The Membership Problem for Regular Expressions with Unordered Concatenation and Numerical Constraints

Dag Hovland

Department of Informatics, University of Oslo, Norway
hovland@ifi.uio.no

Abstract. We study the membership problem for regular expressions extended with operators for *unordered concatenation* and *numerical constraints*. The unordered concatenation of a set of regular expressions denotes all sequences consisting of exactly one word denoted by each of the expressions. Numerical constraints are an extension of regular expressions used in many applications, e.g. text search (e.g., UNIX grep), document formats (e.g. XML Schema). Regular expressions with unordered concatenation and numerical constraints denote the same languages as the classical regular expressions, but, in certain important cases, exponentially more succinct. We show that the membership problem for regular expressions with unordered concatenation (without numerical constraints) is already NP-hard. We show a polynomial-time algorithm for the membership problem for regular expressions with numerical constraints and unordered concatenation, when restricted to a subclass called *strongly 1-unambiguous*.

Keywords: Regular expressions, automata, numerical constraints, unordered concatenation, interleaving, XML, SGML.

1 Introduction

In the ISO standard for the Standard Generalized Markup Language (SGML) [1], the precursor of XML, the operator "&" is used for what in this paper is called *unordered concatenation*, that is, the languages are concatenated, but in any order. For example, $\&(ya, basta)$ denotes $\{yabasta, bastaya\}$. In SGML "&" is infix, but because it is not associative, we find it more convenient to write it prefix. Brüggemann-Klein [4,6] investigates unambiguity of regular expressions extended with such an unordered concatenation operator.

Unordered concatenation is superficially similar to *interleaving*, an extension to regular expressions studied by e.g. Mayer & Stockmeyer in [21]. Interleaving is used to model process-theoretic parallel composition. There is no obvious way to translate the algorithms and the complexity results for interleaving to unordered concatenation.

Numerical constraints allow expressing that a subexpression must be matched a number of times specified by a lower and an upper limit. For example, $(a+b)^{2..3}$ denotes the words of length 2 or 3 consisting only of a's and b's. Numerical constraints are also used in XML Schema, and in addition in applications for text search, e.g. `GNU grep`. The extension has been studied by, among others, Gelade et al.[9], Kilpeläinen & Tuhkanen [18], Hovland [15], and Ghelli et al. [12].

This paper has a theoretical and a more practical motivation. The theoretical motivation is curiosity about the properties of unordered concatenation, especially when used in combination with numerical constraints. Unordered concatenation is intuitive and seems useful for searches and definitions in natural language text. *Membership* is shown to be tractable for the *strongly 1-unambiguous* regular expressions with unordered concatenation *and* numerical constraints. It is interesting to note that for the regular expressions with numerical constraints, the largest known subset where membership can be decided in time linear in the size of the word *and* polynomial in the size of the expression, is the strongly 1-unambiguous subset.

In this paper we will study the regular expressions with unordered concatenation and numerical constraints, and the membership problem for these expressions. In the next section we give a definition of these expressions and their languages, and in Sect. 3 we show that the membership problem is NP-complete already without numerical constraints. The algorithm for membership is based on construction of finite automata with counters, where positions in the term trees of the regular expressions play a central role. Section 4 is therefore devoted to a description of term trees and positions in these trees, while in Sect. 5 we define the finite automata with counters. In Sect. 6 we define strong 1-unambiguity, and state the main theorem. The last section presents some related work and a conclusion.

2 Regular Expressions with Unordered Concatenation and Numerical Constraints

Fix an *alphabet* Σ of *letters*. Assume a, b, and c are members of Σ. $l, l_1, l_2, \ldots$ are used as variables for members of Σ. Let $\mathbb{N} = \{1, 2, \ldots\}$, $\mathbb{N}_1 = \{2, 3, 4, \ldots\} \cup \{\infty\}$, and $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$.

Definition 1. *Given an alphabet Σ, $\mathcal{R}_\Sigma$ is the set of* regular expressions with unordered concatenation and numerical constraints *over Σ, defined by the following grammar:*

$$\mathcal{R}_\Sigma ::= \mathcal{R}_\Sigma + \mathcal{R}_\Sigma \mid \mathcal{R}_\Sigma \cdot \mathcal{R}_\Sigma \mid \mathcal{R}_\Sigma^{\mathbb{N}..\mathbb{N}_1} \mid \&(\mathcal{R}_\Sigma, \ldots, \mathcal{R}_\Sigma) \mid \Sigma \mid \epsilon$$

We only allow $r^{n..u}$ for $n \leq u$. Numerical constraints have the highest precedence, followed by concatenation, choice, and unordered concatenation, which has the least precedence. Parentheses are used, when necessary, to group sub-expressions.

We use $r, r_1, r_2, \ldots$ as variables for regular expressions. The sign for concatenation, $\cdot$, will often be omitted. A regular expression denoting the empty language is not included, as this is irrelevant to the results in this paper. We use $r^{n..}$ as shorthand for $r^{n..\infty}$, $r^{0..n}$ as shorthand for $r^{1..n} + \epsilon$, r^+ as shorthand for $r^{1..\infty}$, r^* as shorthand for $r^{0..}$, and r^n for $r^{n..n}$. We denote the set of letters from Σ occurring in r by $\mathsf{sym}(r)$.

