightarrow 7{:}b^*\epsilon^l\cap a(a+b)^*\epsilon^l$

$(l2)8{:}a^*b^*\epsilon^l\cap(a+b)^*\epsilon^l$ / 1; $(*)9{:}bb^*\epsilon^l\cap a(a+b)^*\epsilon^l$ / $(l1)F \rightarrow 10{:}\epsilon^l\cap a(a+b)^*\epsilon^l$ / $(\epsilon 1)F$

$11: a^*b^*\epsilon^l\cap b(a+b)^*\epsilon^l$

$(*)12{:}aa^*b^*\epsilon^l\cap b(a+b)^*\epsilon^l$ / $(l1)F \rightarrow 13{:}b^*\epsilon^l\cap b(a+b)^*\epsilon^l$

$(*)14{:}bb^*\epsilon^l\cap b(a+b)^*\epsilon^l \rightarrow 15{:}\epsilon^l\cap b(a+b)^*\epsilon^l$

$(l2)16{:}b^*\epsilon^l\cap(a+b)^*\epsilon^l$

$(*)17{:}b^*\epsilon^l\cap(a+b)(a+b)^*\epsilon^l \rightarrow (*)18{:}b^*\epsilon^l\cap\epsilon^l$

$(+)19{:}b^*\epsilon^l\cap b(a+b)^*\epsilon^l \vee \frac{\dots}{F}$; $\frac{(*)\dots}{F} \rightarrow \frac{(l1)19{:}\epsilon^l\cap\epsilon^l}{T}$

$\frac{(*)\dots}{16} \rightarrow \dots \frac{\dots}{F}$

Figure 2. $\mathbf{a^*b^*\cap(a+b)^*}$

$1: ab\epsilon^*\cap\big(a+(b+c)^*c(b+c)(b+c)(b+c)(b+c)\big)b\epsilon^*$

$\frac{\frac{(+)2{:}ab\epsilon^*\cap ab\epsilon^*}{(l2)3{:}\epsilon^l\cap\epsilon^l}}{T} \vee 4: ab\cap(b+c)^*c(b+c)^4b\epsilon^*$

Figure 3. $\boldsymbol{ab\cap\big(a+(b+c)^*c(b+c)(b+c)(b+c)(b+c)\big)b}$

6. Experiments

The decision problem of intersection for regular expressions has been studied in depth based on the automatic technology and the complexity of this problem also has been proved by mathematic. Therefore, our approach cannot reduce the computational complexity of this problem in theory but can improve the practical computational speed significantly due to our approach do not need to convert the expressions to automatics and automatically ignore the no necessary parts of the regular expressions. In this section we present performance measurements for the introduced algorithms that decide the intersection of two regular expressions. The implementations were done in Java; the tests were performed on Core i7 with 2.8 GHz and 4 GB main memory.

Copyright ⓒ 2015 SERSC

A. Experiment of ordinary regular expressions

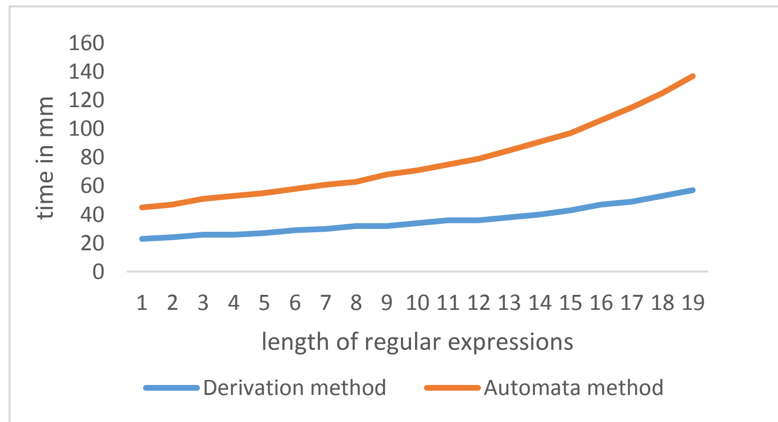


Figure 4. Experiment of Ordinary Regular Expressions

The first experiment tests the intersection algorithm showed in section 5for ordinary expressions such as $a^*b^*\cap(a+b)^*$. We extend the expressions to $a^*b^*c^*\cdots\cap(a+b+c\cdots)^*$ and the characters of those expressions were created randomly from a finite alphabet Σ. We simply define the length of a regular expression as the number of characters of a regular expression and test the length of expressions from 2 to 20. For comparison, we build the automatic approachand the time measurements include the parsingof the regular expressions, the creation of the automatics and the analysis whether the final state is reachable. Inorder to obtain stable values we executed the algorithm thousand times.The measured total time is afterwards divided to get a stableaverage execution time for one run. The execution time of the two algorithmsshowed in figure 4.

