The Inclusion Problem for Forest Languages under Substitutions

Marcial Gaßert
University of Stuttgart, FMI, Germany
marcial.gaissert@gaisseml.de

Manfred Kufleitner
University of Stuttgart, FMI, Germany
kufleitner@fmi.uni-stuttgart.de

Abstract. We consider algorithms and lower bounds for various problems over forest languages; as input models we allow forest algebras, deterministic forest automata and nondeterministic forest automata. For the equivalence problem, we give an almost-linear-time algorithm for both forest algebras and deterministic forest automata; this is complemented by a polynomial time hardness result. The emptiness problem is complete for polynomial time over each of the three models. Additionally, we consider the emptiness of intersection problem for forest algebras and deterministic forest automata; this problem turns out to be complete for exponential time. It is well-known that the corresponding problems for word languages are complete for nondeterministic logarithmic space and for polynomial space, respectively.

Equipped with this toolbox of algorithms and lower bounds, we consider various inclusion problems for regular forest languages under substitutions. The substitutions in this paper replace leaf variables by forest languages. Depending on the direction of the inclusion, the problem for a given substitution is either complete for polynomial time or for exponential time; in particular, the equivalence problem under substitutions is complete for exponential time and, hence, more difficult than the equivalence problem for forest languages without substitutions. If we ask whether there exists a substitution such that a given inclusion holds, then this problem is either complete for \(\text{NP}\) or exponential time, depending on whether we consider inclusion or equivalence; moreover, the problem is undecidable if the substitution is applied on both sides.

1. Introduction

Bojańczyk and Walukiewicz introduced the notion of a forest algebra alongside the recognition of forest languages by finite forest algebras. They also gave an equivalent but more succinct model called forest automata. We add to this the naturally occurring model of nondeterministic forest automata. Forests generalize ordered unranked trees to finite series of unranked trees, thereby admitting a uniform algebraic structure.

We consider relational substitutions on forest languages. Due to the additional structure of forest languages in contrast to word languages, multiple variants arise. We restrict ourselves to substituting leaves independently of one another.

In section 3 we briefly review some of the usual constructions on automata or algebraic models with their runtime or space bounds, and constructions for the aforementioned substitutions. In section 4 we give an almost linear time algorithm for the equivalence of deterministic
forest automata — generalizing the Hopcroft-Karp equivalence test — and a polynomial-time algorithm for checking emptiness of nondeterministic forest automata. In section 5 we then give completeness results for the emptiness, subset and equivalence problems on deterministic forest automata and forest algebras, as well as for the emptiness of the intersection of forest algebras.\footnote{Missing proofs can be found in the appendix of this submission.}

Conway considered substitutions which replace variables in a word by words from a given language. He showed that the set of inclusion maximal substitutions $\sigma$ satisfying $\sigma(L) \subseteq R$ is finite. Furthermore, for regular languages $L$ and $R$, all maximal substitutions are regular, that is, the languages by which the variables are substituted are all regular. The same result can easily be obtained when describing the languages using recognizing monoids and applying a method called saturation; this works by simply extending the substitutions to contain complete classes of the syntactic monoid of $R$.

It is natural to ask whether the principle of saturation extends from word languages recognized by finite monoids to forest languages recognized by finite forest algebras. For substitutions of leaves this is the case, as we will show in section 6.1. In addition, we provide matching lower bounds. Our results hold for both finite forest algebras and deterministic forest automata. Additionally, some results extend to nondeterministic forest automata.

There exists several related work in the literature on trees. For instance, the regular matching problem and the inclusion problem discussed by Boneva, Niewenhuis and Sakho are similar in spirit to the ones described here. In\footnote{Lavaud and Straubing considered constructions on the more succinct automaton models of BUDFAs and BUNFAs from a more practical perspective.} one problem is that string patterns can only be embedded vertically, thus making context variables necessary. With forests, this problem does not occur, since we can embed string patterns horizontally. Delignat-Lavaud and Straubing considered constructions on the more succinct automaton models of BUDFAs and BUNFAs from a more practical perspective.

2. Preliminaries

2.1. Forest Languages, Forest Algebras and Forest Automata

The following definitions of forests, forest languages, forest automata and recognition are due to Bojańczyk and Walukiewicz. We define the set $\mathcal{F}(A)$ of forests and the set $\mathcal{T}(A)$ of trees over a finite, non-empty alphabet $A$ inductively as follows:

- If $f_1, \ldots, f_k \in \mathcal{T}(A)$ for $k \geq 0$, then $f_1 + \cdots + f_k \in \mathcal{F}(A)$.
- For $k = 0$ we denote the resulting forest by $0$ and call it the empty forest.
- If $f \in \mathcal{F}(A)$ and $a \in A$, then $af \in \mathcal{T}(A)$.

For forests $f = f_1 + \cdots + f_k$ and $g = g_1 + \cdots + g_k$ with $f_i, g_i \in \mathcal{T}(A)$, let $f + g = f_1 + \cdots + f_k + g_1 + \cdots + g_k$. Furthermore, $a0 = a$. Note that $+$ is not required to be commutative. A forest induces a partial map $f$: $\text{dom}(f) \to A$ where $\text{dom}(f)$ is defined by

$$\begin{align*}
\text{dom}(0) &= \emptyset \\
\text{dom}(f_1 + \cdots + f_k) &= \{ix \mid 1x \in \text{dom}(f_i), 1 \leq i \leq k\} & \text{for } f_1, \ldots, f_k \in \mathcal{T}(A) \\
\text{dom}(af) &= \{1x \mid x \in \text{dom}(f)\} & \text{for } a \in A, f \in \mathcal{F}(A)
\end{align*}$$

The elements of $\text{dom}(f)$ correspond to positions or nodes in the forest; their images correspond to their label.

A context over an alphabet $A$ is a forest over $A \cup \{1\}$ such that 1 occurs only once, and this occurrence is a leaf. The set of all contexts is denoted by $\mathcal{C}(A)$. For a context $c \in \mathcal{C}(A)$ and a forest $f \in \mathcal{F}(A)$, we define $cf$ as the forest obtained by replacing the occurrence of 1 in $c$ by the forest $f$.\footnote{Missing proofs can be found in the appendix of this submission.}
A forest $g \in \mathcal{F}(A)$ is a subforest of a forest $f \in \mathcal{F}(A)$ if there exists a context $c \in \mathcal{C}(A)$ such that $f = cg$. A subtree is a subforest which is a tree, i.e., a subforest of the form $af$ for some $a \in A, f \in \mathcal{F}(A)$.

A forest algebra $(H, V, \cdot, \text{in}_\ell, \text{in}_r)$ is a 5-tuple consisting of:

- A horizontal monoid $(H, +, 0)$
- A vertical monoid $(V, \cdot, 1)$
- A monoid action $\cdot : V \times H \to H$ of $H$ on $V$.
- Two functions $\text{in}_\ell, \text{in}_r : H \to V$ such that $\text{in}_\ell(g) h = g + h$ and $\text{in}_r(g) h = h + g$.

We also write forest algebras as pairs $(H, V)$, and $\text{in}_\ell$ and $\text{in}_r$ are written as $\text{in}_\ell(g) = g + 1$ and $\text{in}_r(g) = 1 + g$. A forest algebra $(H, V)$ is finite if both $H$ and $V$ are finite.

Over an alphabet $A$, the free forest algebra $A^\Delta$ is $((\mathcal{F}(A), +, 0), (\mathcal{C}(A), \cdot, 1), \cdot, \text{in}_\ell, \text{in}_r)$ with $\text{in}_\ell$ and $\text{in}_r$ uniquely determined by the rest. For a forest algebra $(H, V)$ and $h \subseteq H$ we let $h^* = \{ f_1 + \cdots + f_k \mid k \in \mathbb{N}, f_i \in h \}$. Given two forest algebras $(H, V)$ and $(G, W)$, a forest algebra homomorphism $\varphi : (H, V) \to (G, W)$ is a pair $(\alpha, \beta)$ of monoid homomorphisms with $\alpha(v h) = \beta(v) \alpha(h)$, $\beta(\text{in}_\ell(h)) = \text{in}_\ell(\alpha(h))$ and $\beta(\text{in}_r(h)) = \text{in}_r(\alpha(h))$ for all $v \in V$ and $h \in H$.

A forest language over the alphabet $A$ is a subset $L \subseteq \mathcal{F}(A)$. It is recognized by a forest algebra $(H, V)$ if there exists a homomorphism $\varphi : A^\Delta \to (H, V)$ and a set $E \subseteq H$ such that for all forests $f \in \mathcal{F}(A)$ we have $f \in L$ if and only if $\varphi(f) \in E$. A forest language is recognizable if it is recognized by a finite forest algebra.

Bojańczyk and Walukiewicz require the monoid action to be faithful [2], that is, that for all $u, v \in V$ with $u \neq v$, there exists $h \in H$ such that $uh \neq vh$. Delignat-Lavaud and Straubing [3] Remark 3] showed that the class of recognizable forest languages is independent of the additional requirement of the forest algebra to be faithful: for every forest algebra $(H, V)$ there exists a faithful forest algebra $(H, V/\sim_H)$ such that every forest language $L$ recognized by $(H, V)$ is also recognized by $(H, V/\sim_H)$. The definition of the congruence $\sim_H$ on $V$ is given by $u \sim_H v \iff \forall h \in H : uh = vh$ for elements $u, v \in V$. Then, $(H, V/\sim_H)$ is a faithful forest algebra. The natural homomorphism $\pi : (H, V) \to (H, V/\sim_H)$ satisfies $\varphi(L) = \pi(\varphi(L)) \subseteq H$; thus, $L$ is recognized by $(H, V/\sim_H)$. For a given forest algebra $(H, V)$ and given elements $u, v \in V$, one can check in deterministic logarithmic space whether $u \sim_H v$ for $u, v \in V$.