The reason the unordered concatenation operator is not binary infix, is that, as we will see below, it is not associative. The *star-free regular expressions with unordered concatenation* are the subset of $\mathcal{R}_\Sigma$ with no numerical constraints, that is, no subexpressions of the form $r^{n..u}$.

We lift concatenation of words to sets of words, such that if $L_1, L_2 \subseteq \Sigma^*$, then $L_1 \cdot L_2 = \{w_1 \cdot w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$. Moreover, ϵ denotes the *empty word* of zero length, such that for all $w \in \Sigma^*$, $\epsilon \cdot w = w \cdot \epsilon = w$. Further, we allow non-negative integers as exponents meaning repeated concatenation, such that for an $L \subseteq \Sigma^*$, we have $L^n = L^{n-1} \cdot L$ for $n > 0$ and $L^0 = \{\epsilon\}$. We define that $n < \infty$ for all numbers n. The semantics of unordered concatenation is defined in terms of permutations. By $\mathsf{Perm}(\{1, \ldots, n\})$ we mean the set of permutations of $\{1, \ldots, n\}$. If $\sigma \in \mathsf{Perm}(\{1, \ldots, n\})$, we assume $\sigma = \sigma_1, \ldots, \sigma_n$. For convenience, we recall in Definition 2 the language denoted by a regular expression, and extend it to unordered concatenation and numerical constraints.

Definition 2 (Language). *The language $\|r\|$ denoted by a regular expression $r \in \mathcal{R}_\Sigma$, is defined in the following inductive way:*

$$
\begin{aligned}
&\|r_1 + r_2\| = \|r_1\| \cup \|r_2\| \\
&\|r_1 \cdot r_2\| = \|r_1\| \cdot \|r_2\| \\
&\|\&(r_1, \ldots, r_n)\| = \textstyle\bigcup_{\sigma \in \mathsf{Perm}(\{1,\ldots,n\})} \|r_{\sigma_1}\| \cdots \|r_{\sigma_n}\| \\
&\|r^{l..u}\| = \textstyle\bigcup_{l \le i \le u} \|r\|^i \\
&\text{for } a \in \Sigma \cup \{\epsilon\}, \|a\| = \{a\}
\end{aligned}
$$

Some examples of regular expressions and their languages are: $\|\&(ab, c)\| = \{abc, cab\}$, $\|\&(a, b, c)\| = \{abc, bac, acb, bca, cab, cba\}$, and $\|(a + b)^{1..2}\| = \{a, b, aa, ab, ba, bb\}$. Note that unordered concatenation is not associative, for example: $\|\&(\&(a, b), c)\| = \{abc, bac, cab, cba\} \neq \{abc, acb, bca, cba\} = \|\&(a, \&(b, c))\|$.

3 Complexity of Membership under Unordered Concatenation

The *membership*-problem is to decide, given a regular expression with unordered concatenation $r \in \mathcal{R}_\Sigma$, and a word $w \in \Sigma^*$, whether $w \in \|r\|$. This is also called *matching*. For regular expressions with numerical constraints (without unordered concatenation), the membership problem is known to be in P [17]. NP-hardness of membership for regular expressions with interleaving was shown by Ogden et al. [23]. The proof cannot easily be modified to fit the case for unordered concatenation.

It is not hard to show that the membership problem for regular expressions with numerical constraints and unordered concatenation is in NP. The certificate for an instance of the problem, consists in making all the necessary choices in the regular expression, such that one can see that the word is in the language. The size of the certificate is polynomial in the lengths of the word and the regular expression. An explicit construction is given in [16, p.53].

To show that membership is NP-hard, we use a reduction from satisfiability of propositional formulas, first shown NP-hard by Cook [7]. By a result of Tseitin [25] we can assume the formulas are in conjunctive normal form. In the remainder of this section we will not use the numerical constraints. The usage of the exponents in the expressions and words in this section is only a shorthand for repeated concatenation. The alphabet consists of the names of the Boolean variables. Given a formula with c clauses and v variables, we construct a regular expression r which is a unordered concatenation of $c + v$ expressions. The first c expressions in the unordered concatenation each represent a clause. In these *clause-expressions*, disjunction is represented by choice ($+$), a positive literal is represented by itself, and a negated literal is represented by concatenating the respective letter with itself $c + 1$ times. The last v expressions in the unordered concatenation, one for each variable x, are of the following form $((x + \epsilon)^c x^{c^2}) + (x^{c+1} + \epsilon)^c$. The word w that we will check for membership, is ${x_1}^{c^2+c} \cdots {x_v}^{c^2+c}$, assuming the variables are $x_1, \ldots, x_v$.

Example 3. For the purpose of an example, let the formula be $(x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \wedge (x_3 \vee \neg x_5 \vee x_6) \wedge (x_3 \vee \neg x_6)$. Then $v = 6$, $c = 3$ and $\Sigma = \{x_1, x_2, x_3, x_4, x_5, x_6\}$. The regular expression becomes $\&((x_1 + x_2^4 + x_3^4 + x_4), (x_3 + x_5^4 + x_6), (x_3 + x_6^4), r_1, r_2, r_3, r_4, r_5, r_6)$, where each r_i is $((x_i + \epsilon)^3 x_i^9) + (x_i^4 + \epsilon)^3$. The word to check membership in the language of this regular expression becomes ${x_1}^{12} \cdots {x_6}^{12}$.

