Regular Expressions with Counting: Weak versus Strong Determinism

Non Peer-reviewed author version

GELADE, Wouter; GYSSENS, Marc & MARTENS, Wim (2012) Regular Expressions with Counting: Weak versus Strong Determinism. In: SIAM JOURNAL ON COMPUTING, 41(1), p. 160-190.

DOI: 10.1137/100814196
Handle: http://hdl.handle.net/1942/13185

Regular Expressions with Counting: Weak versus Strong Determinism

Wouter Gelade[1]*, Marc Gyssens[1], and Wim Martens[2]**

[1] Hasselt University and Transnational University of Limburg
School for Information Technology
{firstname.lastname}@uhasselt.be
[2] Technical University of Dortmund
{firstname.lastname}@udo.edu

Abstract. We study deterministic regular expressions extended with the counting operator. There exist two notions of determinism, strong and weak determinism, which almost coincide for standard regular expressions. This, however, changes dramatically in the presence of counting. In particular, we show that weakly deterministic expressions with counting are exponentially more succinct and strictly more expressive than strongly deterministic ones, even though they still do not capture all regular languages. In addition, we present a finite automaton model with counters, study its properties and investigate the natural extension of the Glushkov construction translating expressions with counting into such counting automata. This translation yields a deterministic automaton if and only if the expression is strongly deterministic. These results then also allow to derive upper bounds for decision problems for strongly deterministic expressions with counting.

1 Introduction

The use of regular expressions (REs) is quite widespread and includes applications in bioinformatics [17], programming languages [23], model checking [22], XML schema languages [21], etc. In many cases, the standard operators are extended with additional ones to facilitate usability. A popular such operator is the counting operator allowing for expressions of the form "$a^{2,4}$", defining strings containing at least two and at most four a's, which is used for instance in Egrep [9] and Perl [23] patterns and in the XML schema language XML Schema [21].

In addition to expanding the vocabulary of REs, subclasses of REs have been investigated to alleviate, e.g., the matching problem. For instance, in the context of XML and SGML, the strict subclasses of weakly and strongly deterministic regular expressions have been introduced. Weak determinism (also called one-unambiguity [2]) intuitively requires that, when matching a string from left to

* Research Assistant of the Fund for Scientific Research – Flanders (Belgium).
** Supported by the North-Rhine Westphalian Academy of Sciences, Humanities and Arts; and the Stiftung Mercator Essen.

right against an expression, it is always clear against which position in the expression the next symbol must be matched. For example, the expression $(a+b)^*a$ is not weakly deterministic, but the equivalent expression $b^*a(b^*a)^*$ is. Strong determinism intuitively requires additionally that it is also clear *how* to go from one position to the next. For example, $(a^*)^*$ is weakly deterministic, but not strongly deterministic since it is not clear over which star one should iterate when going from one a to the next.

While weak and strong determinism coincide for standard regular expressions [1][3], this situation changes completely when counting is involved. Firstly, the algorithm for deciding whether an expression is weakly deterministic is non-trivial [13]. For instance, $(b?a^{2,3})^{2,2}b$ is weakly deterministic, but the very similar $(b?a^{2,3})^{3,3}b$ is not. So, the amount of non-determinism introduced depends on the concrete values of the counters. Second, as we will show, weakly deterministic expressions with counting are strictly more expressive than strongly deterministic ones. Therefore, the aim of this paper is an in-depth study of the notions of weak and strong determinism in the presence of counting w.r.t. expressiveness, succinctness, and complexity. In particular, our contributions are the following:

- We give a complete overview of the expressive power of the different classes of deterministic expressions with counting. We show that strongly deterministic expressions with counting are equally expressive as standard deterministic expressions. Weakly deterministic expressions with counting, on the other hand, are more expressive than strongly deterministic ones, except for unary languages, on which they coincide. However, not all unary regular languages are definable by weakly deterministic expressions with counting (Section 3).
- We investigate the difference in succinctness between strongly and weakly deterministic expressions with counting, and show that weakly deterministic expressions can be exponentially more succinct than strongly deterministic ones. This result prohibits an efficient algorithm translating a weakly deterministic expression into an equivalent strongly deterministic one, if such an expression exists. This contrasts with the situation of standard expressions where such a linear time algorithm exists [1] (Section 4).
- We present an automaton model extended with counters, counter NFAs (CNFAs), and investigate the complexity of some related problems. For instance, it is shown that boolean operations can be applied efficiently to CDFAs, the deterministic counterpart of CNFAs (Section 5).
- Bruggemann-Klein [1] has shown that the Glushkov construction, translating regular expressions into NFAs, yields a DFA if and only if the original expression is deterministic. We investigate the natural extension of the Glushkov construction to expressions with counters, converting expressions to CNFAs. We show that the resulting automaton is deterministic if and only if the original expression is strongly deterministic (Section 6).

[3] Brüggemann-Klein [1] did not study strong determinism explicitly, although she did study strong unambiguity. However, she gives a procedure to transform expressions into *star normal form* which rewrites weakly deterministic expressions into equivalent strongly deterministic ones in linear time.

– Combining the results of Section 5, concerning CDFAs, with the latter result then also allows to infer better upper bounds on the inclusion and equivalence problem of strongly deterministic expressions with counting. Further, we show that testing whether an expression with counting is strongly deterministic can be done in cubic time, as is the case for weak determinism [13] (Section 7).

The original motivation for this work comes from the XML schema language XML Schema, which uses weakly deterministic expressions with counting. However, it is also noted by Sperberg-McQueen [20], one of its developers, that *"Given the complications which arise from [weakly deterministic expressions], it might be desirable to also require that they be strongly deterministic as well [in XML Schema]."* The design decision for weak determinism is probably inspired by the fact that it is the natural extension of the notion of determinism for standard expressions, and a lack of a detailed analysis of their differences when counting is allowed. A detailed examination of strong and weak determinism of regular expressions with counting intends to fill this gap.

Related work: Apart from the work already mentioned, there are several automata based models for different classes of expressions with counting with as main application XML Schema validation, by Kilpelainen and Tuhkanen [12], Zilio and Lugiez [4], and Sperberg-McQueen [20]. Here, Sperberg-McQueen introduces the extension of the Glushkov construction which we study in Section 6. We introduce a new automata model in Section 5 as none of these models allow to derive all results in Sections 5 and 6. Further, Sperberg-McQueen [20] and Koch and Scherzinger [14] introduce a (slightly different) notion of strongly deterministic expression with and without counting, respectively. We follow the semantic meaning of Sperberg-McQueen's definition, while using the technical approach of Koch and Scherzinger. Finally, Kilpelainen [10] shows that inclusion for weakly deterministic expressions with counting is coNP-hard; and Colazzo, Ghelli, and Sartiani [3] have investigated the inclusion problem involving subclasses of deterministic expressions with counting. Seidl et al. also investigate counting constraints in XML schema languages by adding Presburger constraints to regular languages [18]. Concerning deterministic languages without counting, the seminal paper is by Bruggemann-Klein and Wood [2] where, in particular, it is shown to be decidable whether a language is definable by a deterministic regular expression. Conversely, general regular expressions with counting have also received quite some attention [7, 8, 11, 16].

2 Preliminaries

Let $\mathbb{N}$ denote the natural numbers $\{0, 1, 2, \ldots\}$. For the rest of the paper, Σ always denotes a finite alphabet. The set of regular expressions over Σ, denoted by $\mathrm{RE}(\Sigma)$, is defined as follows: ε and every Σ-symbol is in $\mathrm{RE}(\Sigma)$; and whenever r and s are in $\mathrm{RE}(\Sigma)$, then so are (rs), $(r + s)$, and $(s)^*$. For readability, we usually omit parentheses in examples. The language defined by a regular expression r, denoted by $L(r)$, is defined as usual. By $\mathrm{RE}(\Sigma,\#)$ we denote $\mathrm{RE}(\Sigma)$

extended with *numerical occurrence constraints* or *counting*. That is, when r is an RE(Σ,#)-expression then so is $r^{k,\ell}$ for $k \in \mathbb{N}$ and $\ell \in \mathbb{N}_0 \cup \{\infty\}$ with $k \leq \ell$. Here, $\mathbb{N}_0$ denotes $\mathbb{N} \setminus \{0\}$. Furthermore, $L(r^{k,\ell}) = \bigcup_{i=k}^{\ell} L(r)^i$. We use $r?$ to abbreviate $(r+\varepsilon)$. Notice that r^* is simply an abbreviation for $r^{0,\infty}$. Therefore, we do not consider the $*$-operator in the context of RE(Σ,#). The *size* of a regular expression r in RE(Σ,#), denoted by $|r|$, is the number of Σ-symbols and operators occurring in r plus the sizes of the binary representations of the integers. An RE(Σ,#) expression r is *nullable* if $\varepsilon \in L(r)$. We say that an RE(Σ,#) r is in *normal form* if for every nullable subexpression $s^{k,l}$ of r we have $k = 0$. Any RE(Σ,#) can easily be normalized in linear time. Therefore, we assume that all expressions used in this paper are in normal form. Sometimes we will use the following observation, which follows directly from the definitions:

Remark 1. A subexpression $r^{k,\ell}$ is nullable if and only if $k = 0$.

Weak determinism. For an RE(Σ,#) r, let Char(r) be the set of Σ-symbols occurring in r. A *marked regular expression with counting over Σ* is a regular expression over $\Sigma \times \mathbb{N}$ in which every $(\Sigma \times \mathbb{N})$-symbol occurs at most once. We denote the set of all these expressions by MRE(Σ,#). Formally, $\overline{r} \in$ MRE(Σ,#) if $\overline{r} \in$ RE($\Sigma \times \mathbb{N}$,#) and, for every subexpression $\overline{s}\,\overline{s}'$ or $\overline{s} + \overline{s}'$ of $\overline{r}$, Char($\overline{s}$) $\cap$ Char($\overline{s}'$) $= \emptyset$. A *marked string* is a string over $\Sigma \times \mathbb{N}$ (in which $(\Sigma \times \mathbb{N})$-symbols can occur more than once). When $\overline{r}$ is a marked regular expression, $L(\overline{r})$ is therefore a set of marked strings.

The demarking of a marked expression is obtained by deleting these integers. Formally, the demarking of $\overline{r}$ is dm($\overline{r}$), where dm : MRE(Σ,#) $\to$ RE(Σ,#) is defined as dm(ε) := ε, dm((a,i)) := a, dm($\overline{rs}$) := dm($\overline{r}$)dm($\overline{s}$), dm($\overline{r+s}$) := dm($\overline{r}$) + dm($\overline{s}$), and dm($\overline{r^{k,\ell}}$) := dm($\overline{r}$)$^{k,\ell}$. Any function m : RE(Σ,#) $\to$ MRE(Σ,#) such that for every $r \in$ RE(Σ,#) it holds that dm(m(r)) $= r$ is a valid *marking* function. For conciseness and readability, we will from now on write a_i instead of (a,i) in marked regular expressions. For instance, a *marking* of $(a+b)^{1,2}a+bc$ is $(a_1+b_1)^{1,2}a_2+b_2c_1$. The markings and demarkings of strings are defined analogously. For the rest of the paper, we usually leave the actual marking function m implicit and denote by $\overline{r}$ a marking of the expression r. Likewise $\overline{w}$ will denote a marking of a string w. We always use overlined letters to denote marked expressions, symbols, and strings.

Definition 2. An RE(Σ,#) expression r is *weakly deterministic* (also called *one-unambiguous*) if, for all strings $\overline{u}, \overline{v}, \overline{w} \in$ Char($\overline{r}$)* and all symbols $\overline{a}, \overline{b} \in$ Char($\overline{r}$), the conditions $\overline{uav}, \overline{ubw} \in L(\overline{r})$ and $\overline{a} \neq \overline{b}$ imply that $a \neq b$.

A regular language is *weakly deterministic with counting* if it is defined by some weakly deterministic RE(Σ,#) expression. The classes of all weakly deterministic languages with counting, respectively, without counting, are denoted by $\mathrm{DET}_W^{\#}(\Sigma)$, respectively, $\mathrm{DET}_W(\Sigma)$.

Intuitively, an expression is weakly deterministic if, when matching a string against the expression from left to right, we always know against which symbol in the expression we must match the next symbol, without looking ahead in the

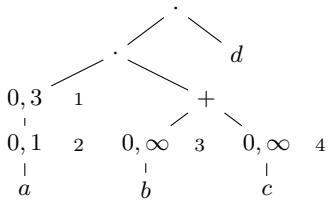

Fig. 1. Parse tree of $(a^{0,1})^{0,3}(b^{0,\infty} + c^{0,\infty})d$. Counter nodes are numbered from 1 to 4.

string. For instance, $(a + b)^*a$ and $(a^{2,3} + b)^{3,3}b$ are not weakly deterministic, while $b^*a(b^*a)^*$ and $(a^{2,3} + b)^{2,2}b$ are.

Strong determinism. Intuitively, an expression is weakly deterministic if, when matching a string from left to right, we always know *where* we are in the expression. For a strongly deterministic expression, we will additionally require that we always know *how* to go from one position to the next. Thereto, we distinguish between going *forward* in an expression and *backward* by *iterating* over a counter. For instance, in the expression $(ab)^{1,2}$ going from a to b implies going forward, whereas going from b to a iterates backward over the counter.

Therefore, an expression such as $((a+\varepsilon)(b+\varepsilon))^{0,2}$ will not be strongly deterministic, although it is weakly deterministic. Indeed, when matching ab, we can go from a to b by either going forward or by iterating over the counter. By the same token, also $(a^{1,2})^{3,4}$ is not strongly deterministic, as we have a choice of counters over which to iterate when reading multiple a's. Conversely, $(a^{2,2})^{3,4}$ is strongly deterministic as it is always clear over which counter we must iterate.

For the definition of strong determinism, we follow the semantic meaning of the definition by Sperberg-McQueen [20], while using the formal approach of Koch and Scherzinger [14] (who called the notion *strong one-unambiguity*)[4]. We denote the *parse tree* of an RE(Σ,#) expression r by $\mathrm{pt}(r)$. Figure 1 contains the parse tree of the expression $(a^{0,1})^{0,3}(b^{0,\infty} + c^{0,\infty})d$.

A *bracketing of a regular expression* r is a labeling of the counter nodes of $\mathrm{pt}(r)$ by distinct indices. Concretely, we simply number the nodes according to the depth-first left-to-right ordering. The bracketing $\widetilde{r}$ of r is then obtained by replacing each subexpression $s^{k,\ell}$ of r with index i with $([_i s]_i)^{k,\ell}$. Therefore, a bracketed regular expression is a regular expression over alphabet $\Sigma \uplus \Gamma$, where $\Gamma := \{[_i,]_i \mid i \in \mathbb{N}\}$. For example, $([_1([_2a]_2)^{0,1}]_1)^{0,3}(([_3b]_3)^{0,\infty} + ([_4c]_4)^{0,\infty})d$ is a bracketing of $(a^{0,1})^{0,3}(b^{0,\infty}+c^{0,\infty})d$, for which the parse tree is shown in Figure 1. We say that a string w in $\Sigma \uplus \Gamma$ is *correctly bracketed* if w has no substring of the form $[_i]_i$. That is, we do not allow a derivation of ε in the derivation tree.

Definition 3. A regular expression r is strongly deterministic with counting if r is weakly deterministic and there do not exist strings u, v, w over $\Sigma \cup \Gamma$, strings

[4] The difference with Koch and Scherzinger is that we allow different derivations of ε while they forbid this. For instance, $a^* + b^*$ is strongly deterministic in our definition, but not in theirs, as ε can be matched by both a^* and b^*.

$\alpha \neq \beta$ over Γ, and a symbol $a \in \Sigma$ such that $u\alpha av$ and $u\beta aw$ are both correctly bracketed and in $L(\widetilde{r})$.

A standard regular expression (without counting) is strongly deterministic if the expression obtained by replacing each subexpression of the form r^* with $r^{0,\infty}$ is strongly deterministic with counting. The class $\mathrm{DET}_S^{\#}(\Sigma)$, respectively, $\mathrm{DET}_S(\Sigma)$, denotes all languages definable by a strongly deterministic expressions with, respectively, without, counting.

3 Expressive power

Brüggemann-Klein and Wood [2] proved that for any alphabet Σ $\mathrm{DET}_W(\Sigma)$ forms a strict subclass of the regular languages, denoted $\mathrm{REG}(\Sigma)$. The complete picture of the relative expressive power depends on the size of Σ, as shown by the following theorem.

