International Journal of Foundations of Computer Science
© World Scientific Publishing Company

COMPRESSED MEMBERSHIP PROBLEMS FOR REGULAR EXPRESSIONS AND HIERARCHICAL AUTOMATA[*]

MARKUS LOHREY
Universität Leipzig, Institut für Informatik, Germany
lohrey@informatik.uni-leipzig.de

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

Membership problems for compressed strings in regular languages are investigated. Strings are represented by straight-line programs, i.e., context-free grammars that generate exactly one string. For the representation of regular languages, various formalisms with different degrees of succinctness (e.g., suitably extended regular expressions, hierarchical automata) are considered. Precise complexity bounds are derived. Among other results, it is shown that the compressed membership problem for regular expressions with intersection is PSPACE-complete. This solves an open problem of Plandowski and Rytter.

Keywords: algorithms on compressed strings; straight-line programs; regular expressions.

1991 Mathematics Subject Classification: 68Q45, 68Q68

1. Introduction

The topic of this paper is algorithms on compressed strings. The goal of such algorithms is to check properties of compressed strings and thereby beat a straightforward "decompress-and-check" strategy. Potential applications for such algorithms can be found for instance in bioinformatics, where massive volumes of string data are stored and analyzed. In this paper we concentrate on compressed membership problems for regular languages, i.e., we want to check whether a compressed string belongs to a given regular language. Here, the input consists of two components: (i) a compressed string and (ii) a regular language. In order to obtain a precise problem description we have to specify the data representation in (i) and (ii). In (i), we choose *straight-line programs* (SLPs); these are context-free grammars that generate exactly one word. Straight-line programs turned out to be a very flexible and mathematically clean compressed representation of strings. Several other dictionary-based compressed representations, like for instance Lempel-Ziv (LZ) factorizations [26], can be converted in polynomial time into straight-line programs

[*]This work is supported by the DFG research project ALKODA.

and vice versa [23]. This implies that complexity results can be transfered from SLP-encoded input strings to LZ-encoded input strings.

For point (ii) we consider various formalisms for describing regular languages with different degrees of succinctness. In Section 3, we consider regular expressions (i.e., expressions using the operators $\cup, \cdot$, and $*$) extended by some of the operators intersection ($\cap$), complement ($\neg$), squaring (^{2}), and shuffle ($\|$). For a set $\mathcal{C}$ of operators, we denote with CMP($\mathcal{C}$) (resp. MP($\mathcal{C}$)) the problem of checking whether an SLP-compressed (resp. uncompressed) word belongs to the language described by an expression over the operators from $\mathcal{C}$.

The complexity of the uncompressed version MP($\mathcal{C}$) is a well studied topic, see, e.g., [10, 20, 21, 24, 25]. Let us recall some of the results (for other operator sets the precise complexity status is not known to the author):

- MP($\{\cdot, \cup\}$) and MP($\{\cdot, \cup, *\}$) are both NL-complete [9, 10].
- MP($\{\cdot, \cup, *, \cap\}$) is complete for LOGCFL (the logspace closure of the class of context-free languages) [21]. The exact complexity of MP($\{\cdot, \cup, \cap\}$) seems to be open.
- MP($\{\cdot, \cup, \neg\}$) and MP($\{\cdot, \cup, *, \neg\}$) are both P-complete [20].
- MP($\mathcal{C} \cup \{\|\}$) is NP-complete for every $\mathcal{C} \subseteq \{\cup, \cdot, *, \cap\}$ that contains at least one of the operators $\cup$, $\cdot$, or $*$ [18, 25].

For some particular operator sets $\mathcal{C}$, the complexity of the compressed version CMP($\mathcal{C}$) was studied in [17, 23]. In [17] it is shown that there exists a fixed regular language L such that the compressed membership problem for L is P-complete. In the uniform version, where a representation of the regular language is also part of the input, first results were obtained in [23]. If the regular language is represented by a finite automaton of size m, compressed membership can be solved in time $O(nm^3)$, where n is the size of the input SLP. Non-trivial hardness results were obtained for more succinct representations of regular languages: Among others, it was shown in [23] that CMP($\{\cdot, \cup, *, \cap\}$) is NP-hard, and it was conjectured that this problem is NP-complete.

In this paper we characterize the complexity of CMP($\mathcal{C}$) for all operator sets $\mathcal{C}$ with $\{\cdot, \cup\} \subseteq \mathcal{C} \subseteq \{\cdot, \cup, *, \cap, \neg, \|, {}^2\}$, see also Section 5 for a summary. In most cases, the complexity turns out to be PSPACE-complete: CMP($\mathcal{C}$) is PSPACE-hard if $\mathcal{C} = \{\cdot, \cup\} \cup \mathcal{D}$ and $\mathcal{D}$ is one the following sets: $\{\neg\}$, $\{^2\}$, $\{*, \cap\}$, $\{*, \|\}$, in particular the conjecture from [23] regarding CMP($\{\cdot, \cup, *, \cap\}$) is false unless $\mathsf{NP} = \mathsf{PSPACE}$. Concerning upper bounds, we show that if $\mathcal{C} \subseteq \{\cdot, \cup, *, \cap, \neg, \|, {}^2\}$ does not contain simultaneously $\neg$ and $\|$, then CMP($\mathcal{C}$) belongs to PSPACE. On the other hand, the presence of both negation and shuffle makes compressed membership more difficult: CMP($\mathcal{C}$) is complete for alternating exponential time with a linear number of alternations ($\mathsf{ATIME}(\exp(n), O(n))$) in case $\{\cdot, \cup, \neg, \|\} \subseteq \mathcal{C}$. Completeness results for $\mathsf{ATIME}(\exp(n), O(n))$ are typical for logical theories [4], but we are not aware of any completeness results for this class in formal language theory.

Technically, the most difficult result in this paper is the PSPACE-hardness of CMP($\{\cdot, \cup, *, \cap\}$). Our proof is based on a technique from [13]. The idea is to encode the transition graph G_n of a linear bounded automaton on configurations of length n by a string W_n, which can be generated by an SLP of size $O(n)$. In [13], we constructed a fixed deterministic context-free language L such that $W_n \in L$ if and only if in the transition graph G there is a path from the initial configuration to a final configuration. Hence, the compressed membership problem for the fixed deterministic context-free language L is PSPACE-complete; another proof for this result (that works even for visibly pushdown languages) was recently given in [15]. Here, we show that reachability in the configuration graph G_n can be also checked by a semi-extended regular expression ρ_n. But in contrast to the language L from [13], the expression ρ_n has to be part of the input (which is clear since compressed membership in a fixed regular language is in P).

Using this technique, we also prove that the compressed membership problem for *hierarchical automata* is PSPACE-complete. In hierarchical automata, states may either represent atomic states or references to submodules, which again may contain references to other submodules and so on, see, e.g., [1]. In this way, automata of exponential size may be represented by a polynomial size structure. PSPACE-completeness of the compressed membership problem for hierarchical automata was already shown in [6]. Our lower bound is slightly stronger, since our construction yields *deterministic* hierarchical automata. For *alternating* hierarchical automata, our main constructions can be adapted in order to prove EXPTIME-completeness of the compressed membership problem.

2. Preliminaries

2.1. *General notations*

Let Γ be a finite alphabet. The *empty word* is denoted by ε. Let $s = a_1 a_2 \cdots a_n \in \Gamma^*$ be a word over Γ, where $a_i \in \Gamma$ for $1 \leq i \leq n$. The *length* of s is $|s| = n$. For $a \in \Gamma$ and $1 \leq i \leq j \leq n$ let $|s|_a = |\{k \mid a_k = a\}|$, $s[i] = a_i$, and $s[i,j] = a_i a_{i+1} \cdots a_j$. If $i > j$ we set $s[i,j] = \varepsilon$. The string s is a *subsequence* of $t \in \Gamma^*$, briefly $s \hookrightarrow t$, if $t \in \Gamma^* a_1 \Gamma^* \cdots a_n \Gamma^*$. For a binary relation $\to$ on some set, let $\overset{*}{\to}$ be the reflexive and transitive closure of $\to$.

A *finite automaton* (with ε-transitions) is a tuple $A = (Q, \Sigma, \delta, q_0, Q_f)$, where Q is the finite set of states, Σ is the alphabet, $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ is the transition relation, $q_0 \in Q$ is the initial state, and $Q_f \subseteq Q$ is the set of final states. The automaton A is *deterministic* if for every state q and all transitions $(q,a,p), (q,b,r) \in \delta$ $(a, b \in \Sigma \cup \{\varepsilon\})$ the following holds:

- If $a = \varepsilon$ then $p = r$ and $b = \varepsilon$.
- If $a = b \neq \varepsilon$ then $p = r$.

In order to define *alternating automata*, let us first define the *alternating graph accessibility problem*, briefly *AGAP*. An instance of AGAP is a tuple (V, E, v_0, F, β), where (V, E) is a directed graph, $v_0 \in V$ is the initial node, $F \subseteq V$ is the set of final nodes, and $\beta : V \to \{\exists, \forall\}$ is a mapping. A node $v \in V$ is called accepting, if

- $v \in F$, or
- $\beta(v) = \exists$ and there exists a successor node of v, which is accepting, or
- $\beta(v) = \forall$ and every successor node of v is accepting.

The instance (V, E, v_0, F, β) is *positive*, if v_0 is accepting.

An *alternating finite automaton* is a tuple $A = (Q, \Sigma, \delta, q_0, Q_f, \alpha)$ such that $(Q, \Sigma, \delta, q_0, Q_f)$ is a finite automaton and $\alpha : Q \to \{\forall, \exists\}$. For $w \in \Sigma^*$, we get an AGAP-instance

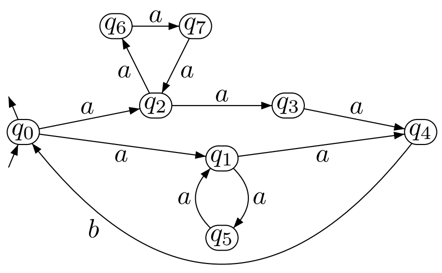
$$I(A, w) = (V, E, v_0, F, \beta),$$

where

- $V = Q \times \{w[1, i] \mid 0 \leq i \leq |w|\}$,
- $E = \{((q, au), (p, u)) \in V \times V \mid (q, a, p) \in \delta\}$,
- $v_0 = (q_0, w)$,
- $F = Q_f \times \{\varepsilon\}$, and
- $\beta(q, u) = \alpha(q)$.

Finally, let $L(A) = \{w \in \Sigma^* \mid I(A, w) \text{ is a positive AGAP-instance}\}$. The problem whether $w \in L(A)$ for a given word w and a given alternating finite automaton A is a classical P-complete problem; it is equivalent to AGAP [7].

Example 1. *Let A be the following alternating finite automaton, where $\alpha(q_0) = \forall$ and $\alpha(q_i) = \exists$ for $1 \leq i \leq 7$ (q_0 is the initial and the unique final state):*



Then we have $L(A) = ((a^6)^+ b)^*$.