B. Experiment of regular expressions with no necessary parts

In a second scenario we evaluated the regular expressions with no necessary partssuch as $ab\cap(a+(b+c)^*c(b+c)^n)b$. We test the number of n from 1 to 20. The execution time of experimentshowed in figure 5. It is easy to see that the time complexity of our algorithm is almost unchanged.

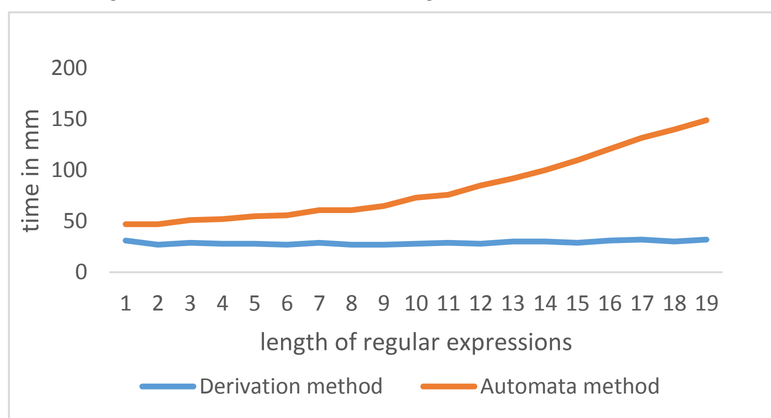


Figure 5. Experiment of Regular Expressions with no Necessary Parts

Copyright ⓒ 2015 SERSC

7. Relation Works

Theoreticalresearch on decision problems of regular expressions has been preceded by intensive research on finite automata in the 1960s and 1970s [11-12]. Martens,Neven and Schwentick study in paper[13] the complexity of the inclusion problem for several sub-classes of the regular expressions used in XML schema languages. Early research on the use of inference rules to solve the decision problems of regular expressions is found in Salomaa's work [14]. Two axiom systems for equality of regular expressions are presented in the paper [14]. Brzozowski [15-16] first proposed a set of inference rules for solving inclusion problem of regular expressions. Antimirov reinvents and details this approach in paper [17], as a term rewriting system for inequalities of regular expressions. To define the schema language DTD for SGML [18], Brüggemann-Klein and Wood [10] gave the definition of 1-unambiguity and showed that 1-unambiguous regular expressions are characterized by deterministic Glushkov automata.

The idea of this paper has some inspiration from the works of paper [19] and paper [20]. In order to solve decision problem of regular tree grammars with disjoint production, paper [19] described an algorithm which checks the intersection between two regular expressions based on constructing finite automata. The time complexity of this algorithm is $O(|E_1| \cdot |E_2| \cdot |\Sigma_1 \cup \Sigma_2|)$, where E_1 and E_2 are the input expressions, Σ_1 and Σ_2 are the character sets of E_1 and E_2. The problem to be solved in paper [19] is similar with ours, but we adopt a different method. Ours algorithm does not need to construct finite automata from regular expressions. Additionally, for some non 1-unambiguous regular expressions, Ours algorithm also may terminate in polynomial time while the paper [19] cannot do. Paper [20] described a polynomial-time algorithm based on a syntax-directed inference system for inclusion of 1-unambiguous regular expression. Our work differs from paper [20] as follows: Firstly, the reference rules presented by our are to address the intersection problem for regular expressions instead of inclusion problem. Secondly, our algorithm can deal with all kinds of regular expressions, while the algorithm described in paper [20] can only be used to 1-unambiguous regular expression. Finally, every step of derivation needs to calculate the set *first* and the function *header* of two input expressions, which significantly increases the cost of computing time. The algorithm of this paper do not need to calculate the set first and the function header and only need to calculate a function which computational complexity is constant time, and therefore more faster than the algorithm described in paper [20].

