One-Unambiguity of Regular Expressions with Numeric Occurrence Indicators

Pekka Kilpeläinen, Rauno Tuhkanen

Report A/2006/2

ISBN 951-781-272-8
ISSN 1795-9195

UNIVERSITY OF KUOPIO
Department of Computer Science

P.O.Box 1627, FI-70211 Kuopio, FINLAND

One-Unambiguity of Regular Expressions with Numeric Occurrence Indicators [⋆]

Pekka Kilpeläinen [∗], Rauno Tuhkanen

University of Kuopio, Department of Computer Science
P.O. Box 1627, FI-70211 Kuopio, Finland

Abstract

Regular expressions with numeric occurrence indicators are an extension of traditional regular expressions, which let the required minimum and the allowed maximum number of iterations of subexpressions be described with numeric parameters. We consider the problem of testing whether a given regular expression E with numeric occurrence indicators is 1-unambiguous or not. This condition means, informally, that any prefix of any word accepted by expression E determines a unique path of matching symbol positions in E. The main contribution of this paper is a polynomial-time method for solving this problem, and a formal proof of its correctness.

Key words: regular expression, numeric iteration, interval expression, one-unambiguity, XML Schema, unique particle attribution

1 Introduction

Regular expressions with numeric occurrence indicators, or #REs for short, are an extension of traditional regular expressions [1]. They let the required minimum and the allowed maximum number of iterations of subexpressions be described with numeric parameters. An expression $E^{m..n}$ denotes, intuitively, the catenation of E with itself at least m and at most n times. #REs appear in a number of established practical variants of regular expressions [2–4]. Numeric iterations are a powerful shorthand notation, which allow some regular expressions be written much more compactly. Any expression $E = F^{m..n}$ can be written equivalently as $E' = FF \cdots F(F|\epsilon) \cdots (F|\epsilon)$, which consists of m copies of expression F followed by $n - m$ copies of $(F|\epsilon)$. If the numeric

[⋆] This work has been supported by the Academy of Finland (Grant no. 102270).
[∗] Corresponding author. E-mail: `Pekka.Kilpelainen@cs.uku.fi`, Tel: +358 17 165 761, Fax: +358 17 162 595

bounds m and n are written in ordinary d-base representation with $d \geq 2$, expression E' is longer than E by a factor which is exponential with respect to the length of E. (We follow the common but slightly imprecise practice of using "exponential" as a synonym for "super-polynomial".)

We consider the problem of testing whether a given regular expression E with numeric occurrence indicators is *1-unambiguous* or not. This condition means, informally, that any prefix of any word $w \in L(E)$ determines a unique path of matching symbol positions in E. That is, no lookahead is required to deterministically match the symbols of w against unique symbol positions in E, while processing the symbols of w one at a time from left to right. The main contribution of this paper is a polynomial-time method for solving this problem, and a formal proof of its correctness. We are not aware that correct polynomial-time algorithms for testing the one-unambiguity of #REs had been published before.

We define the syntax and semantics of regular expressions with numeric occurrence indicators, unambiguity, and related basic concepts more precisely in Section 2. One-unambiguity appears as a validity constraint of expressions used in document schema languages such as SGML and XML DTDs (document type definitions) and XML Schema. Especially, XML Schema both includes numeric occurrence indicators and requires 1-unambiguity of expressions. We discuss the theoretical and practical motivation of the 1-unambiguity problem more in Section 3. We also discuss previously published solutions that we are aware of; these solutions either require exponential amounts of resources or produce erroneous results.

As indicated by the failures of earlier attempts, efficient testing of unambiguity of #REs requires novel methods. In Section 4 we introduce a semantic *flexibility* condition for numeric iterations. This condition has a central role in unambiguity caused by numeric iterations. In Section 5 we develop and prove formally correct a syntactic method, which allows us to recognize flexible iterations of an expression in linear time. Based on the flexibility of iterations we define *Follow relations* for #REs in Section 6, and, based on the Follow relations, introduce and prove formally correct a polynomial-time method for testing the unambiguity of expressions. The last section is a conclusion.

2 Preliminaries

Regular expressions describe languages, which are subsets of Σ^* for a given non-empty set of symbols Σ called the *alphabet*. As usual, we define the *closure* L^* of a language L with $L^* = \bigcup_{i \geq 0} L^i$, where $L^0 = \{\epsilon\}$ and $L^{i+1} = LL^i$; here ϵ denotes the empty word, and catenation of languages L_1 and L_2 is defined by $L_1L_2 = \{uv \mid u \in L_1, v \in L_2\}$. The notation L^+ is also used in its standard meaning $L^+ = \bigcup_{i \geq 1} L^i$. Thus, Σ^* denotes the set of all *words* that can be formed using symbols of alphabet Σ, and Σ^+ is the same set excluding the zero-length empty word.

We use integer superscripts on symbols and words to denote their repetitions in words. For example, a^3bc^3 denotes the word *aaabccc*, and $(a^2b)^2$ denotes the word *aabaab*.

Regular expressions are built of symbols $\emptyset$, ϵ, and $a \in \Sigma$ connected together using sequential catenation, infix operator $|$, postfix operator *, and parentheses for grouping. The language $L(E)$ described by a regular expression E is defined inductively as follows:

$$\begin{array}{ll} L(\emptyset) = \emptyset; & L(\epsilon) = \{\epsilon\}; \\ L(a) = \{a\} \text{ for } a \in \Sigma; & L(FG) = L(F)L(G); \\ L(F|G) = L(F) \cup L(G); & L(F^*) = L(F)^* \ . \end{array}$$

Regular expressions with numeric occurrence indicators (#REs) use, in addition to the above standard constructs, iterative expressions of the form $F^{m..n}$, where m and n are non-negative integers satisfying $m \le n$. For an iteration $F^{m..n}$ we call expression F the *body*, integer m the *minimum bound*, and integer n the *maximum bound* of the iteration. The semantics of numeric occurrence indicators is defined as follows:

$$\begin{aligned} L(F^{m..n}) &= \bigcup_{i=m}^{n} L(F)^i \\ &= \{v_1 \ldots v_i \mid m \le i \le n;\ v_1, \ldots, v_i \in L(F)\} \ . \end{aligned}$$

That is, $L(F^{m..n})$ consists of words that result by concatenating at least m and at most n words of $L(F)$. We use F^n as a shorthand notation for an iteration $F^{n..n}$ whose minimum and maximum bound are both n. In the sequel we often call regular expressions with numeric occurrence indicators simply *expressions*.

For simplicity we exclude the empty set symbol $\emptyset$ from consideration in the expressions. It is straightforward to simplify any expression into an equivalent form that either does not include $\emptyset$ or consists of $\emptyset$ alone. (Such expressions have been called *trim*, e.g., by Brüggemann-Klein and Wood [5].) Thus the symbol $\emptyset$ is only needed for expressing the empty language, which is an uninteresting special case.

Also, we restrict to numeric iterations $F^{m..n}$ whose maximum bound n is at least two, since the other cases can be represented by an equivalent non-iterative form as follows:

$$F^{0..0} \equiv \epsilon,\ F^{0..1} \equiv (F|\epsilon),\ \text{and } F^{1..1} \equiv F$$

As usual, we use $F?$ as a shorthand notation for the expression $(F|\epsilon)$.

Finally, we allow the maximum bound of an iteration to be unbounded, denoted by ∞, and define its semantics as follows:

$$\begin{aligned} L(F^{m..\infty}) &= \bigcup_{i\geq m} L(F)^i \\ &= \{v_1 \dots v_i \mid i \geq m;\ v_1, \dots, v_i \in L(F)\} \ . \end{aligned}$$

In the sequel we restrict to expressions without the Kleene iteration F^*. This can be done without loss of generality, since $L(F^*) = L(F^{0..\infty})$.

Unless stated otherwise, we always consider *marked* expressions, where symbol occurrences are treated as unique *positions.* That is, any occurrence of a symbol $a \in \Sigma$ in an expression is represented as a position a_i, where i is an integer subscript such that a_i occurs exactly once in the marked expression. We denote by $\Pi = \{a_i \mid a \in \Sigma, i \in \mathbb{N}\}$ the marked alphabet for alphabet Σ, and by $()^\natural$ an *unmarking* operation which removes the subscripts of marked symbols. For example, $E = a_1(a_2|b_1)^{5..6}$ is a marked expression, where symbol a occurs at positions a_1 and a_2. The unmarked version is then $(E)^\natural = a(a|b)^{5..6}$. We denote the set of positions of a marked expression E by $Pos(E)$. For example, $Pos(a_1(a_2|b_1)^{5..6}) = \{a_1, a_2, b_1\}$. We sometimes omit subscripts from unique occurrences of symbols for simplicity. For example, we may write $a_1(a_2|b)^{5..6}$ to denote the same marked expression as above.

We follow Brüggemann-Klein and Wood in their treatment of unambiguity:

Definition 2.1 *[5] A marked expression E is* 1-ambiguous *if there are two different positions $x, y \in Pos(E)$ and some words $u, v, w \in Pos(E)^*$ such that*

$$uxv \in L(E), uyw \in L(E), \text{ and } (x)^\natural = (y)^\natural \ .$$

If an expression is not 1-ambiguous, it is 1-unambiguous.

That is, an expression is 1-ambiguous iff there is some input word such that after matching its prefix against a sequence of positions u, the next input symbol could be matched by two different occurrences x and y of that symbol. It may be helpful to think words accepted by a marked expression, and their subwords (such as u, v and w above), as *paths* of positions through the expression. For shortness, we refer to 1-ambiguity and 1-unambiguity using the simpler terms *ambiguity* and *unambiguity* instead.

Example 2.2 *The expression $E_1 = a_1?a_2$ is ambiguous, since an initial symbol a of an input word could be matched either by position a_1 or a_2. The equivalent expression $E_2 = (a_1)^{1..2}$, on the other hand, is trivially unambiguous, since it contains only a single position.*

The expression $E_3 = (a^{2..3}|x_1)^3 x_2$ is seen ambiguous by observing that $a^6 x_2 \in L(E_3)$ and $a^6 x_1 x_2 \in L(E_3)$. Informally this means that the prefix $a^6 x$ of an input word could be matched either (1) as three instances of $a^{2..3}$ followed by

Table 1
Inductive rules for the *First* and *Last* sets of an expression E

E	$\mathit{First}(E)$		$\mathit{Last}(E)$	
ϵ	$\emptyset$		$\emptyset$	
$x \in \Pi$	$\{x\}$		$\{x\}$	
$F\|G$	$\mathit{First}(F) \cup \mathit{First}(G)$		$\mathit{Last}(F) \cup \mathit{Last}(G)$	
$F^{m..n}$	$\mathit{First}(F)$		$\mathit{Last}(F)$	
FG	$\mathit{First}(F)$	if $\epsilon \notin L(F)$	$\mathit{Last}(G)$	if $\epsilon \notin L(G)$
	$\mathit{First}(F) \cup \mathit{First}(G)$	if $\epsilon \in L(F)$	$\mathit{Last}(G) \cup \mathit{Last}(F)$	if $\epsilon \in L(G)$

symbol x matched by position x_2; or (2) as three instances of $(a^{2..3}|x_1)$, the last of which is symbol x matched by position x_1. On the other hand, a rather similar expression $E_4 = (a^{2..3}|x_1)^2 x_2$ is unambiguous. This can be seen by observing that there are no two words that would satisfy the conditions of Definition 2.1 in the language $L(E_4) = \{a^4x_2, a^5x_2, a^6x_2, a^2x_1x_2, a^3x_1x_2, x_1x_1x_2\}$.