We assume some background in complexity theory [19]. In particular, the reader should be familiar with the complexity classes P, $(\mathsf{co})\mathsf{NP}$, PSPACE, and $\mathsf{EXPTIME}$. An *alternating Turing-machine* [3, 19]

$$M = (Q, \Sigma, \delta, q_0, q_f, \alpha)$$

is a nondeterministic Turing-machine (Q is the set of states, Σ is the tape alphabet, $\delta \subseteq Q \times \Sigma \times Q \times \Sigma \times \{\text{left}, \text{right}\}$ is the transition relation, $q_0 \in Q$ is the initial state, and $q_f \in Q$ is the final state) with an additional mapping $\alpha : Q \to \{\forall, \exists\}$. The set of configurations of M is $\Sigma^* Q \Sigma^*$. For a given input $w \in \Sigma^*$ for M, we obtain again an (infinite) AGAP-instance

$$I(M, w) = (\Sigma^* Q \Sigma^*, E, q_0 w, \Sigma^* q_f \Sigma^*, \beta),$$

where $(c_1, c_2) \in E$ if and only if M can move in one step from configuration c_1 to configuration c_2 and $\beta(uqv) = \alpha(q)$ for $u, v \in \Sigma^*$ and $q \in Q$. The input w is accepted by M if $I(M, w)$ is a positive instance. Clearly, if M operates in space $s(n)$ on an input w of size n, then we can restrict the instance $I(M, w)$ to those configurations uqv with $|uv| \leq s(|w|)$ and therefore obtain a finite AGAP-instance.

It is well known that PSPACE (resp. $\mathsf{EXPTIME}$) equals the class of all problems that can be solved on an alternating Turing-machine in polynomial time (resp. polynomial space). $\mathsf{ATIME}(\exp(n), O(n))$ denotes the class of all problems that can be solved on an alternating Turing-machine in exponential time but where the number of alternations (i.e., transitions from an existential state to a universal state or vice versa) is bounded linearly in the input size. Finally, the class $\mathsf{P}_{\parallel}^{\mathsf{NP}}$ consists of all problems that can be accepted by a deterministic polynomial time machine with access to an oracle from NP where all questions to the oracle are asked in parallel [19]. $\mathsf{P}_{\parallel}^{\mathsf{NP}}$ is located between the first and the second level of the polynomial time hierarchy; more precisely $\mathsf{NP} \cup \mathsf{coNP} \subseteq \mathsf{P}_{\parallel}^{\mathsf{NP}} \subseteq \Sigma_2^p \cap \Pi_2^p$.

2.2. *Grammar based compression*

Following [23], a *straight-line program (SLP)* over the terminal alphabet Γ is a context-free grammar $G = (V, \Gamma, S, P)$ (V is the set of nonterminals, Γ is the set of terminals, $S \in V$ is the initial nonterminal, and $P \subseteq V \times (V \cup \Gamma)^*$ is the set of productions) such that: (i) for every $A \in V$ there exists exactly one production of the form $(A, \alpha) \in P$ for $\alpha \in (V \cup \Gamma)^*$, and (ii) the relation $\{(A, B) \in V \times V \mid (A, \alpha) \in P, B \text{ occurs in } \alpha\}$ is acyclic. Clearly, the language generated by the SLP G consists of exactly one word that is denoted by $\mathrm{val}(G)$. The size of G is $|G| = \sum_{(A,\alpha) \in P} |\alpha|$. Every SLP can be transformed in polynomial time into an equivalent SLP in *Chomsky normal form*, i.e., all productions have the form (A, a) with $a \in \Gamma$ or (A, BC) with $B, C \in V$.

Example 2. *Consider the SLP G over the terminal alphabet $\{a, b\}$ that consists of the following productions: $A_1 \to b$, $A_2 \to a$, and $A_i \to A_{i-1} A_{i-2}$ for $3 \leq i \leq 7$. The start nonterminal is A_7. Then $\mathrm{val}(G) = abaababaabaab$, which is the 7^{th} Fibonacci word. The SLP G is in Chomsky normal form and $|G| = 12$.*

We will also allow exponential expressions of the form A^i for $A \in V$ and $i \in \mathbb{N}$ in right-hand sides of productions. Here the number i is coded binary. Such an expression can be replaced by a sequence of ordinary productions, where the length of that sequence is bounded linearly in the length of the binary coding of i.

Without explicit reference, we use the fact that the following tasks can be solved in polynomial time; except for the last point, proofs are straightforward:

- Given an SLP G, calculate $|\mathrm{val}(G)|$.
- Given an SLP G and a number $i \in \{1, \dots, |\mathrm{val}(G)|\}$, calculate $\mathrm{val}(G)[i]$.
- Given an SLP G over the terminal alphabet Γ and a homomorphism $\rho : \Gamma^* \to \Sigma^*$, calculate an SLP H such that $\mathrm{val}(H) = \rho(\mathrm{val}(G))$.
- Given SLPs G_1 and G_2, decide whether $\mathrm{val}(G_1) = \mathrm{val}(G_2)$ [8, 22].

2.3. *Regular expressions*

Regular expressions over the finite alphabet Γ are inductively defined as follows: (i) ε and every $a \in \Gamma$ are regular expressions, and (ii) if ρ and π are regular expressions, then also $\rho \cup \pi$, $\rho \cdot \pi$, and ρ^* are regular expressions. The language $L(\rho) \subseteq \Gamma^*$ is defined as usual. In this paper, we consider several extensions of regular expressions. For a set $\mathcal{C}$ of language operations, $\mathcal{C}$-expressions over the alphabet Γ are built up from constants in $\Gamma \cup \{\varepsilon\}$ using the operations from $\mathcal{C}$. Thus, ordinary regular expressions are $\{\cdot, \cup, *\}$-expressions. The length $|\rho|$ of an expression is defined as follows: For $\rho \in \Gamma \cup \{\varepsilon\}$ set $|\rho| = 1$. If $\rho = \mathrm{op}(\rho_1, \dots, \rho_n)$, where op is an n-ary operator, we set $|\rho| = 1 + |\rho_1| + \cdots + |\rho_n|$. As already defined in the introduction, for a set $\mathcal{C}$ of language operations, the *compressed membership problem* (resp. *membership problem*) is the computational problem of deciding whether $\mathrm{val}(G) \in L(\rho)$ (resp. $w \in L(\rho)$) for a given SLP G (resp. string w) and $\mathcal{C}$-expression ρ. This problem is denoted by $\mathrm{CMP}(\mathcal{C})$ (resp. $\mathrm{MP}(\mathcal{C})$).

Beside the operators $\cdot, \cup, *$, we also consider intersection ($\cap$), complement ($\neg$), squaring (^{2}, where $L^2 = \{uv \mid u, v \in L\}$), and shuffle ($\|$). The latter operator is defined as follows: For words $u, v \in \Gamma^*$ let

$$u \parallel v = \{u_0 v_0 u_1 v_1 \cdots u_n v_n \mid n \geq 0, u = u_0 \cdots u_n, v = v_0 \cdots v_n\}.$$

For $L, K \subseteq \Gamma^*$ let $L \parallel K = \{u \parallel v \mid u \in L, v \in K\}$. It is well known that the class of regular languages is closed under $\cap, \neg, ^2$, and $\|$, but each of these operators leads to more succinct representations of regular languages. Several classes of expressions have their own names in the literature:

- $\{\cdot, \cup, \neg\}$-expressions are called *star-free expressions*.
- $\{\cdot, \cup, *, \cap\}$-expressions are called *semi-extended regular expressions*.
- $\{\cdot, \cup, *, \neg\}$-expressions are called *extended regular expressions*.
- $\{\cdot, \cup, *, \|\}$-expressions are called *shuffle-expressions*.

Due to de Morgan's law, we do not have to distinguish between the problems $\mathrm{CMP}(\mathcal{C})$ and $\mathrm{CMP}(\mathcal{C} \cup \{\cap\})$ in case $\cup, \neg \in \mathcal{C}$.

In the rest of the paper, we use various abbreviations. Instead of $\rho_1 \cdot \rho_2$ we usually just write $\rho_1 \rho_2$. Moreover, for a finite language L we will denote with L also the obvious $\{\cdot, \cup\}$-expression that describes the language L.

2.4. *Hierarchical automata*

A *hierarchical finite automaton* (briefly HFA, see e.g [1], where hierarchical finite automata are called hierarchical state machines) is a tuple $\mathbb{A} = (\Sigma, A_1, \ldots, A_m)$, where Σ is a finite alphabet. Every A_i is a tuple

$$A_i = (Q_i, \Sigma, \delta_i, q_{0,i}, Q_{f,i}, \mu_i),$$

where $(Q_i, \Sigma, \delta_i, q_{0,i}, Q_{f,i})$ is a finite automaton and μ_i is a *partial* mapping $\mu_i : Q_i \to \{i+1, \ldots, m\}$. Note that we must have $\mathrm{dom}(\mu_m) = \emptyset$. We assume that the sets Q_i $(1 \le i \le m)$ are pairwise disjoint. In order to define the unfolded automaton unfold($\mathbb{A}$) let us set $Q = \bigcup_{i=1}^m Q_i$, $\delta = \bigcup_{i=1}^m \delta_i$, and $\mu = \bigcup_{i=1}^m \mu_i$. Let $\mathrm{unfold}(\mathbb{A}) = (P, \Sigma, \gamma, p_0, P_f)$, where the set of states P is the following subset of Q^*:

$$P = \{q_1 q_2 \cdots q_n \mid n \le m, q_1 \in Q_1, q_{i+1} \in Q_{\mu(q_i)} \text{ for } 1 \le i < n, \mu(q_n) \text{ is undefined}\}$$

In order to define the remaining components γ, p_0, and P_f, let us set for a prefix u of some word in P:

$$\mathrm{in}(u) = \{uv \in P \mid v \in \{q_{0,1}, \ldots, q_{0,m}\}^*\}, \quad \mathrm{out}(u) = \{uv \in P \mid v \in \left(\bigcup_{i=1}^m Q_{f,i}\right)^*\}$$

Since we require $uv \in P$, the set in(u) contains only a single element, and we identify this element with in(u). Note also that $u, uv \in P$ implies $v = \varepsilon$. We now set

$$\gamma = \{(uqv, a, upw) \mid (q, a, p) \in \delta, uqv \in \mathrm{out}(uq), upw = \mathrm{in}(up)\}$$

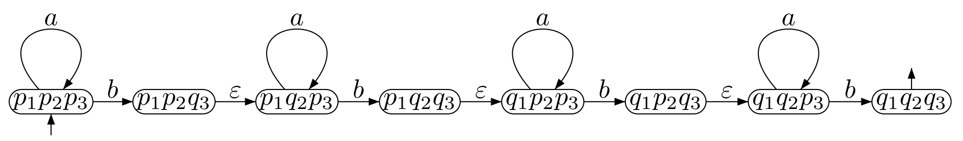
and $p_0 = \mathrm{in}(\varepsilon)$ and $P_f = \mathrm{out}(\varepsilon)$. The HFA $\mathbb{A}$ is deterministic if unfold($\mathbb{A}$) is deterministic. Intuitively, we obtain the automaton unfold($\mathbb{A}$) by replacing as long as possible occurrences of states $q \in Q_i$ with $\mu_i(q) = j > i$ by a copy of the automaton A_j.

An *alternating HFA* is a tuple $\mathbb{A} = (\Sigma, A_1, \ldots, A_m)$ with the same properties as an HFA, except that every A_i is a tuple $(Q_i, \Sigma, \delta_i, q_{0,i}, Q_{f,i}, \alpha_i, \mu_i)$ with $\alpha_i : Q_i \setminus \mathrm{dom}(\mu_i) \to \{\forall, \exists\}$. We set $\mathrm{unfold}(\mathbb{A}) = (P, \Sigma, \gamma, p_0, P_f, \alpha)$, where P, γ, p_0, P_f are defined as above and $\alpha(uq) = \alpha_i(q)$ if $uq \in P$ with $q \in Q_i$.

