Deterministic Regular Expressions with Interleaving [⋆]

Feifei Peng[1,2], Haiming Chen[1] and Xiaoying Mou[1,2]

[1]State Key Laboratory of Computer Science,
Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
[2] University of Chinese Academy of Sciences
`{pengff,chm}@ios.ac.cn`

Abstract. We study the determinism checking problem for regular expressions extended with interleaving. There are two notions of determinism, i.e., strong and weak determinism. Interleaving allows child elements intermix in any order. Although interleaving does not increase the expressive power of regular expressions, its use makes the sizes of regular expressions be exponentially more succinct. We first show an $\mathcal{O}(|\Sigma||E|)$ time algorithm to check the weak determinism of such expressions, where Σ is the set of distinct symbols in the expression. Next, we derive an $\mathcal{O}(|E|)$ method to transform a regular expression with interleaving to its weakly star normal form which can be used to rewrite an expression that is weakly but not strongly deterministic into an equivalent strongly deterministic expression in linear time. Based on this form, we present an $\mathcal{O}(|\Sigma||E|)$ algorithm to check strong determinism. As far as we know, they are the first $\mathcal{O}(|\Sigma||E|)$ time algorithms proposed for solving the weak and strong determinism problems of regular expressions with interleaving.

Key words: Regular Expressions, Interleaving, Strong Determinism, Weak Determinism, Algorithms

1 Introduction

DTD and XML Schema are two widely used schema languages recommended by W3C. The Unique Particle Attribution constraint [1] of DTD and XML Schema requires that all regular expressions used are weakly deterministic. The idea is inherited from the SGML Standard [2] to ensure more efficient parsing. Another definition of determinism which is called strong determinism has also been introduced in the context of XML [5]. Roughly speaking, weak determinism means that a symbol in the input word can be matched uniquely without looking ahead [11]. Meanwhile, strong determinism requires that the use of operators also be unique when matching a word. For example, $(a^*)^*$ is weakly deterministic but not strongly deterministic.

[⋆] Work supported by the National Natural Science Foundation of China under Grant Nos. 61472405, 61070038.

The interleaving operator &, also under the name of unordered concatenation, allows child elements intermix in any order. It existed in SGML but was excluded from the definition of DTDs. RELAX NG resurrects it but in a different way. In SGML, for example, $(ab)\&(cd)$ accepts only sequences "*abcd*" and "*cdab*". For $a\&b^*$, a cannot be present between two b. Usually, the purpose of using & operator is to allow child elements to occur in any order. Hence the above restriction is undesirable. In RELAX NG, $(ab)\&(cd)$ accepts all the sequences in which a occurs before b and c occurs before d, that is, has the interleaving semantics. XML Schema permits a strongly limited interleaving at the top level of a content model but nowhere else [7].

Lots of work (e.g.[4, 6, 8, 9, 12]) focused on testing determinism of standard regular expressions and regular expressions with counting. But little progress has been made in the scope of regular expressions with interleaving (called RE(&)). The most difficult parts are that the transitions of the corresponding Glushkov automata can be exponential and RE(&) do not have the property of locality [10] thus the above algorithms are not capable of dealing with RE(&). Some results for expressions with the SGML interleaving operator are provided in [10], which is not the same with the Relax NG interleaving operator considered here. No study has investigated the weak and strong determinism properties of RE(&) [3].

However, investigating the determinism properties of RE(&) has much significance [3]. In practice, it can help to relax the restriction about interleaving used in XML Schema thus will lead to more succinct and flexible schemas. In this paper, we study both the weak and strong determinism of regular expressions extended with interleaving operator. Here we consider the operator interpreting in the same way with RELAX NG in its more general form. The main contribution of this paper are two $\mathcal{O}(|\Sigma||E|)$ time methods to test weak and strong determinism of RE(&). As far as we know, they are the first $\mathcal{O}(|\Sigma||E|)$ time algorithms proposed for solving the above problems.

Our work about weak determinism is related and inspired by the work of unambiguity of extended regular expressions in SGML document grammars by Brüggemann-Klein [10]. Although having different semantics, the treatment of the & operator in [10] is similar to our first algorithm. To obtain a more efficient algorithm, we get inspiration from [9] which combines the $follow$ sets for the last symbols of each subexpression together into a single $followlast$ set instead of computing $follow$ set for each symbol of the expression. We define $followlast$ for expression with interleaving and establish the relation between $followlast$ and weak determinism for RE(&). As for strong determinism, we extend the notion of weakly star normal form [12] to RE(&) and show that a weakly deterministic RE(&) is strongly deterministic if and only if it is in weakly star normal form. Then we give a $\mathcal{O}(|E|)$ time method to transform an expression to its weakly star normal form. By combining the method and the $\mathcal{O}(|\Sigma||E|)$ time algorithm to check weak determinism, we can check strong determinism in $\mathcal{O}(|\Sigma||E|)$.

The rest of paper is organized as follows. Section 2 contains basic definitions that will be used throughout the paper. Section 3 presents an $\mathcal{O}(|\Sigma||E|^2)$ algorithm for weak determinism based on the $follow^-$ relations and an $\mathcal{O}(|\Sigma||E|)$

time algorithm based on $followlast$. In Section 4 an $\mathcal{O}(|\Sigma||E|)$ time algorithm checking strong determinism is presented. Details of implementation and experimental results are given in Section 5. We conclude in Section 6.

2 Preliminaries

2.1 Regular Expressions with Interleaving

Let u and v be two arbitrary strings. By $u\&v$ we denote the set of strings that is obtained by interleaving of u and v in every possible way. That is, $u\&\epsilon = \epsilon\&u = u$, $v\&\epsilon = \epsilon\&v = v$. If both u and v are non-empty, let $u = au', v = bv'$, where a and b are single symbols, then $u\&v = a(u'\&v) \cup b(u\&v')$. The operator & is then extended to regular languages as a binary operator in the canonical way. It is sufficient enough to say that & obeys the associative law. That is $E\&(F\&G) = (E\&F)\&G = E\&F\&G$ for any expressions E, F, G in RE(&) .

For the rest of the paper, Σ always denotes a finite alphabet. The regular expressions with interleaving over Σ are defined as: $\emptyset, \epsilon$ or $a \in \Sigma$ is a regular expression, E_1^*, E_1E_2, $E_1 + E_2$, or $E_1\&E_2$ is a regular expression for regular expressions E_1 and E_2. They are denoted as RE(&). The language $L(E)$ described by a regular expression with interleaving E is defined in the following inductive way: $L(\emptyset) = \emptyset$; $L(a) = \{a\}$; $L(E_1^*) = L(E_1)^*$; $L(E_1E_2) = L(E_1)L(E_2)$; $L(E_1{+}E_2) = L(E_1){\cup}L(E_2)$; $L(E_1\&E_2) = L(E_1)\&L(E_2)$. $E?$ and E^+ are used as abbreviations of $E{+}\epsilon$ and EE^*, respectively. For example, consider the following expressions and their languages: $L(ab\&cd) = \{acbd, acdb, cabd, cadb, abcd, cdab\}$, $L(a\&(b\&c)) = \{abc, bac, bca, cba, cab, acb\}$.