We include the expression itself in the set of its *subexpressions*, together with its proper subexpressions, defined in the usual way.

Lemma 2.3 *A marked expression E is ambiguous if and only if some of its subexpressions is ambiguous.*

Proof. If none of the subexpressions is ambiguous, then E is not ambiguous (as one of the subexpressions). On the other hand, if any of the subexpressions is ambiguous, then the entire expression is seen ambiguous as a consequence of the next Lemma 2.4. □

Lemma 2.4 *Let E be a trim marked expression. If $\alpha \in L(F)$ and $\beta \in L(F)$ for any subexpression F of E, then*

$$u\alpha v \in L(E) \text{ and } u\beta v \in L(E)$$

for some words $u, v \in \mathit{Pos}(E)^*$.

Proof. A straightforward induction. □

We need to refer to the dual sets of positions $\mathit{First}(F)$ and $\mathit{Last}(F)$ that respectively either start or end an accepting path through an expression F [6,7]. They are defined in the usual way:

$$\begin{aligned} \mathit{First}(F) &= \{x \in \mathit{Pos}(F) \mid xw \in L(F) \text{ for some } w \in \mathit{Pos}(F)^*\} \\ \mathit{Last}(F) &= \{x \in \mathit{Pos}(F) \mid wx \in L(F) \text{ for some } w \in \mathit{Pos}(F)^*\} \end{aligned}$$

Inductive rules for the *First* and *Last* sets of a regular expression (e.g., [8]) extended with the rules for numeric iterations are shown in Table 1. Based on the inductive definition, the *First* and *Last* sets can be computed for the subexpressions of an expression via a bottom-up traversal of its expression tree. Since the sets of disjoint subexpressions are disjoint, their unions can be implemented by simple operations on linked lists. Therefore an implicit representation of the sets can be computed in linear total time, by implementing each union in constant time by setting a few links to point to the lists of the immediate subexpressions. Alternatively we can compute an explicit *First* and *Last* list for each subexpression in quadratic total time, simply by copying the positions from the corresponding lists of the immediate subexpressions.

Observation 2.5 *Let F be a subexpression of a marked expression G. It can be seen by simple induction that if there is any position $x \in \mathit{Pos}(F) \cap \mathit{First}(G)$, then $\mathit{First}(F) \subseteq \mathit{First}(G)$. The corresponding result holds for the Last sets, too: the existence of any $x \in \mathit{Pos}(F) \cap \mathit{Last}(G)$ implies that $\mathit{Last}(F) \subseteq \mathit{Last}(G)$.*

3 Motivation and related work

Regular expressions with numeric occurrence indicators appear in a number of established variants of regular expressions, such as POSIX "interval expressions" [2, Chap.9], XML Schema content models with attributes `minOccurs` and `maxOccurs` [4], and Perl patterns [3,9]. In contrast to the relatively wide adoption of numeric occurrence indicators in practical implementations of regular expressions, little has been written about them in the theoretical literature. Standard literature on regular expression implementation [10–13] seems to ignore numeric occurrence indicators completely.

A special case of numeric iterations called *squaring* has been studied before. A squaring operator applied to subexpression F is equivalent to numeric iteration $(F)^2$. Meyer and Stockmeyer have shown that testing the equivalence of regular expressions with squaring requires exponential space [14]. The expressions used in the proof of this result are ambiguous. More recently, the complexity of decision problems for restricted forms of regular expressions that arise in practical XML schemas have been considered by Martens, Neven, and Schwentick [15], but they do not consider numeric iterations. The problems of inclusion, equivalence and intersection between expressions are seen to remain hard in most of the considered restricted cases, too. For one-unambiguous regular expressions we can compute equivalent deterministic automata in linear time [8], and the inclusion of DFAs can be tested in polynomial time applying the well-known cross-product construction. (See, e.g., [16,17].) Therefore the inclusion and consequently also equivalence of one-unambiguous regular expressions is solvable in polynomial time. We have shown earlier that membership of a word in the language of a #RE can be tested in polynomial time [18]. On the other hand, testing the inclusion and the non-emptiness of intersection between languages represented by two #REs are NP-hard problems even if the expressions are 1-unambiguous [18,19]. The complexity of testing the equivalence of two 1-unambiguous #REs appears to be an open problem.

A regular *language* is 1-unambiguous if it can be described by some 1-unambiguous regular expression. Brüggemann-Klein and Wood have studied the 1-unambiguity of regular languages [5]. They have shown that 1-unambiguous languages form a proper subclass of regular languages; given a Kleene characterization for 1-unambiguous languages; and provided a decision procedure for testing the 1-unambiguity of a given regular language by inspecting structural properties of its minimal accepting automaton.

Various document schema languages are basically grammatical formalisms, which usually describe allowed contents of document elements using variants of regular expressions. For example, XML Schema [4] realizes iteration through numeric attributes `minOccurs` and `maxOccurs` attached to content-describing elements such as `sequence`, `choice`, and `element`, which essentially corresponds to #REs over the alphabet of element tag-names. One-unambiguity was introduced to this application area by the SGML standard, which calls the condition simply "unambiguity" [20]. XML [21] is a simplified offspring of SGML, which was developed by the World Wide Web Consortium (W3C) to support the delivery of document data over the Internet. XML inherited the unambiguity constraint of SGML as "determinism". More recently, XML Schema adopted the same, or at least a similar constraint by the name "unique particle attribution".

From the point of view of language theory, XML Schema can be characterized as an XML-based language to express single-type tree grammars [22]. Web services are an active application area of XML Schema, where schemas are used as a typing mechanism to describe the format of XML messages interchanged between clients and servers [23,24]. XML Schema has also been criticized, for example of its overwhelming complexity (see, e.g., [25,26]), which probably hinders its wide adoption.

The XML Schema Recommendation [4,27] does not give precise formulation for unambiguity stating that "concise expression of this constraint is difficult". Instead, the non-normative Appendix H of the Recommendation outlines a complex sequence of operations for testing unambiguity, which includes unfolding numeric occurrence indicators, translating the expression into an automaton with epsilon transitions, and determinizing the automaton. The problem with this explanation, in addition to its procedural nature, is that a direct implementation is inherently exponential. Both the unfolding of numeric occurrence indicators and the determinization can lead to automata of exponential size, which makes the testing of unambiguity based on the procedure sketched in Appendix H infeasible [26].

We assume that the same notion of 1-unambiguity that has been used to model and to study the unambiguity of SGML and XML content models also applies to XML Schema [28,29]. The XML Schema Recommendation is verbal and verbose, which makes it difficult to verify whether any exact formalization of XML Schema captures the intentions of its authors. W3C did initiate work on the formal description of XML Schema, but the description has remained a draft only. Especially the unambiguity constraint has been left unspecified in the draft formal description of XML Schema [30]. Brown et al. have published an attempt to formalize some core ideas of XML Schema [31], but they refrain from modeling the unambiguity restriction of content models in their article, too.

Restricting to unambiguous expressions and being able to test the unambiguity of expressions is not a main goal by itself. Unambiguity is useful for efficient implementation of #REs, since it allows them to be efficiently matched by a hierarchy of deterministic automata [32]. Thus an efficient test for unambiguity is a precondition for efficient validation of documents using XML Schema. Also string pattern matching using #REs, say, for searching textual files, could be implemented more efficiently in those cases where the expression is recognized to be unambiguous.

Our work is related to, and inspired by the study of unambiguity of extended regular expressions in SGML grammars by Brüggemann-Klein [8,28,33]. SGML content model expressions do not include numeric iterations, but the challenge in Brüggemann-Klein's study [33] was the treatment of the '&' operator, which is used in SGML to describe permutations of subexpressions. A property that is common to marked SGML content model expressions and marked #REs is that both can denote languages that are not *local* [34]. This means, informally, that the possible continuations of a prefix of a word accepted by an expression are not determined by the last matching position alone, but may depend on the entire prefix instead.

XML Schema includes also an `all` construct, which is a strongly restricted version of the SGML '&' operator: an `all` may not be combined with any other operators (`sequence`, `choice`, or iterations of itself), and it can be applied to non-iterative elements only. Thus an XML Schema `all` is essentially equivalent to an SGML content model expression

$$a_1 \& a_2 \& \cdots \& a_m \& b_1? \& b_2? \& \cdots \& b_n?\ ,$$

where $a_1, \ldots, a_m$ and $b_1, \ldots, b_n$ are tag-names of required and optional subelements, respectively. Such an expression is unambiguous if and only if all these tag-names are pairwise disjoint.

A few authors have published methods for testing the unique particle attribution property of XML Schema content models, which is essentially the 1-unambiguity problem of #REs, but these solutions are either exponential or erroneous. We discuss these attempts below.

Thompson and Tobin [35] have described algorithms for testing the unique particle attribution property (UPA) of XML Schema content models. Their algorithms implement the procedure sketched in Appendix H of the XML Schema Recommendation rather directly, and thus require in the worst case exponential time and space with respect to the length of the content model expression.

Fuchs and Brown have published a method for testing the UPA constraint, which is based on computing and analyzing *first* and *follow* sets [26]. Thus their approach is quite close to ours. However, the algorithm of Fuchs and Brown does not recognize the flexibility of iterations (see Section 4), and this makes their method incorrect. When computing *follow* sets for the positions of an iteration $F = G^{m..n}$, Fuchs and Brown include the positions of $\mathit{First}(G)$ in this computation only if $m < n$ or if the body G of the iteration is nullable. (We include them also in other cases of flexible iterations; see Definition 6.1.)

As an example, consider the expression $E_3 = (a^{2..3}|x_1)^3x_2$, which we saw ambiguous in Example 2.2. Consider first its subexpression $F_1 = a^{2..3}$. According to Fuchs and Brown's definition, the set $\mathit{First}(a) = \{a\}$ is included in computing *follow* sets for the positions of subexpression F_1, and thus $\mathit{follow}(a) = \{a, x_2\}$. Then consider the larger subexpression $F_2 = (a^{2..3}|x_1)^3$. Since the bounds of this iteration are equal and its body is not nullable, the set $\mathit{First}(F_2) = \{a, x_1\}$ is not used in its *follow* analysis, and thus $\mathit{follow}(x_1)$ consists of position x_2 alone.

Fuchs and Brown also define for each subexpression F of E a *confusion set*, denoted by $\mathit{confusion}(F)$, which consists of those positions of F which belong in the *follow* set of some position in $\mathit{Last}(F)$. The idea is that these positions of F could be in conflict with some other positions of E. For example, $\mathit{confusion}(a) = \{a\}$ in the above subexpression F_1 of expression E_3.

Fuchs and Brown's UPA test [26, p. 5] for an iteration $G^{m..n}$ consists of testing the body G of the iteration recursively, and checking that $\mathit{First}(G) \cap \mathit{confusion}(G) = \emptyset$. This test as such would deem the subexpression $F_1 = a^{2..3}$ ambiguous, since $\mathit{First}(a) \cap \mathit{confusion}(a) = \{a\}$. This outcome is clearly wrong, since a single-position expression like F_1 is trivially unambiguous. The intended correct meaning of this condition is, obviously, that there are no *two different positions* with the same underlying symbol in $\mathit{First}(G) \cap \mathit{confusion}(G)$. Assuming this minor correction, the lack of observing the flexibility of iterations remains as a more severe flaw. Since the set $\mathit{First}(F_2)$ is not included in the computation of the *follow* sets for the positions in subexpression $F_2 = (a^{2..3}|x_1)^3$, the position x_1 is not recorded in any *follow* set. Thus the conflict between positions x_1 and x_2, which makes expression E_3 ambiguous, remains unnoticed.