8. Conclusion

In this paper we provide an algorithm aimed at intersection checking algorithm for regular expressions, which is a goal-directed, depth-first searching based on inference system. This method is derived directly based on regular expression without constructing of deterministic finite automata. Our approach cannot reduce the computational complexity of this problem in theory but can improve the practical computational speed significantly due to our approach do not need to convert the expressions to automatics and automatically ignore the no necessary parts of the regular expressions. For general regular expressions it is exponential time algorithm and for some special cases, especially for the regular expressions with no necessary parts and one-unambiguous regular expressions often used in XML type checking, it is the polynomial time algorithm.

Acknowledgements

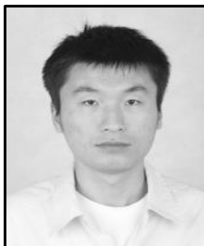
This work was both supported in part by the Beijing Nature Science Foundation underGrant 4122011and the National Science Foundation for Young Scientists of China underGrant 61202074.

Copyright ⓒ 2015 SERSC

References

[1] E. F. Codd, “A relational model of data for large shared data banks”, Communications of the ACM, vol. 13, no.6, **(1975)**, pp. 377-387.
[2] T. Bray, J. Paoli and C. M. Sperberg-McQueen, “Extensible markup language (XML)”, World Wide Web Consortium Recommendation REC-xml-19980210, http://www. w3. org/TR/1998/REC-xml-19980210, **(1998)**.
[3] H. Comon, M. Dauchet and R. Gilleron, “Tree automata techniques and applications”, http://tata.gforge.inria.fr, **(2014)**.
[4] V. Benzaken, G. Castagna and A. Frisch, “CDuce: an XML-centric general-purpose language”, ACM SIGPLAN Notices, vol.38, no.9, **(2003)**, pp. 51-63.
[5] H. S. Thompson, XML schema part 1: structures second edition, http://www. w3. org/TR/2004/REC-xmlschema-1-20041028, **(2004)**.
[6] V. Gapeyev, M. Y. Levin and B. C. Pierce, The Xtatic experience, Technical Reports, University of Pennsylvania, **(2004)**.
[7] H. Hosoya and B. C. Pierce, “XDuce: A statically typed XML processing language”, ACM Transactions on Internet Technology, vol.3, no.2, **(2003)**, pp.117-148.
[8] E. van der Vlist, Relax Ng, O'Reilly Media, **(2003)**.
[9] J. Clark, TREX–tree regular expressions for XML, http://www.thaiopensource.com/trex, **(2001)**.
[10] A. Brüggemann-Klein and D. Wood, One-unambiguous regular languages, Information and computation, vol.140, no.2, **(1998)**, pp.229-253.
[11] L. J. Stockmeyer and A. R. Meyer, Word problems requiring exponential time, Proceedings of the fifth annual ACM symposium on Theory of computing, **(1973)**.
[12] A. Nerode, Linear automaton transformations, Proceedings of the American Mathematical Society**(1958)**.
[13] W. Martens, F. Neven and T. Schwentick, Expressiveness and complexity of XML Schema, ACM Transactions on Database Systems (TODS), vol.31, no.3, **(2006)**, pp.770-813.
[14] A. Salomaa, Two complete axiom systems for the algebra of regular events, Journal of the ACM (JACM), vol.13, no.1, **(1966)**, pp.158-169.
[15] J. A. Brzozowski, Derivatives of regular expressions, Journal of the ACM (JACM), vol.11, no.4, **(1964)**, pp.481-494.
[16] J. A. Brzozowski, “Roots of star events, Switching and Automata Theory”, IEEE Conference Record of Seventh Annual Symposium, **(1966)**.
[17] V. Antimirov, Rewriting regular inequalities, Fundamentals of Computation Theory, Springer Berlin Heidelberg, **(1995)**, pp.116-125.
[18] C. F. Goldfarb and Y. Rubinsky, “The SGML handbook”, Oxford University Press, **(1990)**.
[19] NI Xiao-yong and CHEN Hai-ming, Intersection checking of production rules in regular tree grammar. Computer Engineering and Design, vol.33, no.3, **(2012)**, pp.1197-1202.
[20] D. Hovland, “The inclusion problem for regular expressions”, Journal of Computer and System Sciences, vol.78, no.6, **(2012)**, pp.1795-1813.

Authors



Jia Liu, he was born in 1984. He is a Ph.D. candidate at Beijing University of Technology. His research interests include software theory, data management for XML, etc.

Husheng Liao, he was born in Changchun in 1954. He is a professor and doctoral supervisor at Beijing University of Technology in P.R. China. His research interests include software automation methods and data integration technology, etc.

 Copyright ⓒ 2015 SERSC