2.2 Deterministic Regular Expressions with Interleaving

Marked RE(&) are those symbols marked with subscripts hence each symbol can only occur once. The expression that removes the subscripts of marked symbols of a marked expression E is denoted by $E^\natural$. We denote $(\cdot)^\natural$ as unmarking operator and $(\cdot)'$ as marking operator. For a language L, let $L^\natural$ denotes $\{w^\natural | w \in L\}$, then obviously $(L(E))^\natural = L(E^\natural)$. The set of symbols that occur in E is denoted by $sym(E)$. The size of a RE(&) expression E is denoted by $|E|$. Now a concise definition of weak determinism of expression can be given by its marked form.

Definition 1 ([4]). *A marked expression E is weakly deterministic if and only if for all words $uxv \in L(E), uyw \in L(E)$ where $x, y \in sym(E)$ and $u, v, w \in sym(E)^*$, if $x^\natural \neq y^\natural$ then $x \neq y$. An expression E is weakly deterministic if and only if its marked expression E' is weakly deterministic.*

Intuitively, an expression is weakly deterministic if a symbol in the input word can be matched without looking ahead when matching against the expression. For instance, $(a_1?\&b_2)a_3$ is not weakly deterministic since it does not satisfy the condition if $x^\natural \neq y^\natural$ then $x \neq y$ with $u = b_2$, $v = a_3$ and $w = \epsilon$, and with the competing symbols $x = a_1$ and $y = a_3$. The corresponding unmarked expression $(a?\&b)a$ is not weakly deterministic.

A *bracketing of a RE(&) expression* E is a labeling of the iteration nodes of the syntax tree by distinct indices [6]. The bracketing $\widetilde{E}$ of E is obtained by replacing each subexpression $E_1^{*,+}$ of E with a unique index i with $([_iE_1]_i)^{*,+}$. Therefore, a bracketed RE(&) expression is a RE(&) expression over alphabet $\Sigma \cup \Gamma_E$, where $\Gamma_E = \{[_i,]_i \mid 1 \leq i \leq |E|_\Sigma\}$, $|E|_\Sigma$ is the number of symbol occurrences in E. A string w in $\Sigma \cup \Gamma_E$ is correctly bracketed if w has no substring of the form $[_i]_i$.

Definition 2 ([6]). *An expression E is* strongly deterministic *if E is weakly deterministic and there do not exist strings u, v, w over $\Sigma \cup \Gamma_E$, strings $\alpha \neq \beta$ over Γ_E, and a symbol $a \in \Sigma$ such that $u\alpha av$ and $u\beta aw$ are both correctly bracketed and in $L(\widetilde{E})$.*

For instance, the expression $(a^*)^*$ is weakly deterministic but not strongly deterministic since $[_2[_1a]_1]_2[_2[_1a]_1]_2, [_2[_1a]_1[_1a]_1]_2 \in L(([_2([_1a]_1)^*]_2)^*)$.

For a RE(&) expression E over Σ and for each $z \in sym(E)$, the following definitions are needed to analyze the determinism of expressions.

$first(E) = \{a | au \in L(E), a \in sym(E), u \in sym(E)^*\}$

$last(E) = \{a | ua \in L(E), a \in sym(E), u \in sym(E)^*\}$

$follow(E, z) = \{a | uzav \in L(E), u, v \in sym(E)^*, a \in sym(E)\}$, $z \in sym(E)$

$followlast(E) = \{a | uav \in L(E), u \in L(E), u \neq \epsilon, a \in sym(E), v \in sym(E)^*\}$

It is not hard to see that an expression E is not weakly deterministic if and only if there exist two symbols $x, y \in sym(E')$ with $x^\natural = y^\natural$ such that $x, y \in first(E')$ or there is a symbol $z \in sym(E')$ such that $x, y \in follow(E', z)$.

2.3 Computing $follow^-$ sets

We will need to calculate the $first$ and $follow^-$ sets. The inductive definition of the $first$ set for standard regular expressions can be trivially extended to RE(&). The inductive definition of the $follow^-$ can be found in [10].

Definition 3 ([10]). *For a marked expression E, we define $follow^-(E, x)$ for x in $sym(E)$ by induction on E as follows:*

$follow^-(E, \epsilon) = first(E)$

$E = x : follow^-(E, x) = \emptyset$

$E = F + G :$

$$follow^-(E, x) = \begin{cases} follow^-(F, x) & if\ x \in sym(F) \\ follow^-(G, x) & if\ x \in sym(G) \end{cases}$$

$E = FG :$

$$follow(E, x)^- = \begin{cases} follow^-(F, x) & if\ x \in sym(F), x \notin last(F) \\ follow^-(F, x) \cup first(G) & if\ x \in last(F) \\ follow^-(G, x) & if\ x \in sym(G) \end{cases}$$

$E = F\&G :$

$$follow^{-}(E,x)=\begin{cases} follow^{-}(F,x) & if\ x\in sym(F), x\notin last(F)\\ & or\ if\ x\in last(F), \epsilon\notin L(G)\\ follow^{-}(F,x)\cup first(G) & if\ x\in last(F), \epsilon\in L(F)\\ follow^{-}(G,x) & if\ x\in sym(G), x\notin last(G)\\ & or\ if\ x\in last(G), \epsilon\notin L(F)\\ follow^{-}(G,x)\cup first(F) & if\ x\in last(G), \epsilon\in L(G)\end{cases}$$

$E=F^{*}$:

$$follow^{-}(E,x)=\begin{cases} follow^{-}(F,x) & if\ x\in sym(F), x\notin last(F)\\ follow^{-}(F,x)\cup first(F) & if\ x\in last(F)\end{cases}$$

3 Weak Determinism of RE(&)

The weak determinism problem is to decide, given a regular expression with interleaving, i.e. $r\in$ RE(&), whether r is weakly deterministic or not. The classical way [11] is to compute the $first$ and $follow$ sets to check whether there exist symbols x, y such that $x, y\in first(E')$ or $x, y\in follow(E', z)$ for some symbol z. However, when it comes to & operator, there may be symbols x, y, z such that $x, y\in follow(E', z)$ yet E is weakly deterministic. For example, $E=(a\&b)a$, the corresponding marked expression is $E'=(a_1\&b_2)a_3$, then $follow(E', b_2)=\{a_1, a_3\}$. Thus E might be judged to be not weakly deterministic. Yet $L(E')=\{a_1b_2a_3, b_2a_1a_3\}$, it is not hard to see E is weakly deterministic in fact.