A (deterministic) forest automaton over an alphabet $A$ is a tuple $M = ((Q, +, 0), A, \delta, F)$ such that

- $(Q, +, 0)$ is a finite monoid and its elements are the states,
- $\delta : A \times Q \to Q$ is the transition function and
- $F \subseteq Q$ is the set of accepting states.

The evaluation $f^M$ of a forest $f \in \mathcal{F}(A)$ under the automaton $M$ is inductively defined by

$$0^M = 0$$

$$(af)^M = \delta(a, f^M)$$

$$\left(f_1 + \cdots + f_k\right)^M = f_1^M + \cdots + f_k^M$$

for $a \in A, f \in \mathcal{F}(A)$.

The language accepted by $M$ is $L(M) = \{ f \in \mathcal{F}(A) \mid f^M \in F \}$. It will also be helpful to define $\tilde{\delta} : \mathcal{C}(A) \times Q \to Q$ with $\tilde{\delta}(1, q) = q$, $\tilde{\delta}(f + c, q) = f^M + \tilde{\delta}(c, q)$, $\tilde{\delta}(c + f, q) = \tilde{\delta}(c, q) + f^M$, $\tilde{\delta}(ac, q) = \delta(a, \tilde{\delta}(c, q))$ for all $a \in A, c \in \mathcal{C}(A), f \in \mathcal{F}(A)$.

As [2] show, a forest language is accepted by a forest automaton if and only if it is recognized by a finite forest algebra. Furthermore, the conversion from a forest algebra to a forest automaton is obviously possible using only logarithmic space.

A nondeterministic forest automaton $M = ((Q, +, 0), A, \delta, F)$ is defined like a forest automaton, except with $\delta$ being a function to a set of states, and, accordingly, $0^M = \{ 0 \}$.
\((af)^M = \bigcup_{q \in \mathcal{M}} \delta(a, q)\). \((f_1 + \cdots + f_k)^M = \{q_1 + \cdots + q_k \mid \forall 1 \leq i \leq k: q_i \in f_i^M\}\) for all \(f, f_1, \ldots, f_k \in \mathcal{F}(A), a \in A\) and \(L(M) = \{f \in \mathcal{F}(A) \mid f^M \cap f \neq \emptyset\}\).

We can encode Boolean formula and expressions \(F\) (with \(\top, \bot\) already substituted in for the variables, or with variables from a set \(X\)) as a forest \(\langle F \rangle_F\) over \(\{\wedge, \vee, \neg, \bot, \top\} \cup X\) using:

\[
\begin{align*}
\langle L \land R \rangle_F &= \land(\langle L \rangle_F + \langle R \rangle_F) \\
\langle L \lor R \rangle_F &= \lor(\langle L \rangle_F + \langle R \rangle_F) \\
\langle \neg L \rangle_F &= \neg(\langle L \rangle_F) \\
\langle x \rangle_F &= x \\
\end{align*}
\]

Here \(\top\) and \(\bot\) are meant as representations of true and false, respectively.

Now, let \(\text{True}_F = \{\langle F \rangle_F \mid F\text{ is a true Boolean expression without variables}\}\), which is a recognizable forest language, using a forest algebra whose horizontal elements correspond to pairs of boolean values, with the neutral element being \((\bot, \top)\) and the horizontal operation combining the first component using logical disjunction and the second using conjunction.

### 2.2. Substitutions on Forest Languages

For forests, we define a model of substitutions at the leaves:

**Definition 1.** Let \(\sigma: X \to 2^{\mathcal{F}(A)} \setminus \{\emptyset\}\) with \(X \cap A = \emptyset\). Then we can extend \(\sigma\) to \(\sigma_r\) by

\[
\begin{align*}
\sigma_r: \mathcal{F}(A \cup X) &\to 2^{\mathcal{F}(A)} \\
x &\mapsto \sigma(x) \\
af &\mapsto \{af' \mid f' \in \sigma_r(f)\} \\
g_1 + \cdots + g_k &\mapsto \{g_1' + \cdots + g_k' \mid \forall 1 \leq i \leq k: g_i' \in \sigma_r(g_i)\}
\end{align*}
\]

for all \(x \in X, a \in A, k \in \mathbb{N}\) and all \(f, g_1, \ldots, g_k \in \mathcal{F}(A)\). We call the function \(\sigma_r\) a relational substitution (at the leaves) or sometimes just a substitution. Note that it is a partial function insofar that the value for forests containing variables (i.e., elements of \(X\)) at inner nodes is not defined. Those are also the only forests with undefined values.

We call a substitution \(\sigma\) homomorphic if for all \(x \in X\), we have \(|\sigma(x)| = 1\) and we write \(\sigma \subseteq \theta\) for two substitutions \(\sigma\) and \(\theta\) if, for all \(x \in X\), we have \(\sigma(x) \subseteq \theta(x)\). For forest languages \(L \subseteq \mathcal{F}(A)\) and substitutions \(\sigma: \mathcal{F}(A) \to 2^{\mathcal{F}(A)}\), we write \(\sigma(L) = \bigcup_{f \in L} \sigma(f)\). Additionally, let \(\sigma_r^{-1}(L) = \{f \in \mathcal{F}(A) \mid \sigma_r(f) \cap L \neq \emptyset\} = \bigcup_{l \in L} \{f \in \mathcal{F}(A) \mid l \in \sigma_r(f)\}\).

When relational substitutions are part of the input of an algorithm, we assume the values to be encoded as forest automata or recognizing forest algebras. Note that this especially means that, for problems with given substitutions, we only consider substitutions which map variables to recognizable forest languages.

### 2.3. Complexity Theory Basics

In this paper, we show completeness results for problems in the complexity classes \(P, NP\) and \(EXPTIME\). For \(EXPTIME\)-completeness, we use a characterization in terms of alternating Turing machines: \(\text{APSPACE} = \text{EXPTIME} \[4\]\). For definitions of these classes and an introduction into complexity theory, the reader may refer to \[1\].

### 3. Basic Constructions

First, using the obvious product and complement forest algebra respectively automaton constructions, we get the following result. Here, as in the word case, the complement construction
requires a determinized automaton, but constructions for monotone boolean operations can also be efficiently carried out on nondeterministic forest automata.

**Proposition 2.** The languages recognizable using forest algebras form an effective Boolean algebra. The constructions are possible using only logarithmic space.

This result also holds for deterministic forest automata.

Another useful standard closure property is the closure under inverse homomorphisms:

**Proposition 3.** Let \( L \in \mathcal{F}(B) \) be a forest language recognized by a forest algebra resp. forest automaton and \( \varphi : A^\Delta \to B^\Delta \) be a forest algebra homomorphism. Then we can construct a forest algebra resp. forest automaton for \( \varphi^{-1}(L) \) in logarithmic space.

Also, a nondeterministic forest automaton might be determinized, constructing a deterministic forest automaton:

**Theorem 4.** Given a nondeterministic forest automaton, we can construct a deterministic forest automaton of exponential size accepting the same forest language. This construction is possible in a time polynomial in the size of the output.

In contrast to the case for finite word automata, the construction is not possible using logarithmic space in the output unless \( \text{PSPACE} = \text{EXPTIME} \). This follows from theorem [18] using a similar construction to the \( \text{PSPACE} \)-hardness of universality for nondeterministic finite word automata.

### 3.1. Substitutions

We will now consider constructions for forest languages gotten using substitutions, ways to combine inequalities with substitutions and a result to represent unions succinctly using a substitution.

**Theorem 5.** Given a forest automaton \( M = ((Q, +, 0), \delta, E) \) and a recognizable relational substitution \( \sigma \), we can construct, in polynomial time, a nondeterministic forest automaton \( M' = ((Q', +, 0'), \delta', E') \) such that \( \sigma(L(M)) = L(M') \), and \( |Q'| \) is polynomial.

**Proof.** Let \( M = ((Q, +, 0), \delta, E) \) be a nondeterministic forest automaton and \( \sigma : X \to 2^\mathcal{F}(A) \) a relational substitution with \( M_{\sigma(x)} = ((Q_{\sigma(x)}, +, 0_{\sigma(x)}), \delta_{\sigma(x)}, E_{\sigma(x)}) \) being a nondeterministic forest automaton for every \( x \in X \). Without loss of generality, assume that for all forests \( f \neq 0, 0 \notin f^M \) and \( 0_{\sigma(x)} \notin f^{M_{\sigma(x)}} \) for all \( x \in X \). Then we construct the nondeterministic forest automaton \( M' = ((Q', +, 0'), \delta', E') \) as follows.

The idea of this construction is to let the forest automaton guess nondeterministically for each leaf, whether this is part of a substituted subtree or a direct child, and, at each node in a substituted part, if this is the right-hand side corresponding to a \( x \in X \) and thus should be replaced by that \( x \) (which is marked by a \( s_x \)).