Theorem 4. *For every alphabet Σ,*

$$DET_S(\Sigma) = DET_W(\Sigma) = DET_S^{\#}(\Sigma) = DET_W^{\#}(\Sigma) \subsetneq REG(\Sigma) \ \ (\textit{if } |\Sigma| = 1)$$

$$DET_S(\Sigma) = DET_W(\Sigma) = DET_S^{\#}(\Sigma) \subsetneq DET_W^{\#}(\Sigma) \subsetneq REG(\Sigma) \ \ (\textit{if } |\Sigma| \geq 2)$$

Proof. The equality $\mathrm{DET}_S(\Sigma) = \mathrm{DET}_W(\Sigma)$ is already implicit in the work of Brüggemann-Klein [1]. By this result and by definition, all inclusions from left to right already hold. It therefore suffices to show that (1) $\mathrm{DET}_S^{\#}(\Sigma) \subseteq \mathrm{DET}_S(\Sigma)$ for arbitrary alphabets, (2) $\mathrm{DET}_W^{\#}(\Sigma) \subseteq \mathrm{DET}_S^{\#}(\Sigma)$ for unary alphabets, (3) $\mathrm{DET}_S^{\#}(\Sigma) \subsetneq \mathrm{DET}_W^{\#}(\Sigma)$ for binary alphabets, and (4) $\mathrm{DET}_W^{\#}(\Sigma) \subsetneq \mathrm{REG}(\Sigma)$ for unary alphabets.
(1): We show that each strongly deterministic expression with counting can be transformed into a strongly deterministic expression without counting. This is quite non-trivial, but the crux is to unfold each counting operator in a smart manner, taking special care of nullable expressions.
(2): The crux of this proof lies in Lemma 5. It is well known and easy to see that the minimal DFA for a regular language over a unary alphabet is defined either by a simple chain of states (sometimes also called a *tail* [19]), or a chain followed by a cycle.The languages in $\mathrm{DET}_W^{\#}(\Sigma)$ can be defined in this manner. The following lemma adds to that, that for weakly deterministic regular expressions, only one node in this cycle can be final. The theorem then follows as any such language can be defined by a strongly deterministic expression.

Lemma 5. *Let $\Sigma = \{a\}$, and $L \in REG(\Sigma)$, then $L \in DET_W^{\#}(\Sigma)$ if and only if L is definable by a DFA which is either a chain, or a chain followed by a cycle, for which at most one of the cycle nodes is final.*

(3 and 4): Witnesses for non-inclusion are the languages defined by $(a^{2,3}b?)^*$ and $(aaa)^*(a + aa)$, respectively. Both languages can be shown not to be in $\mathrm{DET}_W(\Sigma)$ [2]. The theorem then follows from the above results.

4 Succinctness

In Section 3 we learned that $\mathrm{DET}^{\#}_{W}(\Sigma)$ strictly contains $\mathrm{DET}^{\#}_{S}(\Sigma)$, prohibiting a translation from weak to strong deterministic expressions with counting. However, one could still hope for an efficient algorithm which, given a weakly deterministic expression known to be equivalent to a strong deterministic one, constructs this expression. However, this is not the case:

Theorem 6. *For every $n \in \mathbb{N}$, there exists an $r \in RE(\Sigma,\#)$ over alphabet $\{a\}$ which is weakly deterministic and of size $\mathcal{O}(n)$ such that every strongly deterministic expression s, with $L(r) = L(s)$, is of size at least 2^n.*

The above theorem holds for the family of languages defined by $(a^{2^n+1,2^{n+1}})^{1,2}$, each of which is weakly deterministic and defines all strings with a's of length from $2^n + 1$ to 2^{n+2}, except for the string $a^{2^{n+1}+1}$. These expressions, in fact, where introduced by Kilpelainen when studying the inclusion problem for weakly deterministic expressions with counting [10].

5 Counter automata

Let C be a set of *counter variables* and $\alpha : C \to \mathbb{N}$ be a function assigning a value to each counter variable. We inductively define *guards* over C, denoted $\mathrm{Guard}(C)$, as follows: for every $\mathrm{cv} \in C$ and $k \in \mathbb{N}$, we have that true, false, $\mathrm{cv} = k$, and $\mathrm{cv} < k$ are in $\mathrm{Guard}(C)$. Moreover, when $\phi_1, \phi_2 \in \mathrm{Guard}(C)$, then so are $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, and $\neg\phi_1$. For $\phi \in \mathrm{Guard}(C)$, we denote by $\alpha \models \phi$ that α models ϕ, i.e., that applying the value assignment α to the counter variables results in satisfaction of ϕ.

An *update* is a set of statements of the form cv++ and reset(cv) in which every $\mathrm{cv} \in C$ occurs at most once. By $\mathrm{Update}(C)$ we denote the set of all updates.

Definition 7. A non-deterministic *counter automaton* (CNFA) is a 6-tuple $A = (Q, q_0, C, \delta, F, \tau)$ where Q is the finite set of states; $q_0 \in Q$ is the initial state; C is the finite set of counter variables; $\delta : Q \times \Sigma \times \mathrm{Guard}(C) \times \mathrm{Update}(C) \times Q$ is the transition relation; $F : Q \to \mathrm{Guard}(C)$ is the acceptance function; and $\tau : C \to \mathbb{N}$ assigns a maximum value to every counter variable.

Intuitively, A can make a transition (q, a, ϕ, π, q') whenever it is in state q, reads a, and guard ϕ is true under the current values of the counter variables. It then updates the counter variables according to the update π, in a way we explain next, and moves into state q'. To explain the update mechanism formally, we introduce the notion of configuration. Thereto, let $\max(A) = \max\{\tau(c) \mid c \in C\}$. A *configuration* is a pair (q, α) where $q \in Q$ is the current state and $\alpha : C \to \{1, \ldots, \max(A)\}$ is the function mapping counter variables to their current value. Finally, an update π transforms α into $\pi(\alpha)$ by setting $\mathrm{cv} := 1$, when $\mathrm{reset(cv)} \in \pi$, and $\mathrm{cv} := \mathrm{cv} + 1$ when $\mathrm{cv{+}{+}} \in \pi$ and $\alpha(\mathrm{cv}) < \tau(\mathrm{cv})$. Otherwise, the value of cv remains unaltered.

Let α_0 be the function mapping every counter variable to 1. The *initial configuration* γ_0 is (q_0, α_0). A configuration (q, α) is *final* if $\alpha \models F(q)$. A configuration $\gamma' = (q', \alpha')$ *immediately follows* a configuration $\gamma = (q, \alpha)$ by reading $a \in \Sigma$, denoted $\gamma \to_a \gamma'$, if there exists $(q, a, \phi, \pi, q') \in \delta$ with $\alpha \models \phi$ and $\alpha' = \pi(\alpha)$.

For a string $w = a_1 \cdots a_n$ and two configurations γ and γ', we denote by $\gamma \Rightarrow_w \gamma'$ that $\gamma \to_{a_1} \cdots \to_{a_n} \gamma'$. A configuration γ is *reachable* if there exists a string w such that $\gamma_0 \Rightarrow_w \gamma$. A string w is *accepted* by A if $\gamma_0 \Rightarrow_w \gamma_f$ where γ_f is a final configuration. We denote by $L(A)$ the set of strings accepted by A.

A CNFA A is *deterministic* (or a CDFA) if, for every reachable configuration $\gamma = (q, \alpha)$ and for every symbol $a \in \Sigma$, there is at most one transition $(q, a, \phi, \pi, q') \in \delta$ such that $\alpha \models \phi$.

The *size* of a transition θ or acceptance condition $F(q)$ is the number of symbols which occur in it plus the size of the binary representation of each integer occcurring in it. By the same token, the size of A, denoted by $|A|$, is $|Q| + \sum_{q \in Q} \log \tau(q) + |F(q)| + \sum_{\theta \in \delta} |\theta|$.

Theorem 8. *1. Given CNFAs A_1 and A_2, a CNFA A accepting the union or intersection of A_1 and A_2 can be constructed in polynomial time. Moreover, when A_1 and A_2 are deterministic, then so is A.*
2. Given a CDFA A, a CDFA which accepts the complement of A can be constructed in polynomial time.
3. MEMBERSHIP *for word w and CDFA A is in time $\mathcal{O}(|w||A|)$.*
4. MEMBERSHIP *for non-deterministic CNFA is* NP*-complete.*
5. EMPTINESS *for CDFAs and CNFAs is* PSPACE*-complete.*
6. Deciding whether a CNFA A is deterministic is PSPACE*-complete.*

6 From RE(Σ,#) to CNFA

In this section, we show how an RE(Σ,#) expression r can be translated in polynomial time into an equivalent CNFA G_r by applying a natural extension of the well-known Glushkov construction. We emphasize at this point that such an extended Glushkov construction has already been given by Sperberg-McQueen [20]. Therefore, the contribution of this section lies mostly in the characterization given below: G_r is deterministic if and only if r is strongly deterministic. Moreover, as seen in the previous section, CDFAs have desirable properties which by this translation also apply to strongly deterministic RE(Σ,#) expressions. We refer to G_r as the *Glushkov counting automaton of r*.

6.1 Notation and terminology

We first provide some notation and terminology needed in the construction below. For an RE(Σ,#) expression r, the set first(r) (respectively, last(r)) consists of all symbols which are the first (respectively, last) symbols in some word defined by r. These sets are inductively defined as follows:

- $\text{first}(\varepsilon) = \text{last}(\varepsilon) = \emptyset$ and $\forall a \in \text{Char}(r), \text{first}(a) = \text{last}(a) = \{a\}$;

– $\text{first}(r_1 + r_2) = \text{first}(r_1) \cup \text{first}(r_2)$ and $\text{last}(r_1 + r_2) = \text{last}(r_1) \cup \text{last}(r_2)$;
– If $\varepsilon \in L(r_1)$, $\text{first}(r_1 r_2) = \text{first}(r_1) \cup \text{first}(r_2)$, else $\text{first}(r_1 r_2) = \text{first}(r_1)$;
– If $\varepsilon \in L(r_2)$, $\text{last}(r_1 r_2) = \text{last}(r_1) \cup \text{last}(r_2)$, else $\text{last}(r_1 r_2) = \text{last}(r_2)$;
– $\text{first}(r^{k,\ell}) = \text{first}(r_1)$ and $\text{last}(r^{k,\ell}) = \text{last}(r_1)$.

For a regular expression r, we say that a subexpression of r of the form $s^{k,\ell}$ is an *iterator* or *iterated subexpression of* r. Let $\text{lower}(s^{k,\ell}) := k$, and $\text{upper}(s^{k,\ell}) := \ell$. We say that $s^{k,\ell}$ is *bounded* when $\ell \in \mathbb{N}$, otherwise it is *unbounded*. For instance, an iterator of the form $s^{0,\infty}$ is a nullable, unbounded iterator.

For a marked symbol $\overline{x}$ and an iterator c we denote by $\text{iterators}(\overline{x}, c)$ the list of all iterated subexpressions of c which contain $\overline{x}$, except c itself. For marked symbols $\overline{x}, \overline{y}$, we denote by $\text{iterators}(\overline{x}, \overline{y})$ all iterated subexpressions which contain $\overline{x}$ but not $\overline{y}$. Finally, let $\text{iterators}(\overline{x})$ be the list of all iterated subexpressions which contain $\overline{x}$. Note that all such lists $[c_1, \ldots, c_n]$ contain a sequence of nested subexpressions. Therefore, we will always assume that they are ordered such that $c_1 \prec c_2 \prec \cdots \prec c_n$. Here $c \prec c'$ denotes that c is a subexpression of c'. For example, if $\overline{r} = ((a_1^{1,2} b_1)^{3,4})^{5,6}$, then $\text{iterators}(a_1, \overline{r}) = [a_1^{1,2}, (a_1^{1,2} b_1)^{3,4}]$, $\text{iterators}(a_1, b_1) = [a_1^{1,2}]$, and $\text{iterators}[a_1] = [a_1^{1,2}, (a_1^{1,2} b_1)^{3,4}, ((a_1^{1,2} b_1)^{3,4})^{5,6}]$.

6.2 Construction

We now define the set $\text{follow}(\overline{r})$ for a marked regular expression $\overline{r}$. As in the standard Glushkov construction, this set lies at the basis of the transition relation of G_r. The set $\text{follow}(\overline{r})$ contains triples $(\overline{x}, \overline{y}, c)$, where $\overline{x}$ and $\overline{y}$ are marked symbols and c is either an iterator or null. Intuitively, the states of G_r will be a designated start state plus a state for each symbol in $\text{Char}(\overline{r})$. A triple $(\overline{x}, \overline{y}, c)$ then contains the information we need for G_r to make a transition from state $\overline{x}$ to $\overline{y}$. If $c \neq \text{null}$, this transition iterates over c and all iterators in $\text{iterators}(\overline{x}, c)$ are reset by going to $\overline{y}$. Otherwise, if c equals null, the iterators in $\text{iterators}(\overline{x}, \overline{y})$ are reset. Formally, the set $\text{follow}(\overline{r})$ contains for each subexpression $\overline{s}$ of $\overline{r}$,

– all tuples $(\overline{x}, \overline{y}, \text{null})$ for $\overline{x}$ in $\text{last}(\overline{s_1})$, $\overline{y}$ in $\text{first}(\overline{s_2})$, and $\overline{s} = \overline{s_1}\,\overline{s_2}$; and
– all tuples $(\overline{x}, \overline{y}, \overline{s})$ for $\overline{x}$ in $\text{last}(\overline{s_1})$, $\overline{y}$ in $\text{first}(\overline{s_1})$, and $\overline{s} = \overline{s_1}^{k,\ell}$.

We introduce a counter variable $\text{cv}(c)$ for every iterator c in $\overline{r}$ whose value will always denote which iteration of c we are doing in the current run on the string. We define a number of tests and update commands on these counter variables:

– $\text{value-test}([c_1, \ldots, c_n]) := \bigwedge_{c_i} (\text{lower}(c_i) \leq \text{cv}(c_i)) \wedge (\text{cv}(c_i) \leq \text{upper}(c_i))$. When we leave the iterators $c_1, \ldots, c_n$ we have to check that we have done an admissible number of iterations for each iterator.
– $\text{upperbound-test}(c) := \text{cv}(c) < \text{upper}(c)$ when c is a bounded iterator and $\text{upperbound-test}(c) := \text{true}$ otherwise. When iterating over a bounded iterator, we have to check that we can still do an extra iteration.
– $\text{reset}(c_1, \ldots, c_n) := \{\text{reset}(\text{cv}(c_1)), \ldots, \text{reset}(\text{cv}(c_n))\}$. When leaving some iterators, their values must be reset. The counter variable is reset to 1, because at the time we reenter this iterator, its first iteration is started.

– update(c) := $\{\text{cv}(c)\text{++}\}$. When iterating over an iterator, we start a new iteration and increment its number of transitions.

We now define the Glushkov counting automaton $G_r = (Q, q_0, C, \delta, F, \tau)$. The set of states Q is the set of symbols in $\overline{r}$ plus an initial state, i.e., $Q := \{q_0\} \uplus \bigcup_{\overline{x} \in \text{Char}(\overline{r})} q_{\overline{x}}$. Let C be the set of iterators occurring in $\overline{r}$. We next define the transition function. For all $\overline{y} \in \text{first}(\overline{r})$, $(q_0, \text{dm}(\overline{y}), \mathit{true}, \emptyset, q_{\overline{y}}) \in \delta$.[5] For every element $(\overline{x}, \overline{y}, c) \in \text{follow}(\overline{r})$, we define a transition $(q_{\overline{x}}, \text{dm}(\overline{y}), \phi, \pi, q_{\overline{y}}) \in \delta$. If $c = \text{null}$, then $\phi := \text{value-test}(\text{iterators}(\overline{x}, \overline{y}))$ and $\pi := \text{reset}(\text{iterators}(\overline{x}, \overline{y}))$. If $c \neq \text{null}$, then $\phi := \text{value-test}(\text{iterators}(\overline{x}, c)) \wedge \text{upperbound-test}(c)$ and $\pi := \text{reset}(\text{iterators}(\overline{x}, c)) \cup \text{update}(c)$. The acceptance criteria of G_r depend on the set $\text{last}(\overline{r})$. For any symbol $\overline{x} \notin \text{last}(\overline{r})$, $F(q_{\overline{x}}) := \text{false}$. For every element $\overline{x} \in \text{last}(\overline{r})$, $F(q_{\overline{x}}) := \text{value-test}(\text{iterators}(\overline{x}))$. Here, we test whether we have done an admissible number of iterations of all iterators in which $\overline{x}$ is located. Finally, $F(q_0) := \text{true}$ if $\varepsilon \in L(r)$. Lastly, for all bounded iterators c, $\tau(\text{cv}(c)) = \text{upper}(c)$ since c never becomes larger than $\text{upper}(c)$, and for all unbounded iterators c, $\tau(\text{cv}(c)) = \text{lower}(c)$ as there are no upper bound tests for $\text{cv}(c)$.

Theorem 9. *For every RE(Σ,#) expression r, $L(G_r) = L(r)$. Moreover, G_r is deterministic iff r is strongly deterministic.*

7 Decidability and Complexity Results

Definition 3, defining strong determinism, is of a semantical nature. Therefore, we provide Algorithm 1 for testing whether a given expression is strongly deterministic, which runs in cubic time. To decide weak determinism, Kilpeläinen and Tuhkanen [13] give a cubic algorithm for RE(Σ,#), while Brüggemann-Klein [1] gives a quadratic algorithm for RE(Σ) by computing its Glushkov automaton and testing whether it is deterministic[6].

Theorem 10. *For any $r \in$ RE(Σ,#), isStrongDeterministic(r) returns true if and only if r is strong deterministic. Moreover, it runs in time $\mathcal{O}(|r|^3)$.*

We next consider the following decision problems, for expressions of class $\mathcal{R}$:
INCLUSION: Given two expressions $r, r' \in \mathcal{R}$, is $L(r) \subseteq L(r')$?
EQUIVALENCE: Given two expressions $r, r' \in \mathcal{R}$, is $L(r) = L(r')$?
INTERSECTION: Given a number of expressions $r_1, \ldots, r_n \in \mathcal{R}$, is $\bigcap_{i=1}^{n} L(r_i) \neq \emptyset$?