It remains to show that the problem instance of the membership problem is only polynomially larger than the propositional formula, and that the word is in the language of the regular expression if and only if the propositional formula is satisfiable. For reasons of space, these proofs must be left out, but they can be found in [16, p.54-57]. The intuition is that in the choices in the last v parts of the regular expressions, the left choice can be used if the corresponding variable can be true, and the right choice if it can be false, and that in the sub-expressions representing the clauses, the chosen subexpression must be true in the formula.

Note that it is enough for NP-hardness with one single top-level unordered concatenation.

4 Term Trees, Positions, and Subscripting

In this section we will define notation necessary for the later sections. Given a regular expression r, we follow [2] and define the term tree of r as the tree where the root is labeled with the main operator (choice, concatenation, or star) and the subtrees are the term trees of the subexpression(s). If $a \in \Sigma \cup \{\epsilon\}$ the term tree is a single root-node with a as label.

We use $\langle n_1, \ldots, n_k \rangle$, a possibly empty sequence of natural numbers, to denote a position in a term tree. We let p, q, including subscripted variants, be variables for such possibly empty sequences of natural numbers. The position of the root is $\langle\rangle$. If $r = r_1 \cdot r_2$ or $r = r_1 + r_2$, and $n_1 \in \{1, 2\}$, the position $\langle n_1, \ldots, n_k \rangle$ in r is the position $\langle n_2, \ldots, n_k \rangle$ in the subtree of child n_1, that is, in the term tree of r_{n_1}. If $r = {r_1}^*$, the position $\langle 1, n_1, \ldots, n_k \rangle$ in r is the position $\langle n_1, \ldots, n_k \rangle$ in the term tree of r_1. Let $\mathsf{pos}(r)$ be the set of positions in r.

$p \odot q$ will be used for the concatenation of positions p and q. We will also use this notation for concatenating a position with each element in a list of positions, and for concatenating a position with each element of a set of lists of positions.

Whenever concatenating with a position of length one, we will often omit the sign $\odot$ and abbreviate, such that for example $p1 = p \odot \langle 1 \rangle$, $2S = \langle 2 \rangle \odot S$, $ir = \langle i \rangle \odot r$, etc.

For a position p in r we will denote the subexpression rooted at this position by $r[p]$. Note that $r[\langle\rangle] = r$. We also set $r[\epsilon] = \epsilon$. Furthermore, given $p_1, \ldots, p_n$ in $\mathsf{pos}(r) \cup \{\epsilon\}$, put $r[p_1 \cdot \cdots \cdot p_n] = r[p_1] \cdot \cdots \cdot r[p_n]$. Lastly, we lift $r[]$ to sets of string, such that if $S \subseteq \mathsf{pos}(r)^*$, then $r[S] = \{r[w] \mid w \in S\}$.

The concept of *marked expressions* will be important in this paper. It was first used in a similar context by Brüggemann-Klein & Wood [5]. For any regular expression r, let $\mu(r)$ be the marked expression, where every instance of any symbol from Σ is substituted by its position in the expression. Note that, e.g., $\mu(b) = \mu(a) = \langle\rangle$, which shows that marking is not injective. Furthermore $\|\mu(r_1 \cdot r_2)\| = 1\|\mu(r_1)\| \cdot 2\|\mu(r_2)\|$, $\|\mu(r_1 + r_2)\| = 1\|\mu(r_1)\| \cup 2\|\mu(r_2)\|$, and $\|\mu(r^*)\| = 1\|\mu(r)^*\|$.

Example 4. Consider $\Sigma = \{a, b\}$ and $r = (\&(a^2, b))^{3..4}$. Then $\mu(r) = (\&(\langle 1, 1, 1 \rangle^2, \langle 1, 2 \rangle)^{3..4}$. The term trees of r and $\mu(r)$ are shown in Fig. 1.

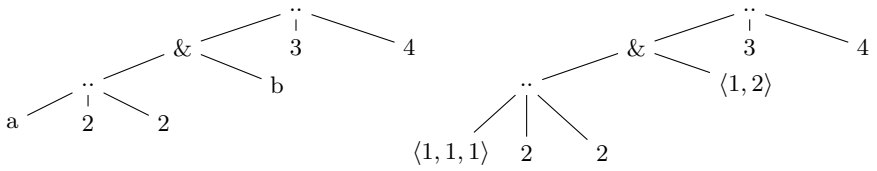

Fig. 1. Term trees for $(\&(a^2, b))^{3..4}$ and $\mu((\&(a^2, b))^{3..4})$

5 Finite Automata with Counters

In this section we describe the finite automata with counters (FAC). FACs are, of course, based on classical finite automata, but extended with a finite set of *counters*. A configuration of the FAC includes a mapping, called *counter state*,

from the counters to the non-negative integers. For subexpressions with numerical constraints we use the counters to keep track of the number of times the subexpression has been matched, and use this to control that the numerical constraints are not violated. For regular expressions with unordered concatenation we use the counters to keep track of which parts of a unordered concatenation have been matched. We keep a counter for every argument in every unordered concatenation. All counters are initially 0. A part of an unordered concatenation can only be used for matching if the corresponding counter is 0, the counter will then be increased to 1. The matching process is only allowed to leave the unordered concatenation when all parts, except those that can match ϵ, have been matched. The counters are then reset to 0.