Example 3. *Let us consider the HFA $\mathbb{A}_n = (\{a, b\}, A_1, \ldots, A_n)$ with*

$$A_n = (\{p_n, q_n\}, \{a, b\}, \{(p_n, a, p_n), (p_n, b, q_n)\}, p_n, \{q_n\}, \emptyset) \text{ and}$$
$$A_i = (\{p_i, q_i\}, \{a, b\}, \{(p_i, \varepsilon, q_i)\}, p_i, \{q_i\}, \{p_i \mapsto i+1, q_i \mapsto i+1\})$$

for $1 \le i < n$. Then unfold($\mathbb{A}_n$) accepts the language $(a^*b)^{2^{n-1}}$. The automaton unfold($\mathbb{A}_3$) is shown below; note that it is deterministic:



3. Compressed membership problems for regular expressions

3.1. *Upper bounds*

There are a few easy upper bounds for $\mathrm{CMP}(\mathcal{C})$. We mentioned already in the introduction that $\mathrm{CMP}(\{\cdot,\cup,*\})$ is P-complete [17]. Moreover, if $\mathcal{C} \subseteq \{\cdot,\cup,\cap,\|\}$, then for every $\mathcal{C}$-expression ρ and every word $w \in L(\rho)$ one has $|w| \leq |\rho|$. It follows that for $\mathcal{C} \subseteq \{\cdot,\cup,\cap,\|\}$, the problems $\mathrm{CMP}(\mathcal{C})$ and $\mathrm{MP}(\mathcal{C})$ are equivalent under polynomial time reductions. Hence, $\mathrm{CMP}(\mathcal{C})$ belongs to P for $\mathcal{C} \subseteq \{\cdot,\cup,\cap\}$ and is NP-complete for $\{\cdot,\|\} \subseteq \mathcal{C} \subseteq \{\cdot,\cup,\cap,\|\}$ due to the results from [18, 25]. The remaining upper bounds are collected in the following theorem:

Theorem 4. *The following holds:*

(a) $\mathrm{CMP}(\{\cdot,\cup,*,\neg,{}^2\}) \in \mathsf{PSPACE}$
(b) $\mathrm{CMP}(\{\cdot,\cup,*,\cap,{}^2,\|\}) \in \mathsf{PSPACE}$
(c) $\mathrm{CMP}(\{\cdot,\cup,*,\neg,{}^2,\|\}) \in \mathsf{ATIME}(\exp(n), O(n))$

Proof. For point (a) let G be an SLP and let ρ be a $\{\cdot,\cup,*,\neg,{}^2\}$-expression. We will check in alternating polynomial time whether $\mathrm{val}(G) \in L(\rho)$. In the following, we will also write down expressions of the form μ^k, where μ^* is a subexpression of ρ and $1 \leq k \leq |\mathrm{val}(G)|$ is a positive integer. Of course, this expression describes the expression

$$\underbrace{\mu \cdot \mu \cdots \mu}_{k \text{ many}}.$$

We define the size of such an expression μ^k as $|\mu| +$ length of the binary coding of k. Our alternating algorithm will store a tuple (π, i, j, b), where:

- π is either a subexpression of ρ or of the form μ^k as described above,
- i and j are positions in $\mathrm{val}(G)$, and
- $b \in \{0, 1\}$.

If $b = 1$ ($b = 0$), then the tuple (π, i, j, b) will represent an accepting configuration if and only if $\mathrm{val}(G)[i, j] \in L(\pi)$ ($\mathrm{val}(G)[i, j] \notin L(\pi)$). If (π, i, j, b) is the current tuple, then the algorithm branches as follows:

- If $\pi = a \in \Sigma$ and $b = 1$ (resp. $b = 0$) then the algorithm accepts (resp. rejects) if and only if $i = j$ and $\mathrm{val}(G)[i] = a$. This can be checked deterministically in polynomial time.
- If $\pi = \neg\mu$ then the algorithm continues with the tuple $(\mu, i, j, 1 - b)$.
- If $\pi = \pi_1 \cup \pi_2$ and $b = 1$ (resp. $b = 0$) then the algorithm guesses existentially (resp. universally) $k \in \{1, 2\}$ and continues with the tuple (π_k, i, j, b).
- If $\pi = \pi_1 \cdot \pi_2$ and $b = 1$ (resp. $b = 0$) then the algorithm guesses existentially (resp. universally) a position $p \in \{i, \ldots, j + 1\}$, then it guesses universally (resp. existentially) $\ell \in \{1, 2\}$ and continues with the tuple $(\pi_1, i, p - 1, b)$ (if $\ell = 1$) or (π_2, p, j, b) (if $\ell = 2$).

- If $\pi = \mu^*$ and $b = 1$ (resp. $b = 0$), then the algorithm accepts (resp. rejects) if $i > j$. Otherwise, it first guesses existentially (resp. universally) a number $k \in \{1, \dots, j - i + 1\}$ and continues with the tuple (μ^k, i, j, b).
- If $\pi = \mu^k$, then if $k = 1$ the algorithm continues with the tuple (μ, i, j, b). If $k \geq 2$ and $b = 1$ (resp. $b = 0$) then it existentially (resp. universally) guesses a position $p \in \{i, \dots, j + 1\}$, then guesses universally (resp. existentially) $\ell \in \{1, 2\}$, and continues with the tuple $(\mu^{\lfloor k/2 \rfloor}, i, p - 1, b)$ (if $\ell = 1$) or $(\mu^{\lceil k/2 \rceil}, p, j, b)$ (if $\ell = 2$).

The running time of this algorithm is bounded by $\mathcal{O}(|\rho| \cdot \log(|\mathrm{val}(G)|))$ which is at most $\mathcal{O}(|\rho| \cdot |G|)$.

For point (b) recall the following simple facts, where $\mathcal{A}_1$ and $\mathcal{A}_2$ are nondeterministic automata with n_1 and n_2 many states, respectively.

- For $L(\mathcal{A}_1) \parallel L(\mathcal{A}_2)$ and $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ there exists an automaton with $n_1 \cdot n_2$ many states (product construction).
- For $L(\mathcal{A}_1) \cdot L(\mathcal{A}_2)$ and $L(\mathcal{A}_1) \cup L(\mathcal{A}_2)$ there exists an automaton with $n_1 + n_2$ many states.
- For $L(\mathcal{A}_1)^2$ (resp. $L(\mathcal{A}_1)^*$) there exists an automaton with $2 \cdot n_1$ (resp. n_1) many states.

As a consequence, a given $\{\cdot, \cup, *, \cap, \parallel, {}^2\}$-expression ρ can be transformed into an automaton $\mathcal{A}(\rho)$ with $2^{O(|\rho|)}$ many states and such that $L(\mathcal{A}(\rho)) = L(\rho)$. In polynomial space we cannot construct this automaton. But this is not necessary. It is only important that a single state of $\mathcal{A}(\rho)$ can be stored in space $|\rho|^{O(1)}$ and that for two given states and a symbol $a \in \Gamma$ it can be checked in polynomial time whether an a-transition exists between the two states. Now we simulate the automaton $\mathcal{A}(\rho)$ in polynomial space on the word $\mathrm{val}(G)$. Thereby, we only store the current position p in the word $\mathrm{val}(G)$, which only needs polynomial space. Recall that the symbol $\mathrm{val}(G)[p]$ can be calculated in polynomial time.

Finally for (c) we can apply a similar strategy as in (a). Let G be an SLP and let ρ be a $\{\cdot, \cup, *, \neg, {}^2, \parallel\}$-expression. We will check with an alternating exponential time algorithm whether $\mathrm{val}(G) \in L(\rho)$. This time the algorithm has to store a tuple (π, w, b), where

- π is a subexpression of ρ and
- w is a subsequence of $\mathrm{val}(G)$.

Note that we store the subsequence w of $\mathrm{val}(G)$ explicitly (whereas in (a) we store a factor of $\mathrm{val}(G)$ by its first and last position), which needs exponential space in the worst case.

If $b = 1$ ($b = 0$), then the triple (π, w, b) will represent an accepting configuration if and only if $w \in L(\pi)$ ($w \notin L(\pi)$). Now, for a current tuple (π, w, b), the algorithm branches in a similar way as in the proof of (a). Only two modifications are necessary:

- If $\pi = \pi_1 \parallel \pi_2$ and $b = 1$ (resp. $b = 0$), then the algorithm guesses existentially (universally) two subsequences w_1 and w_2 of w such that $w \in w_1 \parallel w_2$ (note that this needs exponential time). Then it guesses universally (resp. existentially) $\ell \in \{1, 2\}$ and continues with the tuple (π_1, w_1, b) (if $\ell = 1$) or (π_2, w_2, b) (if $\ell = 2$).
- In order to obtain a linear number of alternations, we have to modify the treatment of an expression $\pi = \mu^*$: If $w = \varepsilon$ and $b = 1$ (resp. $b = 0$) then the algorithm accepts (resp. rejects). If $w \neq \varepsilon$ and $b = 1$ (resp. $b = 0$), then the algorithm first guesses existentially (universally) a factorization $w = w_1 w_2 \cdots w_m$ with $w_i \neq \varepsilon$ for $1 \leq i \leq m$ (again, this needs exponential time). Then it guesses universally (resp. existentially) $\ell \in \{1, \ldots, m\}$ and continues with the tuple (μ, w_i, b).

The number of alternations of this algorithm is bounded by $\mathcal{O}(|\rho|)$ and hence bounded linearly in the input length. $\square$

3.2. *Lower bounds*

In the following, we will prove a series of PSPACE-hardness results.

Theorem 5. CMP$(\{\cdot, \cup, *, \cap\})$ *is* PSPACE-*hard.*

We need a few definitions for the proof: A *semi-Thue system* over an alphabet Γ is a finite subset $S \subseteq \Gamma^* \times \Gamma^*$; it is *length-preserving* if $|\ell| = |r|$ for all $(\ell, r) \in S$. Let $\mathrm{dom}(S) = \{\ell \in \Gamma^* \mid \exists r : (\ell, r) \in S\}$ and $\to_S = \{(u\ell v, urv) \mid (\ell, r) \in S \wedge u, v \in \Gamma^*\}$. A linear order $\succ$ on the alphabet Γ will be extended length-lexicographically to Γ^* as follows: Let $u, v \in \Gamma^*$. Then $u \succ v$ if and only if $|u| > |v|$ or ($|u| = |v|$ and there exit $x, y, z \in \Gamma^*$ and $a, b \in \Gamma$ with $a \succ b$, $u = xay$, and $v = xbz$). We say that the semi-Thue system S over Γ is *length-lexicographic* with respect to $\succ$ if $\ell \succ r$ for all $(\ell, r) \in S$.

Proof of Theorem 5. We will use a construction from [13]. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, q_f)$ be a fixed deterministic linear bounded automaton (Q is the set of states, Σ is the tape alphabet, q_0 (resp. q_f) is the initial (resp. final) state, and $\delta : Q \setminus \{q_f\} \times \Sigma \to Q \times \Sigma \times \{\text{left}, \text{right}\}$ is the transition function) such that the question whether a word $w \in \Sigma^*$ is accepted by $\mathcal{A}$ is PSPACE-complete (such a linear bounded automaton can be easily constructed, see also [2]). The one-step transition relation between configurations of $\mathcal{A}$ is denoted by $\Rightarrow_{\mathcal{A}}$. Let $w \in \Sigma^*$ be an input for $\mathcal{A}$ with $|w| = n$. The number n is the input size in the rest of the construction. The following Steps 1–3 are as in [13].