We describe the state set \( Q' \) as a quotient of the free product (in additive notation) of \( S = \{ s_x : x \in X \}^*, \{ 0, \bot \} \) (with \( \bot + \bot = \bot \)), \( Q \) and all sets \( Q_{\sigma(x)} \) for \( x \in X \) by:

- \( s_x + e + p = x^M + p \) if \( e \in E_{\sigma(x)}, p \notin Q_{\sigma(x)} \),
- \( s_x + b + p = \bot \) if \( b \in Q_{\sigma(x)} \setminus E_{\sigma(x)}, p \notin Q_{\sigma(x)} \),
- \( p + q = \bot \) if \( q \in Q_{\sigma(x)}, p \in (Q \cup \bigcup_{y \in X} Q_{\sigma(y)}) \setminus Q_{\sigma(x)} \) and
- \( p + \bot = \bot, \bot + p = \bot \) for all \( p \in Q' \).

It is important that \( 0 \) is contained in \( Q \) and \( Q_{\sigma(x)} \) for all \( x \in X \). Also, reading those equalities as reduction rules from left to right results in a convergent rewriting system. The set of normal forms has polynomial size since all normal forms have length at most 2.
• $\bot$, $q$ with $q \in Q \cup \bigcup_{x \in X} Q_{\sigma(x)}$,
• $s_x + q$ with $q \in Q_{\sigma(x)}$,
• $p + q$ with $p \in \bigcup_{x \in X} Q_{\sigma(x)}$, $q \in Q$.
To simplify the presentation, we define $r : Q' \rightarrow Q \cup \{\bot\}$ with $r(q) = q$ for $q \in Q$, $r(s_x + p) = x^M$ for $p \in E_{\sigma(x)}$ and $r(q') = \bot$ for all other $q' \in Q'$. Note that $r(x + y) = r(x) + r(y)$ if $y$ is not of the form $p + q$ with $p \in Q_{\sigma(x)}$, $q \in Q$, $x \in X$.

Then, we define $\delta'(q, a) = S + D + S$ with $D$ containing all elements of:
• $\delta(q', a)$ if $q' \in r(S + q + S) \setminus \{\bot\}$ and
• $\delta_{\sigma(x)}(q, a)$ if $q \in Q_{\sigma(x)}$.

The accepting states are all $q \in Q'$ with $r(S + q + S) \cap E \neq \emptyset$. Note that $f^{M'} = S + f^{M'} + S$ for all $f \in F(A) \setminus \{0\}$.

Using a somewhat technical induction we can now show:

**Claim 6.** Let $f \in F(A)$ and $g \in F(A \cup X)$. Then, $f \in \sigma(g)$ if, and only if, $g^M \subseteq r(S + f^{M'} + S)$.

It is easy to see that the $Q'$, the operation in $Q'$, the set of accepting states, and all values of $\delta'$ can be computed in polynomial time when represented using normal forms, which then also holds for the nondeterministic forest automaton.

We can also prove a similar result for the inverse application of substitutions, which we state in the following. The significantly simpler proof, nondeterministically guessing the substitution in the transition for an $x \in X$, is omitted for brevity.

**Theorem 7.** Given a forest automaton $M = ((Q, +, 0), \delta, E)$ and a recognizable relational substitution $\sigma$, we can construct a nondeterministic forest automaton $M' = ((Q, +, 0), \delta', E)$ of equal size such that $L(M') = \sigma^{-1}(L(M))$.

Often, it can be useful to consider systems of inequalities instead of single inequalities. This, however, is equivalent for our purposes by the following result, gotten by distinguishing the single inequalities using an added root node.

**Proposition 8.** Given multiple inequalities of the form $\sigma(L_i) \subseteq \sigma(R_i)$ for $i \in I$ and some index set $I$, we can construct a single inequality $\sigma(L) \subseteq \sigma(R)$ that is satisfied by those $\sigma$ that satisfy all $\sigma(L_i) \subseteq \sigma(R_i)$. Forest automata or algebras for $L$ resp. $R$ can be constructed in time $O(\prod_{i \in I} |L_i|)$ resp. $O(\prod_{i \in I} |R_i|)$.

For the hardness results shown later, we will need to represent a union of forest languages succinctly using a substitution. This can be achieved by adding marks to all subforests, distinguishing the languages to be recognized. Those can then be removed using a substitution. This allows to reduce the required state set from the product to essentially the disjoint union of the individual state sets, so we obtain:

**Proposition 9.** Given recognizable forest languages $L_1, \ldots, L_k \subseteq F(A)$ using forest algebras resp. forest automata, we can construct a polynomially-sized homomorphic relational substitution $\sigma$ and a recognizable forest language $L$ using a forest algebra resp. automata such that $L_1 \cup \cdots \cup L_k = \sigma(L)$.

### 3.2. Globally

For some forest languages, the following construction makes the description simpler:

**Definition 10.** For a forest language $L \subseteq F(A)$, the forest languages $G(L)$ contains all forests $f \in L$ such that all subtrees $ag$ of $f$ with $a \in A$ and $g \in F(A)$ satisfy $g \in L$.

The $G$ is a shorthand for “globally”. Note that for $0 \notin L$, we immediately have $G(L) = \emptyset$. 


Proposition 11. If \( L \) is a recognizable forest language, then so is \( G(L) \). Furthermore, the recognizing forest automaton is polynomial in size.

The automaton for \( G(L) \) can be constructed by simply checking that the state of the subforest is accepting for each application of \( \delta \), and transitioning into a failure state otherwise.

4. Algorithms

We will now give efficient algorithms for the equivalence of (deterministic) forest automata and the emptiness of (possibly nondeterministic) forest automata.

4.1. Equivalence of Deterministic Forest Automata

The equivalence problem for deterministic forest automata can be decided using algorithm 1. If the inputs are not equivalent, the algorithm computes a witness.

Algorithm 1: Equivalence test for forest automata

\[
\begin{align*}
L &\leftarrow \{ (0_1, 0_2, 0) \} \\
M &\leftarrow \emptyset \\
\text{while } L \neq \emptyset \text{ do} \\
&\quad \text{Choose } (p_1, p_2, w) \in L \text{ and remove this triple from } L \\
&\quad \text{if } \text{Find}(p_1) \neq \text{Find}(p_2) \text{ then} \\
&\quad \quad \text{return } w \\
&\quad \text{else} \\
&\quad \quad \text{Union}(p_1, p_2) \\
&\quad \quad \text{Add } (p_1, p_2, w) \text{ to } M \\
&\quad \quad \text{for } a \in A \text{ do} \\
&\quad \quad \quad \text{Add } (\delta_1(p_1, a), \delta_2(p_2, a), aw) \text{ to } L \\
&\quad \quad \text{end} \\
&\quad \quad \text{for } (q_1, q_2, u) \in M \text{ do} \\
&\quad \quad \quad \text{Add } (q_1 + p_1, q_2 + p_2, u + w) \text{ and } (p_1 + q_1, p_2 + q_2, w + u) \text{ to } L \\
&\quad \quad \text{end} \\
&\quad \text{end} \\
\text{return } "L(M_1) = L(M_2)"
\end{align*}
\]

Let Find\(_i(p)\) be the result of Find\(_i(p)\) after the \( i \)-th iteration of the outer loop, and Find\(_i(p)\) the result of Find\(_i(p)\) after termination. To show the correctness of the algorithm we start with two claims about the generated partition.

Claim 12. If Find\(_1(p_1) = \text{Find}(p_2)\), Find\(_1(q_1) = \text{Find}(q_2)\) for \( p_1, q_1 \in Q_1, p_2, q_2 \in Q_2 \), then

1. \( \text{Find}(\delta_1(p_1, a)) = \text{Find}(\delta_2(p_2, a)) \) for all \( a \in A \).
2. \( \text{Find}(p_1 + q_1) = \text{Find}(p_2 + q_2) \).

The proof of claim 12 largely proceeds like for Hopcroft-Karp in the word case, however, for the second part, we have to argue along a path of added elements. Let \( L(p) = \{ c \in C(A) \mid \bar{\delta}(p, c) \in E \} \). An induction over the contexts in \( L(p_1) \) resp. \( L(p_2) \) yields:

Claim 13. If \( \text{Find}(p_1) = \text{Find}(p_2) \), then \( L(p_1) = L(p_2) \).
This leads us to the actual result:

**Theorem 14.** If algorithm \( \mathbb{L} \) returns “\( L(M_1) = L(M_2) \)”, then the forest automata \( M_1 \) and \( M_2 \) are equivalent.

**Proof.** Using claim \( \mathbb{L} \) and \( \text{Find}(0_1) = \text{Find}(0_2) \), we obtain \( L(M_1) = L(M_2) \).

**Theorem 15.** Algorithm \( \mathbb{L} \) executes at most
- \((m + n - 1)\) Union operations, and
- \(1 + (m + n - 1) \cdot (|A| + m + n)\) Find operations.

Here, the first bound is obvious from the number of individual elements managed by the Union-Find-data structure and the second bound follows from counting the number of elements added to \( \mathcal{M} \) for each call to Union and evaluating the resulting arithmetic series. Note that, due to the table for the horizontal operation, the size of the automaton is quadratic in the number of states. Thus, using an appropriate Union-Find-data structure, we achieve an almost-linear runtime.

### 4.2. Emptiness for Nondeterministic Forest Automata

Emptiness of the language recognized by a nondeterministic forest automaton can be decided using the simple marking algorithm \( \mathbb{L} \) which determines the reachable states. Note that, since \( \mathcal{M} \) only gets bigger, we can reject an input as soon as an element of \( E \) gets added to \( \mathcal{M} \). This was omitted in the above algorithm for brevity, and since it does not improve our runtime bound.

