The Language Intersection Problem for Non-Recursive Context-Free Grammars ⋆

Mark-Jan Nederhof [1]

Faculty of Arts, University of Groningen, P.O. Box 716, NL-9700 AS Groningen, The Netherlands

Giorgio Satta [*,2]

Dept. of Information Engineering, University of Padua, via Gradenigo, 6/A, I-35131 Padova, Italy

Abstract

We prove that, given as input two context-free grammars, deciding non-emptiness of intersection of the two generated languages is PSPACE-complete if at least one grammar is non-recursive. The problem remains PSPACE-complete when both grammars are non-recursive and deterministic. Also investigated are generalizations of the problem to several context-free grammars, of which a certain number are non-recursive.

Key words: Formal languages, context-free grammars, computational complexity, parsing algorithms

1 Introduction

The computational complexity of deciding non-emptiness of the intersection of a pair or sequence of languages has been considered for a number of formalisms.

⋆ We thank Harry B. Hunt III for helpful correspondence. We are indebted to an anonymous referee for proposals leading to most of the content of Section 3.

* Corresponding author. Tel: +39-049-8277948. Fax: +39-049-8277799.
Email addresses: markjan@let.rug.nl (Mark-Jan Nederhof), satta@dei.unipd.it (Giorgio Satta).

[1] Supported by the PIONIER Project *Algorithms for Linguistic Processing*, funded by NWO (Dutch Organization for Scientific Research).
[2] Partially supported by MIUR under project PRIN No. 2003091149_005.

Given as input a sequence of deterministic finite automata $M_1, \ldots, M_n$, deciding whether $\cap_{i=1}^{n} L(M_i) \neq \emptyset$ is PSPACE-complete, following [13]. The problem is solvable in polynomial time however for any fixed n; see e.g. [7].

Given as input a finite automaton M and a context-free grammar (CFG) G, the problem of deciding whether $L(M) \cap L(G) \neq \emptyset$ is solvable in polynomial time [10]. Given as input two CFGs G_1 and G_2, it is undecidable whether $L(G_1) \cap L(G_2) \neq \emptyset$, even if the alphabet has size 2 [5]. Given as input two s-grammars G_1 and G_2 that generate languages without arbitrarily long common prefixes, it is decidable whether $L(G_1) \cap L(G_2) \neq \emptyset$, but this problem is not recursively time-bounded [8,9].

The central problem in this paper is that of deciding non-emptiness of the intersection of the languages generated by two input CFGs G_1 and G_2, of which G_2 is non-recursive. (If a grammar does not exhibit recursion, then the generated language is a finite set.) Our main result is that this problem can be solved in polynomial space. We also show that the problem is PSPACE-complete, and remains so even when G_1 is non-recursive as well, and both G_1 and G_2 are deterministic. The problem of intersecting the languages generated by two CFGs, one of which is non-recursive, has been recently investigated in the artificial intelligence literature [14,12], to model applications such as natural language surface generation and machine translation. Practical algorithms for finding strings in the intersection have been reported by [19]. However, none of this work contained a computational analysis of the problem.

The structure of this paper is as follows. We consider the computational complexity of deciding non-emptiness of intersection for a pair of non-recursive CFGs in Section 2. In Section 3 we investigate similar problems on sequences of CFGs, a certain number of which are non-recursive. Definitions of CFGs, deterministic CFGs and linear bounded automata (LBAs) and some of their properties are summarized in the appendix.

2 Non-emptiness of intersection for pairs of grammars

In this section we investigate the computational complexity of the problem of deciding non-emptiness of the intersection of the languages generated by a pair of grammars, at least one of which is non-recursive (see the appendix for the definition of non-recursive CFG). We start by proving a lower bound.[3] The construction we use in the proof is very similar to those published before by e.g. [6,8] and it makes use of the idea of squaring introduced by [18]. Therefore

[3] Harry B. Hunt has communicated to us that he has found a proof of the same theorem, some time before.

we only provide an outline of the proof here.

Theorem 1 *Let the input consist of two non-recursive deterministic CFGs. The problem of deciding non-emptiness of the intersection of the two generated languages is PSPACE-hard.*

Proof. To prove the theorem, we show that the problem of deciding whether a LBA $M = (Q, T, \Sigma, \#, q_0, q_f, \Delta)$ accepts an input string $w = a_1 \cdots a_n \in \Sigma^*$ can be reduced in polynomial time to the problem of deciding whether $L(G_1) \cap L(G_2) \neq \emptyset$ for two non-recursive deterministic CFGs G_1 and G_2.

The idea is that $L(G_1) \cap L(G_2)$ consists of strings that encode accepting computations of M on w by strings of the form

$$v_1 v_2^R v_2 v_3^R \cdots v_{m-1} v_m^R$$

where each string v_k, $1 \leq k \leq m$, is an instantaneous description (i.d.) of M on w (hence $|v_k| = n+1$, $1 \leq k \leq m$), v_1 is the initial i.d. for input w and v_m is the accepting i.d., and $v_k \vdash v_{k+1}$, for each k ($1 \leq k < m$). (R is the string reversal operator; $\vdash$ is the 'move relation' of M as defined in the appendix.) Since there are only $|Q| \cdot |T|^n \cdot (n+1)$ distinct i.d.'s, we may restrict our attention to such strings with $m \leq F$, where we define $E = \lceil \log_2(|Q| \cdot |T|^n \cdot (n+1)) \rceil$ and $F = 2^E$, and due to our assumptions on LBAs given in the appendix, we may even restrict our attention to such strings with m equal to F. The set of such strings is exactly the intersection of the languages L_1 and L_2 defined by:

$$\begin{aligned}
L_1 = \{v_1 v_2^R v_2 \cdots v_{F-1} v_F^R \;\mid\; & v_1, \ldots, v_F \text{ are i.d.'s for input of length } n, \\
& v_1 \text{ is initial i.d. for } w, \\
& v_F \text{ is accepting i.d. of length } n+1\} \\
L_2 = \{v_1 x_2^R v_2 \cdots v_{F-1} x_F^R \;\mid\; & v_1, \ldots, v_F \text{ are i.d.'s of length } n+1, \\
& x_1, \ldots, x_F \text{ are i.d.'s of length } n+1, \\
& v_k \vdash x_{k+1} \text{ for } 1 \leq k < F\}
\end{aligned}$$