Fuchs and Brown's UPA algorithm tries to check the unambiguity of expressions extended with the SGML '&' operator, too. Their definition of *follow* sets for '&'-expressions seems to differ from the corresponding definition of $follow^{-}$ sets by Brüggemann-Klein [33]. We suspect the correctness of their algorithm also with respect to this extension.

Fuchs and Brown also sketch a rather complicated algorithm for testing for given expressions F and G (with numeric iterations and SGML '&' operators) whether one of them subsumes the other, that is, whether $L(F) \subseteq L(G)$ holds. They state that the complexity of their algorithm is exponential with respect to the depth of nested iterations in the expressions only. This claim seems to contradict our proof of the NP-hardness of testing the subsumption between two unambiguous #REs [19], which uses expressions with only two nested iterations. Anyhow, we feel that more extensive analysis of Fuchs and Brown's work [26] is beyond the scope of this paper.

Sperberg-McQueen has recently described applications of so called Brzozowski derivatives to XML Schema processing, including the testing of unique particle attribution [36]. The *(Brzozowski) derivative* of an expression E by symbol $a \in \Sigma$, denoted by $D_a(E)$, is an expression such that $L(D_a(E)) = \{w \in \Sigma^* \mid aw \in L(E)\}$ [37]. That is, the derivative $D_a(E)$ accepts any continuations of an initial a in the words accepted by expression E. Derivatives can be computed by simple symbol manipulation, and they lead to intuitive and elegant algorithms for several problems related to regular expressions and their extensions. For example, a word $w = a_1 \dots a_n$ belongs to $L(E)$ if and only if the expression $D_w(E)$ is nullable, where the word-derivative $D_w(E)$ is obtained by successively deriving E by the symbols $a_1, \dots, a_n$ of the word w.

Sperberg-McQueen's idea is that an expression E is 1-ambiguous if and only if some word-derivative of E has two different occurrences of any symbol in its *First* set. (Actually he calls 1-unambiguity "weak determinism", but the meaning is the same.) Since the number of *characteristic derivatives*, that is, equivalence classes of derivatives of an expression is finite [37], the idea can be implemented as a terminating algorithm.

The above idea is intuitively appealing but, as such, slightly erroneous. For example, consider the expression $E = (a?b?)^2$, which is clearly unambiguous. Its derivative by symbol a is now $D_a(E) = b?(a?b?)$, which is ambiguous by containing two initial occurrences of symbol b. The idea could possibly be corrected as follows: (1) Start from a *marking* E' of the original expression; for example, $E' = (a_1?b_1?)^2$. (2) Define derivatives of marked symbols x in the obvious way, that is, $D_a(x) = \epsilon$ if $(x)^\natural = a$ and $\emptyset$ otherwise. (3) Deem the expression ambiguous if and only if some characteristic derivative of E' has *two different marked symbols* with the same underlying alphabet symbol in its *First* set. For example, in this case $D_a(E') = b_1?(a_1?b_1?)$ would not be considered as an indication of ambiguity, since the two competing occurrences of b originate from the same position b_1 of the original expression.

The main bottleneck of the derivative-based approach is that the number of characteristic derivatives can be large, that is, exponential in the worst case. This is inherent, since the characteristic derivatives are in one-to-one correspondence with the states of the minimal DFA for the expression [37]. As a simple example, the expression $E = a^{1..10^6}$ has the following 1,000,002 characteristic derivatives:

$$E, a^{0..10^6-1}, a^{0..10^6-2}, \ldots, a^{0..1}, \epsilon, \text{ and } \emptyset.$$

Sperberg-McQueen tries to reduce the large number of derivatives for numeric iterations by initially reducing their repetition bounds. His initial transformation replaces any subexpression F^n with $n > 2$ by iteration F^2, and any subexpression $F^{m..n}$ with bounds $1 \leq m < n$ by iteration $F^{1..2}$. Unfortunately his intuition that this transformation would not affect the ambiguity of the expression is wrong. For example, we saw the expression $E_4 = (a^{2..3}|x_1)^2x_2$ unambiguous in Example 2.2. On the other hand, its modified version $E_4' = (a^{1..2}|x_1)^2x_2$ transformed according to Sperberg-McQueen's suggestion is ambiguous, since it accepts both words aax_1x_2 and aax_2. Nevertheless, it might be possible to refine the derivative-based approach into an efficient algorithm for testing the unambiguity of expressions, but this would require some careful consideration.

4 Flexible iterations

In this section we introduce the concept of *flexible iterations*, which plays a central role in analyzing the unambiguity of #REs. A flexible iteration is, intuitively, an iteration that can accept some input without reaching the maximum bound of the iteration. Such flexibility can cause ambiguity of expressions, since in this case further input could be matched either by re-iterating the body of the iteration, or by some other part of the expression. In the simplest cases the flexibility of iterations is quite obvious, but as we'll see, it can also result from rather subtle interaction of minimum and maximum bounds in the expression.

Example 4.1 *Consider the iteration $F_1 = (a^{2..3})^2$. Its body $a^{2..3}$ is obviously flexible in the intuitive sense, since it accepts the word aa as two iterations of its body, while the maximum bound is three. This flexibility would make, for example, an expression like $(a_1)^{2..3}a_2$ ambiguous: the third symbol of input aaa could be matched either by a_1 or a_2. On the other hand, the full expression F_1 is intuitively* not *flexible, since each of the words a^4, a^5 and a^6 accepted by F_1 requires exactly two iterations of its body $a^{2..3}$. A quite similar expression $F_2 = (a^{2..3})^3$ again is flexible: it accepts the word a^6, which can also be matched by iterating its body only twice. For the same reason also the expression $F_3 = (a^{2..3}|x_1)^3$ is flexible. The flexibility of F_3 is the cause to the ambiguity of* *expression* $E_3 = (a^{2..3}|x_1)^3 x_2$*, which was considered in Example 2.2.*

We formalize the intuitive concept of (local) flexibility as follows:

Definition 4.2 *An iterative expression* $F = G^{m..n}$ *is* (locally) flexible*, if there is some word* $w \in L(F) \cap L(G)^k$ *for some* $k < n$*. We call such a word* w *a* witness *to the flexibility of* F*.*

That is, a witness to the flexibility of $F = G^{m..n}$ is a word which can simultaneously be accepted by F and matched by less than n iterations of its body G. We treat ∞ greater than any integer, which means that expressions of the form $G^{m..\infty}$ are flexible.

Notice that the flexibility of $F = G^{m..n}$ implies that for some words $w \in L(F)$ and $v \in L(G)$ also $wv^l \in L(F)$ for some $l > 0$. The converse does not hold, though. As an example, consider again the expression $F_1 = (a^{2..3})^2$. Now $a^4 \in L(F_1)$, $a^2 \in L(a^{2..3})$ and $a^4a^2 \in L(F_1)$, but as discussed in Example 4.1, expression F_1 is not flexible.

Some special cases of flexible iterations are easy to recognize. As the first case we mention iterations with differing maximum and minimum bounds.

Observation 4.3 *Any iteration* $F = G^{m..n}$ *with* $m < n$ *is flexible.*

Proof. The word v^m for any $v \in L(G)$ is a witness to the flexibility of F. $\square$

Nullable iterations are also immediately seen flexible:

Observation 4.4 *An iteration* $F = G^{m..n}$ *with* $\epsilon \in L(G)$ *is flexible.*

Proof. The word $\epsilon \in L(F) \cap L(G)^{n-1}$ is a witness to the flexibility of F. $\square$

Flexibility is not just a local property of expressions: An iteration which is not locally flexible can yet possess flexible behavior in the context of a larger expression. As an example, consider the expression $E = ((a^{2..3})^2)^2$, whose body $F_1 = (a^{2..3})^2$ was stated (locally) non-flexible in Example 4.1. Now E accepts the word a^8, which can be treated either as two full iterations of F_1 (each for a^4), or as a single iteration of F_1 (for a^6) followed by one iteration of its body $a^{2..3}$ (to match a^2). We say now that F_1 is *flexible in* E. Also this kind of flexibility can cause ambiguity in expressions. As an example consider the below extension of the above expression E:

$$E_5 = ((a^{2..3}|x_1)^2)^2 x_2 \ .$$

Now E_5 is ambiguous since it accepts both the words a^8x_2 and $a^8x_1x_2$; that is, the last symbol of the prefix a^8x of an input word could be matched alternatively by position x_1 or by position x_2. The ambiguity is caused by the flexibility of the subexpression $(a^{2..3}|x_1)^2$ in E_5. We define such flexibility of an iteration in an expression as follows:

Definition 4.5 *Let E be a marked #RE. An iterative subexpression $F = G^{m..n}$ of E is* flexible in E *if there is some word $uws \in L(E)$ with $w \in L(F)^l$ for some $l \geq 1$ such that $w \in L(F)^{l'} L(G)^k$ for some $l' < l$ and $k < n$. We call such a word w a* witness *to the flexibility of F in E.*

That is, a witness to the flexibility of $F = G^{m..n}$ in E is a subword w of some word accepted by E, such that w can be matched by a number of iterations of F in such a way that the last iteration isn't "full" ($k < n$). Returning to the above example, the word a^8 is a witness to the flexibility of $F = (a^{2..3}|x_1)^2$ in expression $E_5 = ((a^{2..3}|x_1)^2)^2 x_2$, since $a^8 x_2 \in L(E_5)$, $a^8 \in L(F)^2$, and $a^8 \in L(F)^1 L((a^{2..3}|x_1))^1$ (by treating it as $a^6 a^2$).

Non-local flexibility is related to iterations whose repeated occurrences are sufficient to satisfy the body of an enclosing iteration. As an example, consider the expression $F = ((a^{2..3}|x)^2 b?)^2$. Now two occurrences of the iteration $(a^{2..3}|x)^2$ constitute a single occurrence of the enclosing iteration F. Similarly, four occurrences of the iteration $a^{2..3}$ constitute a single occurrence of F. In this case we say that the sub-iterations are *factors* of the larger one. We define this relationship in general as follows: Let F be a subexpression of a marked expression G. If both $\mathit{First}(F) \subseteq \mathit{First}(G)$ and $\mathit{Last}(F) \subseteq \mathit{Last}(G)$ hold, expression F is a *factor* of expression G. If F is a factor of G, we say that G is a *multiple* of F. Notice that each expression is both a factor and a multiple of itself. If F is a proper subexpression and a factor of G, we say that F is a *proper factor* of G, and that G is a *proper multiple* of F.

The next lemma states that the flexibility in an expression is caused only by the flexibility of the subexpression in some iterative multiple of it.

Lemma 4.6 *Let F be an iterative subexpression of a marked expression E. Then F is flexible in E if and only if F is flexible in some multiple of F which is also an iterative subexpression of E.*

Proof. If F is flexible in a subexpression of E, then Lemma 2.4 implies that it is flexible in E, too.