Step 1. Normalization of the automaton $\mathcal{A}$: We may assume that $\mathcal{A}$ operates in phases, where a single phase consists of a sequence of $2 \cdot n$ transitions of the form $q_1\gamma_1 \overset{*}{\Rightarrow}_{\mathcal{A}} \gamma_2 q_2 \overset{*}{\Rightarrow}_{\mathcal{A}} q_3\gamma_3$, where $\gamma_1, \gamma_2, \gamma_3 \in \Sigma^n$ and $q_1, q_2, q_3 \in Q$. During the transition sequence $q_1\gamma_1 \overset{*}{\Rightarrow}_{\mathcal{A}} \gamma_2 q_2$ only right-moves are made, whereas during the sequence

$\gamma_2 q_2 \overset{*}{\Rightarrow}_{\mathcal{A}} q_3\gamma_3$ only left-moves are made (this normalization can be achieved by simulating one transition of the original automaton by a complete left-right transversal followed by a right-left transversal). The automaton $\mathcal{A}$ accepts if it reaches the final state q_f, otherwise $\mathcal{A}$ does not terminate. There exists a constant $c > 0$ (which only depends on the fixed machine $\mathcal{A}$, but is independent of the input w) such that if w is accepted by $\mathcal{A}$, then $\mathcal{A}$, started on w, reaches the final state q_f after at most $2^{c\cdot n}$ phases.

Step 2. Simulation by a length-preserving and length-lexicographic semi-Thue system S: The technical details of our encoding will be simplified by simulating the automaton $\mathcal{A}$ by a length-preserving semi-Thue system.

Let $\widehat{\Sigma} = \{\widehat{a} \mid a \in \Sigma\}$ be a disjoint copy of Σ and similarly for $\widehat{Q}$. Let

$$\Delta = \Sigma \cup \widehat{\Sigma} \cup \{\lhd, 0, 1, \pounds\} \qquad \text{and} \qquad \Theta = Q \cup \widehat{Q} \cup \Delta. \tag{1}$$

We simulate $\mathcal{A}$ by the following semi-Thue system S over the alphabet Θ:

$$\boxed{\begin{array}{llll} 0\widehat{q} \to \widehat{q}\pounds & \text{for every } q \in Q \setminus \{q_f\} & qa \to \widehat{b}p & \text{if } \delta(q,a) = (p,b,\text{right}) \\ 1\widehat{q} \to 0q & \text{for every } q \in Q \setminus \{q_f\} & \widehat{a}\,\widehat{q} \to \widehat{p}b & \text{if } \delta(q,a) = (p,b,\text{left}) \\ q\pounds \to 1q & \text{for every } q \in Q \setminus \{q_f\} & q\lhd \to \widehat{q}\lhd & \text{for every } q \in Q \setminus \{q_f\} \end{array}}$$

S is length-preserving and

$$\forall v, v_1, v_2 \in \Delta^*(Q \cup \widehat{Q})\Delta^* : (v_1 \;{}_S\!\leftarrow v \to_S v_2) \Longrightarrow v_1 = v_2. \tag{2}$$

If $\succ$ is any linear order on the alphabet Θ that satisfies

$$Q \succ 1 \succ 0 \succ \widehat{\Sigma} \succ \widehat{Q}, \tag{3}$$

then S is length-lexicographic with respect to $\succ$. Let us choose such a linear order on Θ that moreover satisfies

$$Q \succ \Delta \succ \widehat{Q} \tag{4}$$

(this condition will be only important in step 4). In [13] we have argued that

$$\mathcal{A} \text{ accepts } w \iff \exists v \in \Delta^+\{q_f, \widehat{q}_f\}\Delta^+ : 1q_0\pounds^{c\cdot n-1}w\lhd \overset{*}{\to}_S v. \tag{5}$$

We briefly repeat the arguments: First, note that $1q_0\pounds^{c\cdot n-1}w\lhd \overset{*}{\to}_S 1^{c\cdot n}q_0w\lhd$. From the word $1^{c\cdot n}q_0w\lhd$ we can simulate $2^{c\cdot n}$ phases of $\mathcal{A}$. The crucial point is that the prefix from $\{0,1\}^*$ acts as a binary counter: for every $u \in \{0,1\}^i$ ($i < c\cdot n$) and every $q \in Q$ we have: $u10^{c\cdot n-|u|-1}\widehat{q} \overset{*}{\to}_S u1\widehat{q}\pounds^{c\cdot n-|u|-1} \to_S u0q\pounds^{c\cdot n-|u|-1} \overset{*}{\to}_S u01^{c\cdot n-|u|-1}q$. Thus, if $\mathcal{A}$ accepts w, then we can derive from $1q_0\pounds^{c\cdot n-1}w\lhd$ a word $v \in \Delta^+\{q_f, \widehat{q}_f\}\Delta^+$. On the other hand, if $\mathcal{A}$ does not accept w and hence does not terminate, then we can derive from $1q_0\pounds^{c\cdot n-1}w\lhd$ a word of the form $\widehat{q}\pounds^{c\cdot n}u\lhd$ for some $u \in \Sigma^n$ and $q \neq q_f$. Since S is confluent (the left-hand sides of S do not overlap) and $\widehat{q}\pounds^{c\cdot n}u\lhd$ is irreducible with respect to S, we cannot reach from $1q_0\pounds^{c\cdot n-1}w\lhd$ a word from $\Delta^+\{q_f, \widehat{q}_f\}\Delta^+$.

To simplify the following construction, we expand all rules from S in the following sense: The rule $q\pounds \to 1q$ for instance is replaced by all rules of the form $xq\pounds \to x1q$ for all $x \in \Delta$, whereas the rule $0\widehat{q} \to \widehat{q}\pounds$ is replaced by all rules of the form $0\widehat{q}x \to \widehat{q}\pounds x$ for all $x \in \Delta$. Let us call the resulting system again S. Then S is still length-preserving and length-lexicographic with respect to $\succ$ and satisfies (2) and (5). Moreover,

$$\mathrm{dom}(S) \subseteq \Delta(Q \cup \widehat{Q})\Delta.$$

Let

$$m = (c+1)n + 2 \in O(n).$$

Thus, m is the length of words in any derivation starting from $1q_0\pounds^{c\cdot n-1}w\triangleleft$.

Step 3. Reduction to the reachability problem for a directed forest (V, E): Let us now define the directed graph (V, E), where

$$V = \bigcup_{i=1}^{m-2} \Delta^i(Q \cup \widehat{Q})\Delta^{m-i-1} \text{ and } E = \{(v, v') \in V \times V \mid v \to_S v'\}. \tag{8}$$

This graph is basically the transition graph of the automaton $\mathcal{A}$ on configurations of length n. Since S is length-lexicographic, (V, E) is acyclic. Moreover, by (2), every node from V has at most one outgoing edge. Thus, (V, E) is a directed forest. If we order V lexicographically by $\succ$ and write

$$V = \{v_1, \ldots, v_N\} \text{ with } v_1 \succ v_2 \succ \cdots \succ v_N, \tag{9}$$

then $(v_i, v_j) \in E$ implies $i < j$. Note that

$$N = |V| = 2(m-2) \cdot |Q| \cdot |\Delta|^{m-1} \in 2^{O(n)}. \tag{10}$$

Let

$$F = V \cap \Delta^+\{q_f, \widehat{q}_f\}\Delta^+ \qquad \text{and} \qquad v_\xi = 1q_0\pounds^{c\cdot n-1}w\triangleleft. \tag{11}$$

(F for "final"). Elements of F have outdegree 0 in the directed forest (V, E). Moreover, $\xi - 1$ is the number of words from V that are lexicographically larger than $1q_0\pounds^{c\cdot n-1}w\triangleleft$. The number ξ can be easily calculated in logarithmic space from the input word w using simple arithmetic. The automaton $\mathcal{A}$ accepts w if and only if there is a path in (V, E) from v_ξ to a node in F.

Step 4. Reduction to a language membership problem: Note that for all $1 \le i, j \le N$, if $v_i = u_1\ell u_2 \to_S u_1 r u_2 = v_j$ with $(\ell, r) \in S$, then $j - i$ (i.e., the number of words from V that are lexicographically between v_i and v_j) is a number that only depends on the rule (ℓ, r) (and thus $\ell \in \mathrm{dom}(S)$) and $|u_2| \in \{0, \ldots, m-3\}$. We call this number $\lambda(\ell, |u_2|)$; it belongs to the set $2^{O(n)}$ and can be calculated in logarithmic space from ℓ and $|u_2|$ using simple arithmetic. Since $m \in O(n)$ and S is a fixed system, we can construct an injective mapping

$$\mathsf{code} : (\mathrm{dom}(S) \times \{0, \ldots, m-3\}) \cup \{f\} \to \{a\}^+$$

(f for "final"), where all strings in the range of code have length bounded by $O(n)$. Note that the mapping code is not an extension of λ. Define the strings $\delta_1, \ldots, \delta_N \in a^*$ and $W \in \{a, \$\}^*$ as follows:

$$\begin{aligned}
\delta_i &= \begin{cases} \mathsf{code}(\ell, k) & \text{if } v_i = u_1 \ell u_2, \text{ where } \ell \in \mathrm{dom}(S) \text{ and } |u_2| = k \\ \mathsf{code}(f) & \text{if } v_i \in F \text{ (and thus has no outgoing edge)} \\ \varepsilon & \text{if } v_i \in V \setminus F \text{ and } v_i \text{ has no outgoing edge} \end{cases} \\
W &= \delta_1 \$ \delta_2 \$ \cdots \delta_N \$. \qquad (12)
\end{aligned}$$

We make two claims about the data constructed so far:

(a) There is a path in (V, E) from v_ξ to a node in F if and only if W belongs to the language

$$(a^*\$)^{\xi-1} \left(\bigcup_{\substack{0 \le k \le m-3, \\ \ell \in \mathrm{dom}(S)}} \mathsf{code}(\ell, k) \$ (a^*\$)^{\lambda(\ell,k)-1} \right)^* \mathsf{code}(f) \$ \{a, \$\}^*. \qquad (13)$$

(b) The word W can be generated by an SLP of size $O(n)$.

Point (a) is easy to see: For the only if-direction one shows by induction over the length of paths in (V, E) that if a node $v_j \in V$ is reachable in (V, E) from the initial node v_ξ then $\delta_1 \$ \cdots \delta_{j-1} \$$ belongs to

$$(a^*\$)^{\xi-1} \left(\bigcup_{\substack{0 \le k \le m-3, \\ \ell \in \mathrm{dom}(S)}} \mathsf{code}(\ell, k) \$ (a^*\$)^{\lambda(\ell,k)-1} \right)^*. \qquad (14)$$

The induction base is clear since $\delta_1 \$ \cdots \delta_{\xi-1} \$$ belongs to (14). For the induction step assume that $\delta_1 \$ \cdots \delta_{i-1} \$$ belongs to (14) and that $(v_i, v_j) \in E$. Then there exist $0 \le k \le m-3$ and $\ell \in \mathrm{dom}(S)$ such that $\delta_i = \mathsf{code}(\ell, k)$ and $j - i = \lambda(\ell, k)$. Hence $\delta_i \$ \cdots \delta_{j-1} \$ \in \mathsf{code}(\ell, k) \$ (a^*\$)^{\lambda(\ell,k)-1}$. This proves the only if-direction in (a). For the if-direction one can argue similarly.