**Algorithm 2:** Testing emptiness for a nondeterministic forest automaton

\[
\mathcal{M} \leftarrow \{0\}
\]

\[
\text{while } \mathcal{M} \text{ changes do}
\]

\[
\text{for } q \in \mathcal{M} \text{ do}
\]

\[
\text{for } p \in \mathcal{M} \text{ do}
\]

\[
\mathcal{M} \leftarrow \mathcal{M} \cup \{p + q, q + p\}
\]

\[
\text{end}
\]

\[
\text{for } a \in A \text{ do}
\]

\[
\mathcal{M} \leftarrow \mathcal{M} \cup \delta(q, a)
\]

\[
\text{end}
\]

\[
\text{end}
\]

Accept if, and only if, \( \mathcal{M} \cap E = \emptyset \)

**Proposition 16.** Algorithm \( \mathbb{L} \) accepts on the input of a nondeterministic forest automaton \( M = ((Q, +, 0), A, \delta, E) \) if, and only if, \( L(M) = \emptyset \).

This algorithm runs at most \(|H| \cdot (2 \cdot |H| + |A|)\) operations of adding a single element to a set, since the body of the outer for-loop gets executed at most once for each \( h \in H \), and executes at most \( 2 \cdot |H| + |A| \) such operations. Also, since any deterministic forest automaton can be easily converted to a nondeterministic forest automaton, and the problem for deterministic forest automata is P-complete by theorem \( \mathbb{L} \), it is unlikely that a sub-polynomial algorithm exists.
5. Problems Without Substitutions

Before looking at problems using substitutions, it is instructive to look at the problems without any substitution. We will then later use those results directly in reductions or use similar arguments when considering the same problems with an added substitution.

Theorem 17. Given two recognizable forest languages $L$ and $R$, encoded as forest automata, it is $P$-complete to decide:

1. $L = \emptyset$
2. $L \subseteq R$
3. $L = R$

This result also holds if $L$ and $R$ are encoded using recognizing finite forest algebras.

Proof. First, we observe that using Proposition 2, we can reduce the problems onto each other as follows: [4] $\Rightarrow$ [2]: Let $R = \emptyset$. [2] $\Rightarrow$ [2]: Let $L' = L \cup R$ and $R' = L \cap R$. We then have $L' \subseteq R' \iff L \cup R \subseteq L \cap R \iff L = R$. [2] $\Rightarrow$ [2]: by choosing $L' = L \setminus R$. Then $L' = \emptyset \iff L \setminus R = \emptyset \iff L \subseteq R$.

Problem [2] can be solved in polynomial time using algorithm [2] given deterministic forest automata. Problem [2] can even be solved in polynomial time given nondeterministic forest automata using algorithm [2].

Problem [2] is $P$-hard: We proof this by a logspace-reduction from CircuitValueProblem. The CircuitValueProblem asks, given a Boolean circuit and inputs $x_1, \ldots, x_k$, whether the circuit evaluates to $\top$ and was proven to be $P$-complete in [11]. Note that, to prevent confusion with the empty forest or holes, we use $\bot$ and $\top$ as values, instead of the often used 0 and 1.

Let $C$ be a Boolean circuit. Without loss of generality, we assume that the input nodes have already been replaced by their respective values, so instead of variables the respective nodes are now $\top$ or $\bot$. Additionally, we assume all nodes of the boolean circuit to be annotated with their position in the input.

This Boolean circuit can now be described as a sequence $(\beta_1, \ldots, \beta_n)$ where for all $1 \leq i \leq n$, $\beta_i$ is one of $\top, \bot, \land_i(j, k), \lor_i(j, k), \neg_i(j)$ with $j, k < i$.

We then construct a forest algebra $(H, V)$ recognizing $\{(F)_{\infty}\}$, where $F$ is the Boolean formula corresponding to the circuit $C$, with the operators annotated with an index as in the circuit. The encoding is defined analogously to the encoding of Boolean formulas, but preserving the labels. Here, the immediate subformulas are always in the order of the position of their occurrence in the input or, equivalently, their index in the sequence described above. We write $n < n'$ to mean that $n$ comes before $n'$ in that order. Note that, since the conversion from a recognizing forest algebra to a forest automaton is possible using only logarithmic space, this shows the result also for forest automata.

While $F$ may have size exponential in $C$, since the subtrees of gates whose output is used multiple times are duplicated, this is not the case for the forest algebra $(H, V)$.

The elements of horizontal monoid are exactly consecutive sets of nodes in the circuit with a common parent. Consecutive for a set $S$ means that, whenever $n_1, n_3 \in S$, $n_2$ is also a child of every common parent of all elements of $S$, and $n_1 < n_2 < n_3$, then $n_2 \in S$. These sets are thus determined by the parent and the first and last node in the sequence, so there are at most $n^3$ such elements, in addition to the empty set representing a failure state. Since we annotated the nodes to have unique labels, there cannot be any two such elements that coincide, with the exception of equal sets with multiple common parents, which we can easily eliminate by checking if there exists a previous common parent of the same set of nodes.

The elements of the vertical monoid are pairs of elements of the horizontal monoid $(\beta_1, \beta_2)$ with $\beta_1 \neq \beta_2$, corresponding to functions $H \rightarrow H, h_1 \mapsto h_2, h \mapsto \emptyset$ for all $h \in H \setminus \{h_1\}$, and additionally, an identity element $id_H$. 

9
The horizontal operation combines $h_1$ and $h_2$ to $\emptyset$ except when the elements of $h_1$ and $h_2$ all have a common parent, are disjoint and are consecutive, i.e., $h_1 \cup h_2$ is again an element of the horizontal monoid. The vertical operation is function composition.

The forest language $\text{True}_F$ of true Boolean formulas with arbitrarily annotated operators is recognized by the same forest algebra as $\text{True}_F$, by adjusting the homomorphism to ignore the annotations. Now, $\{\langle F \rangle \} \subseteq \text{True}_F$ holds if and only if the Boolean formula, and thus the Boolean circuit, evaluates to $\top$.

**Theorem 18.** Given recognizing finite forest algebras for recognizable forest languages $L_1, \ldots, L_k$ it is EXPTIME-complete to decide whether $L_1 \cap \cdots \cap L_k = \emptyset$.

**Proof.** To decide the problem in exponential time, it suffices to construct the product automaton of the corresponding forest automata and decide emptiness as above.

To show EXPTIME-hardness, using $\text{APSPACE} = \text{EXPTIME}$ \cite{4}, we can equivalently show $\text{APSPACE}$-hardness. Let $L \in \text{APSPACE}$ and $M = (Q, \Gamma, \delta, q_0, g)$ be an alternating turing machine with $L(M) = L$ and a polynomial space bound $p(n)$. Without loss of generality, we assume the following restrictions for $M$:

- In any configuration, there are at most two possible transitions.
- $M$ never moves its head left of the initial position.

Then, on input of a word $w = a_1 \ldots a_n \in \Sigma^*$ we construct forest algebras for multiple languages over the alphabet $\Gamma' = \Gamma \times \{1, \ldots, p(n)\}$, and $\Gamma \times \{1, \ldots, p(n)\} \times Q$. To make those constructions (and the possibility to construct them efficiently) more readable, define the following forest algebra homomorphism from $\Gamma'$:

$$
\begin{align*}
\text{at}_I : \quad & \Gamma^\Delta \to (\Gamma \cup \Gamma \times Q \cup Q)^\Delta \\
& (a_i) 1 \mapsto a1 \\
& (a_j) 1 \mapsto q1 \\
& (a_i) 1 \mapsto (a_j) 1 \\
& (a_i) q \mapsto (a_j) q \\
& \forall i \in I, j \in T
\end{align*}
$$

Now, we can define the languages whose intersection is an accepting computation tree. First, define a language Start of all forests whose upmost level is

\[
\begin{pmatrix}
  a_1 \\
  1 \\
  q_0
\end{pmatrix}
\begin{pmatrix}
  a_2 \\
  2 \\
  n + 1
\end{pmatrix}
\cdots
\begin{pmatrix}
  a_n \\
  n \\
  p(n)
\end{pmatrix}.
\]

Then, we define a language Accept of all forests where the states contained correspond to an accepting computation tree of the alternating turing machine. This can be constructed analogous to the construction for $\text{True}_F$, using a homomorphism mapping all $a1$ where $a$ does not contain a state to $1$, and the states to $\land, \lor, \top$ or $\bot$ according to their type.

Additionally, we define a language Form of all forests with the correct syntactic form, i.e., all levels are

- in the second component, repetitions of $1 \ldots p(n)$
- in the third component, only one state per such repetition.

Then we construct forest languages for checking that we do not modify parts of the tape without a state. Using the closure under inverse homomorphisms with at$_{\{i\}}$ for every $1 \leq i \leq p(n)$, it suffices here to define a single (constant) forest language.

To check the actual transitions, we again use the closure under inverse homomorphisms with at$_{\{i-1,i,i+1\}}$ for every $1 \leq i \leq p(n)$. This can be applied to a language of the
form $G(\bigcup_{d \in \delta} L_d)$, where the languages $L_d$ checks that the transition $d \in \delta$ is applied correctly from the first to the second level, which is easily seen to be recognizable. Note that this also checks that all possible transitions are applied in each state.

Now, if the intersection of these forest languages is empty, then there is no accepting computation tree of $M$ on input $w$, and $w \notin L(M)$. On the other hand, if the intersection is non-empty, it contains (at least) one accepting computation tree of $M$ on input $w$, and $w \in L(M)$.