The best known algorithm to check the weak determinism of standard regular expressions is proposed in [11, 4] to check whether the corresponding Glushkov automata is deterministic. First, they proved that every regular expression can be transformed into its star normal form. Next, they showed that the determinism can be tested in $O(|\Sigma||E|)$ based on Glushkov automaton, via transforming an expression E into its star normal form in linear time. However, although we can transform a regular expression with interleaving using similar techniques as introduced in [11], transitions of the corresponding Glushkov can be exponential. Moreover, the method B.Groz [8] proposed to test whether a regular expression is deterministic in linear time by using a new structural decomposition of the parse tree is also not directly applicable to RE(&).

3.1 The first algorithm

In this section, we consider a subset $follow^{-}(E,z)$ of $follow(E,z)$ and develop a method based on $follow^{-}(E,z)$ to check the determinism of RE(&).

Consider the weak determinism between interleavings first. Suppose we have a marked expression $E=E_1\&...\&E_n$. If some E_i is not weakly deterministic, then E is not weakly deterministic. Assuming each E_i is weakly deterministic, E is also not weakly deterministic if there is some symbol a in both $E_i^{\natural}$ and $E_j^{\natural}$. Because no matter how other symbols intermix, there would always exist two string u, v such that $ua_ia_jv, ua_ja_iv \in L(E)$. So we need to ensure that $sym(E_i^\natural) \cap sym(E_j^\natural) = \emptyset$ for every two subexpressions E_i and E_j. As with the weak determinism between interleaving and other operators, consider a marked expression $H = EF$. Symbols that belong to E but not in $last(E)$ can not be in the same $follow$ set with symbols in F. A violation can only happen if $x, z \in last(E), y \in first(F), x^\natural = y^\natural$, which will cause $follow(H, z) = \{x, y\}$. If $\epsilon \notin L(E)$, x always occurs before y in any string accepted by $L(H)$ thus will not cause nondeterministic. We, therefore, use $follow^-$ to exclude these symbols in $first(F)$.

First, we have the following property about $follow^-$.

Note that $follow^-$ preserves the semantics of $follow$ when not dealing with & operator. That is, the situation $uzyw \in L(E)$, *but* $y \notin follow^-(E, z)$ occurs only if z is in subexpression $M\&N$ of E with $z \in sym(M), y \in sym(N)$. The following lemma is straightforward.

Lemma 1. *Let E be a marked expression. There are strings $u, v \in sym(E)^*$ and symbols $x, y, z \in sym(E)$ with $x^\natural = y^\natural$ such that $uzxv, uzyw \in L(E)$. If $x \in follow^-(E, z)$, $y \notin follow^-(E, z)$, then there exists some subexpression $M\&N$ or $N\&M$ of E, such that $x \in sym(M)$, $y \in sym(N)$.*

Proof. We prove it by contradiction. Suppose that x, y belong to the same side of subexpression $M\&N$ or $N\&M$ of E, then z must be in the other side, otherwise we will have $x, y \in follow^-(E, z)$. Assume $x, y \in sym(M)$ and $z \in sym(N)$ without loss of generality. Since $x \in follow^-(E, z)$, by the definition of $follow^-$, $x \in first(M)$ and $\epsilon \in L(M)$. Since $uzxv, uzyw \in L(E)$, we can see $y \in first(M)$ thus $y \in follow^-(E, z)$. This contradicts with the assumption that $y \notin follow^-(E, z)$. □

In fact, we can see from the above analysis that if $uzxv, uzyw \in L(E), x^\natural = y^\natural$ but $x, y \notin follow^-(E, z)$, then there exists some subexpression $M\&N$ or $N\&M$ of E, such that $x, y \in sym(M)$, $z \in sym(N)$. Let substring u' be the longest prefix of z and substrings v', w' be the longest suffix of z in u, v, w amongst $sym(M)$, then $u'xv', u'yw' \in L(M)$. That is, there exists a symbol $s \in sym(M)$ such that $x, y \in follow^-(M, s)$. Since M is a subexpression of E, then $x, y \in follow^-(E, s)$. This is shown in Lemma 2.

Lemma 2. *Let E be a marked expression. If there are strings $u, v \in sym(E)^*$ and symbols $x, y, z \in sym(E)$ with $x^\natural = y^\natural$ such that $uzxv, uzyw \in L(E)$ but $x, y \notin follow^-(E, z)$, then there exist a symbol $s \in sym(E)$ such that $x, y \in follow^-(E, s)$.*

The following theorem is the main result of this section which states the relation between weak determinism and $first, follow^-$ sets.

Theorem 1. *Let E be a marked expression, $z \in sym(E)$. E is not weakly deterministic if and only if there exist $x, y \in sym(E)$ with $x^\natural = y^\natural$ such that:*
(1) $x, y \in first(E)$ or
(2) $x, y \in follow^-(E, z)$, for some symbol $z \in sym(E)$ or

(3) $F\&G$ or $G\&F$ is a subexpression of E such that $x \in sym(F)$, and $y \in sym(G)$.

Proof. (=>) Assume E is not weakly deterministic, then $x, y \in first(E), x^\natural = y^\natural$ or there are strings $u, v, w \in sym(E)^*, x, y, z \in sym(E)$ such that $uzxv, uzyw \in L(E), x^\natural = y^\natural$. In the first case, $x, y \in first(E)$, condition (1) holds. For the latter case, there are three conditions:
(A) $x, y \in follow^-(E, z)$ or
(B) only one of x and y is in $follow^-(E, z)$ or
(C) $x, y \notin follow^-(E, z)$
For case (A) we are done. As for case (B), we assume $x \in follow^-(E, z)$ and $y \notin follow^-(E, z)$, then by Lemma 1, condition (3) holds. The other case can be proved similarly. For case (C), by Lemma 2, condition (2) or condition (1) holds.

(<=) It is obvious for condition (1). For condition (2), the proof is the same with that of Theorem 1 in [10]. As for condition (3), if $F\&G$ or $G\&F$ is a subexpression of E and $x \in sym(F)$ and $y \in sym(G)$, then $uzxv \in L(F)$ for some $u, v \in sym(F)^*$, $wys \in L(G)$ for some $w, s \in sym(G)^*$. Thus, $uwzxvys, uwzysxv \in L(F\&G)$, E is not weakly deterministic. □

The following Corollary indicates the restrictions put on interleaving in XML Schema might be stronger than necessary.