We can easily construct a non-recursive deterministic CFG with start symbol A_E generating $\{vx^R \mid v, x \text{ are i.d.'s of length } n+1 \text{ and } v \vdash x\}$, of size polynomial in $|M|$ and linear in n. (We may even construct an s-grammar for this language, similarly to a construction from [8]; s-grammars are a subclass of the deterministic grammars.) By adding E rules of the form $A_k \to A_{k+1}\ A_{k+1}$ ($0 \leq k < E$) we obtain a non-recursive deterministic CFG with start symbol A_0 generating L_2. Note that E is polynomial in $|M|$ and linear in n. In a similar way, we may construct a non-recursive deterministic CFG generating L_1, in polynomial time in $|M|$ and linear in n. This completes the proof. $\square$

We now state some well-known facts that will be used in the next theorem and in the next section as well. A CFG is in *canonical two-form* if its rules are of the form $A \to BC$, $A \to B$, $A \to a$ or $S \to \varepsilon$, with A, B, C nonterminals, a a terminal, S the start symbol and ε the empty string.

Lemma 2 *Let G be a CFG.*

(i) *a CFG in canonical two-form and generating the same language as G can be constructed in time $\mathcal{O}(|G|)$;*
(ii) *if G is non-recursive, the length of strings it generates is bounded by $\mathcal{O}(2^{|G|})$.*

A proof of (i) can be found at Theorem 4.20, p. 136, of [20]. Statement (ii) can be easily proved by induction on the maximal depth of parse trees (or subtrees thereof) that can be constructed out of a non-recursive set of rules.

We now provide the main result of this paper, which is an upper bound for a problem more general than the one considered in Theorem 1.

Theorem 3 *Let the input consist of two CFGs, at least one of which is non-recursive. The problem of deciding non-emptiness of the intersection of the two generated languages can be solved by a deterministic Turing machine using space $\mathcal{O}(n^2)$, where n is the length of the input.*

Proof. Let G_1 be a CFG and let G_2 be a non-recursive CFG. We assume that G_1 and G_2 have the same set Σ of terminals. We may also assume that both G_1 and G_2 are in canonical two-form, following Lemma 2(i). If both grammars generate the empty string, then their sets of rules must include $S_1 \to \varepsilon$ and $S_2 \to \varepsilon$, where S_1 and S_2 are the start symbols of G_1 and G_2, respectively, and then $L(G_1) \cap L(G_2)$ is obviously non-empty. Otherwise, we may further ignore any rule of the form $S_1 \to \varepsilon$ or $S_2 \to \varepsilon$.

Let $G_2 = (\Sigma, N_2, S_2, R_2)$ and let M be a push-down automaton (PDA) constructed from G_2 by the top-down parsing strategy in the usual way (see for instance [15]): there is only one state, the stack symbols of M are symbols in $\Sigma \cup N_2$, the stack initially contains only S_2, and M accepts on empty stack. A transition of M is either an *expansion*, which replaces a symbol $A \in N_2$ at the top of the stack with the mirror image of α, for some rule $A \to \alpha$ in R_2, or a *scan*, which pops a terminal $a \in \Sigma$ from the stack while consuming the same terminal from the input.

Since G_2 is non-recursive, each stack in a computation of M has height bounded by $|G_2|$. Let $r, s \in N_2^*$ be two stacks with heights $|r|$ and $|s|$, where $1 \leq |r| \leq |G_2|$ and $0 \leq |s| < |G_2|$, and let $a \in \Sigma$. We write $r \models_a s$ to denote that s can be obtained from r by applying one or more expansions, followed by a scan on terminal a. For a string $v = b_1 \cdots b_m$, $m \geq 1$, we write $r \models_v^+ s$ to denote the existence of a sequence of stacks $r = t_0, t_1, \ldots, t_m = s$ such that $t_0 \models_{b_1} t_1 \models_{b_2} \cdots \models_{b_{m-1}} t_{m-1} \models_{b_m} t_m$. Observe that M recognizes input $w = a_1 \cdots a_n$ if and only if $S_2 \models_w^+ \varepsilon$. We also write $r \models_v^* s$ if and only if $r \models_v^+ s \vee (v = \varepsilon \wedge r = s)$.