6. Substitution of Leaves

Saturated substitutions are an important concept for some of the results obtained in the remainder of this section, whenever we want to decide the existence of a substitution.

Definition 19. Let $\sigma$ be a relational substitution and $R$ a forest language recognized by a forest algebra $(H, V)$ with homomorphism $\varphi$ or accepted by a forest automaton $M$. Then we define the saturated substitutions $\hat{\sigma}_{(H, V)}$ and $\hat{\sigma}_M$ as follows:

\[
\hat{\sigma}_{(H, V)}(x) = \{ f \mid \exists f': f' \in \sigma(x), \varphi(f') = \varphi(f) \}
\]
\[
\hat{\sigma}_M(x) = \{ f \mid \exists f': f' \in \sigma(x), f'M = f^M \}. \]

We also write $\hat{\sigma}_R$ instead of either of those. Note that those notions coincide for a forest algebra and the forest automaton constructed from that forest algebra.

The important property of saturated substitutions is the following:

Lemma 20. Let $\sigma$ be a relational substitution and $L, R$ recognizable forest languages. Then, $\sigma(L) \subseteq R$ if and only if $\hat{\sigma}_R(L) \subseteq R$, where $\hat{\sigma}_R$ is the corresponding saturated substitution.

The proof proceeds by induction on the structure of an element of $L$. One important corollary from this is that, if there exists a substitution $\sigma$ with variables $X$ such that $\sigma(L) \subseteq R$, then there also exists a substitution of polynomial size in the length of the encodings of $X$ and $R$ as forest automata or forest algebras, namely a saturated substitution.

6.1. Relational Substitutions

The following theorem forms the basis for all of the complexity upper bounds shown for problems with relational substitutions. As noted earlier, since the problems given here get a substitution as an input, this result only applies to substitutions which restrict the value of a variable to recognizable forest languages.

Theorem 21. Given two recognizable forest languages $L$ and $R$, encoded as forest automata, and a recognizable relational substitution $\sigma$, also encoded using forest automata:

1. It is P-complete to decide if $\sigma(L) \subseteq R$.
2. It is EXPTIME-complete to decide if $\sigma(L) \supseteq R$.
3. It is EXPTIME-complete to decide if $\sigma(L) = R$.
4. It is EXPTIME-complete to decide if $\sigma(L) \subseteq \sigma(R)$.

The same result holds if the inputs are encoded using finite forest algebras.

The problems can be solved using theorem 17, algorithm 2 and theorem 5 with the hardness results following from simple reductions from the problems in theorem 17 respectively theorem 18, in the latter case using proposition 9.

This now allows us to show the main result for problems with relational substitutions. Note that by Lemma 20, a relational substitution fulfilling the inequalities with substitutions only on one side exists if and only if a saturated one exists. This means that the following result holds for the existence of arbitrary — not only of recognizable — relational substitutions.
Theorem 22. Given two recognizable forest languages $L$ and $R$, encoded as forest automata:

1. It is $\text{NP}$-complete to decide if there exists a substitution $\sigma$ such that $\sigma(L) \subseteq R$.
2. It is $\text{EXPTIME}$-complete to decide if there exists a substitution $\sigma$ such that $\sigma(L) = R$.
3. It is undecidable whether there exists a substitution $\sigma$ such that $\sigma(L) = \sigma(R)$.

The same results hold if the inputs are encoded as finite forest algebras.

Those problems may be solved using the results from theorem 21 using lemma 20 to bound their size. Hardness results are trivial reductions from the corresponding problems in theorem 21 using proposition 8. The result for 3 holds already for star-free word languages [10], and systems of equations over unary word languages [7, 12] or the existence of regular solutions [13]. For forest languages, a system of equations over unary languages can be reduced to a single equation over unary forest languages.

7. Summary and Open Problems

We show $\text{P}$-completeness of emptiness, inclusion and equivalence of deterministic forest automata and forest algebras. Each of these problems is $\text{NL}$-complete for word languages [8, Thm. 26]. For proving membership in $\text{P}$ over forest automata, we give efficient algorithms; for instance, we generalize the Hopcroft-Karp equivalence test to deterministic forest automata.

Another basic problem in automata theory is deciding the emptiness of the intersection; here, we show that the problem is $\text{EXPTIME}$-complete for forest automata. The corresponding problem for word languages is $\text{PSPACE}$-complete [9, Lemma 3.2.3].

We also consider the inclusion and the equivalence problems in the presence of (relational) substitutions. Depending on which side of an inclusion we allow substitutions, the problem is either $\text{P}$-complete or $\text{EXPTIME}$-complete. The latter complexity also applies to the equivalence problem. When asking whether there exists a substitution such that some inclusion or some equivalence holds under this substitution, the inclusion problem is $\text{NP}$-complete and the equivalence problem is $\text{EXPTIME}$-complete. If we allow substitutions on both sides, we can encode language equations and, hence, the problem is undecidable.

In future research, it would be interesting to consider the above problems under more expressive models of substitution such as [3] where substitutions can also be applied to inner nodes. Another interesting mode of substitution is obtained from the restriction where the substitution is applied more synchronously: identical variables need to be replaced by identical forests from a given language (rather than by identical forest languages).

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[2] Mikołaj Bojańczyk and Igor Walukiewicz. Forest algebras. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives*, Texts in Logic and Games, pages 107–132. Amsterdam University Press, 2008.

[3] Iovka Boneva, Joachim Niehren, and Momar Sakho. Regular matching and inclusion on compressed tree patterns with context variables. In Carlos Martín-Vide, Alexander Okhotin, and Dana Shapira, editors, *Language and Automata Theory and Applications - 13th International Conference, LATA 2019, St. Petersburg, Russia, March 26-29, 2019, Proceedings*, volume 11417 of *Lecture Notes in Computer Science*, pages 343–355. Springer, 2019.
[4] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. J. ACM, 28(1):114–133, 1981.

[5] John Horton Conway. Regular algebra and finite machines. Chapman and Hall, London, 1971.

[6] Antoine Delignat-Lavaud and Howard Straubing. An automaton model for forest algebras. internship report, August 2010. URL: https://antoine.delignat-lavaud.fr/doc/report-M1.pdf

[7] Artur Jeż and Alexander Okhotin. Equations over sets of integers with addition only. Journal of Computer and System Sciences, 82(6):1007–1019, 2016.

[8] Neil D. Jones. Space-bounded reducibility among combinatorial problems. J. Comput. Syst. Sci., 11(1):68–85, 1975.

[9] Dexter Kozen. Lower bounds for natural proof systems. In 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977, pages 254–266. IEEE Computer Society, 1977.

[10] Michal Kunc. The power of commuting with finite sets of words. Theory Comput. Syst., 40(4):521–551, 2007.

[11] Richard E. Ladner. The Circuit Value Problem is Log Space Complete for P. SIGACT News, 7(1):18–20, January 1975.

[12] Tommi Lehtinen and Alexander Okhotin. On Language Equations XXK = XXL and XM = N over a Unary Alphabet. In Yuan Gao, Hanlin Lu, Shinnosuke Seki, and Sheng Yu, editors, Developments in Language Theory, pages 291–302, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[13] Alexander Okhotin. Strict language inequalities and their decision problems. In Joanna Jedrzejowicz and Andrzej Szepelewski, editors, Mathematical Foundations of Computer Science 2005, pages 708–719, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
A. Details of the Basic Constructions

Most of the details and correctness proofs for the constructions in were omitted in section. In this section, we will give the concrete substitutions and correctness proofs for the results provided here.

A.1. Closure under Boolean Operations

For boolean operations, as in the word case, we consider constructions for monotonic boolean operations and complementation separately, with the latter only possible for deterministic models.

**Proposition 23.** Given recognizable forest languages $L_1$ and $L_2$, the forest languages $L_1 \cup L_2$ and $L_1 \cap L_2$ are effectively recognizable, i.e., we can construct them in logarithmic space, for deterministic and nondeterministic forest automata and forest algebras.

**Proof.** For $i \in \{1, 2\}$, let $L_i$ be forest languages recognized by the forest algebras $(H_i, V_i)$ using the homomorphism $\varphi_i$ and recognizing set $E_i$.

Then the product forest algebra (without faithfulness) $(H_1 \times H_2, V_1 \times V_2)$ with all operations defined element-wise recognizes $L_1 \cap L_2$ and $L_1 \cup L_2$ using the homomorphism $\varphi_1 \times \varphi_2 : x \mapsto (\varphi_1(x), \varphi_2(x))$ and recognizing set $E_1 \times E_2$ resp. $E_1 \times H_2 \cup H_1 \times E_2$.

Analogous constructions are possible for languages accepted by deterministic or nondeterministic forest automata.

**Proposition 24.** Given a recognizable forest language $L$, the forest language $\mathcal{F}(A) \setminus L$ is effectively recognizable, i.e., we can construct it in logarithmic space, for deterministic forest automata and forest algebras.

**Proof.** Let $L$ be forest languages recognized by the forest algebras $(H, V)$ using the homomorphism $\varphi$ and recognizing set $E$.

Then the complement is recognized by the forest algebra $(H_1, V_1)$ using $\varphi$ and recognizing set $H \setminus E$.

An analogous construction is possible for languages accepted by deterministic forest automata.