Theorem 11. *(1)* INCLUSION *and* EQUIVALENCE *for RE(Σ,#) are* EXPSPACE*-complete [16],* INTERSECTION *for RE(Σ,#) is* PSPACE*-complete [7]. (2)* INCLUSION *and* EQUIVALENCE *for $DET_W(\Sigma)$ are in* PTIME*,* INTERSECTION *for $DET_W(\Sigma)$ is* PSPACE*-complete [15]. (3)* INCLUSION *for $DET_W^{\#}(\Sigma)$ is co*NP*-hard [11].*

[5] Recall that $\text{dm}(\overline{y})$ denotes the demarking of $\overline{y}$.

[6] There sometimes is some confusion about this result: Computing the Glushkov automaton is quadratic in the expression, while linear in the output automaton (consider, e.g., $(a_1 + \cdots + a_n)(a_1 + \cdots + a_n)$). Only when the alphabet is fixed is the Glushkov automaton of a deterministic expression of size linear in the expression.

Algorithm 1 isStrongDeterministic. Returns true if r is strong deterministic, false otherwise.

$\overline{r} \leftarrow$ marked version of r
Initialize Follow $\leftarrow \emptyset$
Compute first($\overline{s}$), last($\overline{s}$), for all subexpressions $\overline{s}$ of $\overline{r}$
if $\exists \overline{x}, \overline{y} \in \text{first}(\overline{r})$ with $\overline{x} \neq \overline{y}$ and $\text{dm}(\overline{x}) = \text{dm}(\overline{y})$ **then return false**
for each subexpression $\overline{s}$ of $\overline{r}$, in bottom-up fashion **do**
    if $\overline{s} = \overline{s_1}\,\overline{s_2}$ **then**
        if $\text{last}(\overline{s_1}) \neq \emptyset$ and $\exists \overline{x}, \overline{y} \in \text{first}(\overline{s_1})$ with $\overline{x} \neq \overline{y}$ and $\text{dm}(\overline{x}) = \text{dm}(\overline{y})$ **then return false**
        $F \leftarrow \{(\overline{x}, \text{dm}(\overline{y})) \mid \overline{x} \in \text{last}(\overline{s_1}), \overline{y} \in \text{first}(\overline{s_2})\}$
    else if $\overline{s} = \overline{s_1}^{[k,\ell]}$, with $\ell \geq 2$ **then**
        if $\exists \overline{x}, \overline{y} \in \text{first}(\overline{s_1})$ with $\overline{x} \neq \overline{y}$ and $\text{dm}(\overline{x}) = \text{dm}(\overline{y})$ **then return false**
        $F \leftarrow \{(\overline{x}, \text{dm}(\overline{y})) \mid \overline{x} \in \text{last}(\overline{s_1}), \overline{y} \in \text{first}(\overline{s_1})\}$
    if $F \cap \text{Follow} \neq \emptyset$ **then return false**
    if $\overline{s} = \overline{s_1}\,\overline{s_2}$ or $\overline{s} = \overline{s_1}^{k,\ell}$, with $\ell \geq 2$ and $k < \ell$ **then**
        Follow $\leftarrow$ Follow $\uplus\, F$
return true

By combining (1) and (2) of Theorem 11 we get the complexity of intersection for $\text{DET}^{\#}_{W}(\Sigma)$ and $\text{DET}^{\#}_{S}(\Sigma)$. This is not the case for the inclusion and equivalence problem, unfortunately. By using the results of the previous sections we can, for $\text{DET}^{\#}_{S}(\Sigma)$, give a pspace upperbound for both problems, however.

Theorem 12. *(1)* equivalence *and* inclusion *for* $DET^{\#}_{S}(\Sigma)$ *are in* pspace. *(2)* intersection *for* $DET^{\#}_{W}(\Sigma)$ *and* $DET^{\#}_{S}(\Sigma)$ *is* pspace*-complete.*

8 Conclusion

We investigated and compared the notions of strong and weak determinism in the presence of counting. Weakly deterministic expressions have the advantage of being more expressive and more succinct than strongly deterministic ones. However, strongly deterministic expressions are expressivily equivalent to standard deterministic expressions, a class of languages much better understood than the weakly deterministic languages with counting. Moreover, strongly deterministic expressions are conceptually simpler (as strong determinism does not depend on intricate interplays of the counter values) and correspond naturally to deterministic Glushkov automata. The latter also makes strongly deterministic expressions easier to handle as witnessed by the pspace upperbound for inclusion and equivalence, whereas for weakly deterministic expressions only a trivial expspace upperbound is known. For these reasons, one might wonder if the weak determinism demanded in the current standards for XML Schema should not be replaced by strong determinism. The answer to some of the following open questions can shed more light on this issue: (1) Is it decidable if a language is definable by a weakly deterministic expression with counting? (2) Can

the Glushkov construction given in Section 6 be extended such that it translates any weakly deterministic expression with counting into a CDFA? (3) What are the exact complexity bounds for inclusion and equivalence of strongly and weakly deterministic expression with counting?

References

1. A. Brüggemann-Klein. Regular expressions into finite automata. *Theor. Comput. Sci.*, 120(2):197–213, 1993.
2. A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.
3. D. Colazzo, G. Ghelli, and C. Sartiani. Efficient asymmetric inclusion between regular expression types. In *ICDT*, pages 174–182, 2009.
4. S. Dal-Zilio and D. Lugiez. XML schema, tree logic and sheaves automata. In *RTA*, pages 246–263, 2003.
5. J. Esparza. Decidability and complexity of Petri net problems – an introduction. In *Petri Nets*, pages 374–428, 1996.
6. M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
7. W. Gelade, W. Martens, and F. Neven. Optimizing schema languages for XML: Numerical constraints and interleaving. In *ICDT*, pages 269–283, 2007.
8. W. Gelade. Succinctness of regular expressions with interleaving, intersection and counting. In *MFCS*, pages 363–374, 2008.
9. A. Hume. A tale of two greps. *Softw., Pract. and Exp.*, 18(11):1063–1072, 1988.
10. P. Kilpeläinen. Inclusion of unambiguous #res is NP-hard, May 2004. Unpublished.
11. P. Kilpeläinen and R. Tuhkanen. Regular expressions with numerical occurrence indicators — preliminary results. In *SPLST 2003*, pages 163–173, 2003.
12. P. Kilpeläinen and R. Tuhkanen. Towards efficient implementation of XML schema content models. In *DOCENG 2004*, pages 239–241. ACM, 2004.
13. P. Kilpeläinen and R. Tuhkanen. One-unambiguity of regular expressions with numeric occurrence indicators. *Inform. Comput.*, 205(6):890–916, 2007.
14. C. Koch and S. Scherzinger. Attribute grammars for scalable query processing on XML streams. *VLDB Journal*, 16(3):317–342, 2007.
15. W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for simple regular expressions. In *MFCS*, pages 889–900, 2004.
16. A.R. Meyer and L.J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *FOCS*, p. 125–129, 1972.
17. D.W. Mount. *Bioinformatics: Sequence and Genome Analysis*. Cold Spring Harbor Laboratory Press, September 2004.
18. H. Seidl, T. Schwentick, A. Muscholl, and P. Habermehl. Counting in Trees for Free. In *ICALP*, pages 1136–1149, 2004.
19. G. Pighizzini and J. Shallit. Unary language operations, state complexity and Jacobsthal's function. *Int. J. Found. Comp. Sc.*, 13(1):145–159, 2002.
20. C.M. Sperberg-McQueen. Notes on finite state automata with counters. http://www.w3.org/XML/2004/05/msm-cfa.html, 2004.
21. C.M. Sperberg-McQueen and H. Thompson. XML Schema. http://www.w3.org/XML/Schema, 2005.
22. M.Y. Vardi. From monadic logic to PSL. In *Pillars of Computer Science*, pages 656–681, 2008.
23. L. Wall, T. Christiansen, and J. Orwant. *Programming Perl*. O'Reilly, 2000.

Appendix

We use $\boxed{\cdots}$ to mark the places where the Appendix continues the proofs from the body of the paper.

Preliminaries for the Appendix

First, Last, and Factors. Some of the preliminaries here are already defined in Section 6.1. We repeat them here for the convenience of the reader. For an RE(Σ,#) expression r, the set first(r) (respectively, last(r)) consists of all symbols which are the first (respectively, last) symbols in some word defined by r. These sets are inductively defined as follows:

- $\text{first}(\varepsilon) = \text{last}(\varepsilon) = \emptyset$ and $\forall a \in \text{Char}(r), \text{first}(a) = \text{last}(a) = \{a\}$;
- If $r = r_1 + r_2$: $\text{first}(r) = \text{first}(r_1) \cup \text{first}(r_2)$ and $\text{last}(r) = \text{last}(r_1) \cup \text{last}(r_2)$;
- If $r = r_1 \cdot r_2$:
  - If $\varepsilon \in L(r_1)$, $\text{first}(r) = \text{first}(r_1) \cup \text{first}(r_2)$, else $\text{first}(r) = \text{first}(r_1)$;
  - If $\varepsilon \in L(r_2)$, $\text{last}(r) = \text{last}(r_1) \cup \text{last}(r_2)$, else $\text{last}(r) = \text{last}(r_2)$;
- If $r = r_1^{k,\ell}$: $\text{first}(r) = \text{first}(r_1)$ and $\text{last}(r) = \text{last}(r_1)$.

For a regular expression r, we say that a subexpression of r of the form $s^{k,\ell}$ is an *iterator of* r. Let $\overline{r}$ and $\overline{s}$ be expressions such that $\overline{s}$ is a subexpression of $\overline{r}$. Then we say that $\overline{s}$ is a *factor* of $\overline{r}$, whenever $\text{first}(\overline{s}) \subseteq \text{first}(\overline{r})$ and $\text{last}(\overline{s}) \subseteq \text{last}(\overline{r})$. When $\overline{r}$ and $\overline{s}$ are iterators, this intuitively means that a number of iterations of $\overline{s}$ are sufficient to satisfy $\overline{r}$. For instance, in the expression $\overline{s} = (a_1^{0,3} b_1^{1,2})^{3,4}$, $b_1^{1,2}$ is a factor of $\overline{s}$, but $a_1^{0,3}$ is not.

Proofs for Section 3

Before giving the proof of Theorem 4, we introduce some lemmas and additional terminology necessary in its proof.

Lemma 13. *Let $r_1 r_2$ be a factor of $r^{k,\ell}$ with r_1, r_2 nullable, $L(r_1) \neq \{\varepsilon\}$, $L(r_2) \neq \{\varepsilon\}$, and $\ell \geq 2$. Then $r^{k,\ell}$ is not strongly deterministic.*

Proof. If $r^{k,\ell}$ is not weakly deterministic, the lemma already holds. Therefore, assume that $r^{k,\ell}$ is weakly deterministic. Let $\overline{r}^{k,\ell}$ be a marked version of $r^{k,\ell}$. Let $\widetilde{s} := [_1\widetilde{\overline{r}}]_1^{k,\ell}$ denote the bracketed version of $\overline{r}^{k,\ell}$. Denote by $\widetilde{\overline{r_1}}$ and $\widetilde{\overline{r_2}}$ the subexpressions of $\widetilde{s}$ that correspond to $\overline{r_1}$ and $\overline{r_2}$.

Let $\widetilde{\overline{u}}$ (respectively, $\widetilde{\overline{v}}$) be a non-empty string in $L(\widetilde{\overline{r_1}})$ (respectively, $\widetilde{\overline{v}}$). These strings exist as $L(r_1)$ and $L(r_2)$ are not equal to $\{\varepsilon\}$. Let $\widetilde{\overline{w}} = [_1\widetilde{\overline{uv}}]_1$. Define $x := \max\{0, k-2\}$. Then, both $[_1\widetilde{\overline{u}}]_1[_1\widetilde{\overline{v}}]_1\widetilde{\overline{w}}^x$ and $[_1\widetilde{\overline{uv}}]_1\widetilde{\overline{w}}^{x+1}$ are in $L(\widetilde{s})$ and thereby show that $r^{k,\ell}$ is not strongly deterministic. □

We abbreviate the term *deterministic finite automaton* with DFA. We let δ denote the transition function of a DFA A, i.e., $\delta(q_1, a) = q_2$ means that A can go from state q_1 to state q_2 while reading a. The following notions come from, e.g., Shallit [19], but we repeat them here for completeness. (Shallit used *tail* to refer to what we call a *chain*.)

Definition 14. We say that a DFA over $\Sigma = \{a\}$ is a *chain* if its start state is q_0 and its transition function is of the form $\delta(q_0, a) = q_1$, ..., $\delta(q_{n-1}, a) = q_n$. A DFA is a *chain followed by a cycle* if its transition function is of the form $\delta(q_0, a) = q_1$, ..., $\delta(q_{n-1}, a) = q_n$, $\delta(q_n, a) = q_{n+1}$, ..., $\delta(q_{n+m-1}, a) = q_{n+m}$, $\delta(q_{n+m}, a) = q_n$. The *cycle states* of the latter DFA are $q_n, \ldots, q_{n+m}$.

We say that a unary regular language L is *ultimately periodic* if L is infinite and its minimal DFA is a chain followed by a cycle, for which at most one of the cycle states is final. We say that L over alphabet $\{a\}$ is (n_0, x)-*periodic* if

(i) $L \subseteq L((a^x)^*)$, and
(ii) for every $n \in \mathbb{N}$ such that $nx \geq n_0$, L contains the string a^{nx}, i.e., the string of a's of length nx.

We say that L is *ultimately x-periodic* if L is (n_0, x)-periodic for some $n_0 \in \mathbb{N}$. Notice that these notions imply that L is infinite. Furthermore, notice the subtle differences between ultimately periodic and ultimately x-periodic. In an ultimately x-periodic language *all* strings have lengths which are multiples of x, i.e. they have length 0 (modulo x). In an ultimately periodic language only all *sufficiently long* string must have the same length y (modulo x), for a fixed y, which, moreover, can be different from 0.

Lemma 15. *Let L be a language, $x \in \mathbb{N}_0$, $k \in \mathbb{N}$, and $\ell \in \mathbb{N}_0 \cup \{\infty\}$ with $k \leq \ell$. If L is ultimately x-periodic, then $L^{k,\ell}$ is also ultimately x-periodic.*

Proof. Every string in $L^{k,\ell}$ is a concatenation of (possibly empty) strings in L. Since the length of every string in L is a multiple of x, it follows that the length of every string in $L^{k,\ell}$ is a multiple of x. Therefore, $L^{k,\ell} \subseteq L((a^x)^*)$.

Furthermore, take $n_0 \in \mathbb{N}$ such that L is (n_0, x)-periodic. Let $k_0 := \max\{k, 1\}$. We will show that $L^{k,\ell}$ is $(k_0(n_0 + x), x)$-periodic, which proves the lemma. Take $n \in \mathbb{N}$ such that $nx \geq k_0(n_0 + x)$. Hence, $(\frac{n}{k_0} - 1)x \geq n_0$ since $k_0 \geq 1$ and therefore $\lfloor \frac{n}{k_0} \rfloor x \geq n_0$. Hence, $a^{nx} = a^{\lfloor n/k_0 \rfloor x} \cdots a^{\lfloor n/k_0 \rfloor x} a^{r_0 x}$, where the dots abbreviate a k_0-fold concatenation and $r_0 := n \mod \lfloor n/k_0 \rfloor$. Since L is (n_0, x)-periodic, $a^{\lfloor n/k_0 \rfloor x} \in L$ and $a^{(\lfloor n/k_0 \rfloor + r_0)x} \in L$. Hence, $a^{nx} \in L^{k_0}$, which implies that $a^{nx} \in L^{k,\ell}$. □

We state Lemma 5 equivalently as in the body of the paper, but using the definitions above:

Proof of Lemma 5: *Let $\Sigma = \{a\}$, and $L \in REG(\Sigma)$, then $L \in DET^{\#}_{W}(\Sigma)$ if and only if L is either finite or ultimately periodic.*

Proof. We only need to show that, if a language is in $\mathrm{DET}^{\#}_{W}(\Sigma)$, then it can be defined with a DFA of the correct form. It is well-known that, for every unary language, the minimal DFA is a chain followed by a cycle. We therefore only need to argue that, in this cycle, at most one of the nodes is final.

Let $\Sigma = \{a\}$ and let r be a weakly deterministic regular expression with counting over $\{a\}$, such that $L(r)$ is infinite. We can assume w.l.o.g. that r does not contain concatenations with ε. Since r is over a unary alphabet, we can make the following observation:

Remark 16. (1) If r has a disjunction of the form r_1+r_2 then either $L(r_1) = \{\varepsilon\}$ or $L(r_2) = \{\varepsilon\}$.
(2) There are no subexpressions of the form $r_1 r_2$ in r in which $|L(r_1)| > 1$.

Indeed, if Remark 16 does not hold, then r is not weakly deterministic.