Corollary 1. *Let $E = E_1\&E_2\&...\&E_n$ be a marked expression. E is weakly deterministic if and only if $E_1, E_2, ...E_n$ are weakly deterministic and $sym(E_i^\natural) \cap sym(E_j^\natural) = \emptyset$ when $j \neq i$.*

The process of this approach is formalized in Algorithm 1. The *compete* function checks if there are two elements a, b in the input word such that $a^\natural = b^\natural$. It returns true if such elements exist, or false otherwise. Below we analyze the time used to test weak determinism.

Theorem 2. *Let E be an expression over a finite alphabet Σ. It can be decided in $\mathcal{O}(|\Sigma||E|^2)$ whether E is weakly deterministic or not.*

Proof. The $first$,$last$ and $follow^-$ sets can be implemented bottom up by converting E into a syntax tree, whose internal nodes are labeled with one of the operators $+, \cdot, ^+, ?, ^*$ or &. If sets are maintained as ordered lists, it can be checked whether there exist x, y such that $x^\natural = y^\natural$ that are included in a $first$ or $follow^-$ set in linear time via merging lists. As soon as this occurs, E is reported to be not weakly deterministic. Hence the maximum length of each $first$ or $follow^-$ set is $|\Sigma|$. Since E has at most $\mathcal{O}(|E|)$ subexpressions, and each subexpression has at most $|E|$ last symbols. At this point, the total time is $\mathcal{O}(|\Sigma||E|^2)$. Condition (3) can be tested by scanning each symbol x in F to see whether there exists a symbol y in G such that $x^\natural = y^\natural$. Emptiness test of $sym(F^\natural) \cap sym(G^\natural)$ can be done in $\mathcal{O}(|\Sigma|)$ time with a hash table. So each subexpression can be examined in $\mathcal{O}(|\Sigma|)$ time. The upper bound of condition (3) is $\mathcal{O}(|\Sigma||E|)$.

Based on the above discussion, it takes $\mathcal{O}(|\Sigma||E|^2)$ time to check the weak determinism of an expression in RE(&). □

Algorithm 1 *weakDeterm1*

```
Input: An expression $E$ in RE(&)
Output: true if $E$ is weakly deterministic or false otherwise
1: construct the corresponding binary tree $T(root)$ of $E$
2: return $weakDeterm\_helper1(root)$
```

3.2 The improved algorithm

Based on the ideas in the previous section, we can have a simpler method that runs in $\mathcal{O}(|\Sigma||E|)$ time by optimizing the examination of the $follow^{-}$ relation used in Theorem 1.

Definition 4. *For a marked expression E, we define $followlast(E)$ by induction on E as follows:*

$E = x : followlast(E) = \emptyset$

$E = F^* : followlast(E) = followlast(F) \cup first(F)$

$E = F + G : followlast(E) = followlast(F) \cup followlast(G)$

$E = FG :$

$$followlast(E) = \begin{cases} followlast(G) & if\ \epsilon \notin L(G) \\ followlast(F) \cup first(G) \cup followlast(G) & if\ \epsilon \in L(G) \end{cases}$$

$E = F\&G :$

$$followlast(E) = \begin{cases} followlast(F) \cup followlast(G) & if\ \epsilon \notin L(F), \epsilon \notin L(G) \\ followlast(F) \cup followlast(G) & if\ \epsilon \notin L(F), \epsilon \in L(G) \\ \cup first(G) & \\ followlast(F) \cup followlast(G) & if\ \epsilon \in L(F), \epsilon \notin L(G) \\ \cup first(F) & \\ followlast(F) \cup followlast(G) & if\ \epsilon \in L(F), \epsilon \in L(G) \\ \cup first(F) \cup first(G) & \end{cases}$$

For example, for $E = F\&G$, $last(E) = last(F) \cup last(G)$. If $\epsilon \in L(F)$ and $\epsilon \in L(G)$, for each $z \in last(F)$, we have $follow^{-}(E, z) = follow^{-}(F, z) \cup first(G)$. For each $z \in last(G)$, we have $follow^{-}(E, z) = follow^{-}(G, z) \cup first(F)$. Together these give that $followlast(E) = followlast(F) \cup first(G) \cup followlast(G) \cup first(F)$.

We can now move to check the weak determinism constraints by computing the $first$ and $followlast$ sets.

Theorem 3. *Let E be a marked expression. E is not weakly deterministic if and only if there exist $x, y \in sym(E)$ with $x^\natural = y^\natural$ and a subexpression F of E such that:*
(A) $x, y \in first(F)$ or
(B) $F = GH$ with $x \in followlast(G)$ and $y \in first(H)$ or
(C) $F = G^*$ or $F = G^+$ with $x \in followlast(G)$ and $y \in first(G)$ or
(D) $F = G\&H$ with $x \in sym(G)$ and $y \in sym(H)$.

Algorithm 2 *weakDeterm_helper1*

```
Input: the root node $F$ of a binary tree $T(root)$
Output: true if the expression of $T(root)$ is weakly deterministic or false otherwise
  if $F = \epsilon, a$ then
    return true
  if $F = F_1|F_2$ then
    if $weakDeterm\_helper1(F_1)$ and $weakDeterm\_helper1(F_2)$ then
      if $first(F_1) \cap first(F_2) \neq \emptyset$ then
        return false
      else return true
    else return false
  if $F = F_1 F_2$ then
    if $weakDeterm\_helper1(F_1)$ and $weakDeterm\_helper1(F_2)$ then
      if $first(F_1) \cap first(F_2) \neq \emptyset$ then
        return false
      for each symbol $a \in last(F_1)$ do
        if $compete(follow^{-}(F, a))$ then
          return false
      return true
    else return false
  if $F = F_1 \& F_2$ then
    if $weakDeterm\_helper1(F_1)$ and $weakDeterm\_helper1(F_2)$ then
      if $sym(F_1)^{\natural} \cap sym(F_2)^{\natural} \neq \emptyset$ then
        return false
      if $first(F_1) \cap first(F_2) \neq \emptyset$ then
        return false
      else return true
    else return false
  if $F = F_1^{*}$ then
    if $weakDeterm\_helper1(F_1)$==false then
      return false
    for each symbol $a \in last(F_1)$ do
      if $compete(follow^{-}(F, a))$ then
        return false
    return true
  else return false
```

Proof. (=>) Assume E is not weakly deterministic, then some of conditions (1)-(3) of Theorem 1 hold. Condition (1) implies condition (A). Condition (2) holds if and only in condition (B) or (C) holds. Condition (3) is equivalent to condition (D).