Let $\mathcal{C}$ be the set of positions of subexpressions we need to keep track of. We model counter states as mappings $\gamma : \mathcal{C} \to \mathbb{N}_0$. Let γ_0 be the counter state in which all counters are 0. We define an *update instruction* ψ as a partial mapping from $\mathcal{C}$ to $\{\mathsf{inc}, \mathsf{res}, \mathsf{one}\}$ (inc for *increment*, res for *reset*, one for setting to 1). Update instructions ψ define mappings f_ψ between counter states in the following way: If $\psi(p) = \mathsf{inc}$, then $f_\psi(\gamma)(p) = \gamma(p) + 1$, if $\psi(p) = \mathsf{res}$ then $f_\psi(\gamma)(p) = 0$, if $\psi(p) = \mathsf{one}$ then $f_\psi(\gamma)(p) = 1$, and otherwise $f_\psi(\gamma)(p) = \gamma(p)$.

Definition 5 (Satisfaction of Update Instructions). *We define a satisfaction relation between update instructions and counter states. Given $\gamma : \mathcal{C} \to \mathbb{N}_0$, $\psi : \mathcal{C} \to \{\mathsf{inc}, \mathsf{res}, \mathsf{one}\}$, $\mathsf{min} : \mathcal{C} \to \mathbb{N}_0$, and $\mathsf{max} : \mathcal{C} \to \mathbb{N}_1$, $\gamma \models^{\mathsf{max}}_{\mathsf{min}} \psi$ is defined by the following inductive rules:*

$$
\begin{array}{lcl}
\gamma \models^{\mathsf{max}}_{\mathsf{min}} \varnothing & & \\
\gamma \models^{\mathsf{max}}_{\mathsf{min}} \psi_1 \cup \{p \mapsto \mathsf{inc}\} & \Leftrightarrow & \gamma \models^{\mathsf{max}}_{\mathsf{min}} \psi_1 \wedge \gamma(p) < \mathsf{max}(p) \\
\gamma \models^{\mathsf{max}}_{\mathsf{min}} \psi_1 \cup \{p \mapsto \mathsf{res}\} & \Leftrightarrow & \gamma \models^{\mathsf{max}}_{\mathsf{min}} \psi_1 \wedge \gamma(p) \geq \mathsf{min}(p) \\
\gamma \models^{\mathsf{max}}_{\mathsf{min}} \psi_1 \cup \{p \mapsto \mathsf{one}\} & \Leftrightarrow & \gamma \models^{\mathsf{max}}_{\mathsf{min}} \psi_1 \wedge \gamma(p) \geq \mathsf{min}(p)
\end{array}
$$

The intuition of Definition 5 is that a counter can only be increased if the value is smaller than the maximum, while a value can only be reset if it's value is at least as large as the minimum. Given mappings max and min, two update instructions are called *overlapping*, if there is a counter state that satisfies both of the update instructions. Overlap can be detected in linear time: For every $p \in \mathcal{C}$ such that p is mapped to inc by one of the update instructions, and p is mapped to either res or one by the other update instruction, it must hold that $\mathsf{min}(p) < \mathsf{max}(p)$.

5.1 Finite Automata with Counters

Definition 6 (Finite Automata with Counters). *A finite automaton with counters (FAC) is a tuple $(\Sigma, Q, \mathcal{C}, \mathcal{A}, \Phi, \mathsf{min}, \mathsf{max}, q^I, \mathcal{F})$. The members of the tuple are described below:*

- *Σ is a finite, non-empty set (the alphabet).*
- *Q and $\mathcal{C}$ are finite sets of* states *and* counters*, respectively.*
- *$q^I \in Q$ is the* initial state*.*

- $\mathcal{A} : Q - \{q^I\} \to \Sigma$ *maps each non-initial state to the letter which is matched when entering the state.*
- Φ *maps each state to a set of pairs of a state and an update instruction.* $\Phi : Q \to \wp(Q \times (\mathcal{C} \to \{\mathsf{inc}, \mathsf{res}, \mathsf{one}\}))$.
- $\mathsf{min} : \mathcal{C} \to \mathbb{N}_0$ *and* $\mathsf{max} : \mathcal{C} \to \mathbb{N}_1$ *are the counter-conditions.*
- $\mathcal{F} \subset Q \times (\mathcal{C} \to \{\mathsf{res}\})$ *describes the* final configurations *(See Definition 7).*

Running or executing an FAC is defined in terms of *transitions* between *configurations*. The configurations of an FAC are pairs, where the first element is a member of Q, and the second element is a counter state.

Definition 7 (Configuration of an FAC). *A* configuration *of an FAC is a pair* (q, γ)*, where* $q \in Q$ *is the current state and* $\gamma : \mathcal{C} \to \mathbb{N}_0$ *is the counter state. A configuration* (q, γ) *is final, if there is* $(q, \psi) \in \mathcal{F}$ *such that* $\gamma \models^{\mathsf{max}}_{\mathsf{min}} \psi$.

Intuitively, the first member of each of the pairs mapped to by Φ, is the state that can be entered, and the second member describes the changes to the current configuration of the automaton in this step. Thus, Φ and $\mathcal{A}$ together describe the possible transitions of the automaton. This is formalized as the transition function δ.

Definition 8 (Transition Function of an FAC). *For an FAC* $(\Sigma, Q, \mathcal{C}, \mathcal{A}, \Phi, q^I, \mathcal{F})$*, the transition function* δ *is defined for any configuration* (q, γ) *and letter* l *by* $\delta((q, \gamma), l) = \{(p, f_\psi(\gamma)) \mid \mathcal{A}(p) = l, (p, \psi) \in \Phi(q), \gamma \models^{\mathsf{max}}_{\mathsf{min}} \psi\}$.