Let then a word w be a witness to the flexibility of F in E. If $w = \epsilon$, expression F is by Observation 4.4 flexible in itself, and F is its own (non-proper) multiple. Assume then that $w \in \mathit{Pos}(F)^+$, and show by induction that w is a witness to the flexibility of F in a multiple of F which is an iterative subexpression of E. We discuss the inductive step only in the case that $E = H^{m..n}$ and E is *not* a multiple of F; the other cases are straightforward. Now we must have that $uws \in L(H)$ for some words $u, s \in \mathit{Pos}(H)^*$, since otherwise Observation 2.5 implies that E *is* a multiple of F. Thus w is a witness to the flexibility of F in H, and the claim holds by induction. □

5 Recognizing flexibility

We stated in Section 4 that an iteration $E = F^{m..n}$ is flexible if $m < n$, or if the body of the iteration is nullable, that is, if $\epsilon \in L(F)$. The first condition is trivial. The second one isn't problematic either, since all nullable subexpressions can be found by a straightforward linear-time traversal of the expression tree. Next we consider recognizing flexibility in the remaining case, where the bounds of the iteration are equal and the expression is not nullable.

We define for non-nullable #REs a numeric *flexibility value* as the main tool for testing whether an iteration is flexible or not. This value is, intuitively, a measure of "stretchability" or "squeezability", which indicates the largest proportional difference between the possible numbers of times that the expression can be satisfied by an input word. As an example consider the expression $E = (a|b^{2..3})^2$, which accepts the language

$$L(E) = \{aa, abb, ab^3, bba, b^3a, b^4, b^5, b^6\} \ .$$

When recognizing an input word like $w = b^{12}$ as a number of occurrences of E, we could "squeeze" the subwords matched by E from b^6 to b^4, giving that both $w \in L(E)^2$ and $w \in L(E)^3$. This ratio $6/4 = 3/2$ is called the flexibility of E, which we formalize below.

Definition 5.1 *Let F be a non-nullable marked #RE. The numeric* flexibility *$fl(F)$ of expression F is defined inductively as follows:*

(1) If $F = x$ for $x \in \Pi$, then $fl(F) = 1$;
(2) If $F = G|H$, then $fl(F) = \max\{fl(G), fl(H)\}$;
(3) If $F = GH$, then $fl(F) = \begin{cases} fl(G) & \text{if } \epsilon \in L(H), \\ fl(H) & \text{if } \epsilon \in L(G), \text{ and} \\ 1 & \text{otherwise}; \end{cases}$
(4) If $F = G^{m..n}$, then $fl(F) = (n/m) \times fl(G)$.

In the last case, if $n = \infty$ or $fl(G) = \infty$, we define that $fl(F) = \infty$. For the second case we define the maximum of ∞ and anything else to be ∞. So, the flexibility is either ∞ or a rational number greater than or equal to one.

Central properties of flexibility values are given below. As preparation for them we present a lemma that states how many times an iteration $F = G^{m..n}$ can be satisfied by a word that satisfies its body G a given number of times:

Lemma 5.2 *Let $F = G^{m..n}$ be a #RE with $m \neq 0$ and $n \neq \infty$.*

(a) *If $w \in L(G)^i$, then $w \in L(F)^j$ for each $j = \lceil i/n \rceil, \ldots, \lfloor i/m \rfloor$.*
(b) *If $w \in L(G)^i$ for each $i = l, \ldots, h$, then $w \in L(F)^j$ for each $j = \lceil l/n \rceil, \ldots, \lfloor h/m \rfloor$.*

Proof. (a) Assume that $w \in L(G)^i$. Let $j \in \{\lceil i/n \rceil, \ldots, \lfloor i/m \rfloor\}$, which requires that $jm \le i \le jn$. To see that $w \in L(F)^j$ we need to show that $w = w_1 \ldots w_j$ such that $w_k \in L(G^{m..n})$ for each $k = 1, \ldots, j$. This holds if

$$i = k_1 + k_2 + \cdots + k_j$$

for some $k_1, \ldots, k_j \in \{m, \ldots, n\}$. This can be shown to hold for any $i \in \{jm, \ldots, jn\}$ by an easy induction on $n - m$.

(b) Assume that $w \in L(G)^i$ for each $i = l, \ldots, h$. Then by item (a) we have that $w \in L(F)^j$ for each $j \in \bigcup_{i=l}^{h} \{\lceil i/n \rceil, \ldots, \lfloor i/m \rfloor\}$. We prove that this union equals the single range $\{\lceil l/n \rceil, \ldots, \lfloor h/m \rfloor\}$ by induction on the length of the range $l, \ldots, h$. For $h = l$ this clearly holds. By the inductive assumption

$$\begin{aligned}&\bigcup_{i=l}^{h+1} \{\lceil i/n \rceil, \ldots, \lfloor i/m \rfloor\} = \\ &\quad \{\lceil l/n \rceil, \ldots, \lfloor h/m \rfloor\} \cup \{\lceil (h+1)/n \rceil, \ldots, \lfloor (h+1)/m \rfloor\} . \qquad (1)\end{aligned}$$

Now $\lceil (h+1)/n \rceil \le \lceil (h+1)/m \rceil = \lfloor h/m \rfloor + 1$, which means that there is no gap between the two ranges in equation (1), and thus their union is

$$\{\lceil l/n \rceil, \ldots, \lfloor (h+1)/m \rfloor\} . \qquad \square$$

Notice that the ranges of j for which $w \in L(F)^j$ can be empty in Lemma 5.2. For example, consider the expression $F = (a^{1..2})^{4..5}$ and its body $G = a^{1..2}$. The word $a^8 \in L(G)^i$ for each $i = 4, \ldots, 8$, and it can be accepted as one or two occurrences of expression F. The corresponding range $\{1, 2\} = \{\lceil 4/5 \rceil, \ldots, \lfloor 8/4 \rfloor\}$ is composed of subranges according to the above proof as follows:

$$\begin{aligned}&\bigcup_{i=4}^{8} \{\lceil i/5 \rceil, \ldots, \lfloor i/4 \rfloor\} = \\ &\quad \{1, \ldots, 1\} \cup \{1, \ldots, 1\} \cup \{2, \ldots, 1\} \cup \{2, \ldots, 1\} \cup \{2, \ldots, 2\} = \\ &\quad \{1\} \cup \{1\} \cup \emptyset \cup \emptyset \cup \{2\}\end{aligned}$$

As the first main result (Corollary 5.4 below) we prove that an infinite flexibility value is a sufficient condition for the flexibility of an iteration. This result follows from the next Lemma 5.3:

Lemma 5.3 *Let F be a non-nullable marked #RE. If $fl(F) = \infty$, there is a word $v \in L(F)$ such that $v^k \in L(F)^i$ for any $k \in \mathbb{N}$ and each $i = 1, \ldots, k$.*

Proof. Assume that $fl(F) = \infty$, and prove the claim by induction on the structure of F. Consider first the case that $F = G^{m..n}$. If $n = \infty$, let $v = u^m$ for any $u \in L(G)$. Then it is easy to see by induction on k that $v^k \in L(F)^i$ for each $i = 1, \ldots, k$. If $n \ne \infty$, then $fl(G) = \infty$. By the inductive assumption there is a word $v \in L(G)$ such that $v^{km} \in L(G)^i$ for each $i = 1, \ldots, km$. Now $v^m \in L(F)$, and $(v^m)^k \in L(F)^j$ (by Lemma 5.2) for each $j = \lceil 1/n \rceil, \ldots, \lfloor (mk)/m \rfloor = 1, \ldots, k$.

In the cases $F = G|H$ and $F = GH$ assume without loss of generality that G is non-nullable and $fl(G) = \infty$. Then the claim follows from the inductive assumption, since $L(G) \subseteq L(F)$. $\square$

Based on the above we see that if the body of an iteration has an infinite flexibility value, the iteration is flexible:

Corollary 5.4 *Let $F = G^{m..n}$ be a non-nullable marked #RE. If $fl(G) = \infty$, then F is flexible.*

Proof. By Lemma 5.3 there is a word $v \in L(G)$ such that $v^n \in L(G)^i$ for each $i = 1, \ldots, n$. Since $v^n \in L(F) \cap L(G)^{n-1}$, this word is a witness to the flexibility of F. $\square$

Next we derive an analogous condition (Corollary 5.6 below) that is sufficient for recognizing the flexibility of iterations whose body has a finite flexibility value. This result is based on the next lemma, which is analogous to Lemma 5.3.

Lemma 5.5 *Let F be a non-nullable marked #RE with $fl(F) \neq \infty$. Then there is a word $v \in L(F)$ such that $v^k \in L(F)^i$ for any $k \in \mathbb{N}$ and each $i = \lceil k/fl(F) \rceil, \ldots, k$.*

Proof. If $fl(F) = 1$, the claim obviously holds. Assume then that $fl(F) > 1$, and show that the claim holds by induction on the structure of F.

For $F = G^{m..n}$ there is, by the inductive assumption, a word $v \in L(G)$ such that $v^{km} \in L(G)^i$ for each $i = \lceil (km)/fl(G) \rceil, \ldots, km$. Now $v^m \in L(F)$, and, according to Lemma 5.2, $v^{km} \in L(F)^j$ for each

$$\begin{aligned} j &= \left\lceil \left\lceil \frac{km}{fl(G)} \right\rceil /n \right\rceil, \ldots, \lfloor (km)/m \rfloor \\ &= \left\lceil \left\lceil \frac{km}{fl(G)} \right\rceil /n \right\rceil, \ldots, k \\ &= \left\lceil \frac{km}{fl(G) \times n} \right\rceil, \ldots, k \qquad (2) \\ &= \lceil k/fl(F) \rceil, \ldots, k \qquad \text{(by } fl(F) = (n/m) \times fl(G)\text{).} \end{aligned}$$

Eq. (2) holds because $f(x) = x/n$ is a continuous and monotonically increasing function such that $f(x) \in \mathbb{Z}$ implies $x \in \mathbb{Z}$, and such functions satisfy $\lceil f(\lceil x \rceil) \rceil = \lceil f(x) \rceil$ (See [38, p. 71]).

In the cases $F = G|H$ and $F = GH$ the claim follows by induction, similarly to the proof of Lemma 5.3. $\square$

Corollary 5.6 *Let $F_1 = G_1^{m_1..n_1}$ be a non-nullable iteration with $fl(G_1) \neq \infty$. Let $F_1 = G_1^{m_1..n_1}, \ldots, F_L = G_L^{m_L..n_L}$ be multiples of F_1 so that F_L is the outermost of them, and let $N = \prod_{i=1}^{L} n_i$. If $fl(G_1) \geq N/(N-1)$, then F_1 is flexible in F_L.*

Proof. By Lemma 5.5 there is a word $v \in L(G_1)$ such that $v^k \in L(G_1)^i$ for any $k \in \mathbb{N}$ and $i = \lceil k/fl(G_1) \rceil, \ldots, k$. Then $v^N \in L(F_L)$ is seen to be a witness to the flexibility of F_1 in F_L as follows: Now $v^N \in L(G_1)^N$ and thus $v^N \in L(F_1)^{N/n_1}$ also. By $N/fl(G_1) \leq N - 1$ and Lemma 5.5 the word v^N belongs to $L(G_1)^{N-1}$, too, and thus $v^N \in L(F_1)^{(N-n_1)/n_1} L(G_1)^{n_1-1}$. □

We continue to show that the numeric condition of Corollary 5.6 gives us not only a sufficient but also a necessary condition (Corollary 5.8 below) for testing the flexibility of iterations with a finite flexibility value. This result follows from the next lemma, which states that the flexibility value is indeed the maximum proportional difference between possible numbers of times that the expression can be satisfied by a word.