Our first goal is to bring r into a normal form. More specifically, we want to write r as

$$r = (r_1(r_2(\cdots(r_n)^{p_n,1}\cdots)^{p_3,1})^{p_2,1})^{p_1,1},$$

where

(a) for each $i = 1, \ldots, n$, $p_i \in \{0, 1\}$;
(b) r has no occurrences of ε;
(c) r_n is a tower of counters, that is, it only has a single occurrence of a;
(d) $L(r_1), \ldots, L(r_{n-1})$ are singletons, and $L(r_n)$ is infinite.

Notice that, if r has the normal form and one of $L(r_i)$, with $1 \leq i \leq n-1$, is not a singleton, then this would immediately violate Remark 16(2). In order to achieve this normal form, we iteratively replace

(i) all subexpressions of the form $(s^{k,k})^{\ell,\ell}$ with $s^{k\ell,k\ell}$;
(ii) all subexpressions of the form $s^{k_1,k_1}s^{k_2,\ell_2}$ with $s^{k_1+k_2,k_1+\ell_2}$; and
(iii) all subexpressions of the form $(s+\varepsilon)$ and $(\varepsilon+s)$ with $s^{0,1}$
(iv) all subexpressions of the form $a^{k,k}s$, where $s \neq a^{k_1,\ell_1}$ with $a^{k,k}(s)^{1,1}$

until no such subexpressions occur anymore. These replacements preserve the language defined by r and preserve weak determinism. Due to Remark 16, these replacements turn r in the normal form above adhering to the syntactic constraints (a)–(c). Furthermore, we know that condition (d) must hold because r is weakly deterministic.

From now on, we therefore assume that $r = (r_1(r_2(\cdots(r_n)^{p_n,1}\cdots)^{p_3,1})^{p_2,1})^{p_1,1}$ in which only $L(r_n)$ is infinite. Notice that we can translate the regular expression

$$r' = (r_1(r_2(\cdots(r_{n-1}(X)^{p_n,1})^{p_{n-1},1}\cdots)^{p_3,1})^{p_2,1})^{p_1,1}$$

over alphabet $\{a, X\}$ into a DFA which is a chain and which reads the symbol X precisely once, at the end of the chain. Therefore, it suffices to prove now that we can translate r_n into a DFA which is a chain followed by a cycle, in which at most one of the cycle nodes is final. Indeed, if A_1 and A_2 are the DFAs for r' and r_n respectively, then we can intuitively obtain the DFA A for r by concatenating

these two DFAs. More formally, if q_1 is the unique state in A_1 which has the transition $\delta(q_1, X) = q_2$ (and q_2 is final in A_1), and q_3 is the initial state of A_2, then we can obtain A by taking the union of the states and transitions of A_1 and A_2, removing state q_2, and adding a transition $\delta(q_1, a) = q_3$. The initial state of A is the initial state of A_1 and its final state set is the union of the final state sets in A_1 and A_2. Since A_1 is a chain, $L(A) = L(r)$ is ultimately periodic if and only if $L(A_2) = L(r_n)$ is ultimately periodic.

Let $s^{k,\infty}$ be the smallest subexpression of r_n in which the upper bound is unbounded. (Such an expression must exist, since $L(r_n)$ is infinite.) We will first prove that $L(s^{k,\infty})$ is ultimately x-periodic for some $x \in \mathbb{N}$. Due to Lemma 15, this also proves that $L(r_n)$ is ultimately x-periodic, which implies that there is a DFA of the right form for $L(r_n)$ and concludes our proof.

It therefore only remains to show that $L(s^{k,\infty})$ is ultimately x-periodic for some $x \in \mathbb{N}$. To this end, there are two possibilities: either s contains a subexpression of the form $(s')^{y_1,y_2}$ with $y_1 < y_2$, or it does not. If not, then $L(s^{k,\infty})$ is of the form $(a^{x,x})^{k,\infty}$ due to the replacement (i) above in our normalization. In this case, $L(s^{k,\infty})$ is clearly (xk, x)-periodic.

If $s^{k,\infty}$ is of the form above, then it is of the form $(((a^{x,x})^{y_1,y_2})^{h_1^1,h_2^1}) \cdots)^{h_1^n,h_2^n})^{k,\infty}$, where $y_1 < y_2$, x can equal 1, and the h_1^i, h_2^i denote a nesting of iterators. It is immediate that $L(s^{k,\infty}) \subseteq L((a^{x,x})^*)$, i.e., the length of every string in $L(s^{k,\infty})$ is a multiple of x. To show that there also exists an n_0 such that $s^{k,\infty}$ defines *all* strings of length mx with $m \in \mathbb{N}$ and $mx \geq n_0$, let $z = h_1^1 \cdot h_1^2 \cdots h_1^n$, or $z = 1$ when $n = 0$. Clearly, $(((a^{x,x})^{y,y+1})^{z,z})^{k,\infty}$ defines a subset of $s^{k,\infty}$ and, hence, it suffices to show that $(((a^{x,x})^{y,y+1})^{z,z})^{k,\infty}$ defines all such strings of length $mx \geq n_0$.

This is proven in the lemma below. The meaning of the variables x, y, z, n_0 is the same in the lemma as in the above paragraph and ℓ denotes the number of iterations of s, i.e., at least k. Moreover, in every such iteration, we must do z iterations of the iterator $((a^{x,x})^{y,y+1})^{z,z}$. Here, we can each time choose whether we do y or $y+1$ iterations of $(a^{x,x})^{y,y+1}$. For every $i \in [1, \ell]$, denoting the ith iteration of $(((a^{x,x})^{y,y+1})^{z,z})^{k,\infty}$, we let $z_{i,1}$ (respectively, $z_{i,2}$) denote the number of times y (respectively, $y+1$) iterations are done. Hence, $z = z_{i,1} + z_{i,2}$ must hold.

Lemma 17. *Let x, y, z be natural numbers. Then, there exist $n_0 \in \mathbb{N}$, such that for all $m \in \mathbb{N}$, with $mx > n_0$, there exists $\ell \in \mathbb{N}$, with $\ell \geq k$, and $z_{1,1}, z_{1,2}, \ldots, z_{\ell,1}, z_{\ell,2} \in \mathbb{N}$ such that*

- *for all $i = 1, \ldots, \ell$: $z_{i,1} + z_{i,2} = z$*
- *$mx = \sum_{i=1}^{\ell} z_{i,1} yx + \sum_{i=1}^{\ell} z_{i,2}(y+1)x$*

Proof. Let x, y, z be fixed, but arbitrary natural numbers. We have that

$$\begin{aligned} mx &= \sum_{i=1}^{\ell} z_{i,1}yx + \sum_{i=1}^{\ell} z_{i,2}(y+1)x \\ \Leftrightarrow m &= \sum_{i=1}^{\ell} z_{i,1}y + \sum_{i=1}^{\ell} z_{i,2}y + \sum_{i=1}^{\ell} z_{i,2} \\ &= \sum_{i=1}^{\ell} (z_{i,1} + z_{i,2})y + \sum_{i=1}^{\ell} z_{i,2} \\ &= \sum_{i=1}^{\ell} zy + \sum_{i=1}^{\ell} z_{i,2} \\ &= \ell zy + \sum_{i=1}^{\ell} z_{i,2} \end{aligned}$$

In the last equality, z and y are fixed but ℓ and the values $z_{i,2}$ can be chosen freely, as long as $\ell \geq k$, and $z_{i,2} \leq z$, for all i. Our objective therefore is to generate all possible values of m for $m > n_0'$, for a certain n_0'.

Notice that $\ell zy + \sum_{i=1}^{\ell} z_{i,2}$ generates precisely the integer values in the interval $[\ell zy..\ell z(y+1)]$, since the sum $\sum_{i=1}^{\ell} z_{i,2}$ can assume all values between 0 and ℓz. Our goal now is to prove that, for ℓ large enough, the intervals $[\ell zy..\ell z(y+1)]$ and $[(\ell+1)zy..(\ell+1)z(y+1)]$ overlap. This happens when

$$\begin{aligned} & \ell z(y+1) \geq (\ell+1)zy \\ \Leftrightarrow\ & \ell(y+1) \geq (\ell+1)y \\ \Leftrightarrow\ & \ell y + \ell \geq \ell y + y \\ \Leftrightarrow\ & \ell \geq y \end{aligned}$$

Since y is fixed, we can indeed always choose ℓ such that $\ell \geq y$ and $\ell \geq k$. □

This concludes the proof of Lemma 5. □

Proof of Theorem 4: *For every alphabet Σ,*

$$DET_S(\Sigma) = DET_W(\Sigma) = DET_S^{\#}(\Sigma) = DET_W^{\#}(\Sigma) \subsetneq REG(\Sigma) \ \ (\textit{if } |\Sigma| = 1)$$

$$DET_S(\Sigma) = DET_W(\Sigma) = DET_S^{\#}(\Sigma) \subsetneq DET_W^{\#}(\Sigma) \subsetneq REG(\Sigma) \ \ (\textit{if } |\Sigma| \geq 2)$$

Proof. The equality $\mathrm{DET}_S(\Sigma) = \mathrm{DET}_W(\Sigma)$ is already implicit in the work of Brüggemann-Klein [1].[7] By this result and by definition, all inclusions from left to right already hold. It therefore suffices to show that

[7] Strong determinism was not explicitly considered in [1]. However, the paper gives a transformation of weakly deterministic regular expressions into equivalent expressions in *star normal form*; and every expression in star normal form is strongly deterministic

(1) $\mathrm{DET}^{\#}_{S}(\Sigma) \subseteq \mathrm{DET}_{S}(\Sigma)$ for arbitrary alphabets,
(2) $\mathrm{DET}^{\#}_{W}(\Sigma) \subseteq \mathrm{DET}^{\#}_{S}(\Sigma)$ for unary alphabets,
(3) $\mathrm{DET}^{\#}_{S}(\Sigma) \subsetneq \mathrm{DET}^{\#}_{W}(\Sigma)$ for binary alphabets, and
(4) $\mathrm{DET}^{\#}_{W}(\Sigma) \subsetneq \mathrm{REG}(\Sigma)$ for unary alphabets.

We now prove these four points.

(1) $\boxed{\cdots}$ Let r be an arbitrary strongly deterministic regular expression with counting. We can assume w.l.o.g. that no subexpressions of the form $s^{1,1}$ or ε occur. Indeed iteratively replacing any subexpression of the form $s^{1,1}$, $s\varepsilon$ or εs by s; $s + \varepsilon$ or $\varepsilon + s$ by $s^{0,1}$ and $\varepsilon^{k,\ell}$ by ε, yields an expression which is still strongly deterministic and which is either equal to ε or does not contain $s^{1,1}$ or ε. As in the former case we are done, we can hence assume the latter.

We recursively transform r into a strongly deterministic expression $\mathrm{EC}(r)$ without counting such that $L(r) = L(\mathrm{EC}(r))$, using the rules below. In these rules, EC stands for "eliminate counter" and ECE for "eliminate counter and epsilon":

(a) for all $a \in \Sigma$, $\mathrm{EC}(a) := a$
(b) $\mathrm{EC}(r_1 + r_2) := \mathrm{EC}(r_1) + \mathrm{EC}(r_2)$
(c) $\mathrm{EC}(r_1 r_2) := \mathrm{EC}(r_1)\mathrm{EC}(r_2)$
(d) if r nullable then $\mathrm{EC}(r^{0,1}) := \mathrm{EC}(r)$
(e) if r not nullable then $\mathrm{EC}(r^{0,1}) := \mathrm{ECE}(r) + \varepsilon$
(f) if $\ell = \infty$ then $\mathrm{EC}(r^{k,\ell}) := \mathrm{ECE}(r) \cdots \mathrm{ECE}(r) \cdot \mathrm{ECE}(r)^*$
(g) if $\ell \in \mathbb{N} \setminus \{1\}$ then $\mathrm{EC}(r^{k,\ell}) := \mathrm{ECE}(r) \cdots \mathrm{ECE}(r) \cdot \big(\mathrm{ECE}(r)(\mathrm{ECE}(r)(\cdots) + \varepsilon) + \varepsilon\big)$

In the last two rules, $\mathrm{ECE}(r) \cdots \mathrm{ECE}(r)$ denotes a k-fold concatenation of $\mathrm{ECE}(r)$, and the recursion in $\big(\mathrm{ECE}(r)(\mathrm{ECE}(r)(\cdots) + \varepsilon) + \varepsilon\big)$ contains $\ell - k$ occurrences of r.

(a') for all $a \in \Sigma$, $\mathrm{ECE}(a) := a$
(b') $\mathrm{ECE}(r_1 + r_2) := \mathrm{ECE}(r_1) + \mathrm{ECE}(r_2)$
(c') $\mathrm{ECE}(r_1 r_2) := \mathrm{EC}(r_1)\mathrm{EC}(r_2)$
(d') if $k \neq 0$, then $\mathrm{ECE}(r^{k,\ell}) := \mathrm{EC}(r^{k,\ell})$
(e') if r is nullable, then $\mathrm{ECE}(r^{0,1}) := \mathrm{ECE}(r)$
(f') if r is not nullable, then $\mathrm{ECE}(r^{0,1}) := \mathrm{EC}(r)$
(g') if $k = 0$ and $\ell = \infty$, then $\mathrm{ECE}(r^{k,\ell}) := \mathrm{ECE}(r)\mathrm{ECE}(r)^*$
(h') if $k = 0$ and $\ell \in \mathbb{N} \setminus \{1\}$, then $\mathrm{ECE}(r^{k,\ell}) := \mathrm{ECE}(r)\big(\mathrm{ECE}(r)(\cdots) + \varepsilon\big)$

Similarly as above, the recursion $\big(\mathrm{ECE}(r)(\cdots) + \varepsilon\big)$ contains $\ell - 1$ occurrences of r. We prove that $L(r) = L(\mathrm{EC}(r))$ and that $\mathrm{EC}(r)$ is a strongly deterministic regular expression.

To show $L(r) = L(\mathrm{EC}(r))$, we prove by simultaneous induction on the application of the rules (a)–(g) and (a')–(h') that $L(\mathrm{EC}(r)) = L(r)$, for any strong deterministic expression r and that $L(\mathrm{ECE}(r)) = L(r) \setminus \{\varepsilon\}$, for all expressions r to which $\mathrm{ECE}(r)$ can be applied. The latter is important as $\mathrm{ECE}(r)$ does not equal $L(r) \setminus \{\varepsilon\}$ for all r. For instance, $\mathrm{ECE}(a^{0,1}b^{0,1}) = (a + \varepsilon)(b + \varepsilon)$, and

$L(a^{0,1}b^{0,1}) = L((a+\varepsilon)(b+\varepsilon))$. However, as EC is always applied to a strongly deterministic expression, we will see that ECE will never be applied to such a subexpression.

The base cases (a) $\mathrm{EC}(a) := a$ and (a') $\mathrm{ECE}(a) := a$ are clear. For the induction, the cases (b)–(e) are trivial. Correctness for cases (f) and (g) is immediate from Remark 1. Case (b') is also trivial.

The first non-trivial case is (c'). If $\varepsilon \notin L(r_1 r_2)$ then (c') is clearly correct. Towards a contradiction, suppose that $\varepsilon \in L(r_1 r_2)$. This implies that r_1 and r_2 are nullable. Notice that we only apply rule (c') when rewriting r if either *(i)* $r_1 r_2$ is a factor of some $s^{0,1}$ with s not nullable (case (e)), or *(ii)* $r_1 r_2$ is a factor of a subexpression of the form $s^{k,\ell}$ with $\ell > 2$ (cases (f,g)). Here, $r_1 r_2$ must always be a factor as whenever ECE is applied to a subexpression, this subexpression is a factor; and the factor relation is clearly transitive. The reason that there must always be such a superexpression to which case (e), (f), or (g) is applied is that the recursive process starts by applying EC. Therefore, we must have applied (e), (f), or (g) to some superexpression of which $r_1 r_2$ is a factor before applying ECE to $r_1 r_2$.

In case *(i)*, r_1 and r_2 nullable implies that r must be nullable (because $r_1 r_2$ is a factor), which contradicts the precondition of rule (e). In case *(ii)*, Lemma 13 claims that $s^{k,\ell}$ and therefore r is not strongly deterministic, which is also a contradiction.

By Remark 1, (d') is also correct. Cases (e')–(h') are also trivial. This concludes the proof that $L(\mathrm{EC}(r)) = L(r)$.

We next prove that $\mathrm{EC}(r)$ and $\mathrm{ECE}(r)$ are strongly deterministic regular expression, whenever r is strongly deterministic. We prove this by induction on the reversed order of application of the rewrite rules. The induction base rules (a) and (a'), and rules (b)–(e) and (b')–(f') are immediate. For instance, for rule (b) we know by induction that $\mathrm{EC}(r_1)$ and $\mathrm{EC}(r_2)$ are strongly deterministic. As $L(r_1) = L(\mathrm{EC}(r_1))$, $L(r_2) = L(\mathrm{EC}(r_2))$, and $r_1 + r_2$ is strongly deterministic, it follows that also $\mathrm{EC}(r_1) + \mathrm{EC}(r_2)$ is strongly deterministic.