Definition 9 (Deterministic FAC). *An FAC* $(\Sigma, Q, \mathcal{C}, \mathcal{A}, \Phi, q^I, \mathcal{F})$ *is deterministic if and only if* $|\delta((q, \gamma), l)| \leq 1$ *for all* $q \in Q, l \in \Sigma$ *and* $\gamma : \mathcal{C} \to \mathbb{N}_0$.

Deciding whether an FAC is deterministic can be done in polynomial time as follows: For each state p, for each two different (p_1, ψ_1), (p_2, ψ_2) both in $\Phi(p)$, verify that either $\mathcal{A}(p_1) \neq \mathcal{A}(p_2)$ or that ψ_1 and ψ_2 are not overlapping.

5.2 Word recognition

An FAC either *accepts* or *rejects* a given input. A deterministic FAC recognizes a word by treating letters in the word one by one. It starts in the *initial configuration* (q^I, γ_0). An FAC in configuration (q, γ), with letter $l \in \Sigma$ next in the word, will reject the word if $\delta((q, \gamma), l)$ is empty. Otherwise it enters the unique configuration $(q', \gamma') \in \delta((q, \gamma), l)$. If the whole word has been read, a deterministic FAC accepts the word if and only if it is in a final configuration. The subset of Σ^* consisting of words being accepted by an FAC A is denoted $\|A\|$. A deterministic FAC accepts or rejects a word in time linear in the length of the word.

Example 10. Let $\Sigma = \{a, b\}$, $Q = \{q^I, a\langle 1, 1, 1\rangle, b\langle 1, 2\rangle\}$, and $\mathcal{C} = \{\langle 1\rangle, \langle 1, 1\rangle, \langle 1, 1, 1\rangle, \langle 1, 2\rangle\}$. Figure 2 illustrates a deterministic FAC $(\Sigma, Q, \mathcal{C}, \mathcal{A}, \Phi, \mathsf{min}, \mathsf{max}, q^I, \mathcal{F})$ which recognizes $\|(\&(a^2, b))^{3..4}\|$. Note that the names of the non-initial states are decorated with the values of $\mathcal{A}$. Every state is depicted as a rectangle

with the name of the state, and with $\mathcal{F}$ described by the reset instructions. Every member of Φ is shown as an arrow, annotated with the corresponding update instruction. $\mathcal{C}$, min, and max are shown in the top of the figure. The sequence of configurations of this FAC while recognizing *aabbaabaa* is :

$$
\begin{array}{ll}
(q^I, & \gamma_0) \\
(a\langle 1,1,1\rangle, & \{\langle 1\rangle \mapsto 1, \langle 1,1\rangle \mapsto 1, \langle 1,1,1\rangle \mapsto 1, \langle 1,2\rangle \mapsto 0\}) \\
(a\langle 1,1,1\rangle, & \{\langle 1\rangle \mapsto 1, \langle 1,1\rangle \mapsto 1, \langle 1,1,1\rangle \mapsto 2, \langle 1,2\rangle \mapsto 0\}) \\
(b\langle 1,2\rangle, & \{\langle 1\rangle \mapsto 1, \langle 1,1\rangle \mapsto 1, \langle 1,1,1\rangle \mapsto 0, \langle 1,2\rangle \mapsto 1\}) \\
(b\langle 1,2\rangle, & \{\langle 1\rangle \mapsto 2, \langle 1,1\rangle \mapsto 0, \langle 1,1,1\rangle \mapsto 0, \langle 1,2\rangle \mapsto 1\}) \\
(a\langle 1,1,1\rangle, & \{\langle 1\rangle \mapsto 2, \langle 1,1\rangle \mapsto 1, \langle 1,1,1\rangle \mapsto 1, \langle 1,2\rangle \mapsto 1\}) \\
(a\langle 1,1,1\rangle, & \{\langle 1\rangle \mapsto 2, \langle 1,1\rangle \mapsto 1, \langle 1,1,1\rangle \mapsto 2, \langle 1,2\rangle \mapsto 1\}) \\
(b\langle 1,2\rangle, & \{\langle 1\rangle \mapsto 3, \langle 1,1\rangle \mapsto 0, \langle 1,1,1\rangle \mapsto 0, \langle 1,2\rangle \mapsto 1\}) \\
(a\langle 1,1,1\rangle, & \{\langle 1\rangle \mapsto 3, \langle 1,1\rangle \mapsto 1, \langle 1,1,1\rangle \mapsto 1, \langle 1,2\rangle \mapsto 1\}) \\
(a\langle 1,1,1\rangle, & \{\langle 1\rangle \mapsto 3, \langle 1,1\rangle \mapsto 1, \langle 1,1,1\rangle \mapsto 2, \langle 1,2\rangle \mapsto 1\})
\end{array}
$$

The last configuration is final, since $\mathsf{min}(\langle 1\rangle) \leq 3$, $\mathsf{min}(\langle 1,1\rangle) \leq 1$, and $\mathsf{min}(\langle 1,1,1\rangle) \leq 2$.