We now show how relation $\models_a$ can be decided in space $\mathcal{O}(\|G_2\|)$ by a deterministic Turing machine. If $r \models_a s$ holds, we must have $r = \alpha A$ and $s = \alpha B_1 B_2 \cdots B_q$, for some $\alpha \in N_2^*$, $A \in N_2$ and $B_i \in N_2$, $1 \leq i \leq q$, $0 \leq q \leq |G_2|$. Let now $P_0 = \{A' \mid A \Rightarrow^* A'\}$ and, for $1 \leq j \leq q$, define $P_j = \{D' \mid C \in P_{j-1},\ (C \rightarrow D\ B_j) \in R_2,\ D \Rightarrow^* D'\})$. It is not difficult to show that $r \models_a s$ if and only if $(E \rightarrow a) \in R_2$ for some $E \in P_q$. Observe that, for each pair $A, A' \in N_2$, $A \Rightarrow^* A'$ can easily be tested by a deterministic Turing machine using space $\mathcal{O}(\|G_2\|)$. Hence, set P_0 can also be computed in space $\mathcal{O}(\|G_2\|)$. Similarly, note that each set P_j, $1 \leq j \leq q$, can be computed in space $\mathcal{O}(\|G_2\|)$ from set P_{j-1}, and that P_{j-1} can be discarded after P_j has been computed. Therefore P_q may be computed from r and s in space $\mathcal{O}(\|G_2\|)$, and $r \models_a s$ can be decided in space $\mathcal{O}(\|G_2\|)$ by a deterministic Turing machine.

We now introduce a set $\mathcal{T}$ that plays a crucial role in our proof. This set consists of all triples (r, A, s), with $r, s \in N_2^*$, $|r| \leq |G_2|$, $|s| < |G_2|$ and $A \in N_1$, such that for some string $v \in \Sigma^+$:

(i) $A \Rightarrow^* v$ in G_1; *and*
(ii) $r \models_v^+ s$ in M.

Note that $(S_2, S_1, \varepsilon) \in \mathcal{T}$ if and only if there is some string that is both generated by G_1 and recognized by M, which indicates non-emptiness of $L(G_1) \cap L(G_2)$.

We can compute the set above using a dynamic programming algorithm that is specified in an abstract way in Figure 1 by means of a deduction system, following [17]. The side conditions of the inference rules refer to rules from G_1 and relation $\models_a$ as defined for M. The algorithm is a straightforward variant of a construction by [2] to compute the intersection of a CFG and a finite automaton, and we therefore omit a proof of correctness here.

A derivation in the deduction system from Figure 1 is a tree with nodes labelled by elements of $\mathcal{T}$. A leaf node corresponds to an application of the axiom (1). A non-leaf node together with its daughter nodes corresponds to the application of inference rule (2). The label at the root of the tree is the triple from $\mathcal{T}$ that

$$\frac{}{(r, A, s)} \begin{cases} r \models_a s, \\ (A' \to a) \in R_1, \\ A \Rightarrow^* A' \end{cases} \tag{1}$$

$$\frac{(r, B, s) \quad (s, C, t)}{(r, A, t)} \begin{cases} (A' \to B\ C) \in R_1, \\ A \Rightarrow^* A' \end{cases} \tag{2}$$

Fig. 1. Deduction system deriving triples $(r, A, s) \in \mathcal{T}$, each of which represents a derivation $A \Rightarrow^* v$ in G_1 and a computation $r \models_v^+ s$ in M involving the same string v.

the derivation derives. Let us specifically consider a derivation τ that derives (S_2, S_1, ε). The number of leaves of τ is bounded by the length of the longest string generated by G_2, and thus is bounded by $2^{|G_2|}$ by Lemma 2(ii). Due to the binary form of inference rule (2), the size of τ, measured in the number of nodes and denoted $|\tau|$, is bounded by $2^{|G_2|+1}$. We define the *height* of a derivation as the length of the longest path from the root to a leaf, measured in the number of edges.

A derivation of item (S_2, S_1, ε) in the deduction system in Figure 1 may have a height that is exponential in $|G_2|$. This is a problem for developing an algorithm that checks in polynomial space whether the set of such derivations is non-empty. For this reason, we turn to a different set $\mathcal{T}_{gap}$ that includes $\mathcal{T}$ and in addition contains pairs of triples of the form $(r, A, s)/(t, B, u)$, which indicate that $(r, A, s) \in \mathcal{T}$ provided $(t, B, u) \in \mathcal{T}$. A pair $(r, A, s)/(t, B, u) \in \mathcal{T}_{gap}$ can best be thought of as representing a *gapped derivation* according to the deduction system in Figure 1, i.e. a derivation that derives (r, A, s) but that has a missing subtree at the node labelled (t, B, u), which acts as a leaf although it was not derived by axiom (1); this node labelled (t, B, u) cannot be the root however. Formally, $(r, A, s)/(t, B, u) \in \mathcal{T}_{gap}$ if and only if there are strings v, w such that:

(i) $A \Rightarrow^+ vBw$ in G_1; *and*
(ii) $r \models_v^* t$ and $u \models_w^* s$ in M.

The elements of $\mathcal{T}_{gap}$ can be computed by the deduction system in Figure 2. This is a straightforward variant of Rytter's parallel algorithm for context-free parsing [4], and we therefore omit a proof of correctness here.