Lemma 5.7 *Let F be a non-nullable marked #RE with $fl(F) \neq \infty$. Then*

$$fl(F) = \max\{h/l \mid \exists w \in \Sigma^+ : w \in L(F)^l \cap L(F)^h\} .$$

The claim can be proved in two parts. For this, denote the above maximum value by M. First, $M \geq fl(F)$ follows rather easily from Lemma 5.5. Second, $M \leq fl(F)$ can be shown to hold by induction on the size of expression F. The full proof is given in Appendix A.

Now we are ready to prove that for iterations with an equal minimum and maximum bound also the converse of Corollary 5.6 holds:

Corollary 5.8 *Let $F_1 = G_1^{m_1..n_1}$ be a non-nullable marked iteration with $m_1 = n_1$ and $fl(G_1) \neq \infty$. Let $F_1 = G_1^{m_1..n_1}, \ldots, F_L = G_L^{m_L..n_L}$ be multiples of F_1 so that F_L is the outermost of them, and let $N = \prod_{i=1}^{L} n_i$. If F_1 is flexible in F_L, then $fl(G_1) \geq N/(N-1)$.*

Proof. Let $w \in L(F_1)^l$ be a witness to the flexibility of F_1 in F_L, which means that $w \in L(F_1)^{l'} L(G_1)^k$ for some $l' < l$ and $k < n_1$. Since $L(F_1) = L(G_1)^{n_1}$, these mean that $w \in L(G_1)^i \cap L(G_1)^j$ for $i = l \times n_1$ and $j = l' \times n_1 + k$. From Lemma 5.7, $j \leq i - 1$, and $i \leq N$ then follows that

$$fl(G_1) \geq i/j \geq i/(i-1) \geq N/(N-1) . \qquad \square$$

We argued in Section 4 about the flexibility of some iterations semantically, based on the existence or non-existence of a witness to the flexibility. This may not be possible in general, since the number of potential witnesses could be infinite. The above results let us decide the flexibility of iterations syntactically, by examining the minimum and maximum bounds of the iterations, for example as follows:

Example 5.9 *The iteration* $F_1 = (a^{2..3}|x_1)^2$ *is not flexible, since* $fl((a^{2..3}|x_1)) = \max\{3/2, 1\} = 3/2$*, which is less than* $2/(2-1)$*. The iteration* $F_2 = (a^{2..3}|x_1)^3$ *on the other hand is flexible, by* $fl((a^{2..3}|x_1)) = 3/2 \geq 3/(3-1)$*. Expression* F_1*, which is not locally flexible, is anyhow flexible in the expression* $E_5 = ((a^{2..3}|x_1)^2)^2 x_2$*. This holds by*

$$fl((a^{2..3}|x_1)) = 3/2 > (2 \times 2)/(2 \times 2 - 1) = 4/3 \ .$$

Let us summarize the above results as a complete set of rules for testing whether a subexpression $F_1 = G_1^{m_1..n_1}$ is flexible in a marked expression E:

(1) If $m_1 < n_1$ or if G_1 is nullable, then F_1 is flexible (in E); Otherwise ...
(2) if $fl(G_1) = \infty$, then F_1 is flexible (in E); Otherwise ...
(3) let $F_1 = G_1^{m_1..n_1}, \ldots, F_L = G_L^{m_L..n_L}$ be all the iterations that are multiples of F_1 in E, in order from the innermost to the outermost, and let $N = \prod_{i=1}^{L} n_i$. Then F_1 is flexible in E if and only if $fl(G_1) \geq N/(N-1)$.

If condition (1) or (2) holds, then iteration F_1 is locally flexible, which by Lemma 4.6 implies that F_1 is flexible in expression E, too. The last rule tests whether F_1 is flexible in any of its iterative multiples $F_1, \ldots, F_L$, which by Lemma 4.6 is equivalent to testing whether F_1 is flexible in the whole expression E. Notice that the last rule covers also the case that F_1 has no proper multiples; then $L = 1$ and $N = n_1$.

Once the nullable subexpressions of expression E have been recognized, a single traversal of the expression tree is sufficient for testing the above rules for each iterative subexpression of E. During the traversal we compute and pass upwards either information of the nullability of subexpressions or their flexibility value, and pass downwards to any factors the product of the maximum bounds of their iterative multiples. Thus, altogether, the flexible iterations can be recognized in linear time with respect to the length of the expression E.

6 Analyzing unambiguity of #REs

In this section we develop methods based on *follow relations* for analyzing the unambiguity of #REs, and formally prove them correct. Follow relations that are used for constructing automata (see, e.g., [8] or [10, Sec. 5.2]) from a marked expression E record all pairs of positions (x, y) such that xy can appear as a subword of some $w \in L(E)$. Such pairs of positions (x, y) are called *transitions* of the expression [6]. We say that a pair (x, y) is a *forward transition* of E if $(x, y) \in \mathit{Last}(G) \times \mathit{First}(H)$ for some subexpression $F = GH$ of E. If (x, y) is not a forward transition but $(x, y) \in \mathit{Last}(G) \times \mathit{First}(G)$ for some iterative subexpression $F = G^{m..n}$ of E, we say that it is a *backward transition* of E. Obviously there are no other transitions than forward or backward transitions, and each forward or backward transition of a trim expression can be shown to be indeed a transition.

The use of follow relations for analyzing unambiguity (for example, [33]) is different: any two distinct positions y and z with a common underlying symbol $(y)^\natural = (z)^\natural$ are considered to be a *conflicting pair* and an indication of ambiguity, if for some position x the transitions (x, y) and (x, z) have been recorded in the follow relation. The follow relations that are used for analyzing unambiguity of #REs need to avoid recording some transitions. As a simple example, the expression $E = a_1^2 a_2$ is unambiguous even though both (a_1, a_1) and (a_1, a_2) are transitions of E. The point is that *after processing any prefix of input* it is always clear whether the next input symbol can be matched by a_1 or a_2.

Definition 6.1 *Let E be a marked #RE. We define the* follow relation $\mathit{Foll}_E(F) \subseteq \mathit{Pos}(F) \times \mathit{Pos}(F)$ *for each subexpression F of E inductively as follows:*

- *If $F = \lambda$ or $F = x$ for any $x \in \Pi$, then $\mathit{Foll}_E(F) = \emptyset$;*
- *If $F = G|H$, then $\mathit{Foll}_E(F) = \mathit{Foll}_E(G) \cup \mathit{Foll}_E(H)$;*
- *If $F = GH$, then $\mathit{Foll}_E(F) = \mathit{Foll}_E(G) \cup \mathit{Foll}_E(H) \cup [\mathit{Last}(G) \times \mathit{First}(H)]$;*
- *If $F = G^{m..n}$, then*

$$\mathit{Foll}_E(F) = \begin{cases} \mathit{Foll}_E(G) \cup [\mathit{Last}(G) \times \mathit{First}(G)] & \textit{if } F \textit{ is flexible in } E, \\ \mathit{Foll}_E(G) & \textit{otherwise}; \end{cases}$$

If it is clear from the context that the expression being analyzed for ambiguity is E, we use a simpler notation $\mathit{Foll}(F)$ instead of $\mathit{Foll}_E(F)$.

Observation 6.2 *Let F and G be such subexpressions of a marked expression E that F is a subexpression of G. Then $\mathit{Foll}_E(F) \subseteq \mathit{Foll}_E(G)$.*

Observation 6.3 *All forward transitions of E are recorded in $\mathit{Foll}(E)$.*

Example 6.4 *Follow relations can be used to test for unambiguity as follows:*

(1) The expression $E = (a_1^{3..4}|b_1)^2 a_2$ is ambiguous, which is notified by the underlined pairs in $\mathit{Foll}(E) = \{\underline{(a_1, a_1)}, \underline{(a_1, a_2)}, (b_1, a_2)\}$.
(2) The expression $E = (a_1^{3..4}|b_1)^2 b_2$ is unambiguous; Since $(a_1^{3..4}|b_1)^2$ is not flexible in E, the transitions (a_1, b_1) and (b_1, b_1) are not recorded in $\mathit{Foll}(E)$. Thus there are no conflicting pairs in $\mathit{Foll}(E) = \{(a_1, a_1), (a_1, b_2), (b_1, b_2)\}$.
(3) The iteration $F = (G)^2$ with $G = a_1 x_1 a_2?$ is ambiguous. Since it is not flexible, $\mathit{Foll}(F) = \mathit{Foll}(G) = \{(a_1, x_1), (x_1, a_2)\}$, which contains no conflicting pairs. In order to observe the ambiguity we need to notice the conflict between positions a_1 and a_2 by

$(x_1, a_2) \in \mathit{Foll}(G)$ *and*
$(x_1, a_1) \in \mathit{Last}(G) \times \mathit{First}(G)$.

As discussed above, the follow relation does not record all transitions. On the other hand, it is necessary that *Foll*(E) records those transitions in any subexpression F of E that could, after accepting some input by F, be followed to continue matching input by further positions of F. The pair (x_1, a_2) above is an example of such a transition. This property is formalized and proved in the following lemma.

Lemma 6.5 *Let F be a subexpression of a marked expression E. If*

$$ux \in L(F) \text{ and } uxyr \in L(F)$$

for some $x, y \in \mathit{Pos}(F)$ and $u, r \in \mathit{Pos}(F)^*$, then $(x, y) \in \mathit{Foll}(F)$.

The rather long inductive proof of Lemma 6.5 is given in Appendix B.

The next theorem, which is the main result of this section, presents a correct and complete method for testing unambiguity of #REs.

Theorem 6.6 *Let E be a marked #RE. Expression E is ambiguous if and only if there are*

(A) *$y, z \in \mathit{First}(E)$ such that $y \neq z$ and $(y)^\natural = (z)^\natural$, or*
(B) *$(x, y), (x, z) \in \mathit{Foll}(E)$ such that $y \neq z$ and $(y)^\natural = (z)^\natural$, or*
(C) *an iteration $F = G^{m..n}$ in E with $(x, y) \in \mathit{Foll}(G)$ and $(x, z) \in \mathit{Last}(G) \times \mathit{First}(G)$ such that $y \neq z$ and $(y)^\natural = (z)^\natural$.*

The proof of Theorem 6.6 is a somewhat tedious case-analysis, which is given in Appendix C. Lemma 6.5 is used for arguing that the conditions (A)–(C) are sufficient for recognizing any ambiguous expression.

Based on Theorem 6.6 we can check the unambiguity of an expression E by computing the *First* sets and *Follow* relations during a bottom-up traversal of the expression tree for E, and checking condition (C) for each iteration. Notice that condition (C) needs to be checked for non-flexible iterations only: If F is flexible in E, then based on the construction of *Foll*(F) and Observation 6.2 this condition is covered by condition (B). All of this can be done in low-order polynomial time:

Theorem 6.7 *Let E be a marked expression over a finite alphabet Σ. It can be decided in time $O(|E|^2)$ whether E is unambiguous or not.*

Proof. We discussed in Section 5 how the flexible iterations of E can be recognized in linear total time. The *Follow* relation can be realized by computing for each position $x \in \mathit{Pos}(E)$ a *follow set*, *foll*(x), of positions such that $y \in \mathit{foll}(x)$ if and only if $(x, y) \in \mathit{Foll}(E)$. The *First*, *Last* and *follow* sets can be implemented as linked lists of positions, which are computed during a bottom-up traversal of the expression tree for E. The rules for computing the *First* and *Last* sets were given in Table 1 on page 5. The *follow* sets are updated in two cases: (1) For a subexpression $F = GH$ the positions of $\mathit{First}(H)$ are added to $\mathit{foll}(x)$ for each position $x \in \mathit{Last}(G)$; and (2) for a flexible iteration $G^{m..n}$ each position of $\mathit{First}(G)$ is similarly added to $\mathit{foll}(x)$ for each $x \in \mathit{Last}(G)$, if it is not already there.