As a corollary of those two propositions, we now get the one given in the main text:

**Proposition 2.** The languages recognizable using forest algebras form an effective Boolean algebra. The constructions are possible using only logarithmic space. This result also holds for deterministic forest automata.

A.2. Determinization

Nondeterministic forest automata can be determinized to deterministic forest automata analogously to the word case:

**Theorem 4.** Given a nondeterministic forest automaton, we can construct a deterministic forest automaton of exponential size accepting the same forest language. This construction is possible in a time polynomial in the size of the output.
Proof. Let $M = ((Q,+), \delta, E)$ be a nondeterministic forest automaton. Then we can construct $M_{\text{det}} = ((2^Q,+), \delta_{\text{det}}, E_{\text{det}})$ with

$$P_1 + P_2 = \{p_1 + p_2 \mid p_1 \in P_1, p_2 \in P_2\} \text{ for } P_1, P_2 \in 2^Q$$

$$\delta_{\text{det}}(P,a) = \bigcup_{p \in P} \delta(p,a)$$

$$E_{\text{det}} = \{ P \in 2^Q \mid P \cap E \neq \emptyset \}$$

The time bound can be achieved by constructing all states reachable from $\{0\}$ using the above operations until there are no more changes. \hfill \qedsymbol

### A.3. Closure under Inverse Homomorphisms

(Effective) closure under inverse homomorphisms is easy to prove:

**Proposition 3.** Let $L \in \mathcal{F}(B)$ be a forest language recognized by a forest algebra resp. forest automaton and $\varphi \colon A^\Delta \rightarrow B^\Delta$ be a forest algebra homomorphism. Then we can construct a forest algebra resp. forest automaton for $\varphi^{-1}(L)$ in logarithmic space.

**Proof.** Given a recognizing forest algebra, we can simply compose the two homomorphisms. For forest automata, we can compute a new $\delta'$ using $\delta'(a,x) = \delta(\varphi(a),x)$. Both of those operations can be carried out element-by-element on the respective function tables. \hfill \qedsymbol

### A.4. Substitutions

In the proof of theorem B we omitted the somewhat technical proof of correctness, which we will now give here:

**Claim 6.** Let $f \in \mathcal{F}(A)$ and $g \in \mathcal{F}(A \cup X)$. Then, $f \in \sigma(g)$ if, and only if, $g^M \subseteq r(S+f^M+S)$.

**Proof of the claim.** Now, we first show by induction on $g$ that, if $f \in \sigma(g)$, then $g^M \subseteq r(S+f^M+S)$. For $g = 0$, we immediately have $g^M = 0^M = \{0\} \subseteq r(S+0^M+S) = r(0+S+0+S) = r(S) \text{ since } 0 \in S$. For $g = x \in X$ we have $x^M \subseteq r(s_x + E_{\sigma(x)}(x)) \subseteq r(S+f_{\sigma(x)}^M + S) \subseteq r(S+f^M+S)$ since $M_{\sigma(x)}$ can be embedded into $Q'$ and $\delta_{\sigma(x)}(q,a) \in \delta(q,a)$ for all $q \in Q_{\sigma(x)}, a \in A$. For $g = aq' \in A$, we have $f = af'$ with some $f' \in \sigma(g')$. By induction hypothesis, $g^M \subseteq r(S+f^M+S)$. Then, $(af')^M = \delta(f' a^M, a) = (af)^M$ since $a^M \subseteq r(S+f^M+S) \setminus \{\bot\}$. For $g = g_1 + g_2$ with $g_1, g_2 \in \mathcal{F}(A) \setminus \{0\}$, we inductively have $f_i \in \sigma(g_i)$ with $g_i^M \subseteq r(S+f_i^M+S)$ for $i \in \{1,2\}$. Thus, $g^M = g_1^M + g_2^M \subseteq r(S+f_1^M+S) + r(S+f_2^M+S) = r(S+f_1^M+S + f_2^M + S) = r(S+f_1^M + f_2^M + S) \text{ where the last step is easily seen by considering separately the cases where } f_j = 0 \text{ for one or both } j \in \{0,1\}$.

On the other hand, show that, if $q \in r(S+f^M+S) \setminus \{\bot\}$, there exists a forest $g$ with $q \in g^M$ and $f \in \sigma(g)$. If $f = 0$, we have $0 \in f^M$, so $q \in r(S)$ and $q = r(s_{x_1} + \cdots + s_{x_k}) = r(s_{x_1}) + \cdots + r(s_{x_k}) = x_1 + \cdots + x_k = (x_1 + \cdots + x_k)^M$ for $x_1, \ldots, x_k \in X, k \in \mathbb{N}$ with $0 \in E_{\sigma(x_i)}$ for all $1 \leq i \leq k$, so $0 \in \sigma(x_1 + \cdots + x_k)$. Now assume $f \neq 0$. Then we have $S + f^M + S = f^M$ and $q \in r(f^M) \setminus \{\bot\}$. We can then write $f = f_1 + \cdots + f_\ell$ and $q = r(q_1 + \cdots + q_\ell)$ for some $\ell > 0$ such that, for all $1 \leq i \leq \ell$, $q_i \in f_i^M$ and $q_i$ is either from $s_x + E_{\sigma(x)}$ for some $x \in X$ or $q_i = \delta(q_i', a)$ for some $a \in A, q_i' \in r(S+f_i^M+S)$. In the former case, we have $f_i \in \sigma(x)$, and choose $g_i = x$. In the latter case, we inductively get a $g_i'$ such that $(g_i')^M = q_i'$ and $f_i' \in \sigma(g_i')$, and choose $g_i = a g_i'$. Then, using $g = g_1 + \cdots + g_\ell$, we have $g^M = q$ and $f = f_1 + \cdots + f_\ell \in \sigma(g_1) + \cdots + \sigma(g_\ell) = \sigma(g_1 + \cdots + g_\ell) = \sigma(g)$. \hfill \qedsymbol
A.5. Inverse Substitutions

We will now give the concrete construction for a nondeterministic forest automaton recognizing the result of inversely applying a relational substitution.

**Theorem 7.** Given a forest automaton $M = ((Q, +, 0), \delta, E)$ and a recognizable relational substitution $\sigma$, we can construct a nondeterministic forest automaton $M' = ((Q, +, 0), \delta', E)$ of equal size such that $L(M') = \sigma^{-1}(L(M))$.

**Proof.** Let $M = ((Q, +, 0), \delta, E)$ be a nondeterministic forest automaton and $\sigma : X \rightarrow 2^F(A)$ a relational substitution with $M_{\sigma(x)} = ((Q_{\sigma(x)}, +, 0_{\sigma(x)}), \delta_{\sigma(x)}, E_{\sigma(x)})$ being a nondeterministic forest automaton for every $x \in X$. Then, we construct the nondeterministic forest automaton $M' = ((Q, +, 0), \delta', E)$ with

$$\delta' : Q \times A \rightarrow 2^Q$$

$q, a \mapsto \delta(q, a) \quad \forall q \in Q, a \in A \setminus X$

$$0, x \mapsto \bigcup \{ f^M \mid f^{M_{\sigma(x)}} \in E_{\sigma(x)} \} \quad \forall x \in X$$

This nondeterministically chooses the substitute whenever reading a variable. By induction over the forest structure, one can easily show that $f^{M'}$ contains an accepting state if, and only if, $M$ accepts an element of $\sigma_r(f)$, i.e., if $f \in \sigma_r^{-1}(L(M))$. \hfill $\square$

A.6. Globally

A forest automaton for $G(L)$ given some recognizable forest language $L$ can be constructed as follows:

**Proposition 11.** If $L$ is a recognizable forest language, then so is $G(L)$. Furthermore, the recognizing forest automaton is polynomial in size.

**Proof.** Given a forest automaton $M = ((Q, +, 0), A, \delta, E)$ for $L$, we add a state $\bot$ with $\bot + x = x + \bot = \bot$ for all states $x$ and modify $\delta$ to $\delta'(q, a) = \delta(q, a)$ if $q \in E$ and $\delta(q, a) = \bot$ otherwise, resulting in a forest automaton $M'$.

Then, if a forest $f$ is accepted by this automaton, then $f \in L$, since all transitions that do not lead to a result of $\bot$ were also possible in the automaton for $L$. Additionally, since $f^{M'} \neq \bot$, this holds also for each subtree. Thus, for each subtree $a f'$ of $f$ with $a \in A, f' \in F(A)$, we have $(a f')^{M'} = \delta'(f^{M'}, a) \neq \bot$, and thus $f^{M'} \in E$ and $f' \in L$.

On the other hand, if a forest $f$ is in $G(L)$, we can show that $f^{M'} = f^M \in E$. First observe that if $f \in G(L)$, then $f' \in G(L)$ for all subtrees $a f'$ of $f$ with $a \in A, f' \in F(A)$. Now the proof proceeds by induction on $f$. Let $f = a_1 f_1 + \cdots + a_k f_k$ for forests $f_1, \ldots, f_k \in F(A)$. Then, we have $f_i \in G(L)$ for each $1 \leq i \leq k$ and inductively $f_i^{M'} \in E$, so $f_i^{M'}$. Given that $f^{M'} = \delta'(f^{M'}, a) + \cdots + \delta'(f_k^{M'}, a) = \delta'(f_1^{M'}, a) + \cdots + \delta'(f_k^{M'}, a) = f^M \in E$. \hfill $\square$

A.7. Combining Inequalities

We obtain the result that allows us to combine multiple inequalities using the following construction.