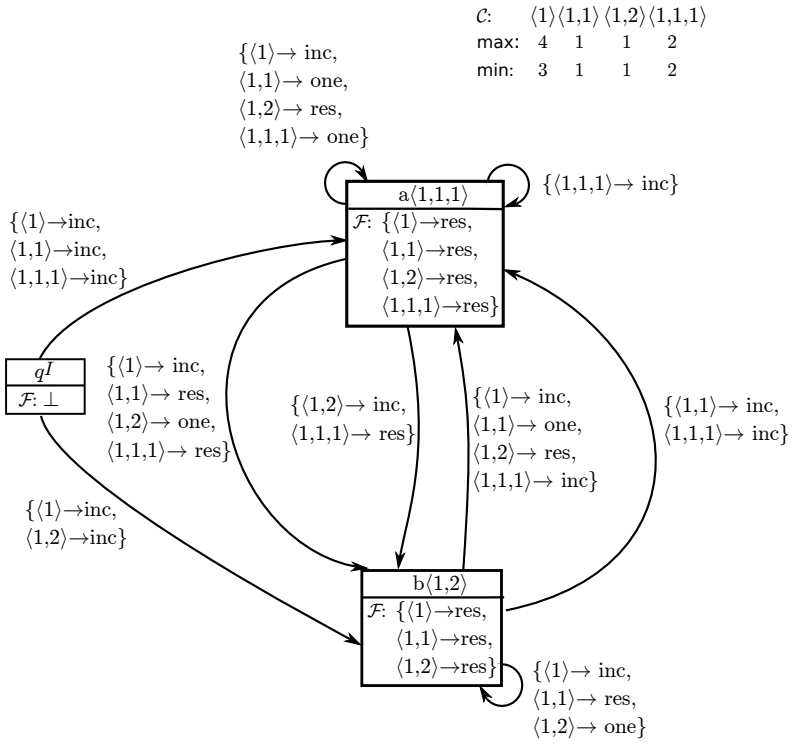

Fig. 2. Illustration of FAC recognizing $\|(\&(a^2, b))^{3..4}\|$

For each letter matched by the FAC, it must test satisfiability of the update instructions corresponding to transitions to the states with a matching letter. Since the sum of these update instructions is smaller than the whole FAC, and testing satisfiability of update instructions is linear, we get the following:

Lemma 11 (Linear-time recognition). *For any deterministic FAC* $A = (\Sigma, Q, \mathcal{C}, \mathcal{A}, \Phi, \mathsf{min}, \mathsf{max}, q^I, \mathcal{F})$*, if* $\sigma(A)$ *is the size of* A*, then for any word* $w \in \Sigma^*$*, the FAC* A *accepts or rejects* w *in time* $O(|w|\sigma(A))$*.*

6 Unambiguity

In this section we will define the right unambiguity we need for constructing deterministic automata. *Strongly 1-unambiguous* regular expressions were first defined by Koch & Scherzinger [20], but the definitions used here also bear on Gelade et al. [9]. A deterministic FAC can be constructed in polynomial time from such expressions. We recall the definition of 1-unambiguity such that the difference with strong 1-unambiguity becomes clear.

Definition 12 (1-unambiguity[3,5]). *A regular expression* r *is 1-unambiguous if for all* $upv, uqw \in \|\mu(r)\|$*, where* $u, v, w \in (\mathsf{pos}(r))^*$ *and* $p, q \in \mathsf{pos}(r)$*,* $r[p] = r[q]$ *implies* $p = q$*.*

Strong 1-unambiguity is needed to prevent unambiguities related to the numerical constraints. For example, $(a^{3..4})^2$ is 1-unambiguous, but there is an ambiguity related to which of the two numerical constraints should be increased when seeing the fourth a in a word. To formalize this ambiguity we will use *subscripted* expressions, and the languages of these. Subscripting is inspired by the *bracketing* used by Koch & Scherzinger [20] and Gelade et al. [9]. The intuition is that for a regular expression r, the subscripted regular expression $\mathsf{ss}(r)$, is such that all subexpressions of the form $r_1^{l..u}$ or $\&(r_1, \ldots, r_n)$ are subscripted with their position in the term tree. For example, $\mathsf{ss}((\&(a^2, b))^{3..4}) = (\&(a^2_{\langle 1,1\rangle}, b)_{\langle 1\rangle})^{3..4}_{\langle\rangle}$.

To define the language of a subscripted expression we will use some more notation: For a position $p = \langle j_1, \ldots, j_n\rangle$, and a positive integer i, pi denotes the position $\langle j_1, \ldots, j_n, i\rangle$. For a regular expression r, let $\Gamma_r = \bigcup_{p\in\mathsf{pos}(r)}\{\uparrow_p, \downarrow_p\}$. For a set L, ϵ^L denotes $\{\epsilon\} \cap L$ and $L^>$ denotes $L - \{\epsilon\}$. The language of a subscripted expression r is a set of strings over $\mathsf{sym}(r) \cup \Gamma_r$. For the not subscripted parts we use the same rules as in Definition 2, while $\|r_p^{l..u}\| = (\bigcup_{i=l}^{u}(\{\uparrow_{p1}\} \cdot \|r\|)^i) \cdot \{\downarrow_{p1}\}$, and $\|\&(r_1, \ldots, r_n)_p\| =$

$$\epsilon^{\|\&(r_1,\ldots,r_n)\|} \cup \left(\bigcup_{\sigma\in\mathsf{Perm}(\{1,\ldots,n\})} \begin{pmatrix} \epsilon^{\|r_{\sigma_1}\|} \cup \{\uparrow_{p\sigma_1}\} \cdot \|r_{\sigma_1}\|^> \\ \cdots \\ \epsilon^{\|r_{\sigma_n}\|} \cup \{\uparrow_{p\sigma_n}\} \cdot \|r_{\sigma_n}\|^> \end{pmatrix}^> \right) \cdot \{\downarrow_{p1} \cdots \downarrow_{pn}\}$$