We can fix an arbitrary ordering for the symbols of Σ, say, based on the dictionary order of their binary representation. Based on this ordering, the lists can be maintained in increasing order of the underlying symbols of the positions, via implementing unions by merging lists. During a merge, it can be checked in linear time with respect to the length of the resulting list whether two different positions with a common underlying symbol would be included in a *First* or a *follow* list. As soon as this happens, we can report that the expression is ambiguous. This restricts the maximum length of each *First* and *follow* list to $|\Sigma|$. Similarly, condition (C) can be tested by scanning for each position $x \in \mathit{Last}(G)$ the lists $\mathit{foll}(x)$ and $\mathit{First}(G)$, in order to see if they contain two different positions with a common underlying symbol.

Since each of the $O(|E|)$ subexpressions of E has at most $|E|$ last positions, and the maximum length of each *First* and *follow* list is limited by a constant, the total time is $O(|E|^2)$. □

Brüggemann-Klein has shown that 1-unambiguity can be tested in *linear* time for both traditional regular expressions [8] and regular expressions extended with the SGML '&' operator [33]. Her linear-time algorithms are based on transforming the expressions first into so-called *star normal form*, which ensures that any unions performed for computing *follow* sets are disjoint. This transformation does not seem to be directly applicable to #REs, though. For example, consider the expressions $E_1 = (F)^*$ and $E_2 = (F)^2$ with $F = a?b?$. The star normal form of expression E_1 is now $(F^\circ)^*$ where $F^\circ = a|b$. A similar transformation applied to the body of a numeric iteration does not seem to lead to an equivalent "star normal form": For example, expression E_2 accepts the word $abab$, while expression $(F^\circ)^2 = (a|b)^2$ accepts words of length two only. On the other hand, expressions of the form $(F^\circ)^{m..4}$ would accept words like a^4 and b^4, which are rejected by expression E_2.

The star-normal-form transformation eliminates directly nested iterations like $(a^*)^*$, which gets replaced by the equivalent form a^*. On the other hand, replacing nested numeric iterations by a single one generally changes the meaning of the expression. For example, the expression $(a^{4..5})^{1..3}$ describes the language $\{a^4, a^5\} \cup \{a^8, a^9, a^{10}\} \cup \{a^{12}, a^{13}, a^{14}, a^{15}\}$, which cannot be described by a single iteration. In general, elimination of nested numeric iterations may lengthen expressions by an exponential factor [18].

In some situations the commonly made assumption of Theorem 6.7 about a finite alphabet is not justified. For example, XML content models use element tag-names of unlimited length as symbols of their alphabet. In such cases, when the alphabet is rather countable than finite, we get a weaker but still polynomial time bound for checking the unambiguity of expressions:

Theorem 6.8 *Let E be a marked expression over an unlimited alphabet Σ. It can be decided in time $O(|E|^3)$ whether E is unambiguous or not.*

Proof. Let $n = |E|$. Expression E can contain at most n occurrences of alphabet symbols. Thus we can replace them by numbers in $\{1, \ldots, n\}$ in $O(n \log n)$ time by applying any standard dictionary (See, e.g., [39]), and use $\{1, \ldots, n\}$ as the alphabet. Obviously this does not affect the unambiguity of E. After this, the computation of the *First* and *follow* lists can proceed similarly to the proof of Theorem 6.7. Now the length of each list is $O(n)$, which yields the time bound of $O(n^3)$. □

7 Conclusions

We have described and formally justified a polynomial-time procedure for testing the 1-unambiguity of regular expressions with numeric occurrence indicators. Previously published solutions [35,26,36] to this problem either require exponential amounts of resources in the worst case or produce erroneous results.

One-unambiguity can be tested for expressions with a finite alphabet in quadratic time, and for expressions with an unlimited alphabet in cubic time. While satisfactory, these results also leave room for potential improvement. As comparison, the unambiguity of standard regular expressions and SGML content models over a finite alphabet can be tested in linear time [8,33], but these methods do not seem directly applicable to #REs.

The usefulness of unambiguity as a constraint on allowed document content models could be discussed in general. The original intent of unambiguity in SGML was to make content model expressions easier for humans to read [40]. On the other hand, unambiguity has also been criticized as an unnecessary restriction [41]. Some XML schema languages like Relax NG do not require unambiguity. Incidentally, Relax NG does not include numeric occurrence indicators either [42].

It seems that unambiguity makes the efficient matching of #REs essentially easier [32]. According to our observations many publicly available implementations of #REs either fail because of exponential use of resources or produce erroneous results with sufficiently complicated expressions. It would be a worthwhile study to identify specific and useful combinations of extensions and restrictions of regular expressions which lead to both theoretically and practically efficient implementations. We have described earlier a polynomial-time algorithm for performing matching with unrestricted #REs, but this algorithm requires in the worst case quadratic space with respect to the length of the input word [18]. What we would like to have is a practical algorithm for matching #REs, which would run in low-order polynomial time and in constant space.

Acknowledgments

We thank prof. Martti Penttonen for his comments on a draft version of this paper.

References

[1] S. Kleene, Realization of events in nerve sets and finite automata, in: C. Shannon, J. McCarthy (Eds.), Automata Studies, Princeton University Press, Princeton, New Jersey, 1956, pp. 3–42.

[2] IEEE, New York, NY, USA, IEEE Std 1003.1-2001 Standard for Information Technology — Portable Operating System Interface (POSIX) Base Definitions, Issue 6 (2001).

[3] L. Wall, R. Schwartz, Programming perl, O'Reilly & Associates, Inc., Sebastopol, CA, 1991.

[4] H. Thompson, D. Beech, M. Maloney, N. Mendelsohn (Eds.), XML Schema Part 1: Structures, W3C Recommendation, 2001.

[5] A. Brüggemann-Klein, D. Wood, One-unambiguous regular languages, Information and Computation 142 (1998) 182–206.

[6] R. McNaughton, H. Yamada, Regular expressions and state graphs for automata, IRE Transactions on Electronic Computers 9 (1) (1960) 39–47.

[7] V. Glushkov, The abstract theory of automata, Russian Mathematical Surveys 16 (1961) 1–53.

[8] A. Brüggemann-Klein, Regular expressions into finite automata, Theoretical Computer Science 120 (1993) 197–213.

[9] P. Hazel, Perl Compatible Regular Expressions, University of Cambridge, `http://www.pcre.org/` (2003).

[10] A. Aho, Algorithms for finding patterns in strings, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science – Volume A: Algorithms and Complexity, Elsevier/MIT Press, 1994, Ch. 5, pp. 255–300.

[11] A. Aho, R. Sethi, J. Ullman, Compilers, principles, techniques, and tools, Addison-Wesley, 1986.

[12] S. Sippu, E. Soisalon-Soininen, Parsing Theory, Vol. I: Languages and Parsing, Springer-Verlag, 1988.

[13] M. Crochemore, T. Lecroq, Pattern matching and text compression algorithms, in: A. Tucker (Ed.), The Computer Science and Engineering Handbook, CRC Press, 2003, Ch. 8.

[14] A. Meyer, L. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, in: 13th Annual IEEE Symp. on Switching and Automata theory, IEEE, 1972, pp. 125–129.

[15] W. Martens, F. Neven, T. Schwentick, Complexity of decision problems for simple regular expressions, in: J. Fiala, V. Koubek, J. Kratochvíl (Eds.), Proc. of the 29th Intl. Symp. on Mathematical Foundations of Computer Science, Springer-Verlag, 2004, pp. 889–900.

[16] J. Hopcroft, R. Motwani, J. Ullman, Introduction to Automata Theory, Languages, and Computation, 2nd Edition, Addison-Wesley, 2001.

[17] D. Wood, Theory of Computation, John Wiley & Sons, Inc., 1987.

[18] P. Kilpeläinen, R. Tuhkanen, Regular expressions with numerical occurrence indicators—preliminary results, in: Proc. of the Eighth Symposium on Programming Languages and Software Tools, University of Kuopio, Department of Computer Science, 2003, pp. 163–173.

[19] P. Kilpeläinen, Inclusion of unambiguous #REs is NP-hard, unpublished note, University of Kuopio. Available at `http://www.cs.uku.fi/~kilpelai/numRE_incl_is_hard.pdf`. (May 2004).

[20] International Organization for Standardization, ISO 8879: Information Processing—Text and Office Systems—Standard Generalized Markup Language (SGML) (Oct. 1986).

[21] T. Bray, J. Paoli, C. M. Sperberg-McQueen (Eds.), Extensible Markup Language (XML) 1.0, W3C Recommendation, 1998, the latest version is available at `http://www.w3.org/TR/REC-xml`.

[22] M. Murata, D. Lee, M. Mani, Taxonomy of XML schema languages using formal language theory, in: Proceedings of Extreme Markup Languages, Montréal, Québec, 2001, pp. 153–166, available at `http://www.mulberrytech.com/Extreme/Proceedings/`.

[23] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001. The latest version is available at `http://www.w3.org/TR/wsdl`.

[24] G. Alonso, F. Casati, H. Kuno, V. Marhiraju, Web Services—Concepts, Architectures and Applications, Springer-Verlag, 2004.

[25] J. Clark, RELAX NG and W3C XML Schema, A message on the posting list `ietf-xml-use.imc.org`. Available at `http://www.imc.org/ietf-xml-use/mail-archive/msg00217.html` (2002).

[26] M. Fuchs, A. Brown, Supporting UPA on an extension of XML Schema, in: Extreme Markup Languages 2003, IDEAlliance, Montréal, Québec, 2003, available at `http://www.mulberrytech.com/Extreme/Proceedings/`.

[27] H. Thompson, D. Beech, M. Maloney, N. Mendelsohn (Eds.), XML Schema Part 1: Structures Second Edition, W3C Recommendation, 2004.

[28] A. Brüggemann-Klein, D. Wood, The validation of SGML content models, Mathematical and Computer Modelling 25 (4) (1997) 73–84.

[29] P. Kilpeläinen, SGML & XML content models, Markup Languages: Theory & Practice 1 (2) (1999) 53–76.

[30] A. Brown, M. Fuchs, J. Robie, P. Wadler (Eds.), XML Schema: Formal Description, W3C Working Draft, 25 September 2001. The latest version is available at `http://www.w3.org/TR/xmlschema-formal/` .

[31] A.Brown, M. Fuchs, J. Robie, P. Wadler, MSL: a model for W3C XML Schema, Computer Networks 39 (5) (2002) 507–521.

[32] P. Kilpeläinen, R. Tuhkanen, Towards efficient implementation of XML schema content models, in: Proc. of the 2004 ACM Symposium on Document Engineering, ACM Press, 2004, pp. 239–241.

[33] A. Brüggemann-Klein, Unambiguity of extended regular expressions in SGML document grammars, in: T. Lengauer (Ed.), Algorithms — ESA 93, Springer-Verlag, 1993, pp. 73–84.

[34] J. Berstel, J.-E. Pin, Local languages and the Berry-Sethi algorithm, Theoretical Computer Science 155 (1996) 439–446.