(<=) Assume some of conditions (A)-(D) hold. If condition (A) holds then condition (1) or (2) of Theorem 1 hold. Condition (B) and condition (C) imply condition (2). Condition (D) is equivalent to condition (3), by Theorem 1, E is not weakly deterministic. □

The process of this approach is formalized in Algorithm 3.

```
Algorithm 3 weakDeterm2
Input: An expression $E$ in RE(&)
Output: true if $E$ is weakly deterministic or false otherwise
    construct the corresponding binary tree $T(root)$ of $E$
    return $weakDeterm\_helper2(root)$
```

Example 1. The expression $E = (a_1^*\&b_2)^*a_3$ is not weakly deterministic, since $first(E) = \{a_1, a_3\}$. It can also be notified by the fact that $followlast((a_1^*\&b_2)^*) = \{a_1, b_2\}$ and $first(a_3) = \{a_3\}$. The expression $E = a_1b_1\&c_1a_2$ is not weakly deterministic, since $a_2 \in sym(c_1a_2)$ and $a_1 \in sym(a_1b_1)$.

The calculation of $first$ and $followlast$ sets is done at the same time using bottom-up on the syntax tree of E. The algorithm will terminate as soon as at least one of the four conditions is satisfied. Thus the length of $first$ set can be at most $\mathcal{O}(|\Sigma|)$. Since each $followlast$ set contains at most $\mathcal{O}(2|\Sigma|)$ symbols, the computation can be performed in $\mathcal{O}(|\Sigma|)$ time. Emptiness test of $sym(F^\natural) \cap sym(G^\natural)$ can be done in $\mathcal{O}(|\Sigma|)$ time with a hash table. So each subexpression can be examined in $\mathcal{O}(|\Sigma|)$ time. An expression E contains at most $\mathcal{O}(|E|)$ subexpressions. Thus the time complexity of the algorithm is $\mathcal{O}(|\Sigma||E|)$. If the size of the alphabet is fixed, the algorithm has linear running time.

Theorem 4. *Let E be an expression over a finite alphabet Σ. It can be decided in $\mathcal{O}(|\Sigma||E|)$ whether E is weakly deterministic or not.*

4 Strong Determinism of RE(&)

In this section, we derive an algorithm checking strong determinism based on a characterization of strong determinism.

In [12], H. Chen et al. proved that a weakly deterministic regular expression with counting is strongly deterministic if and only if it is in weakly star normal form (wSNF). We will show it also holds for RE(&).

Definition 5 ([12]). *An expression E is in weakly star normal form if, for each subexpression H^* of E', $followlast(H) \cap first(H) = \emptyset$, where E' is the marked expression of E.*

We first show that every weakly deterministic expression can be transformed to an equivalent strongly deterministic expression in linear time. The following two definitions are proposed in [4] to transform a standard regular expression to its star normal form. We replace $\epsilon^\circ = \emptyset$ with $\epsilon^\circ = \epsilon$ in [4] and add the rules for $E = F\&G$ to get a transformation method for weakly star normal form.

Definition 6. $E = \emptyset, x, \epsilon : E^\circ = E$
$E = F^* : E^\circ = F^\circ$
$E = F + G : E^\circ = F^\circ + G^\circ$
$E = FG :$

Algorithm 4 *weakDeterm_helper2*

Input: the root node F of a binary tree $T(root)$

Output: true if the expression of $T(root)$ is weakly deterministic or false otherwise

```
if $F = \epsilon, a$ then
    return true
if $F = F_1|F_2$ then
    if $weakDeterm\_helper2(F_1)$ and $weakDeterm\_helper2(F_2)$ then
        if $first(F_1) \cap first(F_2) \neq \emptyset$ then
            return false
        else return true
    return false
if $F = F_1F_2$ then
    if $weakDeterm\_helper2(F_1)$ and $weakDeterm\_helper2(F_2)$ then
        if $followlast(F_1) \cap first(F_2) \neq \emptyset$ then
            return false
        if $\epsilon \in L(F_1)$ then
            if $first(F_1) \cap first(F_2) \neq \emptyset$ then
                return false
        else return true
    else return false
if $F = F_1\&F_2$ then
    if $weakDeterm\_helper2(F_1)$ and $weakDeterm\_helper2(F_2)$ then
        if $sym(F_1)^\natural \cap sym(F_2)^\natural \neq \emptyset$ then
            return false
        if $first(F_1) \cap first(F_2) \neq \emptyset$ then
            return false
        else return true
    else return false
if $F = F_1^*$ then
    if $weakDeterm\_helper2(F_1)$==false then
        return false
    if $followlast(F_1) \cap first(F_1) \neq \emptyset$ then
        return false
    else return true
```

$$E^\circ = \begin{cases} FG & if\ \epsilon \notin L(F), \epsilon \notin L(G) \\ F^\circ G & if\ \epsilon \notin L(F), \epsilon \in L(G) \\ FG^\circ & if\ \epsilon \in L(F), \epsilon \notin L(G) \\ F^\circ + G^\circ & if\ \epsilon \in L(F), \epsilon \in L(G) \end{cases}$$

$E = F\&G:$

$$E^\circ = \begin{cases} F\&G & if\ \epsilon \notin L(F), \epsilon \notin L(G) \\ F^\circ\&G & if\ \epsilon \notin L(F), \epsilon \in L(G) \\ F\&G^\circ & if\ \epsilon \in L(F), \epsilon \notin L(G) \\ F^\circ + G^\circ & if\ \epsilon \in L(F), \epsilon \in L(G) \end{cases}$$

Definition 7. $E = \emptyset, x, \epsilon : E^\bullet = E$

$E = F + G : E^{\bullet} = F^{\bullet} + G^{\bullet}$
$E = FG : E^{\bullet} = F^{\bullet}G^{\bullet}$
$E = F^* : E^{\bullet} = F^{\bullet\circ *}$
$E^{\bullet} = F^{\bullet}\&G^{\bullet}$

For example, $E = (a^*\&b^*)^*$, then we have $E^{\bullet} = (a^*\&b^*)^{*\bullet} = (a^*\&b^*)^{\bullet\circ *} = (a^{*\bullet\circ}\&b^{*\bullet\circ})^* = (a+b)^*$.

Lemma 3. *$E^{\bullet}$ is the weakly star normal form of E which can be computed from E in linear time and $L(E^{\bullet}) = L(E)$.*

The proof can make a direct use of the proof of Theorem 3.1 in [4] so we omit it here. The E° of E has the following property.