Cases (f), (g), (g'), and (h') are more involved, though very similar. Therefore, we only investigate case (f). Let $\overline{\mathrm{EC}(r^{k,\infty})}$ denote a marked version of $\mathrm{EC}(r^{k,\infty})$ and let $\overline{\mathrm{EC}(r^{k,\infty})} = \overline{\mathrm{ECE}(r)}_1 \cdots \overline{\mathrm{ECE}(r)}_k \overline{\mathrm{ECE}(r)}^*_{k+1}$. Further, let $\overline{r}^{k,\infty}$ be a marking of $r^{k,\infty}$ and let $\mathrm{f} : \mathrm{Char}(\overline{\mathrm{EC}(r^{k,\infty})}) \to \mathrm{Char}(\overline{r}^{k,\infty})$ be the natural mapping associating each symbol in $\overline{\mathrm{EC}(r^{k,\infty})}$ to its corresponding symbol in $\mathrm{Char}(\overline{r}^{k,\infty})$. For instance, when $r = (aba)^{1,\infty}$, then $\mathrm{EC}(r) = (aba)(aba)^*$, $\overline{r} = (a_1 b_1 a_2)^{1,\infty}$, $\overline{\mathrm{EC}(r)} = (a_1 b_1 a_2)(a_3 b_2 a_4)^*$, and $\mathrm{f}(a_1) = a_1$, $\mathrm{f}(b_1) = b_1$, $\mathrm{f}(a_2) = a_2$, $\mathrm{f}(a_3) = a_1$, $\mathrm{f}(b_2) = b_1$, and $\mathrm{f}(a_4) = a_2$. By abuse of notation, we also let f denote its natural extension mapping strings to strings.

Now, assume, towards a contradiction, that $\mathrm{EC}(r^{k,\infty})$ is not strongly deterministic, and assume first that this is due to the fact that $\mathrm{EC}(r^{k,\infty})$ is not weakly deterministic. Hence, there exist marked strings $\overline{u}, \overline{v}, \overline{w}$ and marked symbols $\overline{x}$ and $\overline{y}$ such that $\overline{uxv}$ and $\overline{uyw}$ in $L(\overline{\mathrm{EC}(r^{k,\infty})})$ with $\overline{x} \neq \overline{y}$ and $\mathrm{dm}(\overline{x}) = \mathrm{dm}(\overline{y})$. We distinguish two cases. First, assume $\mathrm{f}(\overline{x}) \neq \mathrm{f}(\overline{y})$. Then, as both $\mathrm{f}(\overline{u})\mathrm{f}(\overline{x})\mathrm{f}(\overline{v})$ and $\mathrm{f}(\overline{u})\mathrm{f}(\overline{x})\mathrm{f}(\overline{w})$ are in $L(\overline{r}^{k,\infty})$ we immediately obtain a contradiction with

the weak, and thus strong, determinism of $r^{k,\infty}$. Second, assume $\mathrm{f}(\overline{x}) = \mathrm{f}(\overline{y})$. Then, $\overline{x}$ and $\overline{y}$ cannot occur in the same $\overline{\mathrm{ECE}(r)}_i$, for any $i \in [1, k+1]$. Indeed, otherwise $\mathrm{ECE}(r)$ would not be weakly deterministic, contradicting the induction hypothesis. Hence, assume $\overline{x} \in \mathrm{Char}(\overline{\mathrm{ECE}(r)}_i)$ and $\overline{y} \in \mathrm{Char}(\overline{\mathrm{ECE}(r)}_j)$, with $1 \leq i < j \leq k+1$. But then, consider the strings $w_1 = \mathrm{dm}(\overline{uxv})$ and $w_2 = \mathrm{dm}(\overline{uyw})$ and notice that $\mathrm{dm}(\overline{ux}) = \mathrm{dm}(\overline{uy})$. Let $[_m$ be the opening bracket corresponding to the iterator $\overline{r}^{k,\infty}$ in the bracketed version of $\overline{r}^{k,\infty}$. Then, we can construct two well-bracketed words defined by the bracketing of $\overline{r}^{k,\infty}$ by adding brackets to w_1 and w_2 such that (i) in the prefix $\mathrm{dm}(\overline{ux})$ of w_1, i $[_m$-brackets occur (indicating i iterations of the top iterator) and (ii) in the prefix $\mathrm{dm}(\overline{uy})$ of w_2, j $[_m$-brackets occur. But, as $\mathrm{dm}(\overline{ux}) = \mathrm{dm}(\overline{uy})$ this implies that $\overline{r}^{k,\infty}$ is not strongly deterministic, a contradiction.

Hence, if $\mathrm{EC}(r^{k,\infty})$ is not strongly deterministic, this is due to the second reason in Definition 3. It is easily seen that, as this reason concerns the iterators, and all subexpressions are strongly deterministic due to the induction hypothesis, it must be the case that $\mathrm{ECE}(r)^*$ is not strongly deterministic. However, arguing as above, concerning brackets instead of marked symbols, it then follows that $r^{k,\infty}$ is also not strongly deterministic. This hence leads to the desired contradiction, and shows that $\mathrm{EC}(r)$ is indeed strongly deterministic.

(2) $\boxed{\cdots}$ This follows from Lemma 5. Indeed, any finite language can clearly be defined by a strongly deterministic expression, while an infinite but ultimately periodic language can be defined by a strongly deterministic expression of the form $a^{n_1}(a^{n_2}(\cdots a^{n_{k-1}}(a^{n_k})^* \cdots + \varepsilon) + \varepsilon)$, where a^{n_i} denotes the n_i-fold concatenation of a.

(3) $\boxed{\cdots}$ A witness for non-inclusion is $r = (a^{2,3}b?)^*$. As r is weakly deterministic, $L(r) \in \mathrm{DET}^{\#}_W(\Sigma)$. By applying the algorithm of Brüggemann-Klein and Wood on r for testing whether a regular language $L(r)$ is in $\mathrm{DET}_W(\Sigma)$ [2], it can be seen that $L(r) \notin \mathrm{DET}_W(\Sigma)$. By (1), we therefore have that $L(r) \notin \mathrm{DET}^{\#}_S(\Sigma)$.

(4) $\boxed{\cdots}$ A witness for non-inclusion is $r = (aaa)^*(a + aa)$. We can again easily see that $L(r) \notin \mathrm{DET}_W(\Sigma)$ by applying the algorithm of Brüggemann-Klein and Wood [2] to $L(r)$. As r is over a unary alphabet, from the results above it follows that $L((aaa)^*(a + aa)) \notin \mathrm{DET}^{\#}_W(\Sigma)$, and hence $\mathrm{DET}^{\#}_W(\Sigma) \subsetneq \mathrm{REG}(\Sigma)$. □

Proofs for Section 4

Lemma 18. *Let r be a strongly deterministic regular expression over alphabet $\{a\}$ with only one occurrence of a. Then $L(r)$ can be defined by one of the following expressions:*

(1) $(a^{k,k})^{x,y}$
(2) $(a^{k,k})^{x,y} + \varepsilon$

where $k \in \mathbb{N} \setminus \{0\}$, $x \in \mathbb{N}$, and $y \in (\mathbb{N} \setminus \{0\}) \cup \{\infty\}$.

Proof. First, suppose that $L(r)$ does not contain ε. Since r has one occurrence of a, r is a nesting of iterators. However, since r is strongly deterministic, r can not have subexpressions of the form $(s)^{x,y}$ with $L(s) > 1$ and $(x, y) \neq (1, 1)$. It follows immediately that r is of the form $((\cdots(a^{k_1,k_1})^{k_2,k_2}\cdots)^{k_n,k_n})^{x,y}$. Setting $k := k_1 \times \cdots \times k_n$ proves this case.

Second, suppose that $L(r)$ contains ε. If r is of the form $s + \varepsilon$ and $L(s)$ does not contain ε then we are already done due to the case above. So, the only remaining case is that r is a nesting of iterators. We can assume w.l.o.g. that r does not have subexpressions of the form $s^{1,1}$. Again, since r is strongly deterministic, it cannot have subexpressions of the form $(s)^{x,y}$ with $(x, y) \neq (0, 1)$ and $|L(s) - \{\varepsilon\}| > 1$. Hence, by replacing subexpressions of the form $(s^{k,k})^{\ell,\ell}$ by $s^{k\ell,k\ell}$ and $(s^{0,1})^{0,1}$ by $s^{0,1}$, r can either be rewritten to $(a^{k,k})^{x,y}$ or $((a^{k,k})^{x,y})^{0,1}$. In the first case, the lemma is immediate and, in the second case, we can rewrite r as $((a^k)^{x,y}) + \varepsilon$. □

Proof of Theorem 6: *For every $n \in \mathbb{N}$, there exists an $r \in RE(\Sigma,\#)$ which is weakly deterministic and of size $\mathcal{O}(n)$ such that every strongly deterministic expression s, with $L(r) = L(s)$, is of size at least 2^n.*

Proof. Let r be the weakly deterministic expression $(a^{N+1,2N})^{1,2}$ for $N = 2^n$. Clearly, the size of r is $\mathcal{O}(n)$. Note that r defines all strings of length at least $N + 1$ and at most $4N$, except a^{2N+1}.

We prove that every strongly deterministic expression for $L(r)$ is of size at least 2^n. Thereto, let s be a strongly deterministic regular expression for $L(r)$ with a minimal number of occurrences of the symbol a and, among those minimal expressions, one of minimal size.

We first argue that s is in a similar but more restricted normal form as the one we have in the proof of Lemma 5. If s is minimal and strongly deterministic, then

(a) if s has a disjunction of the form $s_1 + s_2$ then either $L(s_1) = \{\varepsilon\}$ or $L(s_2) = \{\varepsilon\}$;
(b) there are no subexpressions of the form $s_1 s_2$ in s in which $|L(s_1)| > 1$ or $\varepsilon \in L(s_1)$;
(c) there are no subexpressions of the form $s_1^{1,1}$, $(s_1^{0,1})^{0,1}$, $aa^{k,\ell}$, $a^{k,\ell}a$, $a^{k_1,\ell_1}a^{k_2,\ell_2}$, or $(a^{k,k})^{\ell,\ell}$;
(d) there are no subexpressions of the form $(s_1)^{k,\ell}$ with $(k, \ell) \neq (0, 1)$ and $|L(s_1) - \{\varepsilon\}| > 1$;

The reasons are as follows:

- (a),(b): otherwise, s is not weakly deterministic;
- (c): otherwise, s is not minimal; and
- (d): otherwise, by (c), $\ell \geq 2$, which implies that s is not strongly deterministic.

Due to (a), we can assume w.l.o.g. that s does not have any disjunctions. Indeed, when $L(s_2) = \{\varepsilon\}$, we can replace every subexpression $s_1 + s_2$ or $s_2 + s_1$ with

$(s_1)^{0,1}$ since the latter expression has the same or a smaller size as the former ones. From (a)–(d), and the fact that $\varepsilon \notin L(s)$, it then follows that s is of the form

$$s = s_1(s_2(\cdots(s_m)^{0,1}\cdots)^{0,1})^{0,1}$$

where each s_i $(i = 1, \ldots, m-1)$ is of the form $a^{k,k}$. Notice that s_m can still be non-trivial according to (a)–(d), such as $(a^{7,7})^{8,9}$ or $a^{7,7}((a^{2,2})^{0,1})^{2,2}$. We now argue that s_m is either of the form s'_m or $s''_m s'_m$, where s'_m and s''_m have only one occurrence of the symbol a. We already observed above that s_m does not have any disjunctions. So, the only remaining operators are counting and conjunction.

Suppose, towards a contradiction, that s_m has at least two conjunctions (and, therefore, at least three occurrences of a). Because of (b) and (c), s_m cannot be of the form $p_1p_2p_3$, since then $|L(p_1p_2)| = 1$ and p_1p_2 can be rewritten. Hence, s_m is of the form p_1p_2, where either p_1 or p_2 is an iterator which contains a conjunction. If p_1 contains a conjunction we know due to (b) that $|L(p_1)| = 1$ and then p_1 can be rewritten. If p_2 is an iterator that contains a conjunction then p_2 is of the form $p_3^{0,1}$ due to (d) and since $|L(s_m)| = \infty$. Due to (c), p_3 cannot be of the form $p_4^{0,1}$. Due to (d), p_3 cannot be of the form $p_4^{k,\ell}$ with $(k, \ell) \neq (0, 1)$. Hence, $p_3 = p_4p_5$. But this means that $s_m = p_1(p_4p_5)^{0,1}$ which violates the definition of s_m, being the innermost expression in the normal form for s above (i.e., s_m should have been p_4p_5). This shows that s_m has at most one conjunction.

It now follows analogously as in the reasoning for p_3 above that s_m is either of the form s'_m or $s''_m s'_m$, where s'_m and s''_m have only one occurrence of the symbol a.

We will now argue that, for each string of the form $a^{N+1}, \ldots, a^{2N}$ its last position is matched onto a different symbol in s. Formally, let $\overline{u_{N+1}}, \ldots, \overline{u_{2N}}$ be the unique strings in $L(\overline{s})$ such that $|\overline{u_i}| = i$, for every $i \in [N+1, 2N]$. We claim that the last symbol in each $\overline{u_i}$ is different. This implies that s contains at least N occurrences of a, making the size of s at least $N = 2^n$, as desired. Thereto, let $\overline{w_1}$ and $\overline{w_2}$ be two different strings in $\{\overline{u_{N+1}}, \ldots, \overline{u_{2N}}\}$. W.l.o.g., assume $\overline{w_1}$ to be the shorter string. Towards a contradiction, assume that $\overline{w_1}$ and $\overline{w_2}$ end with the same symbol $\overline{x}$. Let $\overline{s} = \overline{s_1}(\overline{s_2}(\cdots(\overline{s_m})^{0,1}\cdots)^{0,1})^{0,1}$. Due to the structure of $\overline{s}$ and since both $\overline{w_1}$ and $\overline{w_2}$ are in $L(\overline{s})$, this implies that $\overline{x}$ cannot occur in $\overline{s_1}, \ldots, \overline{s_{m-1}}$ and that $\overline{x}$ must be the rightmost symbol in $\overline{s}$. As $\overline{s_m}$ is either of the form $\overline{s_m}'$ or $\overline{s_m}''\overline{s_m}'$ this implies $\overline{x}$ occurs in $\overline{s_m}'$. Again due to the structure of s and s_m, this means that s'_m must always define a language of the form $\{w \mid vw \in L(r)\}$, where v is a prefix of $\mathrm{dm}(\overline{w_1})$. Considering the language defined by s, $L(s'_m)$ hence contains the strings $a^i, \ldots, a^{i+k}, a^{i+k+2}, \ldots, a^{i+k+2N}$ for some $i \geq 1$ and $k \geq 0$. However, as s'_m only contains a single occurrence of the symbol a, Lemma 18 says that it cannot define such a language. This is a contradiction.

It follows that the size of s is at least 2^n. □

Proofs for Section 5

Proof of Theorem 8:

1. *Given CNFAs A_1 and A_2, a CNFA A accepting the union or intersection of A_1 and A_2 can be constructed in polynomial time. Moreover, when A_1 and A_2 are deterministic, then so is A.*
2. *Given a CDFA A, a CDFA which accepts the complement of A can be constructed in polynomial time.*
3. MEMBERSHIP *for word w and CDFA A is in time $\mathcal{O}(|w||A|)$.*
4. MEMBERSHIP *for non-deterministic CNFA is* NP*-complete.*
5. EMPTINESS *for CDFAs and CNFAs is* PSPACE*-complete.*
6. *Deciding whether a CNFA A is deterministic is* PSPACE*-complete.*

We first note that a CNFA $A = (Q, q_0, C, \delta, F, \tau)$ can easily be completed. For $q \in Q$, $a \in \Sigma$ define $\text{Formulas}(q, a) = \{\phi \mid (q, a, \phi, \pi, q') \in \delta\}$. We say that A is complete if, for any value function α, it holds that $\alpha \models \bigvee_{\phi \in \text{Formulas}(q,a)} \phi$. That is, for any configuration (q, α) and symbol a, there is always a transition which can be followed by reading a.

Define the completion A^c of A as $A^c = (Q^c, q_0, C, \delta^c, F^c, \tau^c$ with $Q^c = Q \cup \{q_s\}$, δ^c is δ extended with $(q_s, a, \text{true}, \emptyset, q_s)$ and, for all $q \in Q$, $a \in Sigma$, with the tuples $(q, a, \phi^c_{a,q}, \phi, q_s)$, where $\phi^c_{a,q} = \neg \bigvee_{\phi \in \text{Formulas}(q,a)} \phi$. Finally, F^c is F extended with $F^c(q_s) = \text{false}$. Note also that A^c is deterministic if and only if A is deterministic. Hence, from now on we can assume that all CNFAs and CDFAs under consideration are complete.

Proof. (1) Given two complete CNFAs A_1, A_2, where $A_i = (Q_i, q_i, C_i, \delta_i, F_i, \tau_i)$, their union can be defined as $A = (Q_1 \times Q_2, (q_1, q_2), C_1 \uplus C_2, \delta, F, \tau_1 \cup \tau_2)$. Here,

- $\delta = \{((s_1, s_2), a, \phi_1 \wedge \phi_2, \pi_1 \cup \pi_2, (s'_1, s'_2)) \mid (s_i, a, \phi_i, \pi_i, s'_i) \in \delta_i \text{ for } i = 1, 2\}$
- $F(s_1, s_2) = F_1(s_1) \vee F_2(s_2)$.