[35] H. Thompson, R. Tobin, Using finite state automata to implement W3C XML Schema content model validation and restriction checking, in: XML Europe 2003, IDEAlliance, London, UK, 2003, available at `http://www.ltg.ed.ac.uk/~ht/XML_Europe_2003.html`

[36] C. M. Sperberg-McQueen, Applications of Brzozowski derivatives to XML Schema processing, in: Extreme Markup Languages 2005, IDEAlliance, Montréal, Québec, 2005, available at `http://www.mulberrytech.com/Extreme/Proceedings/`.

[37] J. Brzozowski, Derivatives of regular expressions, Journal of the ACM 11 (4) (1964) 481–494.

[38] R. L. Graham, D. E. Knuth, O. Patashnik, Concrete Mathematics, Addison-Wesley, 1989.

[39] K. Melhorn, A. Tsakalidis, Data structures, in: J. van Leeuwen (Ed.), Handbook of Theoretical Computer Science – Volume A: Algorithms and Complexity, Elsevier/MIT Press, 1994, Ch. 6, pp. 301–341.

[40] C. F. Goldfarb, The SGML Handbook, Clarendon Press, Oxford, 1990.

[41] M. Mani, Keeping chess alive: Do we need 1-unambiguous content models?, talk given at *Extreme Markup Languages 2001* in Montréal. Slides are available at `http://web.cs.wpi.edu/~mmani/publications/extreme2001chess.ppt` (Aug. 2001).

[42] J. Clark, M. Murata (Eds.), RELAX NG Specification, OASIS, 2001, http://www.oasis-open.org/committees/relax-ng/spec-20011203.html.

A Proof of Lemma 5.7

The claim was that if F is a non-nullable marked #RE with $fl(F) \neq \infty$, then

$$fl(F) = \max\{h/l \mid \exists w \in \Sigma^+ : w \in L(F)^l \cap L(F)^h\} \ .$$

Proof. Let $M = \max\{h/l \mid \exists w \in \Sigma^+ : w \in L(F)^l \cap L(F)^h\}$.

In the case that $F = x$ for some $x \in \Pi$ we have $fl(F) = 1$. Thus the claim $fl(F) = M$ clearly holds, since then $w \in L(F)^l \cap L(F)^h$ only if $l = h$. For the inductive cases assume that the claim holds for the proper subexpressions of F. Take an integer k large enough such that $\lceil k/fl(F)\rceil = k/fl(F)$. By Lemma 5.5 there is a word $v \in L(F)$ such that $v^k \in L(F)^{k/fl(F)} \cap L(F)^k$, which implies that $M \geq fl(F)$. Next we show that $M \leq fl(F)$, which entails the claim. To see this, let l and h be integers and $w \in \Sigma^+$ such that $w \in L(F)^l \cap L(F)^h$ and $h/l = M$.

In the case $F = G|H$ the condition $w \in L(F)^l$ implies that w consists of l_1 subwords belonging to $L(G)$ and of l_2 subwords belonging to $L(H)$, for some integers l_1 and l_2 with $l_1 + l_2 = l$. Similarly, $w \in L(F)^h$ means that w consists of h_1 subwords belonging to $L(G)$ and of h_2 subwords belonging to $L(H)$ with $h_1 + h_2 = h$. By the inductive assumption $h_1 \leq l_1 \times fl(G)$ and $h_2 \leq l_2 \times fl(H)$, and thus $M \leq fl(F)$ is seen as follows:

$$\begin{aligned} h/l &= (h_1 + h_2)/(l_1 + l_2) \\ &\leq (l_1 \times fl(G) + l_2 \times fl(H))/(l_1 + l_2) \\ &\leq (l_1 + l_2) \times \max\{fl(G), fl(H)\}/(l_1 + l_2) = fl(F) \end{aligned}$$

Next consider the case $F = GH$. If neither G nor H is nullable, the word $w \in L(F)^l$ consists of subwords $w_1, \ldots, w_l \in L(F)$ of the form $w_i = u_i v_i$ with $u_i \in Pos(G)^+$ and $v_i \in Pos(H)^+$. Therefore $w \in L(F)^l \cap L(F)^h$ only if $l = h$, and thus $M = fl(F) = 1$. Assume then that $\epsilon \in L(G)$. (The case $\epsilon \in L(H)$ is analogous.) Now the word $w \in L(F)^l$ consists of a unique sequence of one or more "blocks" β_i as $w = \beta_1 \ldots \beta_k$, where any non-empty occurrence of G denotes the beginning of the next block β_{i+1}. (For an example, see Fig. A.1.) The boundary of two blocks cannot be spanned by any occurrence of F. Now let the division of w into $w_1 \ldots w_l$ consist of l_i occurrences of F in block β_i, for each $i = 1, \ldots, k$. Similarly, let $w \in L(F)^h$ be witnessed by each of the blocks β_i consisting of h_i occurrences of F. By the inductive assumption

$$\begin{aligned} w &= b\ b\ b \| a\ b\ b\ b \| a\ b\ b\ b | b\ b\ b \in L(F)^4 \\ &= b|b|b\|a\ b|b|b\|a\ b|b|b|b|b|b \in L(F)^{12} \end{aligned}$$

Fig. A.1. A coarse (above) and a fine (below) division of the three blocks b^3, ab^3 and ab^6 of a word $w = b^3ab^3ab^6$ into occurrences of expression $F = a?b^{1..3}$

$h_i \leq fl(H) \times l_i$ for each $i = 1, \ldots, k$. The result $M \leq fl(F)$ is then seen as follows:

$$h/l = \left(\sum_{i=1}^{k} h_i \right) / \ l \leq \left(fl(H) \times \sum_{i=1}^{k} l_i \right) / \ l = fl(F)$$

The last equation above holds by $l = \sum_{i=1}^{k} l_i$ and $fl(F) = fl(H)$.

Finally consider the case $F = G^{m..n}$. Now $w \in L(F)^h$ implies that $w \in L(G)^{h'}$ for some $h' \geq hm$. Similarly $w \in L(F)^l$ implies that $w \in L(G)^{l'}$ for some $l' \leq ln$. Then $M \leq fl(F)$ is seen as follows:

$$\begin{aligned} h/l &\leq \frac{h'/m}{l'/n} = (h'/l') \times (n/m) \\ &\leq fl(G) \times (n/m) = fl(F) \end{aligned}$$ □

B Proof of Lemma 6.5

The claim was that if F is a subexpression of a marked expression E such that

$$ux \in L(F) \text{ and } uxyr \in L(F)$$

for some $x, y \in Pos(F)$ and $u, r \in Pos(F)^*$, then $(x, y) \in Foll(F)$.

Proof. If (x, y) is a forward transition of F, then $(x, y) \in Foll(F)$.

Assume then that (x, y) is a backward transition. That is, there is some iteration $F_i = G_i^{m_i..n_i}$ in F, for which $(x, y) \in Last(G_i) \times First(G_i)$. Let all such subexpressions of F be, in order from the innermost to the outermost, $F_1 = G_1^{m_1..n_1}, \ldots, F_k = G_k^{m_k..n_k}$. If $m_i < n_i$ for any $i = 1, \ldots, k$, then F_i is flexible in E (by Observation 4.3 and by Lemma 4.6), and thus $(x, y) \in Foll(F_i) \subseteq Foll(F)$.

Assume then that $m_i = n_i$ for all $i = 1, \ldots, k$. We show that nevertheless some of iterations $F_1, \ldots, F_k$ is flexible in F, and thus $(x, y) \in Foll(F)$. To see this, let us divide the word ux for each $i = 1, \ldots, k$ as $ux = \alpha_i u_i x$ so that $u_i x$ is the longest suffix of ux that consists of positions of F_i only. This can be done because $x \in Pos(F_i)$ for all $i = 1, \ldots, k$. (Since $Pos(F_{i-1}) \subseteq Pos(F_i)$, we have that $u_{i-1}x$ is a suffix of $u_i x$.) We prove by induction that the following holds for all $i = 1, \ldots, k$: If $u_i x \in L(F_i)^l$ and $u_i xyr_i \in L(F_i)^{l'}$ for some $l' \leq l$ and some prefix r_i of r, then some of iterations $F_1, \ldots, F_i$ is flexible in F. Because $u_k x \in L(F_k)^1$ and $u_k xyr_k \in L(F_k)^1$ for some prefix r_k of r, this entails the claim.

For the base case assume that $u_1 x \in L(F_1)^l$ and $u_1 xyr_1 \in L(F_1)^{l'}$ for some $l' \leq l$ and some prefix r_1 of r. Since $F_1 = (G_1)^{m_1}$ and (x, y) is not a transition of G_1, we have for some $v'_1, \ldots, v'_{l' \times m_1} \in L(G_1)$ that $v'_1 \ldots v'_{h'} = u_1 x$ and $v'_{h'+1} \ldots v'_{l' \times m_1} = yr_1$, for some $h' < l' \times m_1$. Now $u_1 x = v'_1 \ldots v'_{h'} \in L(F_1)^{\lfloor h'/m_1 \rfloor} L(G_1)^{h' \bmod m_1}$. Since $\lfloor h'/m_1 \rfloor < l$ and $h' \bmod m_1 < m_1$, this means that $u_1 x$ is a witness to the flexibility of F_1 in F.

Let then $i > 1$. Assume that $u_i x \in L(F_i)^l$ and $u_i xyr_i \in L(F_i)^{l'}$ where $l' \leq l$ and r_i is some prefix of r. Since $F_i = (G_i)^{m_i}$, this means that $u_i x = v_1 \ldots v_{l \times m_i}$ for some $v_1, \ldots, v_{l \times m_i} \in L(G_i)$, and $u_i xyr_i = v'_1 \ldots v'_{l' \times m_i}$ for some $v'_1, \ldots, v'_{l' \times m_i} \in L(G_i)$. Let $p \in \mathbb{N}$ be minimal for which $u_i x \in L(G_i)^p$. Since $u_i x \in L(G_i)^{l \times m_i}$, we know that $p \leq l \times m_i$. Assume first that $p < l \times m_i$. Then $u_i x$ is seen to be a witness to the flexibility of F_i in F, similarly to the base case. Assume then that $p = l \times m_i$. This implies that $l' = l$ and that $u_i x$, which is a prefix of $v'_1 \ldots v'_{l \times m_i}$, is not a prefix of $v'_1 \ldots v'_{l \times m_i - 1}$. Thus xy is a subword of $v'_{l \times m_i} \in L(G_i)$. Since (x, y) is not a forward transition, xy must be a subword of a word in $L(F_{i-1})$.

Let $h < l \times m_i$ be maximal such that $v_{h+1} \ldots v_{l \times m_i}$ has the word $u_{i-1} x$ as a suffix. Then $u_{i-1} x \in L(F_{i-1})^{l \times m_i - h}$. Similarly, let $h' < l \times m_i$ be maximal such that $v'_{h'+1} \ldots v'_{l \times m_i}$ has $u_{i-1} xyr_i$ as a suffix.

Now both $v_1 \ldots v_h$ and $v'_1 \ldots v'_{h'}$ are prefixes of u_i. If $v_1 \ldots v_h \neq v'_1 \ldots v'_{h'}$, either of the words $v_{h+1}, v'_{h'+1} \in L(G_i)$ contains a symbol of $\mathit{Last}(G_i) - \mathit{Pos}(F_{i-1})$ followed by a word of $L(F_{i-1})$. From this we can show (in the next Lemma B.1) that $\epsilon \in L(G_i)$, and thus F_i is flexible.