Lemma 4. *Let E be an RE(&) expression. If $followlast(E') \cap first(E') = \emptyset$, $E = E^{\circ}$.*

Proof. This can be proved by induction on the structure of E. The cases for $E = \emptyset, a(a \in \Sigma), \epsilon$ are straightforward, where $E = E^{\circ}$.

$E = F + G$: From the computations of $first$ and $followlast$, we have $followlast(E') = followlast(F') \cup followlast(G')$ and $first(E') = first(F') \cup first(G')$. Since $followlast(E') \cap first(E') = \emptyset$, we have $followlast(F') \cap first(F') = \emptyset$ and $followlast(G') \cap first(G') = \emptyset$. By the inductive hypothesis we have $F = F^{\circ}$ and $G = G^{\circ}$, therefore, $E = E^{\circ}$.

$E = FG$: If $\epsilon \notin L(F), \epsilon \notin L(G)$, by Definition 6 we have $E^{\circ} = FG$. Therefore $E = E^{\circ}$. If $\epsilon \in L(F), \epsilon \notin L(G)$, from the computations of $first$ and $followlast$, we have $followlast(E') = followlast(G')$ and $first(E') = first(F') \cup first(G')$. Since $followlast(E') \cap first(E') = \emptyset$, we get $followlast(G') \cap first(G') = \emptyset$. By the inductive hypothesis we have $G = G^{\circ}$, then $E^{\circ} = FG^{\circ} = FG = E$. If $\epsilon \notin L(F), \epsilon \in L(G)$, from the computations of $first$ and $followlast$, we have $followlast(E') = followlast(F') \cup followlast(G') \cup first(G')$ and $first(E') = first(F')$. Since $followlast(E') \cap first(E') = \emptyset$, we can get $followlast(F') \cap first(F') = \emptyset$. By the inductive hypothesis we have $F = F^{\circ}$, then $E^{\circ} = F^{\circ}G = FG = E$. The situation when $\epsilon \in L(F), \epsilon \in L(G)$ can never happens. Otherwise, $followlast(E') = followlast(F') \cup followlast(G') \cup first(G')$ and $first(E') = first(F') \cup first(G')$, $followlast(E') \cap first(E')$ cannot be $\emptyset$, which is a contradiction.

$E = F\&G$: If $\epsilon \notin L(F), \epsilon \notin L(G)$, by Definition 6 we have $E^{\circ} = FG$. Therefore $E = E^{\circ}$. The situation when $\epsilon \in L(F), \epsilon \in L(G)$ can never happens. Otherwise, $followlast(E') = followlast(F') \cup followlast(G') \cup first(G') \cup first(F')$ and $first(E') = first(F') \cup first(G')$, $followlast(E') \cap first(E')$ cannot be $\emptyset$, which is a contradiction. The other cases can be proved similarly.

$E = F^*$: This case can never happen. Since $followlast(E') = followlast(F') \cup first(F')$ and $first(E') = first(F')$, $E = F^*$ will contradict to $followlast(E') \cap first(E') = \emptyset$. □

It is not hard to see that for any expression E, the weakly star normal of E is unique. We prove it by the next lemma.

Lemma 5. *Let E be an RE(&) expression. E is in weakly star normal form iff $E = E^\bullet$.*

Proof. (=>) We prove it by induction on the structure of E. The cases for $E = \emptyset, a(a \in \Sigma), \epsilon$ are straightforward, where $E = E^\bullet$.

$E = F + G, E = FG$ *or* $E = F\&G$: Suppose E is in wSNF, then for each subexpression H^* of E', $followlast(H) \cap first(H) = \emptyset$. For each subexpression H_1^* of F' and each subexpression H_2^* of G', since F', G' are subexpressions of E', we have $H_1, H_2 \subseteq H$. Therefore, $followlast(H_1) \cap first(H_1) = \emptyset$ and $followlast(H_2) \cap first(H_2) = \emptyset$ thus F, G are in wSNF. By the inductive hypothesis we have $F = F^\bullet, G = G^\bullet$, then $E = F^\bullet + G^\bullet, E = F^\bullet G^\bullet$ *or* $E = F^\bullet \& G^\bullet$. Therefore $E = E^\bullet$.

$E = F^*$: Suppose E is in wSNF, then for each subexpression H^* of E', $followlast(H) \cap first(H) = \emptyset$. For any subexpression H_1 of F', we have $H_1 \subseteq H$ and $F' \subseteq H$. Therefore, $followlast(H_1) \cap first(H_1) = \emptyset$ thus F is in wSNF. By the inductive hypothesis we have $F = F^\bullet$. Since $F' \subseteq H$, we have $followlast(F') \cap first(F') = \emptyset$. By Lemma 4, $F = F^\circ$, then $E^\bullet = F^{\bullet\circ *} = F^{\circ *} = F^* = E$.

(<=) By Lemma 3, $E^\bullet$ is in weakly star normal form. Since $E = E^\bullet$, E is in weakly star normal form. □

Then from Lemma 5, we have

Corollary 2. *If E is not the same with its weakly star normal form $E^\bullet$, E is not in weakly star normal form.*

The following characterization of strong determinism can be found in [12], which can be trivially extended to expressions in RE(&).

Lemma 6. *Let E be an expression in RE(&).*
(1) $E = \epsilon, a \in \Sigma$: E is strongly deterministic.
(2) $E = F+G$: E is strongly deterministic iff F and G are strongly deterministic and $first(F) \cap first(G) = \emptyset$.
(3) $E = FG$:
(a) If $\epsilon \in L(F)$, then E is strongly deterministic iff F and G are strongly deterministic, $first(F) \cap first(G) = \emptyset$, and $followlast(F) \cap first(G) = \emptyset$.
(b) If $\epsilon \notin L(F)$, then E is strongly deterministic iff F and G are strongly deterministic, and $followlast(F) \cap first(G) = \emptyset$.
(4) $E = F\&G$: E is strongly deterministic iff F and G are strongly deterministic, and $sym(F) \cap sym(G) = \emptyset$.
(5) $E = F^*$: E is strongly deterministic iff F is strongly deterministic and $followlast(F) \cap first(F) = \emptyset$.

Proof. The proof is by induction on the structure of E. We only show the induction step for interleaving. Others can be found in [12].

$E = F\&G$: If E is strongly deterministic, then E is weakly deterministic. By Corollary 1, $sym(F) \cap sym(G) = \emptyset$.