We now prove the following claim. With reference to the deduction system in Figure 1, let τ_1 be a derivation with root labelled (r, A, s), or a gapped deriva-

$$\frac{}{(r, A, s)} \begin{cases} r \models_a s, \\ (A' \to a) \in R_1, \\ A \Rightarrow^* A' \end{cases} \tag{3}$$

$$\frac{(r, B, s)}{(r, A, t)/(s, C, t)} \begin{cases} (A' \to B\ C) \in R_1, \\ A \Rightarrow^* A', \\ |t| \leq |G_2| \end{cases} \tag{4}$$

$$\frac{(s, C, t)}{(r, A, t)/(r, B, s)} \begin{cases} (A' \to B\ C) \in R_1, \\ A \Rightarrow^* A', \\ |r| \leq |G_2| \end{cases} \tag{5}$$

$$\frac{(r, A, s)/(t, B, u) \quad (t, B, u)}{(r, A, s)} \tag{6}$$

$$\frac{(r, A, s)/(t, B, u) \quad (t, B, u)/(x, C, y)}{(r, A, s)/(x, C, y)} \tag{7}$$

Fig. 2. Alternative deduction system based on Rytter's algorithm.

tion with root labelled (r, A, s) and with a missing subtree at a distinguished leaf labelled (t, B, u). Then there exists a derivation τ_2, according to the deduction system in Figure 2, deriving (r, A, s) or $(r, A, s)/(t, B, u)$, respectively, with height bounded by $6 \cdot \log_2(|\tau_1|) + 1$. Our proof of this claim is by induction on $|\tau_1|$. The intuition is that τ_1 is recursively divided into smaller (gapped) derivations, each of size at most half the original size, until we end up with (gapped) derivations of size 1 or 3. Based on this decomposition, we can create a derivation τ_2 according to Figure 2, pasting subderivations together again, increasing the height by at most 6 nodes for each level of recursion. Note that $|\tau_1|$ is always odd, since τ_1 is a binary tree.

The simplest base case arises when $|\tau_1| = 1$. This means that τ_1 is a tree consisting of a single node that was derived by axiom (1) in Figure 1, which also occurs as axiom (3) in Figure 2. Therefore we can take $\tau_2 = \tau_1$. Since $|\tau_2| = 1 \leq 6 \cdot \log_2(|\tau_1|) + 1 = 1$, we have proven this base case. When $|\tau_1| = 3$, then τ_1 can be a derivation or a gapped derivation. Both these cases can be proven easily, similarly to the case $|\tau_1| = 1$.

We use the induction hypothesis for the case $|\tau_1| \geq 5$. Let us first identify a node labelled (x_1, C, x_3) in τ_1 that has two daughter nodes labelled (x_1, C_1, x_2) and (x_2, C_2, x_3) and that is such that the number of nodes in the subtree rooted in (x_1, C, x_3) is at least $|\tau_1|/2$, but the number of nodes in each of the subtrees rooted in (x_1, C_1, x_2) and (x_2, C_2, x_3) is smaller than $|\tau_1|/2$. There is always exactly one node that satisfies this description. We can now distinguish a large number of subcases. We will explicitly treat one of the more complicated ones; the others are left to the reader. For this subcase, we assume that τ_1 is a gapped derivation, with root (r, A, s) and distinguished leaf (t, B, u) where a subtree is missing. We also assume that (t, B, u) is not a descendant of (x_1, C, x_3), but that there is a node labelled (y_1, D, y_3) with daughter nodes labelled (y_1, D_1, y_2) and (y_2, D_2, y_3), where (t, B, u) is a descendant of (y_1, D_1, y_2) and (x_1, C, x_3) is a descendant of (y_2, D_2, y_3).

This case is sketched in Figure 3. Note that we have now divided τ_1 into a number of smaller derivations and gapped derivations, each of size at most $|\tau_1|/2$. There is for example a gapped derivation with root (y_2, D_2, y_3) and distinguished leaf (x_1, C, x_3), and a (non-gapped) derivation with root (x_1, C_1, x_2). The induction hypothesis now states that there are derivations according to Figure 2 that derive e.g. $(y_2, D_2, y_3)/(x_1, C, x_3)$ and (x_1, C_1, x_2), and each of these derivations has height at most $6 \cdot \log_2(|\tau_1|/2) + 1$.

We can put these derivations together again into a derivation τ_2 of $(r, A, s)/(t, B, u)$ as sketched in Figure 4. This derivation has height 6, plus the maximal height of the derivations obtained from the induction hypothesis. The total height is therefore at most $6 + 6 \cdot \log_2(|\tau_1|/2) + 1 = 6 \cdot \log_2(|\tau_1|) + 1$. This ends the proof of our claim.

On the basis of this claim, we may conclude that (S_2, S_1, ε) is derivable in the deduction system from Figure 1 if and only if (S_2, S_1, ε) is derivable in the deduction system from Figure 2 by a derivation of height at most

$$6 \cdot \log_2(2^{|G_2|+1}) + 1 = 6 \cdot |G_2| + 7 \tag{8}$$

which is linear in $|G_2|$.

We follow an idea from [16], developed further in [3], which allows us to formulate a recursive function that tries to find a derivation according to the deduction system from Figure 2. The function takes as input argument the triple or pair of triples at the root of the derivation that is to be found. In the initial call, this argument is (S_2, S_1, ε). The function enumerates over all possible instantiations of inference rules that have the argument as consequent, and then applies the function recursively on the antecedents. We also restrict

$$\dfrac{\begin{array}{c}\dfrac{\begin{array}{c}(t,B,u)\\ \vdots\end{array}}{(y_1,D_1,y_2)} \quad \dfrac{\begin{array}{c}\dfrac{\dfrac{\vdots}{(x_1,C_1,x_2)} \quad \dfrac{\vdots}{(x_2,C_2,x_3)}}{(x_1,C,x_3)}\\ \vdots\end{array}}{(y_2,D_2,y_3)}\\ \hline (y_1,D,y_3)\\ \vdots\end{array}}{(r,A,s)}$$

Fig. 3. Gapped derivation according to the deduction system from Figure 1, with root labelled (r, A, s) and missing subtree at the leaf labelled (t, B, u).

$$\dfrac{(r,A,s)/(y_1,D,y_3) \quad \dfrac{\dfrac{\dfrac{(y_2,D_2,y_3)/(x_1,C,x_3) \quad \dfrac{\dfrac{(x_1,C_1,x_2)}{(x_1,C,x_3)/(x_2,C_2,x_3)} \quad (x_2,C_2,x_3)}{(x_1,C,x_3)}}{(y_2,D_2,y_3)}}{(y_1,D,y_3)/(y_1,D_1,y_2)} \quad (y_1,D_1,y_2)/(t,B,u)}{(y_1,D,y_3)/(t,B,u)}}{(r,A,s)/(t,B,u)}$$

Fig. 4. Subderivation according to the deduction system from Figure 2, deriving $(r, A, s)/(t, B, u)$. At the leaves we attach subderivations obtained from the induction hypothesis of our proof.

the maximal depth of recursion to $6 \cdot |G_2| + 7$.