For point (b) let us assume that $Q = \{p_1, \ldots, p_{n_1}\}$ and $\Delta = \{a_1, \ldots, a_{n_2}\}$ with $p_i \succ p_{i+1}$, $\widehat{p}_i \succ \widehat{p}_{i+1}$, and $a_i \succ a_{i+1}$.[a] We now define an SLP G by the following productions:[b]

$$\begin{aligned}
A_i &\to \prod_{j=1}^{n_2} B_{i,j} A_{i+1} \widehat{B}_{i,j} \text{ for } 0 \le i < m-3 \\
A_{m-3} &\to \prod_{j=1}^{n_2} B_{m-3,j} \widehat{B}_{m-3,j}
\end{aligned}$$

[a]Note that the order $\succ$ on the subalphabets Q, $\widehat{Q}$, and Δ, respectively, is arbitrary except that $1 \succ 0$. Note also that Q and Δ are fixed alphabets, hence, n_1 and n_2 are fixed constants and do not depend on n.

[b]The expression $\prod_{i=1}^{k} w_i$ is an abbreviation for $w_1 w_2 \cdots w_k$.

$$B_{i,j} \to \prod_{k=1}^{n_1}\prod_{\ell=1}^{n_2}(C_{i,j,k,\ell}\$)^{|\Delta|^{m-i-3}} \text{ for } 0 \le i \le m-3,\ 1 \le j \le n_2$$

$$C_{i,j,k,\ell} \to \begin{cases} \mathsf{code}(a_j p_k a_\ell, m-i-3) & \text{if } a_j p_k a_\ell \in \mathrm{dom}(S) \\ \mathsf{code}(f) & \text{if } p_k = q_f \\ \varepsilon & \text{if } a_j p_k a_\ell \notin \mathrm{dom}(S) \text{ and } p_k \neq q_f \end{cases}$$

$$\widehat{B}_{i,j} \to \prod_{k=1}^{n_1}\prod_{\ell=1}^{n_2}(\widehat{C}_{i,j,k,\ell}\$)^{|\Delta|^{m-i-3}} \text{ for } 0 \le i \le m-3,\ 1 \le j \le n_2$$

$$\widehat{C}_{i,j,k,\ell} \to \begin{cases} \mathsf{code}(a_j \widehat{p}_k a_\ell, m-i-3) & \text{if } a_j \widehat{p}_k a_\ell \in \mathrm{dom}(S) \\ \mathsf{code}(f) & \text{if } \widehat{p}_k = \widehat{q}_f \\ \varepsilon & \text{if } a_j \widehat{p}_k a_\ell \notin \mathrm{dom}(S) \text{ and } \widehat{p}_k \neq \widehat{q}_f \end{cases}$$

The exponents that appear in the right-hand sides of the productions for the nonterminal $B_{i,j}$ and $\widehat{B}_{i,j}$, namely $|\Delta|^{m-i-3}$, are of size $2^{O(n)}$ and can therefore be replaced by sequences of $O(n)$ many ordinary productions. Hence, the total size of the SLP G is $O(m^2) = O(n^2)$. As in [13] it can be shown that $\mathrm{val}(G) = W$. The constraint $Q \succ \Delta \succ \widehat{Q}$ from (4) implies that the δ_i appear in $\mathrm{val}(G)$ in the order corresponding to the lexicographic order of the v_i ($1 \le i \le N$). This proves point (b).

The proof of the theorem would be complete if we could construct in polynomial time a $\{\cdot, \cup, *, \cap\}$-expression for the language (13). Instead of this, we will construct a $\{\cdot, \cup, *, \cap\}$-expression for a slightly modified language that serves our purpose as well. Choose prime numbers $p_1, \ldots, p_r$ with $N < p_1 p_2 \cdots p_r$. Note that we can assume $p_1, \ldots, p_r, r \in O(\log(N)) \subseteq O(n)$ and that these prime numbers can be computed in polynomial time. For a number $M < p_1 p_2 \cdots p_r$ we define the $\{\cdot, \cup, *, \cap\}$-expression

$$\rho[M] = \bigcap_{1 \le i \le r} \left((a^*\$)^{p_i}\right)^* (a^*\$)^{M \bmod p_i}.$$

Hence, $L(\rho[M]) = (a^*\$)^M \left((a^*\$)^{p_1 p_2 \cdots p_r}\right)^*$ by the Chinese remainder theorem. Finally, set

$$\rho = \rho[\xi - 1]\left(\bigcup_{\substack{0 \le k \le m-3,\\ \ell \in \mathrm{dom}(S)}} \mathsf{code}(\ell, k)\$\rho[\lambda(\ell, k) - 1]\right)^* \mathsf{code}(f)\$\{a, \$\}^*.$$

Since $N < p_1 p_2 \cdots p_r$, the language (13) contains W if and only if $W \in L(\rho)$. This completes the proof of Theorem 5. □

The language (13) can be described by a $\{\cdot, \cup, *, {}^2\}$-expression of polynomial size. Moreover, occurrences of $*$ can be replaced by suitable exponents. Hence, we obtain:

Theorem 6. $\mathrm{CMP}(\{\cdot, \cup, {}^2\})$ *is* PSPACE*-hard.*

Proof. Consider the language (13) and the word W from (12). First, we can replace every occurrence of a^* by a $\{\cup, \cdot\}$-expression μ for $\{\varepsilon\} \cup \mathrm{ran}(\mathsf{code})$, where $\mathrm{ran}(\mathsf{code})$ is the (finite) range of the mapping code. The resulting expression

$$(\mu\$)^{\xi-1}\left(\bigcup_{\substack{0\le k\le m-3,\\ \ell\in\mathrm{dom}(S)}} \mathsf{code}(\ell,k)\$(\mu\$)^{\lambda(\ell,k)-1}\right)^{*}\mathsf{code}(f)\$\{a,\$\}^* \tag{16}$$

generates the string W if and only if the language (13) contains W. The exponents $\xi - 1$ and $\lambda(\ell, k) - 1$ in this expression can be easily eliminated using the squaring-operator. For instance, an expression π^{1010}, where the exponent is binary coded, can be replaced by $((\pi^2)^2)^2 \cdot \pi^2$. It remains to eliminate the two remaining Kleene stars in (16) in such a way that the resulting expression generates the string W if and only if the language (13) contains W. Recall that the length of W is bounded by $2^{O(n)}$. This allows us to replace, e.g., $\{a, \$\}^*$ in (16) by $(\cdots((a \cup \$ \cup \varepsilon)^2)^2 \cdots)^2$ where the number of applications of the squaring-operator is $\lceil\log(|W|)\rceil \in O(n)$. This expression generates all words over the alphabet $\{a, \$\}$ of length at most 2^m, where m is the smallest number with $2^m \ge |W|$. The other Kleene star in (16) can be eliminated in the same way. The resulting $\{\cdot, \cup, {}^2\}$-expression generates the string W if and only if the language (13) contains W. □

In order to prove PSPACE-hardness for star-free expressions, we can reuse a construction from [16], where the authors proved PSPACE-completeness of the problem of checking whether an SLP-compressed word satisfies a given LTL-formula.

Theorem 7. *$\mathrm{CMP}(\{\cdot, \cup, \neg\})$ is PSPACE-hard.*

Proof. The following construction is mainly taken from [16]; we present it for completeness. We prove the theorem by a reduction from the PSPACE-complete *quantified boolean satisfiability problem* (QSAT), see, e.g., [19]. Let $\theta = Q_1x_1 \cdots Q_nx_n : \varphi$ be a quantified boolean formula, where $Q_i \in \{\exists, \forall\}$ and $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ is a formula in 3-CNF, i.e., every C_i is a disjunction of three literals. Let the SLP G_n be defined as $G_n = (\{A_1, \dots, A_n\}, \Sigma_n, A_1, P_n)$, where $\Sigma_n = \{b_i, e_i, t_i, f_i \mid 1 \le i \le n\}$ and P_n contains the following productions:

$$\begin{aligned} A_i &\to b_iA_{i+1}t_ie_ib_iA_{i+1}f_ie_i \text{ for } 1 \le i < n \\ A_n &\to b_nt_ne_nb_nf_ne_n \end{aligned}$$

The idea is that every pair b_i, e_i is a pair of matching brackets. The symbol t_i (resp. f_i) indicates that the variable x_i is set to true (resp. false). Then, the word $\mathrm{val}(G_n)$ can be seen as a binary tree of height n, whose leaves correspond to valuations for the variables $\{x_1, \dots, x_n\}$, i.e., to mappings $v : \{x_1, \dots, x_n\} \to \{\mathrm{true}, \mathrm{false}\}$. More generally, for every $0 \le j \le n$ and every position $1 \le p \le |\mathrm{val}(G_n)|$ such that $\mathrm{val}(G_n)[p-1] = b_j$ (here, we assume a hypothetical position 0 labeled with the hypothetical symbol b_0) we define a (partial) valuation $v_p : \{x_1, \ldots, x_j\} \to \{\text{true}, \text{false}\}$ as follows:

$$v_p(x_k) = \begin{cases} \text{true} & \text{if } \exists r > p : \text{val}(G)[p,r] \in (\Sigma \setminus \{e_k\})^* t_k \\ \text{false} & \text{if } \exists r > p : \text{val}(G)[p,r] \in (\Sigma \setminus \{e_k\})^* f_k \end{cases}$$

Example 8. *We have*

$$\text{val}(G_2) = b_1 b_2 t_2 e_2 b_2 f_2 e_2 t_1 e_1 b_1 b_2 t_2 e_2 b_2 f_2 e_2 f_1 e_1.$$

For the position $p = 6$ in the word $\text{val}(G_2)$ we have $v_6(x_1) =$ true and $v_6(x_2) =$ false.

Now let us define a star-free expression $\rho(\theta)$ such that θ is a positive QSAT-instance if and only if $\text{val}(G_n) \in L(\rho(\theta))$.

Let the formula θ_j ($0 \le j \le n$) be $Q_{j+1}x_{j+1} \cdots Q_n x_n : \varphi$. Thus, $\theta_n = \varphi$ and $\theta_0 = \theta$. The set of free variables in θ_j is $\{x_1, \ldots, x_j\}$. Inductively, we will construct for every $0 \le j \le n$ a star-free expression ρ_j such that for every position $1 \le p \le |\text{val}(G_n)|$ with $\text{val}(G_n)[p-1] = b_j$ we have

$$v_p \text{ satisfies } \theta_j \iff \text{val}(G_n)[p, |\text{val}(G_n)|] \in L(\rho_j). \tag{17}$$

Let us start with ρ_n. Assume that $\varphi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, where

$$C_i = (x_{i,1}^{a_{i,1}} \vee x_{i,2}^{a_{i,2}} \vee x_{i,3}^{a_{i,3}}).$$

Here $x_{i,j} \in \{x_1, \ldots, x_n\}$ and $a_{i,j} \in \{1, -1\}$. Moreover, we set $x^1 = x$ and $x^{-1} = \neg x$. Let us first define for a clause $C = (x_i^a \vee x_j^b \vee x_k^c)$ the star-free expression $\rho(C)$ as

$$\rho(C) = \Big((\Sigma \setminus \{e_i\})^* t_i^a \ \cup \ (\Sigma \setminus \{e_j\})^* t_j^b \ \cup \ (\Sigma \setminus \{e_k\})^* t_k^c \Big) \Sigma^*,$$

where t_i^{-1} is defined as f_i. Here, for a subalphabet $\Gamma \subseteq \Sigma_n$, we set

$$\Gamma^* = \bigcap_{a \in \Sigma_n \setminus \Gamma} \neg(\Sigma_n^* a \Sigma_n^*), \tag{18}$$

where $\Sigma_n^* = \neg(a \cap \neg a)$ for some a. Then, the expression

$$\rho_n = \bigcap_{i=1}^{m} \rho(C_i)$$

satisfies (17). Now assume that ρ_{j+1} is already defined such that (17) holds for $j+1$ and let us define ρ_j. We have to distinguish the cases $Q_{j+1} = \exists$ and $Q_{j+1} = \forall$. If $Q_{j+1} = \exists$, then

$$\rho_j = (\Sigma \setminus \{e_1, \ldots, e_j\})^* b_{j+1} \rho_{j+1}.$$

If $Q_{j+1} = \forall$, then

$$\rho_j = \neg\Big((\Sigma \setminus \{e_1, \ldots, e_j\})^* b_{j+1} \neg \rho_{j+1} \Big).$$

In both cases, it is not hard to verify (17).