If F, G are strongly deterministic, then F, G are weakly deterministic. Since $sym(F) \cap sym(G) = \emptyset$, E is weakly deterministic from Corollary 1. If E is not strongly deterministic, then there are strings u, v, w over $\Sigma_E \cup \Gamma_E$, strings $\alpha \neq \beta$ over Γ_E, and a symbol $a \in sym(E)$ such that $u\alpha av$ and $u\beta aw$ are both correctly bracketed and in $L(\widetilde{E})$. Assume $a \in sym(F)$ without loss of generality. Let u', v', w' be the substrings of u, v, w amongst $sym(F)$ and α', β' be the substrings of α, β amongst Γ_F, then both of $u'\alpha' av'$ and $u'\beta' aw'$ are bracketed correctly and in $L(\widetilde{F})$, which implies that F is not strongly deterministic. This is a contradiction. So E is strongly deterministic. □

Then we can establish the relation between weak determinism and strong determinism by the following lemma.

Lemma 7. *Let E be a weakly deterministic expression. E is in wSNF iff E is strongly deterministic.*

The proof from right to left is based on Lemma 6 and Definition 5 by contradiction. The details are omitted here. The proof from left to right is by induction of E. For instance, we briefly prove the interesting case $E = F\&G$ in the inductive step. Suppose E is in wSNF. Thus F, G is clearly in wSNF. Since E is weakly deterministic, we have F, G are weakly deterministic and $sym(F) \cap sym(G) = \emptyset$ by Corollary 1. By the inductive hypothesis we have F, G are strongly deterministic. Hence E is strongly deterministic from Lemma 6.

From the above analysis we can get an algorithm to check strong determinism of RE(&). First, check weak determinism of E using $weakDeterm2(E)$. If E is weakly deterministic, compute the weakly star normal form $E^\bullet$ of E. If $E^\bullet$ is the same with E, E is strongly deterministic. Otherwise, E is not strongly deterministic. The time complexity of the algorithm is also $\mathcal{O}(|\Sigma||E|)$. The process is formalized in Algorithm 5. For instance, $E = (a^*)^*$. E is weakly deterministic because it contains only one symbol. $E^\bullet = a^{\bullet\circ*\bullet\circ*} = a^{\bullet\circ*\circ*} = a^{*\circ*} = a^*$. Since $E \neq E^\bullet$, E is not strongly deterministic. By Lemma 3, the equivalent strongly deterministic expression of E is $E^\bullet = a^*$.

Algorithm 5 *StrongDeterm*

```
Input: An expression in RE(&)
Output: true if $E$ is strongly deterministic or false otherwise
if $weakDeterm2(E)$ is true then
    compute the weakly star normal form $E^\bullet$ of $E$
    if $equal(E, E^\bullet)$ then
        return true
return false
```

Theorem 5. *$StrongDeterm(E)$ returns true iff E is strongly deterministic.*

Proof. It follows from Lemma 7 and Lemma 5. □

Theorem 6. *StrongDeterm*(E) *runs in time* $\mathcal{O}(|\Sigma||E|)$.

The proof follows from Theorem 4 and Lemma 3.

5 Implementations and experiments

In this section we first study the performance of the $followlast$ algorithm by comparing it with the $follow^-$ algorithm. Experiments were performed on a computer with a Intel Core 2 Duo CPU(2.67GHz) and 4G memory. Next, we discuss implementation of our algorithm for transforming an expression to its weakly star normal form. We have implemented all our algorithms and made them available at `http://lcs.ios.ac.cn/~pengff/projects.html`.

5.1 Weak determinism

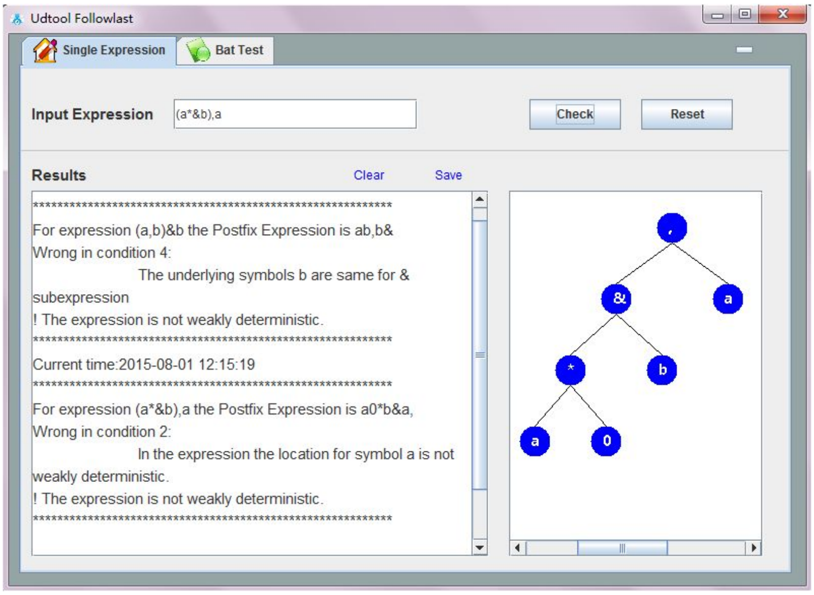
In this section, we describe experiments for verifying the correctness of algorithms based on $followlast$ and $follow^-$ sets. Complex content models are designed to test the efficiency of the above determinism algorithms.

All algorithms are implemented in Java. First, scan for an input expression and convert it into a syntax tree. Next, make a post order traversal of the syntax tree for computing $first$, $followlast$ and *sym* sets in subexpressions for every symbol. These contents are stored in ArrayList objects as attributes of nodes. At the same time, conditions in Theorem 1 or Theorem 3 can be checked for each subexpression. Once a subexpression meets the condition, the program is interrupted and shows information for nondeterministic symbols. A general overview of the interfaces of the $followlast$ algorithm is presented in Figure 1.

We design three complex content models for testing the efficiency and scalability of the above algorithms. The design for increasingly large content models is inspired by P. Kilpeläinen [9].

For sequence operator, the content model with interleaving can be defined by the form: $F_1\&F_2\&...\&F_n$, where the repeated subexpression F_i is of the sequence form $a_ib_i?c_i^*d_i^+$. Figure 2 shows how the algorithms scales up as the number of subexpressions is increased. It can be easily observed that it takes more time for $follow^-$ algorithms in these cases. Nonetheless, the gap is not very large. The reason is that the main difference between the two algorithms lies in sequence operator. We can see it from Algorithm 2 and Algorithm 4.

We further study the scalability for choice operator. The content model with interleaving can be defined by the form: $G_1\&G_2\&...\&G_n$, where the repeated subexpression G_i is of the sequence form $a_i|b_i?|c_i^*|d_i^+$. The weak determinism checking times are shown in Figure 3. As for choice form, the time usage of $follow^-$ algorithm increases quadratically, and the time usage of $followlast$ algorithm increases nearly linearly. However, both of them are slower than $followlast$ algorithm for sequence which implies sequence is easier to implement.