The function can be executed on a deterministic Turing machine through the use of a stack. The maximal number of elements on the stack is the maximal depth of recursion. Note also that each function call requires space $\mathcal{O}(\|G_1\| + \|G_2\|)$, since that amount of space is sufficient to enumerate all instances of inference rules. The total stack size is therefore $\mathcal{O}((6 \cdot |G_2| + 7) \cdot (\|G_1\| + \|G_2\|))$, or quadratic in the complete input length. This ends the proof of the theorem. $\square$

By combining Theorems 1 and 3, we obtain the following results.

Theorem 4 *Let the input consist of two CFGs, at least one of which is non-recursive. The problem of deciding non-emptiness of the intersection of the two generated languages is PSPACE-complete. The problem remains PSPACE-complete if both CFGs are constrained to be non-recursive and deterministic.*

3 Non-emptiness of intersection for any number of grammars

In this section we investigate generalizations of the problems presented in the previous section. We allow sequences of CFGs with at least one non-recursive CFG.

Theorem 5 *Let the input consist of a CFG and a sequence of non-recursive CFGs. The problem of deciding non-emptiness of the intersection of all the generated languages can be solved by a deterministic Turing machine using space $\mathcal{O}(n^2)$, where n is the length of the input.*

Outline of the proof. We only provide an outline of the proof here, since the argument is closely related to the one presented in the proof of Theorem 3. Let $G = (\Sigma, N, S, R)$ be a CFG and let $G_i = (\Sigma, N_i, S_i, R_i)$, $1 \leq i \leq k$, be non-recursive CFGs.

We may assume that all these grammars are in canonical two-form, following Lemma 2(i). Let M_i, $1 \leq i \leq k$, be PDAs constructed from G_i as in the proof of Theorem 3. Similarly to that proof, we define items $(r_1, \ldots, r_k, A, s_1, \ldots, s_k)$ and items $(r_1, \ldots, r_k, A, s_1, \ldots, s_k)/(t_1, \ldots, t_k, B, u_1, \ldots, u_k)$, with $A, B \in N$, $r_i, s_i, t_i, u_i \in N_i^*$, and $|r_i|, |s_i|, |t_i|, |u_i| \leq |G_i|$, $1 \leq i \leq k$. Strings r_i, s_i, t_i, u_i represent possible stacks of M_i. We define a set of items $\mathcal{T}_{gap}$ such that $(r_1, \ldots, r_k, A, s_1, \ldots, s_k) \in \mathcal{T}_{gap}$ if and only if there is a string v with

(i) $A \Rightarrow^* v$ in G; *and*
(ii) $r_i \models_v^+ s_i$ in M_i, $1 \leq i \leq k$.

Furthermore, we have $(r_1, \ldots, r_k, A, s_1, \ldots, s_k)/(t_1, \ldots, t_k, B, u_1, \ldots, u_k) \in \mathcal{T}_{gap}$ if and only if there are strings v, w such that:

(i) $A \Rightarrow^+ vBw$ in G; *and*
(ii) $r_i \models_v^* t_i$ and $u_i \models_w^* s_i$ in M_i, $1 \leq i \leq k$.

Note that $(S_1, \ldots, S_k, S, \varepsilon, \ldots, \varepsilon) \in \mathcal{T}_{gap}$ if and only if the intersection of the language generated by G and all the languages generated by G_i, $1 \leq i \leq k$, is non-empty.

By using the deduction system in Figure 2, adapted to the new kinds of items, we can compute set $\mathcal{T}_{gap}$. Let i^* be an integer such that $1 \leq i^* \leq k$ and $|G_{i^*}| \geq |G_i|$ for $1 \leq i \leq k$. The corresponding argument from the proof of Theorem 3 can be adapted to show that item $(S_1, \ldots, S_k, S, \varepsilon, \ldots, \varepsilon)$ is derivable in our deduction system if and only if there is a derivation for this item of height at most $6 \cdot |G_{i^*}| + 7$, which is a linear bound in the size of the input. We can then define a deterministic Turing machine that searches for all derivations of the above item within the given bound on the height. This can be realized by a stack of function calls, each of which requires an amount of space $\mathcal{O}(\sum_{i=1}^{k} \|G_i\|)$, resulting in overall quadratic space in the size of the input. □

If all input CFGs are restricted to be non-recursive, we can prove a tighter upper bound.