Now, let us choose $j = 0$ and $p = 1$ in (17). Then v_p is the empty valuation. Hence, $\theta_0 = \theta$ is true if and only if $\mathrm{val}(G_n) \in L(\rho_0)$. Finally, note that the expression ρ_0 can be constructed in logspace from the formula θ. This concludes the proof of Theorem 7. $\square$

By [25] the uncompressed membership problem $\mathrm{MP}(\{\cdot, \cup, *, \|\})$ is NP-complete. The compressed variant of this problem becomes again PSPACE-complete. For the proof we use the fact that non-emptiness of the intersection of a list of $\{\cdot, \cup, *\}$-expressions is PSPACE-complete [11] together with a trick, which was already applied in [18] in order to encode an intersection as a shuffle.

Theorem 9. $\mathrm{CMP}(\{\cdot, \cup, *, \|\})$ *is* PSPACE*-hard.*

Proof. We use a trick, which was already applied in [18] in order to encode an intersection as a shuffle. It is well known that for given $\{\cup, \cdot, *\}$-expressions $\rho_1, \ldots, \rho_r$ over the alphabet $\{0, 1\}$ it is PSPACE-complete to check whether

$$\bigcap_{i=1}^{r} L(\rho_i) \neq \emptyset$$

[11]. Let $N = n_1 \cdots n_r$, where n_i is the number of states in a finite automaton for ρ_i, the binary representation of this number can be easily computed in polynomial time from $\rho_1, \ldots, \rho_r$. Define the homomorphism $\phi : \{0, 1\}^* \to \{0, 1, \#\}^*$ by $\phi(x) = x\#$ for $x \in \{0, 1\}$. We claim that the following four conditions are equivalent:

(a) $\bigcap_{i=1}^{r} L(\rho_i) \neq \emptyset$

(b) $\exists w \in \bigcap_{i=1}^{r} L(\rho_i) : |w| \leq N$

(c) $(0^r\#^r1^r\#^r)^N \in L(\phi(\rho_1)) \| \cdots \| L(\phi(\rho_r)) \| \{0^r\#^r, 1^r\#^r\}^*$

(d) $\{0^r\#^r, 1^r\#^r\}^* \cap \left(L(\phi(\rho_1)) \| \cdots \| L(\phi(\rho_r)) \| \{0^r\#^r, 1^r\#^r\}^* \right) \neq \emptyset$

The implication (a) $\Rightarrow$ (b) follows from the choice of N. Now assume that (b) holds, i.e., there exists $w \in \bigcap_{i=1}^{r} L(\rho_i)$ with $|w| \leq N$. Let $w = a_1 a_2 \cdots a_n$ with $a_i \in \{0, 1\}$ and $n \leq N$. Then we have

$$\begin{aligned}(0^r\#^r1^r\#^r)^N \quad \in \quad & a_1\#a_2\# \cdots a_n\# \| \cdots \| a_1\#a_2\# \cdots a_n\# \| \\ & (1-a_1)^r\#^r \cdots (1-a_n)^r\#^r(0^r\#^r1^r\#^r)^{N-n},\end{aligned}$$

where r copies of $a_1\#a_2\# \cdots a_n\#$ are shuffled on the right-hand side. This proves (c). The implication (c) $\Rightarrow$ (d) is trivial.

Finally, assume that (d) holds. We will deduce (a). So, assume that

$$a_1^r\#^r \cdots a_k^r\#^r \in \phi(w_1) \| \cdots \| \phi(w_r) \| (b_1^r\#^r \cdots b_\ell^r\#^r),$$

where $w_i \in L(\rho_i)$ for $1 \leq i \leq r$ and $a_1, \ldots, a_k, b_1, \ldots, b_\ell \in \{0, 1\}$. We will show that $w_1 = w_2 = \cdots = w_r$.

Assume that, w.l.o.g., $w_1 = \cdots = w_i = \varepsilon$ and $w_{i+1}, \ldots, w_r \neq \varepsilon$, where $0 \leq i \leq r$. Let $w_j = c_j v_j$ with $c_j \in \{0, 1\}$ for $i + 1 \leq j \leq r$. Thus

$$a_1^r \#^r \cdots a_k^r \#^r \in (c_{i+1} \# \phi(v_{i+1})) \parallel \cdots \parallel (c_r \# \phi(v_r)) \parallel (b_1^r \#^r \cdots b_\ell^r \#^r). \tag{19}$$

If $k = 0$ then we must have $i = r$ (and $\ell = 0$), i.e., $w_1 = w_2 = \cdots = w_r = \varepsilon$. Hence, assume that $k > 0$. If $\ell = 0$, then

$$a_1^r \#^r \cdots a_k^r \#^r \in (c_{i+1} \# \phi(v_{i+1})) \parallel \cdots \parallel (c_r \# \phi(v_r)).$$

It follows that there exist factorizations $c_j \# \phi(v_j) = \alpha_j \beta_j$ $(i + 1 \leq j \leq r)$ such that

$$a_1^r \#^r \in \alpha_{i+1} \parallel \cdots \parallel \alpha_r \tag{20}$$

$$a_2^r \#^r \cdots a_k^r \#^r \in \beta_{i+1} \parallel \cdots \parallel \beta_r. \tag{21}$$

The only possibility for (20) is that $i = 0$, $c_1 = \cdots = c_r = a_1$, and $\alpha_j = a_1 \#$ for $1 \leq j \leq r$. Then (21) implies

$$a_2^r \#^r \cdots a_k^r \#^r \in \phi(v_1) \parallel \cdots \parallel \phi(v_r).$$

Induction over k implies that $v_1 = \cdots = v_r$ and thus $w_1 = \cdots = w_r$.

Finally, assume that $k > 0$ and $\ell > 0$ in (19). Again, we obtain factorizations $c_j \# \phi(v_j) = \alpha_j \beta_j$ $(i + 1 \leq j \leq r)$ and $b_1^r \#^r \cdots b_\ell^r \#^r = \alpha\beta$ such that

$$a_1^r \#^r \in \alpha_{i+1} \parallel \cdots \parallel \alpha_r \parallel \alpha \tag{22}$$

$$a_2^r \#^r \cdots a_k^r \#^r \in \beta_{i+1} \parallel \cdots \parallel \beta_r \parallel \beta. \tag{23}$$

Membership (22) implies that the string α must be a prefix of $b_1^r \#^r$. If $\alpha = \varepsilon$, then, as in case $\ell = 0$, we get $i = 0$, $c_1 = \cdots = c_r = a_1$, and $\alpha_j = a_1 \#$ for $1 \leq j \leq r$. Then (23) becomes

$$a_2^r \#^r \cdots a_k^r \#^r \in \phi(v_1) \parallel \cdots \parallel \phi(v_r) \parallel b_1^r \#^r \cdots b_\ell^r \#^r.$$

Induction over k yields $v_1 = \cdots = v_r$.

If $\alpha = b_1^r \#^r$ then (22) implies $a_1 = b_1$ and $\alpha_j = \varepsilon$ for $i + 1 \leq j \leq r$. Hence (23) implies that

$$a_2^r \#^r \cdots a_k^r \#^r \in \phi(w_{i+1}) \parallel \cdots \parallel \phi(w_r) \parallel (b_2^r \#^r \cdots b_\ell^r \#^r).$$

Now, induction over ℓ (or k) implies $w_1 = \cdots = w_r$.

Finally, assume that α is a proper and non-empty prefix of $b_1^r \#^r$. Clearly, (22) implies $a_1 = b_1$. If $\alpha = a_1^r \#^p$ for some $p < r$, then (22) implies $\alpha_{i+1} = \cdots = \alpha_r = \varepsilon$ (in order to have only r many occurrences of a_1 in the shuffle). But then, the number of occurrences of $\#$ in the right-hand side of (22) is only $p < r$. If $\alpha = a_1^p$ for some $0 < p < r$, then in order to get r many occurrences of $\#$ in the shuffle, we must have $i = 0$ and $\alpha_1 = \cdots = \alpha_r = a_1 \#$. But then, we have $r + p > r$ many occurrences of a_1 in the right-hand side of (22). This proves (a).

The equivalence of (a) and (c) means that

$$\bigcap_{i=1}^{r} L(\rho_i) = \emptyset \iff (0^r \#^r 1^r \#^r)^N \in L(\phi(\rho_1)) \parallel \cdots \parallel L(\phi(\rho_r)) \parallel \{0^r \#^r, 1^r \#^r\}^*.$$

The latter is an instance of CMP($\{\cdot, \cup, *, \|\}$), since the string $(0^r \#^r 1^r \#^r)^N$ can be generated by an SLP of size $O(r + \log(N)) = O(r + \log(n_1) + \cdots \log(n_r))$. This number is bounded polynomially in $|\rho_1| + \cdots + |\rho_r|$. $\square$

For expressions involving shuffle and complement, the computational complexity of the compressed membership problem increases considerably: We can encode in CMP($\{\cdot, \cup, \neg, \|\}$) the problem of checking whether a given monadic second-order formula is true in a given unary word a^N, where N is given binary. This problem was shown to be complete for $\mathsf{ATIME}(\exp(n), O(n))$ in [14].

Theorem 10. CMP($\{\cdot, \cup, \neg, \|\}$) *is* $\mathsf{ATIME}(\exp(n), O(n))$*-hard.*

Proof. In a first step, we will prove an $\mathsf{ATIME}(\exp(n), O(n))$ lower bound for the problem CMP($\{\cdot, \cup, *, \neg, \|\}$). In a second step we show how to eliminate the Kleene star $*$.

For $n \geq 1$, let S_n be the relational structure $(\{0, \ldots, n-1\}, s)$ with $s = \{(i, i+1) \mid 0 \leq i < n-1\}$. Monadic second-order formulas (briefly MSO-formulas) over such a structure are built up from atomic formulas of the form $s(x, y)$ (where x and y are first-order variables ranging over $\{0, \ldots, n-1\}$) and $x \in X$ (where x is a first-order variable and X is a second-order variable, ranging over subsets of $\{0, \ldots, n-1\}$) using boolean connectives and quantification over variables. For an MSO-sentence ψ, i.e., an MSO-formula where all variables are quantified, we write $S_n \models \psi$, if ψ evaluates to true in the structure S_n. In [14], it was shown that the following problem is complete for $\mathsf{ATIME}(\exp(n), O(n))$:[c]

INPUT: An MSO-sentence ψ and a binary coded number N.

QUESTION: $S_N \models \psi$?