For the intersection of A_1 and A_2, the definition is completely analogue, only the acceptance condition differs: $F(s_1, s_2) = F(s_1) \wedge F(s_2)$. It is easily verified that if A_1 and A_2 are both deterministic, then A is also deterministic.
(2) Given a complete CDFA $A = (Q, q, C, \delta, F, \tau)$, the CDFA $A' = (Q, q, C, \delta, F', \tau)$, where $F'(q) = \neg F(q)$, for all $q \in Q$, defines the complement of A.
(3) Since A is a CDFA, from every reachable configuration only one transition can be followed when reading a symbol. Hence, we can match the string w from left to right, maintaining at any time the current configuration of A. Hence, we need to make $|w| + 1$ transitions, while every transition can be made in time $\mathcal{O}(|A|)$. Therefore, testing MEMBERSHIP can be done in time $\mathcal{O}(|w||A|)$.
(4) Obviously, we can decide in non-deterministic polynomial time whether a string w is accepted by a CNFA A. It suffices to guess a sequence of $|w|$ transitions, and check whether this sequence forms a valid run of A on w.

To show that it is NP-hard, we do a reduction from BIN PACKING [6]. This is the problem, given a set of weights $W = \{n_1, \ldots, n_k\}$, a packet size p, and the maximum number of packets m; decide whether there is a partitioning of $W = S_1 \uplus S_2 \uplus \cdots \uplus S_m$ such that for each $i \in [1, m]$, we have that $\sum_{n \in S_i} n \leq p$. The latter problem is NP-complete, even when all integers are given in unary.

We will construct a string w and a CNFA A such that $w \in L(A)$ if and only if there exists a proper partitioning for $W = \{n_1, \ldots, n_k\}$, p, and m. Let $\Sigma = \{\#, 1\}$, and $w = \#1^{n_1}\#\#1^{n^2}\#\cdots\#1^{n_k}\#$. Further, A will have an initial state q_0 and for every $j \in [1, m]$ a state q_j and countervariable cv_j. When reading the string w the automaton will non-deterministically guess for each n_i to which of the m sets n_i is assigned (by going to its corresponding state) and maintaining the running sum of the different sets in the countervariables. (As the initial value of the countervariables is 1, we actually store the running sum plus 1) In the end it can then easily be verified whether the chosen partitioning satisfies the desired properties.

Formally, $A = (Q, q_0, \{\mathrm{cv}_1, \ldots, \mathrm{cv}_m\}, \delta, F, \tau)$ can be defined as follows:

- $Q = \{q_0, \ldots, q_m\}$;
- For all $i \in [1, m]$, $(q_0, \#, \mathrm{true}, \emptyset, q_i), (q_i, 1, \mathrm{true}, \{\mathrm{cv}_i{+}{+}\}, q_i), (q_i, \#, \mathrm{true}, \emptyset, q_0) \in \delta$
- $F(q_0) = \bigwedge_{i \in [1,m]} \mathrm{cv}_i \leq p + 1$, and for $q \neq q_0$, $F(q) = \mathrm{false}$; and
- for all $i \in [1, m]$, $\tau(\mathrm{cv}_i) = p + 2$.

(5) We show that EMPTINESS is in PSPACE for CNFAs, and PSPACE-hard for CDFAs. The theorem then follows.

The algorithm for the upperbound guesses a string w which is accepted by A. Instead of guessing w at once, we guess it symbol by symbol and store one configuration γ, such that A can be in γ after reading the already guessed string w. When γ is an accepting configuration, we have guessed a string which is accepted by A, and have thus shown that $L(A)$ is non-empty. Since any configuration can be stored in space polynomial in A (due to the maximum values τ on the countervariables), and PSPACE is closed under complement, the result follows.

We show that the EMPTINESS problem for deterministic CNFAs is PSPACE-hard by a reduction from REACHABILITY of 1-safe petri nets.

A *net* is a triple $N = (S, T, E)$, where S and T are finite sets of *places* and *transitions*, and $E \subseteq (S \times T) \cup (T \times S) \to \{0, 1\}$. The *preset* of a transition t is denoted by $\bullet t$, and defined by $\bullet t = \{s \mid E(s, t) = 1\}$. A *marking* is a mapping $M : S \to \mathbb{N}$. A *Petri net* is a pair $\mathcal{N} = (N, M_0)$, where N is a net and M_0 is the initial marking. A transition t is enabled at a marking M if $M(s) > 0$ for every $s \in \bullet t$. If t is enabled at M, then it can *fire*, and its firing leads to the successor marking M' which is defined for every place s by $M'(s) = M(s) + E(t, s) - E(s, t)$. The expression $M \to_t M'$ denotes that M enables transition t, and that the marking reached by the firing of t is M'. Given a (firing) sequence $\sigma = t_1 \cdots t_n$, $M \Rightarrow_\sigma M'$ denotes that there exist markings $M_1, M_2, \ldots, M_{n-1}$ such that $M \to_{t_1} M_1 \cdots M_{n-1} \to_{t_n} M'$. A Petri net is *1-safe* if $M(s) \leq 1$ for every place s and every reachable marking M.

REACHABILITY for 1-safe petri nets is the problem, given a 1-safe petri net (N, M_0) and a marking M, is M reachable from M_0. That is, does there exist a sequence σ such that $M_0 \Rightarrow_\sigma M$. The latter problem is PSPACE-complete [5].

We construct a CDFA A such that $L(A) = \emptyset$ if and only if M is not reachable from M_0. That is, A will accept all strings $at_1 \cdots t_k$ such that $M_0 \Rightarrow_{t_1 \cdots t_k} M$. To achieve this, A will simulate the working of the Petri net on the firing sequence given by the string. Therefore, we maintain a countervariable for every place s, which will have value 1 when $M'(s) = 0$ and 2 when $M'(s) = 1$, where M' is the current marking. With every transition of the Petri net, we associate a transition in A. Here, the guard ϕ is used to check whether the preconditions for the firing of this transition are satisfied, and the updates are used to update the values of the countervariables according to the transition. The string is accepted if after executing this firing sequence, the countervariables correspond to the given marking M.

Formally, let $N = (S, T, E)$. Then, the CDFA $A = (\{q_0, q_1\}, q_0, S, \delta, F, \tau)$ is defined as follows:

- $(q_0, a, \text{true}, \pi, q_1) \in \delta$, where $\pi = \{s{+}{+} \mid M_0(s) = 1\}$;
- For every $t \in T$: $(q_1, t, \phi, \pi, q_1) \in \delta$, where $\phi = \bigwedge_{s \in {}^\bullet t} s = 2$ and $\pi = \{\text{reset}(s) \mid E(s,t) > E(t,s)\} \cup \{s{+}{+} \mid E(t,s) > E(s,t)\}$;
- $F(q_0) = \text{false}$, and $F(q_1) = \bigwedge_{M(s)=0} s = 1 \wedge \bigwedge_{M(s)=1} s = 2$; and
- For every $s \in S$, $\tau(s) = 2$.

(6) To show that it is in PSPACE, we guess a string w character by character and only store the current configuration. If at any time it is possible to follow more than one transition at once, A is non-deterministic. Since PSPACE is closed under complement, the result follows.

To show that the problem is PSPACE-hard, we do a reduction from ONE RUN of 1-safe petri nets. This is the problem, given a 1-safe petri net $\mathcal{N} = (N, M_0)$, is there exactly one run on N. The latter problem is PSPACE-complete [5].

We construct a CNFA A which is deterministic if and only if ONE RUN is true for $\mathcal{N}$. To do this, we set the alphabet of A to $\{a, t\}$ and A will accept strings of the form at^*, and simulates the working of $\mathcal{N}$. The set of states $Q = \{q_0, q_c, t_1, \ldots, t_n\}$, when $T = \{t_1, \ldots, t_n\}$. Furthermore, there is one countervariable for every place: $C = S$.

The automaton now works as follows. From its initial state q_0 it goes to its central state q_c and sets the values of the countervariables to the initial marking M_0. From q_c, there is a transition to every state t_i. This transition can be followed by reading a t if and only if the values of the countervariables satisfy the necessary conditions to fire the transition t_i. Then, from the states t_i, there is a transition back to q_c, which can be followed by reading a t and is used to update the countervariables according to the firing of t_i. As in the previous proof, the countervariable for place s will have value 1 when $M'(s) = 0$ and 2 when $M'(s) = 1$, where M' is the current marking.

More formally, $A = (\{q_0, q_c\} \cup T, q_0, S, \delta, F, \tau)$ is constructed as follows:

- $(q_0, a, \text{true}, \pi, q_c) \in \delta$, where $\pi = \{s{+}{+} \mid M_0(s) = 1\}$;

– For all $t_i \in T$, $(q_c, t, \phi, \emptyset, t_i) \in \delta$, where $\phi = \bigwedge_{s \in \bullet t_i} s = 2$;
– For all $t_i \in T$, $(t_i, t, \text{true}, \pi, q_c) \in \delta$, where $\pi = \{\text{reset}(s) \mid E(s, t_i) > E(t_i, s)\} \cup \{s{+}{+} \mid E(t_i, s) > E(s, t_i)\}$;
– For all $s \in S$, $\tau(s) = 2$; and
– $F(q) = \text{true}$, for all $q \in Q$. □

Proofs for Section 6

To state Lemma 19, from which Theorem 9 will follow, we first introduce some notation. The goal will be to associate transition sequences of G_r with correctly bracketed words in $\widetilde{\overline{r}}$, the bracketing of $\overline{r}$. A *transition sequence* $\sigma = t_1 \cdots t_n$ of G_r is simply a sequence of transitions of G_r. It is *accepting* if there exists a sequence of configurations $\gamma_0 \gamma_1 \cdots \gamma_n$ such that γ_0 is the inital configuration, γ_n is a final configuration, and, for every $i = 1, \ldots, n-1$, $\gamma_{i-1} \rightarrow_{a_i} \gamma_i$ by using transition $t_i = (q_{i-1}, a_i, \phi_i, \pi_i, q_i)$. Here, for each i, $\gamma_i = (q_i, \alpha_i)$.

Recall that the bracketing $\widetilde{r}$ of an expression r is obtained by associating indices to iterators in r and by replacing each iterator $c := s^{k,\ell}$ of r with index i with $([_i s]_i)^{k,\ell}$. In the following, we use $\text{ind}(c)$ to denote the index i associated to iterator c. As every triple $(\overline{x}, \overline{y}, c)$ in $\text{follow}(\overline{r})$ corresponds to exactly one transition t in G_r, we also say that t *is generated by* $(\overline{x}, \overline{y}, c)$.

We now want to translate transition sequences into correctly bracketed words. Thereto, we first associate a bracketed word $\text{word}_{G_r}(t)$ to each transition t of G_r. We distinguish a few cases:

(i) If $t = (q_0, a, \phi, \pi, q_{\overline{y}})$, with $\text{iterators}(\overline{y}) = [c_1, \ldots, c_n]$, then $\text{word}_{G_r}(t) = [_{\text{ind}(c_n)} \cdots [_{\text{ind}(c_1)} \overline{y}$.
(ii) If $t = (q_{\overline{x}}, a, \phi, \pi, q_{\overline{y}})$ and is generated by $(\overline{x}, \overline{y}, c) \in \text{follow}(\overline{r})$, with $c \neq \text{null}$. Let $\text{iterators}(\overline{x}, c) = [c_1, \ldots, c_n]$ and $\text{iterators}(\overline{y}, c) = [d_1, \ldots, d_m]$. Then, $\text{word}_{G_r}(t) =]_{\text{ind}(c_1)} \cdots]_{\text{ind}(c_n)} [_{\text{ind}(d_m)} \cdots [_{\text{ind}(d_1)} \overline{y}$.
(iii) If $t = (q_{\overline{x}}, a, \phi, \pi, q_{\overline{y}})$ and is generated by $(\overline{x}, \overline{y}, \text{null}) \in \text{follow}(\overline{r})$. Let $\text{iterators}(\overline{x}, \overline{y}) = [c_1, \ldots, c_n]$ and let $\text{iterators}(\overline{y}, \overline{x}) = [d_1, \ldots, d_m]$. Then, $\text{word}_{G_r}(t) =]_{\text{ind}(c_1)} \cdots]_{\text{ind}(c_n)} [_{\text{ind}(d_m)} \cdots [_{\text{ind}(d_1)} \overline{y}$.

Finally, to a marked symbol $\overline{x}$ with $\text{iterators}(\overline{x}) = [c_1, \ldots, c_n]$, we associate $\text{word}_{G_r}(\overline{x}) =]_{\text{ind}(c_1)} \cdots]_{\text{ind}(c_n)}$. Now, for a transition sequence $\sigma = t_1, \ldots, t_n$, where $t_n = (q_{\overline{x}}, a, \phi, \pi, q_{\overline{y}})$, we set $\text{brack-seq}_{G_r}(\sigma) = \text{word}_{G_r}(t_1) \cdots \text{word}_{G_r}(t_n)$ $\text{word}_{G_r}(\overline{y})$. Notice that $\text{brack-seq}_{G_r}(\sigma)$ is a bracketed word over a marked alphabet. We sometimes also say that σ *encodes* $\text{brack-seq}_{G_r}(\sigma)$. We usually omit the subscript G_r from word and brack-seq when it is clear from the context.

For a bracketed word $\widetilde{w}$, let $\text{strip}(w)$ denote the word obtained from $\widetilde{w}$, by removing all brackets. Then, for a transition sequence σ, define $\text{run}(\sigma) = \text{strip}(\text{brack-seq}(\sigma))$.

Lemma 19. *Let $\overline{r}$ be a marked regular expression with counting and G_r the corresponding counting Glushkov automaton for r.*

(1) For every string $\widetilde{\overline{w}}$, we have that $\widetilde{\overline{w}}$ is a correctly bracketed word in $L(\widetilde{\overline{r}})$ if and only if there exists an accepting transition sequence σ of G_r such that $\widetilde{\overline{w}} = \textit{brack-seq}(\sigma)$.
(2) $L(\overline{r}) = \{\textit{run}(\sigma) \mid \sigma$ is an accepting transition sequence of $G_r\}$.

Proof. We first prove (1) by induction on the structure of $\overline{r}$. First, notice that brack-seq(σ) is a correctly bracketed word, for every accepting transition sequence σ on G_r. Hence, we can restrict attention to correctly bracketed words below.

For the induction below, we first fix a bracketing $\widetilde{\overline{r}}$ of r and we assume that all expressions $\widetilde{\overline{s}}$, $\widetilde{\overline{s_1}}$, $\widetilde{\overline{s_2}}$ are correctly bracketed subexpressions of $\widetilde{\overline{r}}$.

- $\overline{s} = \overline{x}$. Then, also $\widetilde{\overline{s}} = \overline{x}$, and $L(\widetilde{\overline{s}}) = \overline{x}$. It is easily seen that the only accepting transition sequence σ of G_s consists of one transition t, with brack-seq(σ) $= \overline{x}$.
- $\overline{s} = \overline{s_1} + \overline{s_2}$. Clearly, the set of all correctly bracketed words in $L(\widetilde{\overline{s}})$ is the union of all correctly bracketed words in $L(\widetilde{\overline{s_1}})$ and $L(\widetilde{\overline{s_2}})$. Further, observe that G_s is constructed from G_{s_1} and G_{s_2} by identifying their initial states and taking the disjoint union otherwise. As the initial states only have outgoing, and no incoming, transitions, the set of accepting transition sequences of G_s is exactly the union of the accepting transition sequences of G_{s_1} and G_{s_2}. Hence, the lemma follows from the induction hypothesis.
- $\overline{s} = \overline{s_1} \cdot \overline{s_2}$. Let $\widetilde{\overline{w}}$ be a correctly bracketed word. We distinguish a few cases. First, assume $\widetilde{\overline{w}} \in \text{Char}(\widetilde{\overline{s_1}})^*$, i.e., $\widetilde{\overline{w}}$ contains only symbols of $\widetilde{\overline{s_1}}$. Then, if $\varepsilon \notin L(s_2)$, $\widetilde{\overline{w}} \notin L(\widetilde{\overline{s}})$, and, by construction of G_s, there is no accepting transition sequence σ on G_s such that brack-seq(σ) $= \widetilde{\overline{w}}$. This is due to the fact that, with $\varepsilon \notin L(\overline{s_2})$, for all $\overline{x} \in \text{Char}(\overline{s_2})$, we have that $\overline{x} \notin \text{last}(\overline{s})$ and hence $F(q_{\overline{x}}) = \text{false}$. Hence, assume $\varepsilon \in L(\overline{s_2})$. Then, $\widetilde{\overline{w}} \in L(\widetilde{\overline{s}})$ if and only if $\widetilde{\overline{w}} \in L(\widetilde{\overline{s_1}})$ if and only if, by the induction hypothesis, there is an accepting transition sequence σ on G_{s_1}, with brack-seq(σ) $= \widetilde{\overline{w}}$. By construction of G_s, and the fact that $\varepsilon \in L(\overline{s_2})$, brack-seq($\sigma$) $= \widetilde{\overline{w}}$ if and only if σ is also an accepting transition sequence on G_s. This settles the case $\widetilde{\overline{w}} \in \text{Char}(\widetilde{\overline{s_1}})^*$. The case $\widetilde{\overline{w}} \in \text{Char}(\widetilde{\overline{s_2}})^*$ can be handled analogously.
  Finally, consider the case that $\widetilde{\overline{w}}$ contains symbols from both $\widetilde{\overline{s_1}}$ and $\widetilde{\overline{s_2}}$. If $\widetilde{\overline{w}}$ is not of the form $\widetilde{\overline{w}} = \widetilde{\overline{w_1}}\widetilde{\overline{w_2}}$, with $w_1 \in \text{Char}(\widetilde{\overline{s_1}})^*$ and $w_2 \in \text{Char}(\widetilde{\overline{s_2}})^*$, then we immediately have that $\widetilde{\overline{w}} \notin L(\widetilde{\overline{s}})$ nor can there be a transition sequence σ on G_s encoding $\widetilde{\overline{w}}$. Hence, assume $\widetilde{\overline{w}}$ is of this form. Then, $\widetilde{\overline{w}} \in L(\widetilde{\overline{s}})$ if and only if $\widetilde{\overline{w_i}} \in L(\widetilde{\overline{s_i}})$, for $i = 1, 2$ if and only if, by the induction hypothesis, there exist accepting transition sequences σ_1 (resp. σ_2) on G_{s_1} (resp. G_{s_2}) encoding $\widetilde{\overline{w_1}}$ (resp. $\widetilde{\overline{w_2}}$). Let $\sigma_1 = t_1, \ldots, t_n$ and $\sigma_2 = t_{n+1}, \ldots, t_m$, with $q_{\overline{x}}$ the target state of t_n, and $q_{\overline{y}}$ the target state of t_{n+1}. Further, let $t = (q_{\overline{x}}, a, \phi, \pi, q_{\overline{y}})$ be the unique transition of G_s generated by the tuple $(\overline{x}, \overline{y}, \text{null}) \in \text{follow}(\overline{s})$. We claim that $\sigma = t_1, \ldots, t_n, t, t_{n+2}, \ldots, t_m$ is an accepting transition sequence on G_s with brack-seq(σ) = brack-seq(σ_1)brack-seq(σ_2), and hence brack-seq(σ) $= \widetilde{\overline{w}}$. To see that σ is an accepting transition sequence on G_s note that the guard $F(q_{\overline{x}})$ (in G_{s_1}) is equal to ϕ, the guard of t. Hence, the