Theorem 6 *Let the input consist of a sequence of non-recursive CFGs. The problem of deciding non-emptiness of the intersection of all the generated languages can be solved by a linear space nondeterministic Turing machine.*

Proof. Let $G_i = (\Sigma, N_i, S_i, R_i)$, $1 \leq i \leq k$, be non-recursive CFGs in two-normal form. Let also M_i, $1 \leq i \leq k$, be PDAs constructed from G_i as in the proof of Theorem 3. We define a nondeterministic finite automaton $M_\cap = (\Sigma, Q, \delta, q_0, F)$ that recognizes the intersection of the languages $L(G_i)$, $1 \leq i \leq k$. The set of states Q contains all items of the form $(r_1, \ldots, r_k)$, with $r_i \in N_i^*$ and $|r_i| \leq |G_i|$, $1 \leq i \leq k$. Each r_i represents a possible stack in a computation of M_i. The initial state q_0 is $(S_1, \ldots, S_k)$ and the set of final states F contains only the state $(\varepsilon, \ldots, \varepsilon)$. The transition function δ is defined as follows. For each $a \in \Sigma$ and for each $(r_1, \ldots, r_k), (s_1, \ldots, s_k) \in Q$ such that $r_i \models_a s_i$ in M_i, $1 \leq i \leq k$, we let $(s_1, \ldots, s_k) \in \delta((r_1, \ldots, r_k), a)$.

We can now specify a nondeterministic Turing machine M that takes as input a sequence of grammars G_i, $1 \leq i \leq k$, and accepts if and only if the intersection of the generated languages is non-empty or, equivalently, if and only if there is at least one string accepted by $M_\cap$. At each moment in the computation, the working tape of M will contain a state from Q that we will refer to as the *hypothesis*. The hypothesis is initially q_0. Given hypothesis q, M uses its nondeterministic states for guessing an input symbol $a \in \Sigma$ and a new state $q' \in Q$, which is written on the working tape. It then checks whether $q' \in \delta(q, a)$. If this is the case, M erases q from the working tape, keeps q' as the hypothesis and the step is repeated; otherwise, the computation halts with failure. M accepts whenever the hypothesis is $(\varepsilon, \ldots, \varepsilon)$. Let i^* be an integer such that $1 \leq i^* \leq k$ and $|G_{i^*}| \geq |G_i|$ for $1 \leq i \leq k$. From Lemma 2(ii) we have that $2^{|G_{i^*}|}$ is a bound on the length of any string accepted by some M_i. M uses a binary counter as a clock, and stops with a failure whenever more than $2^{|G_{i^*}|} + 1$ states of M have been visited.

To conclude the proof, we observe that M uses an amount of space $\mathcal{O}(\sum_{i=1}^{k} \|G_i\|)$. This is because the i-th component of each state $q \in Q$ can be represented in space $\mathcal{O}(\|G_i\|)$, $1 \leq i \leq k$, and there are at most two states in the working tape at each step in the computation. Furthermore, checking a single instance $r_i \models_a s_i$ can be done in linear space, as discussed in the proof of Theorem 3. Finally, the counter used by M can be implemented using space $\log_2(2^{|G_{i^*}|} + 1) = \mathcal{O}(|G_{i^*}|)$. $\square$

We now turn to cases where the input includes more than one arbitrary CFG. The class NEXPTIME below is the class of all languages recognized by some nondeterministic Turing machine in time $\mathcal{O}(2^{c \cdot n})$ for some constant c, where n is the length of the input. We only provide an outline of the proof, since we use constructions that have already appeared in the literature.

Theorem 7 *Let the input consist of a non-recursive CFG G and a sequence of CFGs G_i, $1 \leq i \leq k$. The problem of deciding non-emptiness of the intersection of all the generated languages is NEXPTIME-complete. The problem remains NEXPTIME-complete if G is constrained to be deterministic, and the sequence G_i, $1 \leq i \leq k$, is constrained to consist of exactly $k = 2$ deterministic CFGs.*

Outline of the proof. We first prove NEXPTIME-hardness for the restricted problem of three deterministic CFGs, one of which is non-recursive. Let M be an arbitrary nondeterministic Turing machine running in time $f(n)$ in the length n of its input, with $f(n)$ an exponential function. Hence $L(M) \in$ NEXPTIME. From here on, M is fixed. We show a polynomial time reduction from the problem of deciding whether an input string w belongs to $L(M)$ to the problem of deciding whether $L(G) \cap L(G_1) \cap L(G_2)$ is non-empty for a non-recursive deterministic CFG G and for deterministic CFGs G_1 and G_2.

Without loss of generality, we can assume that each i.d. of M on input w has length $f(|w|)$, and each valid computation of M on input w requires exactly $f(|w|)$ steps. We can construct two deterministic CFGs G_1 and G_2 such that $L(G_1) \cap L(G_2)$ is the set of all valid computations of M on input w; see for instance [8]. The construction can be carried out in linear time in the size of the input w. We then construct a non-recursive deterministic CFG G that accepts all and only the strings defined over the common alphabet of G_1 and G_2 and having length exactly $f(|w|)^2$. This can be easily done using the squaring technique already exploited in the proof of Theorem 1. The construction of G can be carried out in time $\mathcal{O}(|w| \cdot \log(|w|))$. This completes the hardness part of the proof.