Input Expression (a*&b),a

```
**********************************************************
For expression (a,b)&b the Postfix Expression is ab,b&
Wrong in condition 4:
                    The underlying symbols b are same for &
subexpression
! The expression is not weakly deterministic.
**********************************************************
Current time:2015-08-01 12:15:19
**********************************************************
For expression (a*&b),a the Postfix Expression is a0*b&a,
Wrong in condition 2:
                    In the expression the location for symbol a is not
weakly deterministic.
! The expression is not weakly deterministic.
**********************************************************
```

Figure 1: Checking the weak determinism of $(a^*\&b)a$

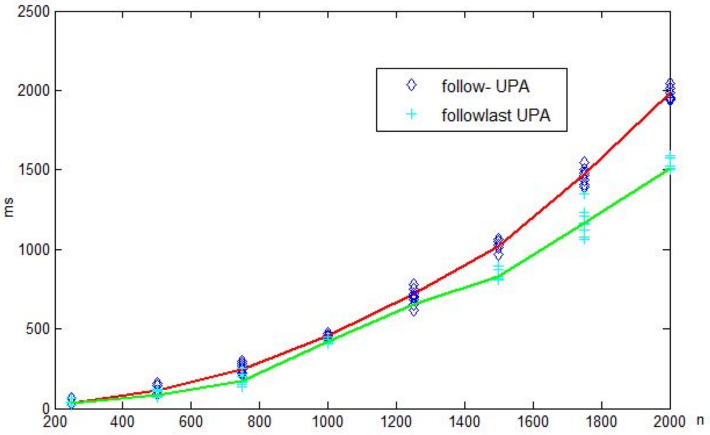


Figure 2: Scale-up: Number of subexpressions for sequence

5.2 Weakly star normal form and strong determinism

$E^{\bullet}$ is built up from $H^{\bullet\circ}$ for subexpressions H^* of E during a post order traversal through the syntax tree of E. Some tricks must be mentioned. The expression is processed to a postfix expression for keeping the order of operations and removing parentheses, and during this preprocessing, a special symbol 0 is added before a unary operator. When constructing binary trees, the leaf nodes are

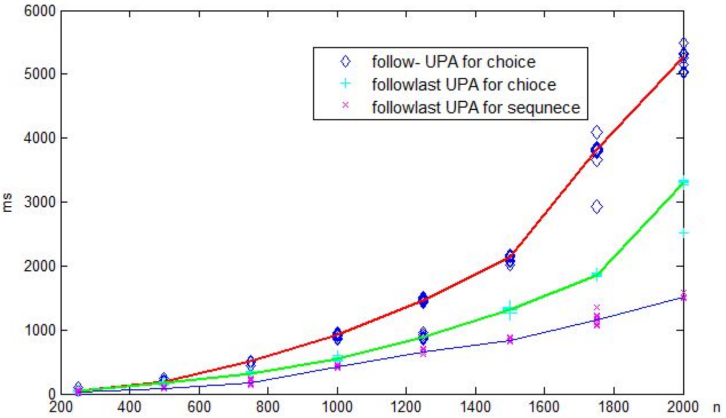


Figure 3: Scale-up: Number of subexpressions for choice

labeled with 0 or a symbol, and the internal nodes are labeled with operators, i.e. interleaving, choice, sequence or a unary operator. For every subexpression H^* of E, if $H = M^*$, parent node $*$ of node M is deleted. If M is the left node of its parent, we add $parent(node(*)).left = node(*).left$, otherwise we add $parent(node(*)).right = node(*).left$. If $H = FG, \epsilon \in L(F)$ and $\epsilon \in L(G)$, then parent node , of nodes F, G is replaced with $+$. In the end, we can get the postfix expression of $E^\bullet$ by another post order traversal of the syntax tree.

The strong determinism algorithm can be easily implemented by simply combing the above two algorithms. Note that the process of converting a postfix expression into an infix expression may add parentheses to the original expression. For example, from the above algorithm, the postfix expression for wSNF of $E = a, b, c$ is $(ab, c,)$. Thus $E^\bullet = ((a, b), c)$, but actually nothing has been changed in E. Therefore, function $equal(E, E^\bullet)$ is implemented by comparing whether the postfix expressions of E and $E^\bullet$ are equal or not.

6 Conclusion

In this paper, we have investigated the determinism problem for regular expressions extended with interleaving. Weak determinism is a property required by W3C XML Schema Recommendation. An $\mathcal{O}(|\Sigma||E|)$ time algorithm is proposed based on examination of $first$ and $followlast$ sets. We then explored the transformation from weakly deterministic RE(&) to strongly deterministic RE(&). Based on this form, we modify the weakly determinism algorithm to strong determinism. As for future work, we want to investigate whether there is a natural extension of the Glushkov construction for RE(&) and the relation between such automata and determinism.

References

1. World Wide Web Consortium. `http://www.w3.org/wiki/UniqueParticleAttribution`
2. ISO 8879. Information processingtext and office systems-standard generalized markup language (SGML) (1986)
3. Gelade W, Martens W, Neven F. Optimizing schema languages for XML: numerical constraints and interleaving. Database Theory ICDT. Springer Berlin Heidelberg, 2006: 269-283 (2007)
4. A. Brüggemann-Klein. Regular expressions into finite automata. Theoretical Computer Science, 120(2):197-213 (1993)
5. Koch C, Scherzinger S. Attribute grammars for scalable query processing on XML streams. The VLDB journal, 16(3): 317-342 (2007)
6. Gelade W, Gyssens M, Martens W. Regular expressions with counting: weak versus strong determinism. SIAM Journal on Computing, 41(1): 160-190 (2012)
7. Matthew Fuchs, and Allen Brown. Supporting UPA and restriction on an extension of XML Schema. Extreme Markup Languages®(2003)
8. B. Groz, S. Maneth, S. Staworko. Deterministic regular expressions in linear time. PODS, pp. 49-60 (2012)
9. P. Kilpeläinen. Checking determinism of XML Schema content models in optimal time. Informat. Systems, 36(3), pp. 596-617 (2011)
10. A. Brüggemann-Klein. Unambiguity of extended regular expressions in SGML document grammars. ESA, pp. 73-84 (1993)
11. A. Brüggemann-Klein, D. Wood. One-unambiguous regular languages. Information and Computation 142(2), 182-206 (1998)
12. H. Chen, P. Lu. Checking Determinism of Regular Expressions with Counting. Information and Computation 241, 302-320 (2015)