**Proposition 8.** Given multiple inequalities of the form $\sigma(L_i) \subseteq \sigma(R_i)$ for $i \in I$ and some index set $I$, we can construct a single inequality $\sigma(L) \subseteq \sigma(R)$ that is satisfied by those $\sigma$ that satisfy all $\sigma(L_i) \subseteq \sigma(R_i)$. Forest automata or algebras for $L$ resp. $R$ can be constructed in time $O(\prod_{i \in I} |L_i|)$ resp. $O(\prod_{i \in I} |R_i|)$.
Proof. We can construct $L$ and $R$ as follows:

\[ L = \{i\ell_i \mid i \in I, \ell_i \in L_i\} \quad R = \{ir_i \mid i \in I, r_i \in R_i\} \]

The construction in the given time bound is easily possible as a simple extension of the product automaton or algebra. \hfill \Box

A.8. Using Substitutions to Represent Unions

One construction central to the EXPTIME-hardness results was that we can use substitutions to represent unions of forest languages succinctly:

**Proposition 9.** Given recognizable forest languages $L_1, \ldots, L_k \subseteq F(A)$ using forest algebras resp. forest automata, we can construct a polynomially-sized homomorphic relational substitution $\sigma$ and a recognizable forest language $L$ using a forest algebra resp. automata such that $L_1 \cup \cdots \cup L_k = \sigma(L)$.

**Proof.** Assume forest automata $M_i = ((Q_i, +, 0_i), \delta_i, E_i)$ for $1 \leq i \leq k$ and the $Q_i$ pairwise disjoint. Then we can construct $M = ((Q, +, 0), \delta, E)$ with

\[
\begin{align*}
Q &= \{0, \bot\} \cup \bigcup_{1 \leq i \leq k} Q_i \times \{i, i'\} \\
E &= \bigcup_{1 \leq i \leq k} E_i \times \{i\} \\
(p, i) + (q, j) &= (p + q, i) & i \in \{1, \ldots, k\}, j \in \{i, i'\} \\
(p, i') + (q, j) &= (p + q, i') & i \in \{1, \ldots, k\}, j \in \{i, i'\} \\
0 + x &= x + 0 = x & x \in Q \\
x + y &= \bot & \text{unless defined otherwise} \\
\delta((q, i'), a) &= (\delta_i(q, a), i) & i \in \{1, \ldots, k\} \\
\delta(0, m_i) &= (0, i', i')
\end{align*}
\]

This expects a marker for the appropriate $L_i$ as the leftmost child of every node in the forest.

The forest language recognized by this forest automaton is mapped to $L_1 \cup \cdots \cup L_k$ using the substitution $\sigma: \{m_1, \ldots, m_k\} \mapsto A$ with $\sigma(m_i) = \{0\}$.

For the construction on forest algebras, the main observation is that any context that does not contain any of $m_1, \ldots, m_k$ does either have the form 1 or $a1$ for some $a \in A$ or corresponds to the constant $\bot$ function. \hfill \Box

B. Omitted Proofs for Algorithms

Earlier, we gave an algorithm for the equivalence of deterministic forest automata and the emptiness of nondeterministic forest automata. Here, we will expand on some parts of the correctness and runtime proofs.

B.1. Equivalence of Deterministic Automata

For the equivalence test on deterministic forest automata, we will now expand on the details of the correctness proof, and show the stated runtime bound. In the correctness proof of algorithm 1, we omitted the proof of the following two claims:

**Claim 12.** If $\text{Find}(p_1) = \text{Find}(p_2)$, $\text{Find}(q_1) = \text{Find}(q_2)$ for $p_1, q_1 \in Q_1, p_2, q_2 \in Q_2$, then

1. $\text{Find}(\delta_1(p_1, a)) = \text{Find}(\delta(p_2, a))$ for all $a \in A$. 

17
2. Find\((p_1 + q_1) = Find(p_2 + q_2)\).

**Proof of the claim.** Assume the contrary and choose \(i\) minimal such that there exist \(p_1, q_1 \in Q_1, p_2, q_2 \in Q_2, a \in A\) with \(Find_i(p_1) = Find_i(q_1), Find_i(p_2) = Find_i(q_2)\) and

1. \(Find_i(\delta_1(p_1, a)) \neq Find_i(\delta_2(p_2, a))\). Since \(i\) is minimal, \(Find_{i-1}(p_1) \neq Find_{i-1}(p_2)\).

Thus, \(Find_i(p_1) = Find_i(p_2)\) is due to a call to Union in the \(i\)-th iteration. Then the triple \((p_1, p_2, u)\) for some \(u\) was added to \(L\). Since we eventually have \(Find(z_1) = Find(z_2)\) for all \((z_1, z_2, w)\) added to \(L\), this is a contradiction.

2. \(Find_i(p_1 + q_1) \neq Find_i(p_2 + q_2)\). Since \(i\) is minimal, \(Find_{i-1}(p_1) \neq Find_{i-1}(p_2)\) or \(Find_{i-1}(q_1) \neq Find_{i-1}(q_2)\).

First, assume \(Find_{i-1}(p_1) \neq Find_{i-1}(q_2)\). Now, there exist \((z_0, z_1, u_1)\), \((z_2, z_3, u_2)\), \ldots, \((z_{k-1}, z_k, u_k)\) in \(M\) with \(z_0 = q_1\) and \(z_k = q_2\). Then, we add triples \((p_1 + z_i, p_1 + z_i + 1, u + u_i)\) resp. \((p_2 + z_i, p_1 + z_i + 1, u + u_i)\) to \(L\). Thus, we eventually have: \(Find(p_1 + q_1) = Find(p_1 + z_0) = Find(p_2 + z_1) = \cdots = Find(p_2 + z_k) = Find(p_2 + q_2)\), which is a contradiction. The case for \(Find_{i-1}(q_1) \neq Find_{i-1}(q_2)\) is symmetric.

**Claim 13.** If \(Find(p_1) = Find(p_2)\), then \(L(p_1) = L(p_2)\).

**Proof of the claim.** Show \(c \in L(p_1) \iff c \in L(p_2)\) by induction on the size of \(c\). For \(c = 1\), the result is immediately obvious.

Assume \(c = c'(a)\) for \(c' \in C(A)\) and \(a \in A\). By claim [12], we have \(Find(\delta_1(p_1, a)) = Find(\delta_2(p_2, a))\) and thus, by induction hypothesis, \(c' \in L(\delta_1(p_1, a)) \iff c' \in L(\delta_2(p_2, a))\).

Now let \(c = c'(f + 1)\) or \(c = c'(1 + f)\) for some \(c' \in C(A)\) and \(f \in F(a) \setminus \{0\}\). Then, \(Find(f^{M_1}) = Find(f^{M_2})\) and thus, \(Find(f^{M_1} + p_1) = Find(f^{M_2} + p_2)\) by claim 1. Using the induction hypothesis and rewriting concludes the proof.

Also, we only stated the runtime bound without proof:

**Theorem 15.** Algorithm 7 executes at most

- \((m + n - 1)\) Union operations, and
- \(1 + (m + n - 1) \cdot (|A| + m + n)\) Find operations.

**Proof.** The first bound is obvious because we only unite disjoint subsets.

For the second bound, we first note that every Find is due to some triple in \(L\). After the Union, we call add \(|A| + 2M\) elements to \(L\), where \(M\) is the current number of elements in \(M\). In total, this leads to \(f\) additions to \(L\) where

\[
\begin{align*}
f &= \sum_{i=1}^\ell |A| + 2M_i \\
&\leq \sum_{i=1}^\ell |A| + 2i \\
&\leq \sum_{i=1}^{n+m-1} |A| + 2i \\
&= (n + m - 1) \cdot |A| + (n + m - 1) \cdot (n + m) \\
&= (n + m - 1) \cdot (|A| + n + m)
\end{align*}
\]

when \(M_i\) is the size of \(M\) after the \(i\)-th call to Union and \(\ell\) is the total number of such calls. Together with the initial element, we get \(1 + f = 1 + (n + m - 1) \cdot (|A| + n + m)\) calls to Find.

**B.2. Emptiness for Nondeterministic Forest Automata**

For emptiness of nondeterministic automata, we omitted the correctness proof, which we will now give here.

**Proposition 16.** Algorithm 2 accepts on the input of a nondeterministic forest automaton \(M = ((Q, +, 0), A, \delta, E)\) if, and only if, \(L(M) = \emptyset\).
Proof. If the algorithm does not accept an input, then after its execution, there exists some \( e \in \mathcal{M} \cap E \). By straightforward induction over the execution of the algorithm, it is easily seen that for each \( m \in \mathcal{M} \), there exists a forest \( f \in \mathcal{F}(A) \) such that \( m \in f^M \). This also holds for \( e \), so there exists \( f \in \mathcal{F}(A) \) with \( f^M \cap E \neq \emptyset \) and \( f \in L(M) \).

On the other hand, assume there exists some forest \( f \in \mathcal{F}(A) \) and some \( q \in Q \) with \( q \in f^M \). By induction, show that \( q \in \mathcal{M} \) after the execution of the algorithm. If \( f = 0 \), this is obvious. If \( f = f_1 + f_2 \) for two smaller forests \( f_1 \in \mathcal{F}(A) \) and \( f_2 \in \mathcal{F}(A) \), then \( q = q_1 + q_2 \) for some \( q_1 \in f_1^M \) and \( q_2 \in f_2^M \). Inductively, \( q_1, q_2 \in \mathcal{M} \). In the first iteration before which both \( q_1 \in \mathcal{M} \) and \( q_2 \in \mathcal{M} \), we then add \( \{q_1 + q_2, q_2 + q_1\} \) to \( \mathcal{M} \), and thus \( q = q_1 + q_2 \in \mathcal{M} \). If \( f = af' \) for some \( a \in A, f' \in \mathcal{F}(A) \), we have \( q = \delta(q', a) \) for some \( q' \in f'^M \). Inductively, \( q' \in \mathcal{M} \). In the first iteration before which \( q' \in \mathcal{M} \), we add all elements of \( \delta(q', a) \) to \( \mathcal{M} \), thus also \( q = \delta(q', a) \).