We now prove membership in NEXPTIME for the general problem. Let G be a non-recursive CFG, and let G_i, $1 \leq i \leq k$, be arbitrary CFGs. Since G is non-recursive, Lemma 2(ii) implies that $2^{|G|}$ is an upper bound on the length of all strings in $L(G)$. Let M be a nondeterministic Turing machine specified as follows. Using its nondeterministic states, M guesses a string w

of length smaller than or equal to $2^{|G|}$. Then M checks whether w belongs to $L(G)$ and to each $L(G_i)$, $1 \leq i \leq k$, and answers accordingly. Since the word recognition problem for general context-free grammars can be carried out in deterministic polynomial time (more precisely, cubic time on a random access machine; see [1]), the overall running time of M is dominated by the guessing of the string w and is therefore in $\mathcal{O}(2^{c \cdot n})$, for certain constant c, where n is the size of the input. This proves that the general problem is in NEXPTIME. □

4 Discussion

In this paper we have investigated the problem of deciding non-emptiness for the intersection of the languages generated by a sequence of arbitrary input CFGs and a sequence of non-recursive input CFGs, under various restrictions on the lengths of the two sequences. All of the upper bounds we have presented also hold for the complement problem of deciding whether the intersection of the languages generated by the input grammars is empty, rather than non-empty. This is because these results refer to complexity classes that are closed under complementation. In particular, the class of nondeterministic linear space languages, mentioned in Theorem 6, is closed under complementation, as shown in [11]. The lower bounds in Theorems 1 and 7 can also be extended to the empty intersection problem with only minor changes to the proofs.

We have considered here the class of non-recursive CFGs, motivated by recent interest in the literature for applications based on this problem, as discussed in the introduction. It is very easy however to extend all of our results by replacing non-recursive CFGs by CFGs generating finite languages. (Note that if a CFG generates a finite language, recursion might still be found in derivations of the form $A \Rightarrow^+ A$, involving unary rules and/or epsilon rules.) This directly follows from Lemma 2(i), which would eliminate recursion from a CFG without changing the generated language and using only linear space.

A natural question is this: Up to what degree can the constraint of absence of recursion, as expressed in Theorem 4, be relaxed while preserving membership in the class PSPACE? Note that membership in PSPACE was established by exploiting the following two properties: (i) the non-recursive CFG can be translated into a PDA with stack of bounded height not dependent on the input string, and (ii) the length of the shortest string in the intersection, if any exists, is bounded by an exponential function in the size of the input. If the constraint of absence of recursion is replaced by the constraint that all recursion should be right recursion (i.e. for each nonterminal A, if $A \Rightarrow^* \alpha A \beta$, some α and β, then $\beta = \epsilon$), then property (i) still holds, as this also leads to PDAs with bounded stack height. (Conversely, if a PDA realizing the top-down strategy has bounded stack height, then there cannot be other types of recursion but right-recursion.) However, property (ii) above no longer holds in general, and the proof techniques we used are no longer applicable. We leave as an open problem whether non-emptiness of the intersection of two languages remains in PSPACE if they are represented by two input CFGs, one of which contains right recursion as only type of recursion.

References

[1] A.V. Aho and J.D. Ullman. *Parsing*, volume 1 of *The Theory of Parsing, Translation and Compiling*. Prentice-Hall, 1972.

[2] Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. In Y. Bar-Hillel, editor, *Language and Information: Selected Essays on their Theory and Application*, chapter 9, pages 116–150. Addison-Wesley, 1964.

[3] S.A. Cook. Path systems and language recognition. In *ACM Symposium on Theory of Computing*, pages 70–72, 1970.

[4] A. Gibbons and W. Rytter. *Efficient Parallel Algorithms*. Cambridge University Press, 1988.

[5] M.A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, 1978.

[6] J. Hartmanis. Context-free languages and Turing machine computations. In J.T. Schwartz, editor, *Proceedings of a Symposium in Applied Mathematics*, volume 19, pages 42–51, 1967.

[7] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[8] H.B. Hunt III. On the decidability of grammar problems. *Journal of the ACM*, 29(2):429–447, 1982.

[9] H.B. Hunt III. Terminating Turing machine computations and the complexity and/or decidability of correspondence problems, grammars, and program schemes. *Journal of the ACM*, 31(2):299–318, 1984.

[10] H.B. Hunt III, D.J. Rosenkrantz, and T.G. Szymanski. On the equivalence, containment, and covering problems for the regular and context-free languages. *Journal of Computer and System Sciences*, 12:222–268, 1976.

[11] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[12] K. Knight and I. Langkilde. Preserving ambiguities in generation via automata intersection. In *Proceedings of the Seventeenth National Conference on Artificial*

Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence, pages 697–702, Austin, Texas, USA, July–August 2000.

[13] D. Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science*, pages 254–266, 1977.

[14] I. Langkilde. Forest-based statistical sentence generation. In *6th Applied Natural Language Processing Conference and 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 170–177 (Section 2), Seattle, Washington, USA, April–May 2000.

[15] H.R. Lewis and C.H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, 1981.

[16] P.M. Lewis II, R.E. Stearns, and J. Hartmanis. Memory bounds for recognition of context-free and context-sensitive languages. In *Conference Record of the 6th Annual Symposium on Switching Circuit Theory and Logical Design*, pages 191–202, 1965.

[17] D. McAllester. On the complexity analysis of static analyses. *Journal of the ACM*, 49(4):512–537, 2002.

[18] A.R. Meyer and L.J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Conference Record of the 13th IEEE Symposium on Switching and Automata Theory*, pages 125–129, 1972.

[19] M.-J. Nederhof and G. Satta. Parsing non-recursive context-free grammars. In *40th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 112–119, Philadelphia, Pennsylvania, USA, July 2002.

[20] S. Sippu and E. Soisalon-Soininen. *Parsing Theory, Vol. I: Languages and Parsing*, volume 15 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1988.

Appendix

Here we summarize the definitions of CFGs and LBAs, and some of their properties, which were used in the preceding sections.