We encode this problem into CMP($\{\cdot, \cup, *, \neg, \|\}$). Let us take an MSO-sentence

$$Q_1 X_1 \cdots Q_n X_n : \varphi(X_1, \ldots, X_n),$$

where $Q_i \in \{\exists, \forall\}$, φ does not contain quantifiers, and every X_i is either a first-order or a second-order variable (it will simplify notation to denote first-order variables with capital letters as well). Let $F = \{i \mid 1 \leq i \leq n, X_i \text{ is a first-order variable}\}$ be the set of indices of first-order variables. We will define $\{\cdot, \cup, *, \neg, \|\}$-expressions over the alphabet $\Sigma = \{X_1, \ldots, X_n, a\}$ (using a simple encoding, we can accomplish the encoding also with a binary alphabet). For every $0 \leq i \leq n$ let

$$\rho_i = \Big((X_1 \cup \varepsilon) \cdots (X_i \cup \varepsilon) X_{i+1} \cdots X_n a\Big)^* \cap \tag{24}$$

$$\bigcap_{j \in F, 1 \leq j \leq i} (\Sigma \setminus \{X_j\})^* X_j (\Sigma \setminus \{X_j\})^*. \tag{25}$$

[c]Actually, in [14] it was shown that for every fixed number k, there exists a fixed MSO-sentence of quantifier alternation depth k for which the model-checking problem is complete for the k^{th} level of the exponential time hierarchy, i.e., for $\mathsf{ATIME}(\exp(n), k)$. The proof in [14] immediately leads to $\mathsf{ATIME}(\exp(n), O(n))$-hardness for the case of a variable MSO-sentence.

This expression describes all unary strings over the alphabet $\{a\}$ together with a valuation for the variables $X_1, \ldots, X_i$. The idea is that if a certain string-position belongs to the value of variable X_j then the symbol X_j will occur between the $(j-1)^{\text{th}}$ and j^{th} occurrence of a. To make our encoding possible, we have to demand that those variables that are already quantified (and hence are not part of a valuation) label every position; this explains the subexpression $X_{i+1} \cdots X_n$ in line (24). In (25) we express that the value of a first-order variable contains exactly one position.

Example 11. *Assume we have variables $X_1, X_2, X_3, X_4, X_5, X_6$, where X_2 and X_4 are first-order variables, i.e., $F = \{2, 4\}$. Then the string*

$$X_1X_3X_5X_6\, a\, X_5X_6\, a\, X_1X_2X_5X_6\, a\, X_3X_4X_5X_6\, a\, X_1X_5X_6\, a\, X_3X_5X_6\, a$$

belongs to $L(\rho_4)$. It encodes the string a^6 together with the following valuation for the variables X_1, X_2, X_3, and X_4 (recall that the universe is $\{0, 1, 2, 3, 4, 5\}$):

$$X_1 \mapsto \{0, 2, 4\},\ X_2 \mapsto 2,\ X_3 \mapsto \{0, 3, 5\},\ X_4 \mapsto 3.$$

Now let us translate the MSO-sentence $Q_1X_1 \cdots Q_nX_n : \varphi(X_1, \ldots, X_n)$ into a $\{\cdot, \cup, *, \neg, \|\}$-expression. We start with the boolean part φ, for which we define a $\{\cdot, \cup, *, \neg, \|\}$-expression $\rho[\varphi]$ inductively as follows:

$$\rho[X_j \in X_i] = \rho_n \cap \Sigma^* X_i (\Sigma \setminus \{a\})^* X_j \Sigma^* \text{ if } i < j, \text{ analogously for } j > i \quad (26)$$
$$\rho[s(X_i, X_j)] = \rho_n \cap \Sigma^* X_i (\Sigma \setminus \{a\})^* a (\Sigma \setminus \{a\})^* X_j \Sigma^* \quad (27)$$
$$\rho[\varphi_1 \wedge \varphi_2] = \rho[\varphi_1] \cap \rho[\varphi_2] \quad (28)$$
$$\rho[\neg\varphi] = \rho_n \cap \neg\rho[\varphi] \quad (29)$$

Then $\rho[\varphi]$ defines all a-strings together with a valuation for the variables $X_1, \ldots, X_n$ which satisfy the boolean formula $\varphi(X_1, \ldots, X_n)$. Next, for every $0 \leq i \leq n$, we define an expression π_i, which defines all a-strings together with a valuation for the variables $X_1, \ldots, X_i$ which satisfy the formula $Q_{i+1}X_{i+1} \cdots Q_nX_n : \varphi(X_1, \ldots, X_n)$. Clearly, $\pi_n = \rho[\varphi]$. Now assume that π_i is already defined for $i > 0$. If $Q_i = \exists$, then we set

$$\pi_{i-1} = \rho_{i-1} \cap (\pi_i \parallel X_i^*). \quad (30)$$

In case $Q_i = \forall$ we can use double negation and define π_{i-1} as

$$\pi_{i-1} = \rho_{i-1} \cap \neg(\rho_{i-1} \cap ((\rho_i \cap \neg\pi_i) \parallel X_i^*)). \quad (31)$$

It is easy to see that every π_i indeed defines the desired language. Note also that π_0 can be constructed in polynomial time from the formula $Q_1X_1 \cdots Q_nX_n : \varphi(X_1, \ldots, X_n)$. Finally, we have $S_N \models Q_1X_1 \cdots Q_nX_n : \varphi(X_1, \ldots, X_n)$ if and only if $(X_1 \cdots X_n a)^N \in L(\pi_0)$. Note that $(X_1 \cdots X_n a)^N$ can be defined by an SLP of polynomial size if N is binary coded.

Finally, let us show how to eliminate the star operator in (24)–(31). First, consider the subexpression $((X_1 \cup \varepsilon) \cdots (X_i \cup \varepsilon) X_{i+1} \cdots X_m a)^*$ in (24). For $i < m$

let

$$F = \{X_j X_k \mid j > k \text{ or } i+1 \le j < m, k > j+1\} \cup \{xa \mid x \in \Sigma \setminus \{X_m\}\}.$$

Then

$$\Big((X_1 \cup \varepsilon) \cdots (X_i \cup \varepsilon) X_{i+1} \cdots X_m a\Big)^* = \\ \varepsilon \;\cup\; \Big(\{X_1, \ldots, X_{i+1}\}\Sigma^* \;\cap\; \bigcap_{f \in F} \neg\Sigma^* f \Sigma^* \;\cap\; \Sigma^* a\Big).$$

For $i = m$, we have

$$\Big((X_1 \cup \varepsilon) \cdots (X_m \cup \varepsilon) a\Big)^* = \varepsilon \;\cup\; \Big(\bigcap_{j>k} \neg\Sigma^* X_j X_k \Sigma^* \;\cap\; \Sigma^* a\Big).$$

After doing these replacements, the only remaining Kleene stars are of the form Γ^* for some subalphabet $\Gamma \subseteq \Sigma$ and can be eliminated as in the proof of Theorem 7, see (18). This proves Theorem 10. $\square$

Since it is PSPACE-complete to check whether a given monadic second-order formula is true in a given word a^n, where n is given *unary*,[d] the proof of Theorem 10 shows that the uncompressed variant $\mathrm{MP}(\{\cdot, \cup, \neg, \|\})$ is already PSPACE-complete.

3.3. *Shuffle of compressed words*

It is NP-complete to check for given strings $w, w_1, \ldots, w_n$ whether w belongs to the shuffle $w_1 \| \cdots \| w_n$ [25]. If we fix n to some constant, then this question can be easily solved in logarithmic space. We next show that in the compressed setting the situation changes. For the proof of the following problem we need the following computational problem COMPRESSED-EMBEDDING:

INPUT: Two SLPs G and H over the alphabet $\{a, b\}$

QUESTION: $\mathrm{val}(G) \hookrightarrow \mathrm{val}(H)$?

In [12], it was shown that COMPRESSED-EMBEDDING is hard for $\mathsf{P}_{\|}^{\mathsf{NP}}$.

Theorem 12. *The problem of checking* $\mathrm{val}(G) \in \mathrm{val}(H_1) \| \mathrm{val}(H_2) \| \mathrm{val}(H_3)$ *for four given SLPs* G, H_1, H_2, H_3 *over a binary alphabet belongs to* PSPACE *and is hard for* $\mathsf{P}_{\|}^{\mathsf{NP}}$.

Proof. Membership in PSPACE is obvious. Now, for two given SLPs G and H over the alphabet $\{a, b\}$ we have $\mathrm{val}(G) \hookrightarrow \mathrm{val}(H)$ if and only if $\mathrm{val}(H) \in \mathrm{val}(G) \| a^{|\mathrm{val}(H)|_a - |\mathrm{val}(G)|_a} \| b^{|\mathrm{val}(H)|_b - |\mathrm{val}(G)|_b}$. $\square$

Below, we will present a simplified proof for the $\mathsf{P}_{\|}^{\mathsf{NP}}$-hardness of COMPRESSED-EMBEDDING. In [12], this result is shown in three steps:

[d]The upper bound of PSPACE can be established with a simple alternating polynomial time algorithm. The lower bound of PSPACE follows from the PSPACE-hardness of QSAT.

(i) COMPRESSED-EMBEDDING is NP-hard,
(ii) COMPRESSED-EMBEDDING can be reduced to its own complement (and hence is also coNP-hard), and
(iii) boolean AND and OR can be simulated with COMPRESSED-EMBEDDING.

The most difficult part in [12] is (i). This step can be simplified by proving first coNP-hardness and then using (ii):

Theorem 13. *COMPRESSED-EMBEDDING is* coNP*-hard.*

Proof. Recall that in the SUBSETSUM problem the input consists of binary-coded integers $w_1, \ldots, w_n, t$ and it is asked whether there exist $x_1, \ldots, x_n \in \{0,1\}$ with $\sum_{i=1}^n x_i \cdot w_i = t$. This problem is NP-complete, see [5].

For an instance $\overline{w} = (w_1, \ldots, w_n), t$ of SUBSETSUM, we will construct SLPs G and H over the alphabet $\{a,b\}$ such that $\mathrm{val}(G) \hookrightarrow \mathrm{val}(H)$ if and only if $\forall x_1, \ldots, x_n \in \{0,1\} : \sum_{i=1}^n x_i \cdot w_i \neq t$.

We begin with some notation. Let $s = w_1 + \cdots + w_n$. We can assume that $t < s$. Let $x \in \{0, 1, \ldots, 2^n - 1\}$ be an integer. With x_i $(1 \leq i \leq n)$ we denote the i^{th} bit in the binary representation of x, where x_1 is the least significant bit. Thus, $x = \sum_{i=1}^n x_i 2^{i-1}$. We define $x \circ \overline{w} = \sum_{i=1}^n x_i w_i$. Hence, $x \circ \overline{w}$ is the sum of the subset of $\{w_1, \ldots, w_n\}$ encoded by the integer x. Thus, $\overline{w}, t$ is a *negative* instance of SUBSETSUM if and only if $\forall x \in \{0, \ldots, 2^n - 1\} : x \circ \overline{w} \neq t$. In [13] it was shown that the string $\prod_{x=0}^{2^n-1} (0^{x \circ \overline{w}} 1 0^{s - x \circ \overline{w}})$ can be generated by an SLP of polynomial size with respect to the size of the input $\overline{w}, t$. Hence, the same is true for the strings

$$u = (a^t (ab) a^{s-t})^{2^n} \text{ and } v = \prod_{x=0}^{2^n-1} (ab)^{x \circ \overline{w}} a (ab)^{s - x \circ \overline{w}}.$$

It suffices to show that $u \hookrightarrow v$ if and only if $\forall x \in \{0, \ldots, 2^n - 1\} : x \circ \overline{w} \neq t$.