\( \square \)

C. Proof of the Saturation Lemma

In this section, we show the following — somewhat technical — lemma for inclusion and saturated substitutions that allows us to limit the size of substitutions that fulfill some inequality with a substitution on only one of the sides.

Lemma 20. Let \( \sigma \) be a relational substitution and \( L, R \) recognizable forest languages. Then, \( \sigma(L) \subseteq R \) if and only if \( \bar{\sigma}(L) \subseteq R \), where \( \bar{\sigma} \) is the corresponding saturated substitution.

Proof. First, we note that, by definition of \( \bar{\sigma} \), we have \( \sigma(L) \subseteq \bar{\sigma}(L) \) and thus, the direction from right to left is immediately obvious.

Let \( L \) be recognized by a finite forest algebra \((H, V)\) using a homomorphism \( \varphi \) and recognizing set \( E \), and let \( \varphi(L) \subseteq R \). Now, for every \( x \in X \) and \( r \in \varphi(\sigma(x)) \), we choose a representative \( s_{x, r} \in \sigma(x) \) with \( \varphi(s_{x, r}) = r \). This exists by the definition of \( \bar{\sigma} \).

We now show for every \( f_L \in \mathcal{F}(A \cup X) \) with variables only occurring at the leaves and every \( \hat{f}_R \in \bar{\sigma}_R(f_L) \), there exists \( f_R \in \mathcal{F}(A) \) such that \( f_R \in \sigma(f_L) \) and \( \varphi(\hat{f}_R) = \varphi(f_R) \). For 0 this is immediately obvious. For the case \( f_L = x \in X \), we can choose \( f_R = s_{x, r} \) with \( r = \varphi(\hat{f}_R) \).

Now, if \( f_L = af'_L \) for some \( f'_L \in \mathcal{F}(A \cup X) \) with variables only at the leaves, we have \( \hat{f}_R = a\hat{f}'_R \) for some \( \hat{f}'_R \in \bar{\sigma}(f'_L) \) by the definition of relational substitutions. Inductively, there exists \( f_R \in \sigma(f_L) \) with \( \varphi(\hat{f}_R) = \varphi(f_R) \), thus now \( \varphi(\hat{f}_R) = \varphi(a\hat{f}'_R) = \varphi(a\hat{f}_R) = \varphi(f_R) \) with \( f_R = af_R \).

In the case \( f_L = f_{L1} + \cdots + f_{Lk} \) with forests \( f_{L1}, \cdots, f_{Lk} \in \mathcal{F}(A \cup X) \), \( k \in \mathbb{N} \), there again exist \( \hat{f}_R \) for \( 1 \leq i \leq k \) and \( \hat{f}_R = \hat{f}_{R1} + \cdots + \hat{f}_{Rk} \). Thus, inductively there exist \( f_{R1} \) with \( \varphi(\hat{f}_{R1}) = \varphi(f_{R1}) \) for \( 1 \leq i \leq k \) and thus with \( f_R = f_{R1} + \cdots + f_{Rk} \), we get \( \varphi(\hat{f}_R) = \varphi(\hat{f}_{R1}) + \cdots + \varphi(\hat{f}_{Rk}) = \varphi(f_{R1}) + \cdots + \varphi(f_{Rk}) = \varphi(f_R) \).

Now, assume there exists \( f_R \in \sigma(L) \). Then, \( \hat{f}_R \in \bar{\sigma}(f_L) \) for some \( f_L \in L \), and thus there exists also \( f_R \in \sigma(f_L) \) with \( \varphi(\hat{f}_R) = \varphi(f_R) \). Using this and \( \hat{f}_R \notin R \), we can now conclude \( f_R \in \sigma(L) \). \( \square \)

D. Proofs for Problems With Relational Substitutions

Now, we will show how the other results of the paper can be put together to obtain the completeness results in section 6.3.

Theorem 21. Given two recognizable forest languages \( L \) and \( R \), encoded as forest automata, and a recognizable relational substitution \( \sigma \), also encoded using forest automata:

1. It is \( \mathsf{P} \)-complete to decide if \( \sigma(L) \subseteq R \).
2. It is \( \mathsf{EXPTIME} \)-complete to decide if \( \sigma(L) \supseteq R \).
3. It is \( \mathsf{EXPTIME} \)-complete to decide if \( \sigma(L) = R \).
4. It is EXPTIME-complete to decide if \( \sigma(L) \subseteq \sigma(R) \).

The same result holds if the inputs are encoded using finite forest algebras.

Proof. We get P-hardness of \( \text{I} \) by reduction from the corresponding problems in Theorem \( \text{I} \) choosing \( X = \emptyset \), and solve problem \( \text{I} \) directly using Theorem \( \text{I} \) and then deciding \( \sigma(L) \cap \overline{R} = \emptyset \) using algorithm \( \text{I} \).

Problem \( \text{I} \) is EXPTIME-hard by reduction from Theorem \( \text{I} \). Since emptiness of intersection is equivalent to the universality of a union using the closure under complement, and we can denote the union using a substitution by Proposition \( \text{I} \). Since \( \sigma(L) \subseteq R \) holds trivially in this construction, this also shows that \( \text{I} \) is EXPTIME-hard. Problem \( \text{I} \) immediately follows by reduction from \( \text{I} \) since \( \sigma(L) = L \) for \( L \subseteq F(A) \) (that is, not containing any \( x \in X \)).

To solve \( \text{I} \) or \( \text{I} \) in exponential time, we can simply construct a NFA as in Theorem \( \text{I} \) do a powerset construction (Theorem \( \text{I} \) and then check the subset relation like in Theorem \( \text{I} \).

Problem \( \text{I} \) is solvable in exponential time by solving problems \( \text{I} \) and \( \text{I} \) as above. \( \square \)

Theorem 22. Given two recognizable forest languages \( L \) and \( R \), encoded as forest automata:

1. It is NP-complete to decide if there exists a substitution \( \sigma \) such that \( \sigma(L) \subseteq R \).
2. It is EXPTIME-complete to decide if there exists a substitution \( \sigma \) such that \( \sigma(L) = R \).
3. It is undecidable whether there exists a substitution \( \sigma \) such that \( \sigma(L) = \sigma(R) \).

The same results hold if the inputs are encoded as finite forest algebras.

Proof. To prove that the problems are in NP resp. APSPACE, we can simply guess a saturated substitution and check if the respective inequality holds using the respective algorithm from Theorem \( \text{I} \). Note that the saturated substitutions have right hand sides that are recognized by forest automata, have polynomial size in \( L \) and \( R \), and every substitution satisfying the inequality can be converted to a saturated one using Lemma \( \text{I} \). This also holds for problem \( \text{I} \) since \( \sigma(L) = R \) and \( \sigma(L) \subseteq \sigma(L) \subseteq R \) implies \( \hat{\sigma}(L) = R \).

We first show that problem \( \text{I} \) is NP-hard by reduction from Satisfiability. Given a Boolean formula \( F \), we construct its encoding as a forest \( (F)_F \) and a recognizing forest algebra for \( L = \{ (F)_F \} \). As noted above, this also shows the result for the problem with forest automata as input, since the forest algebra may be converted to a forest automaton using only logarithmic space. If given a model \( A \) for \( F \) we can construct \( \sigma : x \mapsto \{ A(x) \} \) which satisfies \( \sigma(L) \subseteq \text{True}_F \). Given a substitution \( \sigma(L) \subseteq \text{True}_F \), there exists a homomorphic substitution \( \sigma' \) such that \( \sigma'(x) \subseteq \sigma(x) \). For this homomorphic substitution, the variables are substituted consistently, so this substitution \( \sigma' \) also defines an assignment, which is a model for \( F \) by the definition of \( \text{True}_F \).

We can show EXPTIME-hardness of \( \text{I} \) by reduction from the problem with a given substitution in Theorem \( \text{I} \) by using proposition \( \text{I} \) to force the substitution to be the given one. Note that the resulting forest algebras or automata can be constructed to be of polynomial size, since the (homomorphic) substitution used in the hardness proof substitutes all \( x \in X \) by the same forest language.

One way of showing that \( \text{I} \) is undecidable already for unary forest languages is by reduction from the problem given in Theorem \( \text{I} \) which asks, given \( C, D, E, F \subseteq \mathbb{N} \), whether there exists \( X \subseteq \mathbb{N} \) such that \( X + X + C = X + X + D \) and \( X + E = F \). Identifying sets of natural numbers with subsets of \( \{ a \}^* \subseteq F(A) \), this is the case if and only if there exists a relational substitution \( \sigma : \{ x \} \rightarrow F(A) \) such that \( \sigma_r(a(x + x + C) + a(x + E)) = \sigma_r(a(x + x + D) + aF) \).

If we do not restrict the problem to unary languages, all results are generalizations of the respective problem for word languages \( \text{I} \). \( \square \)