fact that σ_1 is an accepting transition sequence on G_{s_1} ensures that t can be followed in G_s. Further, note that after following transition t, all counters are reset, as they were after following t_{n+1} in G_{s_2}. This ensures that σ_1 and σ_2 can indeed be composed to the accepting transition sequence σ in this manner. Further, to see that $\text{brack-seq}_{G_s}(\sigma) = \text{brack-seq}_{G_{s_1}}(\sigma_1)\text{brack-seq}_{G_{s_2}}(\sigma_2)$ it suffices to observe that $\text{word}_{G_{s_1}}(q_{\overline{x}})\text{word}_{G_{s_2}}(t_{n+1}) = \text{word}_{G_s}(t)$, by definition. Hence, σ is an accepting transition sequence on G_s, with $\text{brack-seq}_{G_s}(\sigma) = \widetilde{\overline{w}}$, as desired. Conversely, we need to show that any such transition sequence σ can be decomposed in accepting transition sequences σ_1 and σ_2 satisfying the same conditions. This can be done using the same reasoning as above.

– $\overline{s} = \overline{s_1}^{[k,\ell]}$. Let $\widetilde{\overline{s}} = ([_j\widetilde{\overline{s_1}}]_j)^{k,\ell}$ and $\widetilde{\overline{w}}$ be a correctly bracketed word.

First, assume $\widetilde{\overline{w}} \in L(\widetilde{\overline{s}})$, we show that there is an accepting transition sequence σ on G_s encoding $\widetilde{\overline{w}}$. As $\widetilde{\overline{w}}$ is correctly bracketed, we can write $\widetilde{\overline{w}} = [_j\widetilde{\overline{w_1}}]_j[_j\widetilde{\overline{w_2}}]_j \cdots [_j\widetilde{\overline{w_n}}]_j$, with $n \in [k,\ell]$, and $\widetilde{\overline{w_i}} \in L(\widetilde{\overline{s_1}})$, $\widetilde{\overline{w_i}} \neq \varepsilon$, for all $i \in [1,n]$. Then, by the induction hypothesis, there exist valid transition sequences $\sigma_1, \ldots, \sigma_n$ on G_{s_1} encoding w_i, for each $i \in [1,n]$. For all $i \in [1,n]$, let $\sigma_i = t_1^i, \ldots, t_{m_i}^i$, for some m_i, the target state of t_1^i be $q_{\overline{y}_i}$ and the target state of $t_{m_i}^i$ be $q_{\overline{x}_i}$. For all $i \in [1,n-1]$, let t^i be the unique transition generated by $(\overline{x}_i, \overline{y}_{i+1}, \overline{s}) \in \text{follow}(\overline{s})$, and define $\sigma = t_1^1 \cdots t_{m_1}^1 t^1 t_2^2 \cdots t_{m_2}^2 t^2 t_2^3 \cdots t^{n-1} t_2^n \cdots t_{m_n}^n$. It now suffices to show that σ is an accepting transition sequence on G_s and $\text{brack-seq}(\sigma) = \widetilde{\overline{w}}$. The reasons are analogous to the ones for the constructed transition sequence σ in the previous case ($\overline{s} = \overline{s_1} \cdot \overline{s_2}$). To see that σ is an accepting transition sequence, note that we simply execute the different transition sequences σ_1 to σ_n one after another, separated by iterations over the topmost iterator $\overline{s}$, by means of the transitions t^1 to t^n. As these transitions reset all counter variables (except $\text{cv}(\overline{s})$) the counter variables on each of these separate runs always have the same values as they had in the runs on G_{s_1}. Therefore, it suffices to argue that the transitions t^i can safely be followed in the run σ, and that we finally arrive in an accepting configuration. This is both due to the fact that we do n such iterations, with $n \in [k,\ell]$, and each iteration increments $\text{cv}(\overline{s})$ by exactly 1. To see that $\text{brack-seq}_{G_s}(\sigma) = \widetilde{\overline{w}}$ note that, by induction, $\widetilde{\overline{w}} = [_j\text{brack-seq}_{G_{s_1}}(\sigma_1)]_j[_j\text{brack-seq}_{G_{s_1}}(\sigma_2)]_j \cdots [_j\text{brack-seq}_{G_{s_1}}(\sigma_n)]_j$. Therefore, it suffices to observe that $\text{word}_{G_s}(t_1^1) = [_j\text{word}_{G_{s_1}}(t_1^1)$, $\text{word}_{G_s}(t_{m_n}^n) = \text{word}_{G_{s_1}}(t_{m_n}^n)]_j$, for all $i \in [1,n-1]$, $\text{word}_{G_s}(t^i) = \text{word}_{G_{s_1}}(\overline{x}_i)]_j[_j\text{word}_{G_{s_1}}(t_1^{i+1})$, and $\text{word}_{G_s}(t) = \text{word}_{G_{s_1}}(t)$, for all other transitions t occurring in σ.

Conversely, assume that there exists an accepting transition sequence σ on G_s encoding $\widetilde{\overline{w}}$. We must show $\widetilde{\overline{w}} \in L(\widetilde{\overline{s}})$. This can be done using arguments analogous to the ones above. It suffices to note that σ contains n transitions t^1 to t^n, for some $n \in [k,\ell]$, generated by a tuple of the form $(\overline{x}, \overline{y}, \overline{s}) \in \text{follow}(\overline{s})$. This allows to decompose σ into n accepting transition sequences σ_1 to σ_n and apply the induction hypothesis to obtain the desired result.

To prove the second point, i.e. $L(\overline{s}) = \{\text{run}(\sigma) \mid \sigma \text{ is an accepting transition sequence on } G_s\}$, it suffices to observe that

$$\begin{aligned} L(\overline{s}) &= \{\text{strip}(\widetilde{\overline{w}}) \mid \widetilde{\overline{w}} \in L(\widetilde{\overline{s}}) \text{ and } \widetilde{\overline{w}} \text{ correctly bracketed}\} \\ &= \{\text{strip}(\text{brack-seq}(\sigma)) \mid \sigma \text{ is an accepting transition sequence on } G_s\} \\ &= \{\text{run}(\sigma) \mid \sigma \text{ is an accepting transition sequence on } G_s\} \end{aligned}$$

Here, the first equality follows immediately from Lemma 20 below, the second equality from the first point of this lemma, and the third is immediate from the definitions. □

The following lemma is immediate from the definitions. Notice that it is important in this lemma that, for subexpressions $s^{k,\ell}$ with s nullable, we have $k = 0$, as required by the normal form for RE(Σ,#).

Lemma 20. *Let $r \in$ RE(Σ,#), and $\widetilde{r}$ be the bracketing of r. Then,*

$$L(r) = \{\textit{strip}(\widetilde{w}) \mid \widetilde{w} \in L(\widetilde{r}) \textit{ and } \widetilde{w} \textit{ is correctly bracketed}\}.$$

We are now ready to prove Theorem 9.

Proof of Theorem 9: *For every RE(Σ,#) expression r, $L(G_r) = L(r)$. Moreover, G_r is deterministic if and only if r is strongly deterministic.*

Proof. Let $r \in \text{RE}(\Sigma,\#)$, $\overline{r}$ a marking of r, and G_r its corresponding counting Glushkov automaton.

We first show that $L(r) = L(G_r)$. Thereto, observe that (1) $L(r) = \{\text{dm}(\overline{w}) \mid \overline{w} \in L(\overline{r})\}$, by definition of $\overline{r}$, and (2) $L(G_r) = \{\text{dm}(\text{run}(\sigma)) \mid \sigma \text{ is an accepting transition sequence on } G_r\}$ by definition of G_r and the run predicate. As, by Lemma 19, $L(\overline{r}) = \{\text{run}(\sigma) \mid \sigma \text{ is an accepting transition sequence on } G_r\}$, we hence obtain $L(r) = L(G_r)$.

We next turn to the second statement: r is strongly deterministic if and only if G_r is deterministic. For the right to left direction, suppose r is not strongly deterministic, we show that then G_r is not deterministic. Here, r can be not strongly deterministic for two reasons: either it is not weakly deterministic, or it is but violates the second criterion in Definition 3.

First, suppose r is not weakly deterministic, and hence there exists words $\overline{u}, \overline{v}, \overline{w}$ and symbols $\overline{x}, \overline{y}$, with $\overline{x} \neq \overline{y}$ but $\text{dm}(\overline{x}) = \text{dm}(\overline{y})$ such that $\overline{uxv} \in L(\overline{r})$, and $\overline{uyw} \in L(\overline{r})$. Then, by Lemma 19 there exist accepting transition sequences $\sigma_1 = t_1, \ldots, t_n$ and $\sigma_2 = t'_1, \ldots, t'_m$ such that $\text{run}(\sigma_1) = \overline{uxv}$ and $\text{run}(\sigma_2) = \overline{uyw}$. Let $|\overline{u}| = i$. Then, $t_{i+1} \neq t'_{i+1}$, as t_{i+1} is a transition to $q_{\overline{x}}$, and t'_{i+1} to $q_{\overline{y}}$, with $q_{\overline{x}} \neq q_{\overline{y}}$. Let $j \in [1, i+1]$ be the smallest number such that $t_j \neq t'_j$, which must exist as $t_{i+1} \neq t'_{i+1}$. Let $u = \text{dm}(\overline{ux})$. Then, after reading the prefix of u of length $j-1$, we have that G_r can be in some configuration γ and can both follow transition t_j and t'_j while reading the jth symbol of u. Hence, G_r is not deterministic.

Second, suppose r is weakly deterministic, but there exist words $\widetilde{u}, \widetilde{v}, \widetilde{w}$ over $\Sigma \cup \Gamma$, words $\alpha \neq \beta$ over Γ, and symbol $a \in \Sigma$ such that $\widetilde{u}\alpha a\widetilde{v}$ and $\widetilde{u}\beta a\widetilde{w}$ are correctly bracketed and in $L(\widetilde{r})$. Consequently, there exist words $\widetilde{\overline{u}}$, $\widetilde{\overline{v}}$, and $\widetilde{\overline{w}}$ and $\overline{x}$ such that $\widetilde{\overline{u}}\alpha\overline{x}\widetilde{\overline{v}}$ and $\widetilde{\overline{u}}\beta\overline{x}\widetilde{\overline{w}}$ are correctly bracketed and in $L(\widetilde{\overline{r}})$. Here, both words must use the same marked symbol $\overline{x}$ for a as r is weakly deterministic. Then, by Lemma 19 there exist transition sequences $\sigma_1 = t_1, \ldots, t_n$, $\sigma_2 = t_1, \ldots, t'_n$ such that brack-seq$(\sigma_1) = \widetilde{\overline{u}}\alpha\overline{x}\widetilde{\overline{v}}$ and brack-seq$(\sigma_2) = \widetilde{\overline{u}}\beta\overline{x}\widetilde{\overline{w}}$. W.l.o.g. we can assume that α and β are chosen to be maximal (i.e. $\widetilde{\overline{u}}$ is either empty or ends with a marked symbol). But then, let $i = |\text{strip}(\widetilde{\overline{u}})|$ and observe that word$(t_{i+1}) = \alpha\overline{x}$ and word$(t'_{i+1}) = \beta\overline{x}$, and thus as word$(t_{i+1}) \neq$ word(t'_{i+1}) also $t_{i+1} \neq t'_{i+1}$. By reasoning exactly as in the previous case, it then follows that G_r is not deterministic.

Before proving the converse direction, we first make two observations. First, for any two transitions t and t', with $t \neq t'$ who share the same source state $q_{\overline{x}}$, we have that word$(t) \neq$ word(t'). Indeed, observe that if the target state of t is $q_{\overline{y}}$, and the target of t' is $q_{\overline{y}'}$, that then word$(t) = \alpha\overline{y}$ and word$(t') = \beta\overline{y}'$, with α, β words over Γ. Hence, if $\overline{y} \neq \overline{y}'$, we immediately have word$(t) \neq$ word(t'). Otherwise, when $\overline{y} = \overline{y}'$, we know that t is computed based on either (1) $(\overline{x}, \overline{y}, \text{null}) \in \text{follow}(\overline{r})$ and t' on $(\overline{x}, \overline{y}, c) \in \text{follow}(\overline{r})$ or (2) $(\overline{x}, \overline{y}, c) \in \text{follow}(\overline{r})$ and t' on $(\overline{x}, \overline{y}, c') \in \text{follow}(\overline{r})$. In both cases it is easily seen that $\alpha \neq \beta$. The second observation we make is that G_r is *reduced*, i.e., for every reachable configuration γ, there is some string which brings γ to an accepting configuration. The latter is due to the upper bound tests present in G_r.

Now, assume that G_r is not deterministic. We show that r is not strongly deterministic. As G_r is reduced and not deterministic there exist words u, v, w and symbol a such that $uav, uaw \in L(G_r)$ witnessed by accepting transition sequences $\sigma_1 = t_1, \ldots, t_n$ and $\sigma_2 = t'_1, \ldots, t'_m$ (for uav and uaw, respectively), such that $t_i = t'_i$, for all $i \in [1, |u|]$, but $t_{|u|+1} \neq t'_{|u|+1}$. Let $q_{\overline{x}}$ be the target of transition $t_{|u|+1}$ and $q_{\overline{y}}$ the target of transition $t'_{|u|+1}$, and $\overline{x}$ and $\overline{y}$ their associated marked symbols. Note that $\text{dm}(\overline{x}) = \text{dm}(\overline{y}) = a$. We now distinguish two cases.

First, assume $\overline{x} \neq \overline{y}$. By Lemma 19, both run$(\sigma_1) \in L(\overline{r})$ and run$(\sigma_2) \in L(\overline{r})$. Writing run$(\sigma_1) = \overline{u}\overline{x}\overline{v}$ and run$(\sigma_2) = \overline{u}\overline{y}\overline{w}$ (such that $|u| = |\overline{u}|$, $|v| = |\overline{v}|$ and $|w| = |\overline{w}|$) we then obtain the strings $\overline{u}$, $\overline{v}$, $\overline{w}$ and symbols $\overline{x}, \overline{y}$ sufficient to show that r is not weakly deterministic, and hence also not strongly deterministic.

Second, assume $\overline{x} = \overline{y}$. We then consider brack-seq(σ_1) and brack-seq(σ_2) which, by Lemma 19, are both correctly bracketed words in $L(\widetilde{\overline{r}})$. Further, note that $t_{|u|+1}$ and $t'_{|u|+1}$ share the same source state, and hence, by the first observation above, word$(t_{|u|+1}) = \alpha\overline{x} \neq$ word$(t'_{|u|+1}) = \beta\overline{y}$. In particular, as $\overline{x} = \overline{y}$, it holds that $\alpha \neq \beta$. But then, writing brack-seq$(\sigma_1) = \widetilde{\overline{u}}\alpha\overline{x}\widetilde{\overline{v}}$, and brack-seq$(\sigma_2) = \widetilde{\overline{u}}\alpha\overline{y}\widetilde{\overline{w}}$, we can see that $\text{dm}(\widetilde{\overline{u}})$, $\text{dm}(\widetilde{\overline{v}})$, $\text{dm}(\widetilde{\overline{w}})$, α, β, and a, violate the condition in Definition 3 and hence show that r is not strongly deterministic. □

Proofs for Section 7

In the following, for two symbols $\overline{x}$, $\overline{y}$ of a marked expression $\overline{r}$, we denote by $\mathrm{lca}(\overline{x},\overline{y})$ the smallest subexpression of $\overline{r}$ containing both $\overline{x}$ and $\overline{y}$. Further, we write $\overline{s} \preceq \overline{r}$ when $\overline{s}$ is a subexpression of $\overline{r}$ and $\overline{s} \prec \overline{r}$ when $\overline{s} \preceq \overline{r}$ and $\overline{s} \neq \overline{r}$.