Since $|u|_a = |v|_a$, we have $u \hookrightarrow v$ if and only if $a^t(ab)a^{s-t} \hookrightarrow (ab)^{x \circ \overline{w}} a (ab)^{s - x \circ \overline{w}}$ for every $x \in \{0, \ldots, 2^n - 1\}$. Thus, it suffices to show for every $x \in \{0, \ldots, 2^n - 1\}$: $a^t(ab)a^{s-t} \hookrightarrow (ab)^{x \circ \overline{w}} a (ab)^{s - x \circ \overline{w}}$ if and only if $x \circ \overline{w} \neq t$. Let us fix $x \in \{0, \ldots, 2^n - 1\}$. First, assume that $x \circ \overline{w} \neq t$. Then, either $x \circ \overline{w} < t$ or $x \circ \overline{w} > t$. In both cases, $a^t(ab)a^{s-t}$ is easily seen to be a subsequence of $(ab)^{x \circ \overline{w}} a (ab)^{s - x \circ \overline{w}}$. Now assume that $x \circ \overline{w} = t$ and $a^t(ab)a^{s-t} \hookrightarrow (ab)^{x \circ \overline{w}} a (ab)^{s - x \circ \overline{w}} = (ab)^t a (ab)^{s-t}$. Hence, the prefix $a^t ab$ of $a^t(ab)a^{s-t}$ has to be embedded into the prefix $(ab)^t aab$ of $(ab)^t a (ab)^{s-t}$. But then, $a^{s-t} \hookrightarrow (ab)^{s-t-1}$, which is impossible. $\square$

The words u and v from the previous proof have the same number of a's. Hence, $u \hookrightarrow v$ if and only if $v \in u \parallel b^{s \cdot 2^n - 2^n}$. It follows that it is already coNP-hard to check for only three SLPs G, H_1, and H_2 over the alphabet $\{a,b\}$, whether $\mathrm{val}(G) \in \mathrm{val}(H_1) \parallel \mathrm{val}(H_2)$. The SLPs G and H constructed in the $\mathsf{P}_{\parallel}^{\mathsf{NP}}$-hardness proof of COMPRESSED-EMBEDDING, see [12], do no longer have the property of generating the same number of a's. Therefore we need a forth SLP in the statement of Theorem 12.

4. Compressed membership problems for hierarchical automata

Let us now move from regular expressions to hierarchical finite automata. In [6] it was proved that the compressed membership problem for hierarchical automata is PSPACE-complete. Using our main lower bound construction from the proof of Theorem 5, we can sharpen the lower bound by proving PSPACE-completeness even for deterministic HFAs:

Theorem 14. *It is* PSPACE*-complete to check for a given SLP G and a deterministic HFA $\mathbb{A}$, whether* $\mathrm{val}(G) \in L(\mathit{unfold}(\mathbb{A}))$.

Proof. Let us use the notations from the proof of Theorem 5. For every $M < N$ we can easily construct in polynomial time a deterministic HFA, which generates the language $(a^*\$)^M$, see Example 3. Hence, the same is also true for the language (13). □

Finally, for alternating HFAs an adaptation of our main construction yields EXPTIME-completeness:

Theorem 15. *It is* EXPTIME*-complete to check for a given SLP G and an alternating HFA $\mathbb{A}$ whether* $\mathrm{val}(G) \in L(\mathit{unfold}(\mathbb{A}))$.

Proof. For the EXPTIME upper bound, let us take an SLP G and an alternating HFA $\mathbb{A}$. Note that a state of unfold($\mathbb{A}$) can be stored in polynomial space. Hence, we can check in alternating polynomial space (= EXPTIME) whether unfold($\mathbb{A}$) accepts val(G). An alternating polynomial space machine just has to store a state of unfold($\mathbb{A}$) and a position in val(G).

For the EXPTIME lower bound, let us start with an alternating Turing-machine

$$M = (Q, \Sigma, \delta, q_0, q_f, \alpha)$$

with a linear space bound that accepts an EXPTIME-complete language [3] and let w be an input for M. We follow the steps in the proof of Theorem 5. Let Δ and Θ be defined as in (1). Then, a construction similar to those in step 1 and step 2 yields a length-preserving semi-Thue system S over the alphabet $\Theta = Q \cup \widehat{Q} \cup \Delta$ with $\mathrm{dom}(S) \subseteq \Delta(Q \cup \widehat{Q})\Delta$, which is moreover length-lexicographic with respect to any linear order that satisfies (3) and (4). Since M is not deterministic, property (2) is not longer true.

Define the graph (V, E) as in (8) and assume (9)–(11). This time, (V, E) is in general not a forest but only a dag. From this dag, we define an AGAP-instance (V, E, v_ξ, F, β) by setting $\beta(v_i) = \alpha(q)$ if $v_i \in \Delta^+\{q, \widehat{q}\}\Delta^+$. This instance is positive if and only if w is accepted by M; this replaces the acceptance condition (5).

For all $1 \leq i \leq N$ the following holds: If $\{v_{j_1}, \ldots, v_{j_k}\}$ is the set of E-successors of node $v_i = u_1 \ell u_2$ (with $\ell \in \mathrm{dom}(S)$), then the set of integers $\{j_1 - i, \ldots, j_k - i\}$ only depends on the left-hand side ℓ and the length $|u_2|$. We call this set $\Lambda(\ell, |u_2|)$;

it can be calculated in logarithmic space from ℓ and $|u_2|$. The finite set $\Lambda(\ell, |u_2|)$ replaces the number $\lambda(\ell, |u_2|)$ in the proof of Theorem 5.

Define the word W as in (12). Finally, we describe an alternating HFA $\mathbb{A}$ that accepts the word W if and only if (V, E, v_ξ, F, β) is a positive AGAP-instance. In a first step, $\mathbb{A}$ reads a prefix from $(a^*\$)^{\xi-1}$ (which is possible with a small deterministic HFA). Then $\mathbb{A}$ enters a loop, in which it first reads the next block from $a^*\$$ from the input. If $u = \mathsf{code}(f)$ then $\mathbb{A}$ accepts. Otherwise, $u = \mathsf{code}(\ell, k)$ for some $\ell \in \mathrm{dom}(S)$ and $0 \leq k \leq m-3$. Let $\ell \in \Delta\{q, \widehat{q}\}\Delta$ for a state $q \in Q$. Then $\mathbb{A}$ guesses existentially (if $\alpha(q) = \exists$) resp. universally (if $\alpha(q) = \forall$) a number $\delta \in \Lambda(\ell, k)$ and jumps over the next $\delta - 1$ many \$'s in the input word. $\square$

5. Summary and open problems

By combining our upper and lower bounds for regular expressions from Section 3, we obtain the following results:

- CMP($\mathcal{C}$) belongs to P if $\mathcal{C} \subseteq \{\cdot, \cup, \cap\}$ or $\mathcal{C} = \{\cdot, \cup, *\}$.
- CMP($\mathcal{C}$) is NP-complete if $\{\cdot, \cup, \|\} \subseteq \mathcal{C} \subseteq \{\cdot, \cup, \cap, \|\}$.
- CMP($\mathcal{C}$) is PSPACE-complete if (i) $\{\cdot, \cup\} \cup \mathcal{D} \subseteq \mathcal{C} \subseteq \{\cdot, \cup, *, \cap, \neg, \|, {}^2\}$ for $\mathcal{D}$ one of sets $\{\neg\}$, $\{^2\}$, $\{*, \cap\}$, or $\{*, \|\}$ and (ii) $\{\neg, \|\} \not\subseteq \mathcal{C}$.
- CMP($\mathcal{C}$) is $\mathsf{ATIME}(\exp(n), O(n))$-complete if $\{\neg, \|\} \subseteq \mathcal{C} \subseteq \{\cdot, \cup, *, \cap, \neg, \|, {}^2\}$.

This characterizes all cases with $\{\cdot, \cup\} \subseteq \mathcal{C} \subseteq \{\cdot, \cup, *, \cap, \neg, \|, {}^2\}$. Of course, one might also consider operator sets which do not include $\{\cdot, \cup\}$.

We conjecture that the problem in Theorem 12 is PSPACE-complete. Another interesting problem is the membership problem for finite automata with compressed constants. In this problem it is asked whether an SLP-compressed word is accepted by a finite automaton A, where transitions of A are labeled with SLPs. An NP lower bound is shown in [23], and the problem is easily seen to be in PSPACE.

Acknowledgments The author would like to thank the referees for their many useful comments.

References

[1] R. Alur and M. Yannakakis. Model checking of hierarchical state machines. *ACM Trans. Program. Lang. Syst.*, 23(3):273–303, 2001.

[2] G. Bauer and F. Otto. Finite complete rewriting systems and the complexity of the word problem. *Acta Inf.*, 21:521–540, 1984.

[3] A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *J. Assoc. Comput. Mach.*, 28(1):114–133, 1981.

[4] K. J. Compton and C. W. Henson. A uniform method for proving lower bounds on the computational complexity of logical theories. *Ann. Pure Appl. Logic*, 48:1–79, 1990.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP–completeness.* Freeman, 1979.

[6] B. Genest and A. Muscholl. Pattern matching and membership for hierarchical message sequence charts. *Theory Comput. Syst.*, 42(4):536-567, 2008.
[7] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, 1995.
[8] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoret. Comput. Sci.*, 158(1&2):143–159, 1996.
[9] B. Jenner. Knapsack problems for NL. *Inform. Process. Lett.*, 54(3):169–174, 1995.
[10] T. Jiang and B. Ravikumar. A note on the space complexity of some decision problems for finite automata. *Inform. Process. Lett.*, 40(1):25–31, 1991.
[11] D. Kozen. Lower bounds for natural proof systems. In *Proc. FOCS 77*, pages 254–266. IEEE Computer Society Press, 1977.
[12] Y. Lifshits and M. Lohrey. Querying and embedding compressed texts. In *Proc. MFCS 2006*, LNCS 4162, pages 681–692. Springer, 2006.
[13] M. Lohrey. Word problems and membership problems on compressed words. *SIAM J. Comput.*, 35(5):1210–1240, 2006.
[14] M. Lohrey. Model-checking hierarchical structures. To appear in *J. Comput. System Sci.*, 2008. A preliminary version appeared in *Proc. LICS 2005*, pages 168–177. IEEE Computer Society Press, 2005.
[15] M. Lohrey. Leaf languages and string compression. In *Proc. FSTTCS 2008*, pages 292–303, 2008.
[16] N. Markey and P. Schnoebelen. Model checking a path. In *Proc. CONCUR 2003*, LNCS 2761, pages 248–262. Springer, 2003.
[17] N. Markey and P. Schnoebelen. A PTIME-complete matching problem for SLP-compressed words. *Inform. Process. Lett.*, 90(1):3–6, 2004.
[18] A. J. Mayer and L. J. Stockmeyer. The complexity of word problems—this time with interleaving. *Inform. and Comput.*, 115(2):293–311, 1994.
[19] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.
[20] H. Petersen. Decision problems for generalized regular expressions. Proc. of DCAGRS 2000, pages 22–29, 2000.
[21] H. Petersen. The membership problem for regular expressions with intersection is complete in LOGCFL. In *Proc. STACS 2002*, LNCS 2285, pages 513–522. Springer, 2002.
[22] W. Plandowski. Testing equivalence of morphisms on context-free languages. In *Proc. ESA'94*, LNCS 855, pages 460–470. Springer, 1994.
[23] W. Plandowski and W. Rytter. Complexity of language recognition problems for compressed words. In *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 262–272. Springer, 1999.
[24] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time (preliminary report). In *Proc. STOCS 73*, pages 1–9. ACM Press, 1973.
[25] M. K. Warmuth and D. Haussler. On the complexity of iterated shuffle. *J. Comput. System Sci.*, 28(3):345–358, 1984.
[26] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Trans. Information Theory*, 23(3):337–343, 1977.