Assume then that $v_1 \ldots v_h = v'_1 \ldots v'_{h'}$. Now it must be that $h' \geq h$, since otherwise $u_i x = v'_1 \ldots v'_{h'} v_{h+1} \ldots v_{l \times m_i}$ would consist of less than $p = l \times m_i$ words of G_i, contrary to the previous assumption. Now $u_{i-1} x \in L(F_{i-1})^{l \times m_i - h}$ and $u_{i-1} xyr'_i \in L(F_{i-1})^{l \times m_1 - h'}$ for some prefix r'_i of r_i. By $h' \geq h$ we have that $l \times m_1 - h' \leq l \times m_1 - h$, and thus by the inductive assumption some of iterations $F_1, \ldots, F_{i-1}$ is flexible in F. □

The below lemma completes the above proof for the case where we argued G_i to be nullable:

Lemma B.1 *Let G_i be a marked expression and F_{i-1} a subexpression of G_i which is not a proper subexpression of any iteration in G_i, and $\mathit{First}(F_{i-1}) \subseteq \mathit{First}(G_i)$. If $\alpha z \beta v \gamma \in L(G_i)$ for some words $\alpha, \beta, \gamma \in \mathit{Pos}(G_i)^*$, some position $z \in \mathit{Last}(G_i) - \mathit{Pos}(F_{i-1})$ and some word $v \in L(F_{i-1})$, then $\epsilon \in L(G_i)$.*

Proof. Only two cases arise in the structural induction on G_i. Consider first the case that $G_i = H_1H_2$. If $\alpha z\beta v\gamma' \in L(H_1)$ for some prefix γ' of γ, then $\epsilon \in L(H_1)$ by the inductive assumption. Also $\epsilon \in L(H_2)$ must hold because of $z \in \mathit{Pos}(H_1)$ and $z \in \mathit{Last}(G_i)$. Therefore $\epsilon \in L(H_1)L(H_2) = L(G_i)$. The case that $\alpha' z\beta v\gamma \in L(H_2)$ for some suffix α' of α is similar: Then $\epsilon \in L(H_2)$ by induction, and $\epsilon \in \mathit{First}(H_1)$ must hold since F_{i-1} is a subexpression of H_2 and $\mathit{First}(F_{i-1}) \subseteq \mathit{First}(G_i)$. The third possibility is that $z \in \mathit{Pos}(H_1)$ and F_{i-1} is a subexpression of H_2. By $z \in \mathit{Last}(G_i)$ then $\epsilon \in L(H_2)$, and $\epsilon \in L(H_1)$ holds by $\mathit{First}(F_{i-1}) \subseteq \mathit{First}(G_i)$.

The second case is that $G_i = H_1|H_2$. Then $\alpha z\beta v\gamma \in L(H_j)$ for $j = 1$ or $j = 2$, which by the inductive assumption implies that $\epsilon \in L(H_j) \subseteq L(G_i)$. □

C Proof of Theorem 6.6

The claim was the sufficiency and the correctness of the following conditions for testing the ambiguity of a marked expression E:

(A) $y, z \in \mathit{First}(E)$ such that $y \neq z$ and $(y)^\natural = (z)^\natural$, or
(B) $(x, y), (x, z) \in \mathit{Foll}(E)$ such that $y \neq z$ and $(y)^\natural = (z)^\natural$, or
(C) an iteration $F = G^{m..n}$ in E with $(x, y) \in \mathit{Foll}(G)$ and $(x, z) \in \mathit{Last}(G) \times \mathit{First}(G)$ such that $y \neq z$ and $(y)^\natural = (z)^\natural$.

Proof. Assume first that E is ambiguous, that is, for some $u, v, r \in \mathit{Pos}(E)^*$ and some $y, z \in \mathit{Pos}(E)$ such that $y \neq z$ we have $uyr, uzv \in L(E)$ and $(y)^\natural = (z)^\natural$. We show that then some of conditions (A), (B) or (C) holds. First, if $u = \epsilon$, then obviously condition (A) holds. Assume then that $uxyr, uxzv \in L(E)$, where x is some position of E. If both of (x, y) and (x, z) are forward transitions of E, then condition (B) holds.

Assume then that (x, y) is a backward transition of E. That is, there is some subexpression $F_i = G_i^{m_i..n_i}$ of E, for which $(x, y) \in \mathit{Last}(G_i) \times \mathit{First}(G_i)$. Let all the subexpressions for which this holds be, in order from the innermost to the outermost, $F_1 = G_1^{m_1..n_1}, \ldots, F_k = G_k^{m_k..n_k}$. Similarly to the proof of Lemma 6.5, let us divide the word ux as $ux = \alpha_k u_k x$ such that $u_k x$ is the longest suffix of ux that consists of positions of F_k only.

Consider different possibilities that can cause $uxzv \in L(E)$. First assume that $z \notin \mathit{Pos}(F_k)$. There are two possibilities. First, E can contain a subexpression FG such that F_k is a subexpression of F with $x \in \mathit{Last}(F)$ and $z \in \mathit{First}(G)$. Since $u_k x \in L(F_k)$ and $u_k xyr' \in L(F_k)$ for some prefix r' of r, we have $(x, y) \in \mathit{Foll}(F_k) \subseteq \mathit{Foll}(E)$ by Lemma 6.5, and $(x, z) \in \mathit{Foll}(E)$ by construction. Thus condition (B) holds. Second, E can contain an iteration $F = G^{m..n}$ such that F_k is a subexpression of G and $(x, z) \in \mathit{Last}(G) \times \mathit{First}(G)$. Now $y \notin \mathit{First}(G)$, since otherwise F would be one of $F_1, \ldots, F_k$, which is not possible by $z \notin \mathit{Pos}(F_k)$. Thus again, similarly to the above, we have $(x, y) \in$

$\mathit{Foll}(F_k) \subseteq \mathit{Foll}(G)$, and condition (C) holds.

Then assume that $z \in \mathit{Pos}(F_k)$. If (x, z) is a forward transition in any $G_1, \ldots, G_k$, then $(x, z) \in \mathit{Foll}(G_k)$ and condition (C) holds on iteration F_k. The last possibility is that (x, z) is a backward transition in F_k. Let all the iterations $H_j = I_j^{o_j..p_j}$ for which $(x, z) \in \mathit{Last}(I_j) \times \mathit{First}(I_j)$ be, in order from the innermost to the outermost, $H_1 = I_1^{o_1..p_1}, \ldots, H_l = I_l^{o_l..p_l}$. If $F_i = H_j$ for any $i = 1, \ldots, k$ and $j = 1, \ldots, l$, then $y, z \in \mathit{First}(F_i)$, and either condition (A) or (B) holds. (See Lemma C.1 below.) Let then finally H_l be a proper subexpression of G_1 and $z \notin \mathit{First}(G_1)$. Let $u_l x$ be the longest prefix of ux that consists of positions of H_l only. Then $u_l x \in L(H_l)$, and $u_l x z v' \in L(H_l)$ for some prefix v' of v. This implies by Lemma 6.5 that $(x, z) \in \mathit{Foll}(H_l) \subseteq \mathit{Foll}(G_1)$, and thus condition (C) holds on iteration F_1.

Assume then that some of conditions (A), (B) or (C) holds. If (A) holds, E is obviously ambiguous.

Assume then that condition (B) holds for transitions (x, y) and (x, z). Let F be the minimal subexpression of E for which both $(x, y), (x, z) \in \mathit{Foll}(F)$. The first possibility is that $F = GH$. In this case assume without loss of generality that $(x, z) \in \mathit{Last}(G) \times \mathit{First}(H)$. If $y \in \mathit{First}(H)$, too, then expression F (and thus E) is clearly ambiguous. Otherwise $(x, y) \in \mathit{Foll}(G)$. If (x, y) is a forward transition, F is again easily seen ambiguous. Assume then that (x, y) is a backward transition, that is, $(x, y) \in \mathit{Last}(G_1) \times \mathit{First}(G_1)$ for some subexpression $F_1 = G_1^{m_1..n_1}$ of G which is flexible in E. Let $F_1 = G_1^{m_1..n_1}, \ldots, F_k = G_k^{m_k..n_k}$ be subexpressions of E, in order from the innermost to the outermost, such that $(x, y) \in \mathit{Last}(G_i) \times \mathit{First}(G_i)$ for all $i = 1, \ldots, k$, and $k \geq 1$ is the smallest for which F_1 is flexible in F_k. By Observation 2.5 we have that $\mathit{First}(F_1) \subseteq \mathit{First}(F_k)$ and $\mathit{Last}(F_1) \subseteq \mathit{Last}(F_k)$, and thus some word $w \in L(F_k)$ is a witness to the flexibility of F_1 in F_k. Now there are two possibilities. Either (i) F_k is a subexpression of G, or (ii) $F = GH$ is a subexpression of G_i for some $i = 2, \ldots, k$ such that F_{i-1} is a subexpression of G. In case (i) the word w can be continued, as a subword of a word accepted by $F = GH$, both by $z \in \mathit{First}(H)$ and by $y \in \mathit{First}(G_1)$ (since w is a witness to the flexibility of $F_1 = G_1^{m_1..n_1}$). This means that F is ambiguous. In case (ii) the expression H must be nullable, because $F = GH$ is a subexpression of G_i and $x \in \mathit{Pos}(G) \cap \mathit{Last}(G_i)$. Let then $uv \in L(G_i)$ be a word such that $v \in L(G)$, and consider how it could be continued as a word accepted by $F_i = G_i^{m_i..n_i}$. By $F = GH$ the word uv can continue with $z \in \mathit{First}(H)$. Since the word uv also constitutes an iteration of the body of F_i, it may continue also by $y \in \mathit{First}(G_i)$ (as another iteration of F_i). Therefore F_i is ambiguous.

The second possibility is that the minimal subexpression F of E for which both $(x, y), (x, z) \in \mathit{Foll}(F)$ is an iteration $F = G^{m..n}$. Without loss of generality assume that $(x, z) \in \mathit{Last}(G) \times \mathit{First}(G)$. If (x, y) is a forward transition in G, the expression is again seen ambiguous. Assume then that $(x, y) \in \mathit{Last}(G_1) \times \mathit{First}(G_1)$ for some subexpression $F_1 = G_1^{m_1..n_1}$ of G such that F_1 is flexible in

E. There are two possibilities. First, F_1 can be flexible in G. By Observation 2.5 we have that $\mathit{Last}(F_1) \subseteq \mathit{Last}(G)$. Therefore there is a word $uw \in L(G)$ such that w is a witness to the flexibility of F_1 in G. Then F is ambiguous since it accepts input that begins with uw and continues either by $y \in \mathit{First}(G_1)$ (by the flexibility of $F_1 = G_1^{m_1..n_1}$) or by $z \in \mathit{First}(G)$ (as another iteration of F). Second, F_1 may be flexible in some multiple I of F_1 such that F is a subexpression of I. Then both $y, z \in \mathit{First}(F)$, which makes expression F clearly ambiguous.

Finally, assume that condition (C) holds for transitions (x, y) and (x, z) in an iteration $F = G^{m..n}$. Then F is seen ambiguous similarly to the above analysis. □

Lemma C.1 *Let E be a marked #RE, and let $y, z \in \mathit{First}(F)$ for a subexpression F of E. Then $y, z \in \mathit{First}(E)$ or $(x, y), (x, z) \in \mathit{Foll}(E)$ for some $x \in \mathit{Pos}(E)$.*

Proof. Straightforward induction. □