A context-free grammar (CFG) is a 4-tuple $G = (\Sigma, N, S, R)$, where Σ is a finite set of *terminals*, N is a finite set of *nonterminals*, including the *start symbol* S, and R is a finite set of *rules*, each of the form $A \to \alpha$, where $A \in N$ and $\alpha \in (\Sigma \cup N)^*$. The 'derives' relation associated with G is denoted by $\Rightarrow$, its transitive closure by $\Rightarrow^+$, and its reflexive and transitive closure by $\Rightarrow^*$. We define the language generated by G to be $L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$.

A CFG is said to be in Chomsky normal form (CNF) if each rule is of one of the forms $S \to \varepsilon$, $A \to B\ C$ or $A \to a$, where $A, B, C \in N$ and $a \in \Sigma$. For each CFG G we can find a CFG G' in CNF such that $L(G') = L(G)$, by a transformation that can be carried out in quadratic time [5].

We say G is *non-recursive* if $A \Rightarrow^+ \alpha A \beta$ does not hold for any $A \in N$ and $\alpha, \beta \in (\Sigma \cup N)^*$. A non-recursive CFG always generates a finite language. Conversely, if $L(G)$ is finite, then there is a non-recursive CFG G' such that $L(G') = L(G)$. Furthermore, such a CFG G' can be found in polynomial time, e.g. by a transformation to CNF.

Following [5] we say G is *deterministic* if there is a partition π of $\Sigma \cup N$ and a set $\mathcal{E} \subset N$ such that $\Sigma \in \pi$ and for every $A, A', B \in N$ and $\alpha, \beta, \beta' \in (\Sigma \cup N)^*$:

(i) if $(A \to \alpha B \beta) \in R$ and $B \in \mathcal{E}$ then $\beta = \varepsilon$ and $A \in \mathcal{E}$;
(ii) if $(A \to \alpha\beta), (A' \to \alpha\beta') \in R$ and if A and A' are in the same equivalence class, then at least one of the following four cases is true:
  (a) $\beta \neq \varepsilon$ and $\beta' \neq \varepsilon$ and the first symbol from β and the first symbol from β' are in the same equivalence class;
  (b) $\beta = \beta' = \varepsilon$ and $A = A'$;
  (c) $\beta = \varepsilon$ and $A \in \mathcal{E}$;
  (d) $\beta' = \varepsilon$ and $A' \in \mathcal{E}$.

A deterministic grammar can be straightforwardly mapped to a deterministic push-down automaton that recognizes the same language.

We define the *size* of G as $|G| = \sum_{(A \to \alpha) \in R} |A\alpha|$, where $|A\alpha|$ denotes the string length of $A\alpha$. Following [5], we define the *norm* of G as $\|G\| = |G| \cdot \log_2(|\Sigma| + |N|)$, where $|\Sigma|$ and $|N|$ denote the number of elements in Σ and N, respectively. The norm of a CFG is a reasonable measure of its size when it is represented by means of a fixed alphabet.

The following definitions were taken from [7], with at most notational changes. A *linear bounded automaton* (LBA) M is a nondeterministic Turing machine denoted by a 7-tuple $(Q, T, \Sigma, \#, q_0, q_f, \Delta)$, where Q is a finite set of *states*, T is a finite set of *tape symbols*, $\Sigma \subset T$ is a finite set of *input symbols*, $\# \in T \backslash \Sigma$ is the *blank*, $q_0 \in Q$ is the initial state, $q_f \in Q$ is the final state, Δ is the next-move relation, which is a finite subset of $(Q \times T) \times (Q \times T \times \{\mathcal{L}, \mathcal{R}, \mathcal{S}\})$ (where the symbols $\mathcal{L}$, $\mathcal{R}$ and $\mathcal{S}$ stand for 'left', 'right' and 'stationary'), subject to the following: Δ is such that the blank $\#$ cannot be overwritten with something other than a blank, nor can the tape head move to the right of a blank. More precisely, for each $((q, \#), (q', b, d)) \in \Delta$, we have $b = \#$ and $d \in \{\mathcal{L}, \mathcal{S}\}$.

A computation of a LBA on input $w = a_1 \cdots a_n \in \Sigma^*$ starts with tape content $a_1 \cdots a_n \#$, with the tape head on the first tape square, and with state q_0. The constraint we placed on Δ above implies that the tape head can only visit the first $n+1$ positions of the tape. We say input w is accepted by a LBA if and only if at least one computation on w reaches the final state q_f. We assume, without loss of generality, that before acceptance, all tape squares are overwritten with blanks, and the tape head returns to the first tape square. We may also assume that $q_0 \neq q_f$. Lastly, we assume that there is a single transition applicable in state q_f, given by $((q_f, \#), (q_f, \#, \mathcal{S}))$. This ensures that an accepting computation can be made unboundedly long. The problem of determining whether a given LBA accepts a given input string is PSPACE-complete [7].

For a fixed LBA and an input string of length n, an *instantaneous description* (i.d.) is a string in $T^* \cdot (Q \times T) \cdot T^*$ of length $n+1$. An i.d. $b_1 \cdots b_{j-1}(q, b_j)b_{j+1} \cdots b_{n+1}$ denotes that the current tape content is $b_1 \cdots b_{n+1}$, the current state is q and the tape head is currently on the j-th tape square. Symbol b_{n+1} is always $\#$. For a fixed LBA, we define the binary relation $\vdash$ on the set of instantaneous descriptions such that $\alpha \vdash \beta$ if and only if i.d. β follows from i.d. α by applying one transition from Δ, as e.g. in [7]. If we define $\vdash^+$ to be the transitive closure of $\vdash$, then a string $w = a_1 \cdots a_n$ is accepted by the LBA if and only if $(q_0, a_1)a_2 \cdots a_n\# \ \vdash^+ \ (q_f, \#)\#^n$.