The ambiguity observed in $(a^{3..4})^2$ corresponds to the fact that there are u, v, w such that both $u \cdot a \cdot \uparrow_{\langle 1,1\rangle} \cdot a \cdot v$ and $u \cdot a \cdot \downarrow_{\langle 1,1\rangle} \cdot \uparrow_{\langle 1\rangle} \cdot \uparrow_{\langle 1,1\rangle} \cdot a \cdot w$ are words in $\|\mathsf{ss}((a^{3..4})^2)\|$.

Definition 13 (Strong 1-unambiguity[9,20]). *A regular expression r is strongly 1-unambiguous if it is 1-unambiguous, and for all $u\alpha av, u\beta bw \in \|\mathsf{ss}(r)\|$, where $a, b \in \mathsf{sym}(r)$, $\alpha, \beta \in \Gamma_r^*$ and $u, v, w \in (\Sigma \cup \Gamma_r)^*$, $a = b$ implies $\alpha = \beta$.*

Examples of expressions that are not strongly 1-unambiguous are $(a^{1..2})^{1..2}$, $(a^*a)^{2..3}$ and $(\&(a^{1..2}, b))^{1..2}$, while $(a + b)^{1..4}$ is strongly 1-unambiguous.

We can now formulate the main result of this paper. The construction of an FAC from a regular expression is based on first, last, and follow sets. There is not space for it in this paper, but it can be seen in [16, p.86-107].

Theorem 14. *For any regular expression r, we can in polynomial time construct an FAC recognizing exactly $\|r\|$. For any word w, and any strongly 1-unambiguous regular expression r, we can in polynomial time decide whether $w \in \|r\|$.*

7 Related Work

The present paper is based on Chapter 3 of Hovland [16]. Proofs, definitions, and examples left out of the present paper for reasons of space can be found in [16]

Sperberg-McQueen [24] has studied regular expressions with numerical constraints and a translation to finite automata with counters, though no proofs are given. Gelade et al. [10,11] and Gelade et al. [9] also wrote about this, including full proofs. The latter was published simultaneously with the paper [15]. The present paper is based on [15], but also incorporates ideas from [9], most notably the bracketing, which was inspiration for the subscripted expressions. Section 6 from [9], including the proofs for Sect. 6 in the Appendix of [9] has inspired some of the content concerning subscripting, strong 1-unambiguity, and proving correctness of the construction of FACs.

Kilpeläinen & Tuhkanen [17,18,19], Gelade [8], Gelade et al. [10,11], and Gelade et al. [9] also investigated properties of the regular expressions with numerical constraints, and give algorithms for membership. Stockmeyer & Meyer [22] study the regular expressions with squaring, a subclass of the regular expressions with numerical constraints. Colazzo, Ghelli & Sartiani, describe in [13] an algorithm for linear-time membership in a subclass of regular expressions called collision-free. The collision-free regular expressions have at most one occurrence of each symbol from Σ, and the counters (and the Kleene star) can only be applied directly to letters or disjunctions of letters. The latter class is strictly included in the class of strongly 1-unambiguous regular expressions.

Extensions of finite automata similar to Finite Automata with Counters have been studied by many authors. The earliest is the treatment of multicounter automata by Greibach [14]. In the multicounter-automata counters can only increase or decrease by 1, and the transition function cannot read the values of the counters. An instruction corresponding to res or one does therefore not exist. Sperberg-McQueen [24] describe the *Counter-extended Finite-state Automata* (CFA) and Gelade et al. [9] describe the *CNFA*. Both of the latter automata

classes use separate expressions for the update instructions (called *actions* by Sperberg-McQueen) and for specifying the conditions/guards. In the FACs in the present paper these guards (or conditions) are implicit, and calculated directly from the update instructions. The language for guards is also quite expressive, and this leads to higher expressive power in the CNFAs and CFAs compared to FACs. Gelade et al. [10,11] describe *NFA(#)*s which have a counter for each state. The FACs described in the present paper are a variant of those described by the author in [15], modified to fit the combination of numerical constraints and unordered concatenation. In the case of unordered concatenation, only the values 0 and 1 are used. The update instruction `one` is new.

Brüggemann-Klein [4,6] gives an algorithm for deciding 1-unambiguity of regular expressions with unordered concatenation. Unordered concatenation is also mentioned in [5,3]. Strong 1-unambiguity has also been mentioned by Brüggemann-Klein & Wood [5,3] and Sperberg-McQueen [24], and Gelade et al. [9]. The first in-depth study of strong 1-unambiguity was by Koch & Scherzinger [20].

8 Conclusion

We have studied the membership problem for regular expressions extended with numerical constraints and with unordered concatenation, an operator similar to "&" in SGML. The membership problem was shown to be NP-complete already without the numerical constraints. We defined *Finite Automata with Counters* (FAC). There is a polynomial-time translation from the regular expressions with numerical constraints and unordered concatenation to FACs. Further we defined *strongly 1-unambiguous regular expressions*, a subset of the regular expressions with numerical constraints and unordered concatenation in constraint normal form, and for which the FAC resulting from the translation is deterministic. The deterministic FAC can recognize the language of the given regular expression in time linear in the size of word to be tested. Testing whether an FAC is deterministic can be done in polynomial time.