Proof of Theorem 10: *For every $r \in RE(\Sigma,\#)$, $isStrongDeterministic(r)$ returns true if and only if r is strongly deterministic. Moreover, it runs in time $\mathcal{O}(|r|^3)$.*

Proof. Let $r \in \mathrm{RE}(\Sigma,\#)$, $\overline{r}$ a marking of r, and G_r the corresponding Glushkov counting automaton. By Theorem 9 it suffices to show that $isStrongDeterministic(r)$ returns true if and only if G_r is deterministic. Thereto, we first extract from Algorithm 1 the reasons for $isStrongDeterministic(r)$ to return false. This is the case if and only if there exist marked symbols $\overline{x},\overline{y},\overline{y}'$, with $\mathrm{dm}(\overline{y}) = \mathrm{dm}(\overline{y}')$ such that either

1. $\overline{y},\overline{y}' \in \mathrm{first}(\overline{s})$ and $\overline{y} \neq \overline{y}'$ (Line 4)
2. $(\overline{x},\overline{y},c) \in \mathrm{follow}(\overline{s})$, $(\overline{x},\overline{y}',c) \in \mathrm{follow}(\overline{s})$, $\overline{y} \neq \overline{y}'$ and $\mathrm{upper}(c) \geq 2$ (Line 10)
3. $(\overline{x},\overline{y},c) \in \mathrm{follow}(\overline{s})$, $(\overline{x},\overline{y}',c') \in \mathrm{follow}(\overline{s})$, $c \prec c'$, and $\mathrm{upper}(c) \geq 2$, $\mathrm{upper}(c') \geq 2$, and $\mathrm{upper}(c) > \mathrm{lower}(c)$ (Line 12)
4. $(\overline{x},\overline{y},\mathrm{null}) \in \mathrm{follow}(\overline{s})$, $(\overline{x},\overline{y}',c') \in \mathrm{follow}(\overline{s})$, $\mathrm{lca}(\overline{x},\overline{y}) \prec c'$ and $\mathrm{upper}(c') \geq 2$ (Line 12)
5. $(\overline{x},\overline{y},c) \in \mathrm{follow}(\overline{s})$, $(\overline{x},\overline{y}',\mathrm{null}) \in \mathrm{follow}(\overline{s})$, $c \prec \mathrm{lca}(\overline{x},\overline{y}')$, $\mathrm{upper}(c) \geq 2$, and $\mathrm{upper}(c) > \mathrm{lower}(c)$ (Line 12)
6. $(\overline{x},\overline{y},\mathrm{null}) \in \mathrm{follow}(\overline{s})$, $(\overline{x},\overline{y}',\mathrm{null}) \in \mathrm{follow}(\overline{s})$, $\overline{y} \neq \overline{y}'$ and $\mathrm{lca}(\overline{x},\overline{y}) = \mathrm{lca}(\overline{x},\overline{y}')$ (Line 7)
7. $(\overline{x},\overline{y},\mathrm{null}) \in \mathrm{follow}(\overline{s})$, $(\overline{x},\overline{y}',\mathrm{null}) \in \mathrm{follow}(\overline{s})$, and $\mathrm{lca}(\overline{x},\overline{y}) \prec \mathrm{lca}(\overline{x},\overline{y}')$ (Line 12)

We now show that G_s is not deterministic if and only if one of the above seven conditions holds. We first verify the right to left direction by investigating the different cases.

Suppose case 1 holds, i.e., $\overline{y},\overline{y}' \in \mathrm{first}(\overline{s})$ and $\overline{y} \neq \overline{y}'$. Then, there is a transition from q_0 to $q_{\overline{y}}$ and one from q_0 to $q_{\overline{y}'}$, with $q_{\overline{y}} \neq q_{\overline{y}'}$. These transitions can both be followed when the first symbol in a string is $\mathrm{dm}(\overline{y})$. Hence, G_s is not deterministic.

In each of the six remaining there are always two distinct tuples in $\mathrm{follow}(\overline{s})$ which generate distinct transitions with as source state $q_{\overline{x}}$ and target states $q_{\overline{y}}$ and $q_{\overline{y}'}$, and $\mathrm{dm}(\overline{y}) = \mathrm{dm}(\overline{y}')$. Therefore, it suffices, in each of the cases, to construct a reachable configuration $\gamma = (q_{\overline{x}},\alpha)$ from which both transitions can be followed by reading $\mathrm{dm}(\overline{y})$. The reachability of this configuration γ will always follow from Lemma 21, but its precise form depends on the particular case we are in. In each of the following cases, let $\mathrm{iterators}(\overline{x}) = [c_1,\ldots,c_n]$ and, when applicable, set $c = c_i$ and $c' = c_j$. When both c and c' occur, we always have $c \prec c'$, and hence $i < j$.

Case 2: Set $\gamma = (q_{\overline{x}}, \alpha)$ with $\alpha(\mathrm{cv}(c_m)) = \mathrm{lower}(c_m)$, for all $m \in [1, i-1]$, and $\alpha(\mathrm{cv}(c_m)) = 1$, for all $m \in [i, m]$. We need to show that the transitions generated by $(\overline{x}, \overline{y}, c)$ and $(\overline{x}, \overline{y}', c)$ can both be followed from γ. This is due to the fact that $\alpha \models \text{value-test}_{[c_1,\ldots,c_{i-1}]}$ and $\alpha \models \text{upperbound-test}_{[c_i]}$. The latter because $\mathrm{upper}(c_i) \geq 2 > \alpha(\mathrm{cv}(c_i))$.

Case 3: Set $\gamma = (q_{\overline{x}}, \alpha)$ with $\alpha(\mathrm{cv}(c_m)) = \mathrm{lower}(c_m)$, for all $m \in [1, j-1]$, and $\alpha(\mathrm{cv}(c_m)) = 1$, for all $m \in [j, n]$. To see that the transition generated by $(\overline{x}, \overline{y}, c)$ can be followed, note that again $\alpha \models \text{value-test}_{[c_1,\ldots,c_{i-1}]}$ and $\alpha \models \text{upperbound-test}_{[c_i]}$. The latter due to the fact that $\mathrm{upper}(c) > \mathrm{lower}(c) = \alpha(\mathrm{cv}(c_i))$. On the other hand, also $\alpha \models \text{value-test}_{[c_1,\ldots,c_{j-1}]}$ and $\alpha \models \text{upperbound-test}_{[c_j]}$ as $\mathrm{upper}(c') \geq 2 > \alpha(\mathrm{cv}(c_j))$. Hence, G_s can also follow the transition generated by $(\overline{x}, \overline{y}', c')$.

Case 4: Set $\gamma = (q_{\overline{x}}, \alpha)$ with $\alpha(\mathrm{cv}(c_m)) = \mathrm{lower}(c_m)$, for all $m \in [1, j-1]$, and $\alpha(\mathrm{cv}(c_m)) = 1$, for all $m \in [j, n]$. Arguing as above, and using the facts that (1) $\mathrm{upper}(c') \geq 2$ and (2) $\mathrm{iterators}(\overline{x}, \overline{y}) = [c_1, \ldots, c_{i'}]$, for some $i' < j$ (as $\mathrm{lca}(\overline{x}, \overline{y}) \prec c'$); we can deduce that G_s can again follow both transitions.

Cases 5, 6, and 7: In each of these cases we can set $\gamma = (q_{\overline{x}}, \alpha)$ with $\alpha(\mathrm{cv}(c_m)) = \mathrm{lower}(c_m)$, for all $m \in [1, n]$. Arguing as before it can be seen that in each case both transitions can be followed from this configuration.

We next turn to the left to right direction. Assume G_s is not deterministic and let γ be a reachable configuration such that two distinct transitions t_1, t_2 can be followed by reading a symbol a. We argue that in this case the conditions for one of the seven cases above must be fulfilled. First, if $\gamma = (q_0, \alpha)$ then t_1 and t_2 must be the initial transitions of a run, as there are no transitions returning to q_0. As, furthermore, q_0 has exactly one transition to each state $q_{\overline{x}}$, when $\overline{x} \in \mathrm{first}(\overline{s})$, we can see that the conditions in case 1 hold.

Therefore, assume $\gamma = (q_{\overline{x}}, \alpha)$, with $\overline{x} \in \mathrm{Char}(\overline{r})$. We will investigate the tuples in $\mathrm{follow}(\overline{s})$ which generated t_1 and t_2 and show that they fall into one of the six remaining cases mentioned above, hence forcing $isStrongDeterministic(s)$ to return false. There are indeed six possibilities, ignoring symmetries, which we immediately classify to the case they will belong to:

2. t_1 is generated by $(\overline{x}, \overline{y}, c)$ and t_2 by $(\overline{x}, \overline{y}', c)$,
3. t_1 is generated by $(\overline{x}, \overline{y}, c)$ and t_2 by $(\overline{x}, \overline{y}', c')$ with $c \prec c'$.
4. t_1 is generated by $(\overline{x}, \overline{y}, \mathrm{null})$ and t_2 by $(\overline{x}, \overline{y}', c')$ with $\mathrm{lca}(\overline{x}, \overline{y}) \prec c'$.
5. t_1 is generated by $(\overline{x}, \overline{y}, c)$ and t_2 by $(\overline{x}, \overline{y}', \mathrm{null})$ and $c \prec \mathrm{lca}(\overline{x}, \overline{y})$.
6. t_1 is generated by $(\overline{x}, \overline{y}, \mathrm{null})$ and t_2 by $(\overline{x}, \overline{y}', \mathrm{null})$ with $\mathrm{lca}(\overline{x}, \overline{y}) = \mathrm{lca}(\overline{x}, \overline{y}')$.
7. t_1 is generated by $(\overline{x}, \overline{y}, \mathrm{null})$ and t_2 by $(\overline{x}, \overline{y}', \mathrm{null})$ with $\mathrm{lca}(\overline{x}, \overline{y}) \prec \mathrm{lca}(\overline{x}, \overline{y}')$.

Note that all subexpressions under consideration (i.e. $\mathrm{lca}(\overline{x}, \overline{y})$, $\mathrm{lca}(\overline{x}, \overline{y}')$, c, and c') contain x, and that $\mathrm{lca}(\overline{x}, \overline{y})$ and $\mathrm{lca}(\overline{x}, \overline{y}')$ are always subexpressions whose topmost operator is a concatenation. These are the reasons why all these expressions are in a subexpression relation, and why we never have, for instance, $\mathrm{lca}(\overline{x}, \overline{y}) = c$.

However, these six cases only give us the different possibilities of how the transitions t_1 and t_2 can be generated by tuples in $\mathrm{follow}(\overline{s})$. We still need to

argue that the additional conditions imposed by the cases mentioned above apply. Thereto, we first note that for all iterators c and c' under consideration, $\text{upper}(c) \geq 2$ and $\text{upper}(c') \geq 2$ must surely hold. Indeed, suppose for instance $\text{upper}(c) = 1$. Then, for any transition t generated by $(\overline{x}, \overline{y}, c)$ the guard ϕ contains the condition $\text{upperbound-test}_c := \text{cv}(c) < 1$, which can never be true. Hence, such a transition t can never be followed and is not relevant. This already shows that possibilities 4 and 7 above, indeed imply cases 4 and 7, respectively. For the additional cases, we argue on a case by case basis.

Cases 2 and 6: We additionally need to show $\overline{y} \neq \overline{y}'$, which is immediate from the fact that $t_1 \neq t_2$. Indeed, assuming $\overline{y} = \overline{y}'$, implies that t_1 and t_2 are generated by the same tuple in $\text{follow}(\overline{s})$ and are hence equal.

Cases 3 and 5: We need to show $\text{upper}(c) > \text{lower}(c)$. In both cases, the guard of transition t_1 contains the condition $\text{cv}(c) < \text{upper}(c)$ as upperbound test, whereas the guard of transition t_2 contains the condition $\text{cv}(c) \geq \text{lower}(c)$. These can only simultaneously be true when $\text{upper}(c) > \text{lower}(c)$.

This settles the correctness of the algorithm. We conclude by arguing that the algorithm runs in time $\mathcal{O}(|r|^3)$. Computing the first and last sets for each $\overline{s} \preceq \overline{r}$ can easily be done in time $\mathcal{O}(|r|^3)$ as can the test on Line 4. Further, the for loop iterates over a linear number of nodes in the parse tree of $\overline{r}$. To do each iteration of the loop in quadratic time, one needs to implement the set Follow as a two-dimensional boolean table. In each iteration we then need to do at most a quadratic number of (constant time) lookups and writes to the table. Altogether this yields a quadratic algorithm.

In the proof below, we use $\text{base}(s^{k,\ell})$ to denote s.

Lemma 21. *Let $\overline{x} \in \mathit{Char}(\overline{r})$, and $\mathit{iterators}(\overline{x}) = [c_1, \ldots, c_n]$. Let $\gamma = (q_{\overline{x}}, \alpha)$, be a configuration such that $\alpha(\mathit{cv}(c_i)) \in [1, \mathit{upper}(c_i)]$, for $i \in [1, n]$, and $\alpha(\mathit{cv}(c)) = 1$, for all other countervariables c. Then, γ is reachable in G_r.*

Proof. This lemma can easily be proved by induction on the structure of r. However, as this a bit tedious, we only provide some intuition by constructing, given such a configuration γ, a string w which brings G_s from its initial configuration to γ.

We construct w by concatenating several substrings. Thereto, for every $i \in [1, n]$, let v_i be a non-empty string in $L(\text{base}(c_i))$. Concatenating such a string v_i with itself allows to iterate c_i. We further define, for every $i \in [1, n]$, a marked string $\overline{w_i}$ which, intuitively, connects the different iterators. Thereto, let $\overline{w_n}$ be a minimal (w.r.t. length) string such that $\overline{w_n} \in (\text{Char}(\overline{s}) \setminus \text{Char}(\overline{c_n}))^*$ and such that there exist $\overline{u} \in \text{Char}(\overline{c_n})^*$ and $\overline{v} \in \text{Char}(\overline{s})^*$ such that $\overline{w_n u x v} \in L(\overline{s})$. Similarly, for any $i \in [1, n-1]$, let $\overline{w_i}$ be a minimal (w.r.t. length) string such that $\overline{w_i} \in (\text{Char}(\overline{c_{i+1}}) \setminus \text{Char}(\overline{c_i}))^*$ and there exist $\overline{u} \in \text{Char}(\overline{c_i})^*$ and $\overline{v} \in \text{Char}(\overline{c_{i+1}})^*$ such that $\overline{w_n u x v} \in L(\overline{c_{i+1}})$. Finally, let $\overline{w_0}$ be a string such that there exists a $\overline{u}$ such that $\overline{w_0 x u} \in L(\text{base}(c_1))$. We require these strings $\overline{w_1}$ to $\overline{w_n}$ to be minimal to be sure that they do not allow to iterate over their corresponding counter.

Then, the desired string w is

$$\text{dm}(\overline{w_n})(v_n)^{\alpha(\text{cv}(c_n))}\text{dm}(\overline{w_{n-1}})(v_{n-1})^{\alpha(\text{cv}(c_{n-1}))} \cdots \text{dm}(\overline{w_0})\text{dm}(\overline{x})\,.$$

Proof of Theorem 12:

1. EQUIVALENCE *and* INCLUSION *for* $DET_S^{\#}(\Sigma)$ *are in* PSPACE.
2. INTERSECTION *for* $DET_W^{\#}(\Sigma)$ *and* $DET_S^{\#}(\Sigma)$ *is* PSPACE*-complete.*

Proof. (1): We show that INCLUSION for $\mathrm{DET}_S^{\#}(\Sigma)$ is in PSPACE. Given two strongly deterministic $\mathrm{RE}(\Sigma,\#)$ expressions r_1, r_2, we construct two CDFAs A_1, A_2 for r_1 and r_2 using the construction of section 6, which by Theorem 9 are indeed deterministic. Then, we construct the CDFAs A_2', A such that A_2' accepts the complement of A_2, and A is the intersection of A_1 and A_2'. This can all be done in polynomial time and preserves determinism according to Theorem 8. Finally, $L(r_1) \subseteq L(r_2)$ if and only if $L(A) \neq \emptyset$, which can be decided in PSPACE by Theorem 8(1).

(2): The result for $\mathrm{DET}_W^{\#}(\Sigma)$ is immediate from Theorem 11(1) and (2). This result also carries over to $\mathrm{DET}_S^{\#}(\Sigma)$. For the upper bound this is trivial, whereas the lower bound follows from the fact that standard weakly deterministic regular expressions can be transformed in linear time into equivalent strongly deterministic expressions [1]. □