References

1. ISO 8879. Information processing — text and office systems — standard generalized markup language (SGML) (October 1986)
2. Bezem, M., Klop, J.W., de Vrijer, R. (eds.): Term Rewriting Systems. Cambridge University Press (2003), `http://www.cs.vu.nl/~terese`
3. Brüggemann-Klein, A.: Regular expressions into finite automata. Theoretical Computer Science 120(2), 197–213 (1993)
4. Brüggemann-Klein, A.: Unambiguity of extended regular expressions in SGML document grammars. In: Lengauer, T. (ed.) ESA. Lecture Notes in Computer Science, vol. 726, pp. 73–84. Springer (1993)
5. Brüggemann-Klein, A., Wood, D.: One-unambiguous regular languages. Information and Computation 140(2), 229–253 (1998)
6. Brüggemann-Klein, A.: Compiler-construction tools and techniques for SGML parsers: Difficulties and solutions (May 1994), `http://xml.coverpages.org/brugg-standardEP-ps.gz`

7. Cook, S.A.: The complexity of theorem-proving procedures. In: STOC. pp. 151–158. ACM (1971)
8. Gelade, W.: Succinctness of regular expressions with interleaving, intersection and counting. In: Ochmanski, E., Tyszkiewicz, J. (eds.) MFCS. Lecture Notes in Computer Science, vol. 5162, pp. 363–374. Springer (2008)
9. Gelade, W., Gyssens, M., Martens, W.: Regular expressions with counting: Weak versus strong determinism. In: Královic, R., Niwinski, D. (eds.) MFCS. Lecture Notes in Computer Science, vol. 5734, pp. 369–381. Springer (2009), http://lrb.cs.uni-dortmund.de/~martens/data/mfcs09-appendix.pdf
10. Gelade, W., Martens, W., Neven, F.: Optimizing schema languages for XML: Numerical constraints and interleaving. In: Schwentick, T., Suciu, D. (eds.) Proceedings of ICDT. Lecture Notes in Computer Science, vol. 4353, pp. 269–283. Springer (2007)
11. Gelade, W., Martens, W., Neven, F.: Optimizing schema languages for XML: Numerical constraints and interleaving. SIAM J. Comput. 38(5), 2021–2043 (2009)
12. Ghelli, G., Colazzo, D., Sartiani, C.: Linear time membership for a class of XML types with interleaving and counting. In: PLAN-X (2008)
13. Ghelli, G., Colazzo, D., Sartiani, C.: Linear time membership in a class of regular expressions with interleaving and counting. In: Shanahan, J.G., Amer-Yahia, S., Manolescu, I., Zhang, Y., Evans, D.A., Kolcz, A., Choi, K.S., Chowdhury, A. (eds.) CIKM. pp. 389–398. ACM (2008)
14. Greibach, S.A.: Remarks on blind and partially blind one-way multicounter machines. Theor. Comput. Sci. 7, 311–324 (1978)
15. Hovland, D.: Regular expressions with numerical constraints and automata with counters. In: Leucker, M., Morgan, C. (eds.) ICTAC. Lecture Notes in Computer Science, vol. 5684, pp. 231–245. Springer (2009)
16. Hovland, D.: Feasible Algorithms for Semantics — Employing Automata and Inference Systems. Ph.D. thesis, Universitetet i Bergen (2010), http://hdl.handle.net/1956/4325
17. Kilpeläinen, P., Tuhkanen, R.: Regular expressions with numerical occurrence indicators - preliminary results. In: Kilpeläinen, P., Päivinen, N. (eds.) SPLST. pp. 163–173. University of Kuopio, Department of Computer Science (2003)
18. Kilpeläinen, P., Tuhkanen, R.: Towards efficient implementation of XML schema content models. In: Munson, E.V., Vion-Dury, J.Y. (eds.) ACM Symposium on Document Engineering. pp. 239–241. ACM (2004)
19. Kilpeläinen, P., Tuhkanen, R.: One-unambiguity of regular expressions with numeric occurrence indicators. Information and Computation 205(6), 890–916 (2007)
20. Koch, C., Scherzinger, S.: Attribute grammars for scalable query processing on XML streams. VLDB J. 16(3), 317–342 (2007)
21. Mayer, A.J., Stockmeyer, L.J.: Word problems-this time with interleaving. Inf. Comput. 115(2), 293–311 (1994)
22. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential space. In: Proceedings of FOCS. pp. 125–129. IEEE (1972)
23. Ogden, W.F., Riddle, W.E., Rounds, W.C.: Complexity of expressions allowing concurrency. In: POPL. pp. 185–194 (1978)
24. Sperberg-McQueen, C.M.: Notes on finite state automata with counters (2004), http://www.w3.org/TR/xml/
25. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in Constructive Mathematics and Mathematical Logic, part 2 pp. 115–225 (1968)

Appeared in: Adrian-Horia Dediu, Carlos Martín-Vide (Eds): Language and Automata Theory and Applications, 6th International Conference, LATA 2012, A Coruña, Spain, March 5-9, 2012. The original publication is available at `www.springerlink.com`.