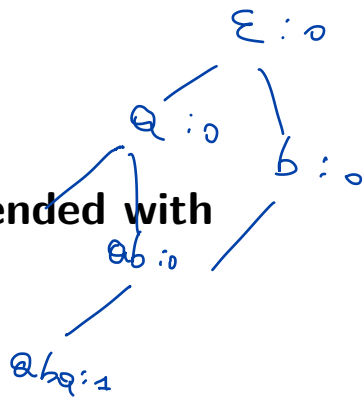


$$\underline{aba + abb}$$



# On polynomial grammars extended with substitution

Janusz Schmude ☒

Institute of Informatics,  
University of Warsaw, Poland

## Abstract

We investigate decidability of equivalence of register transducers, also called copyful Streaming String Transducers in case of string input, extended with an operation of substituting a register for all occurrences of a given letter in another register. We reduce to zeroness of polynomial grammars (over ring of polynomials) extended with analogous substitution operation by encoding strings into polynomials; a similar method was used successfully by Seidl et al. in 2018. We give two restrictions under which register transducers with substitution have decidable equivalence. They seem to be very restrictive but on the other hand, they seem to be on the edge of the scope of this “polynomial” method, as in the third result we give a rather restricted model of polynomial grammars with substitution that has undecidable equivalence.

**2012 ACM Subject Classification** Theory of computation → Formal languages and automata

**Keywords and phrases** equivalence problem, register transducers, word substitution, polynomial grammar

**Funding** Supported by ERC Consolidator Grant LIPA, agreement no. 683080.

**Acknowledgements** I would like to thank Mikołaj Bojańczyk for the support in obtaining the results and writing this paper. I would also like to thank James Worrell, Mahsa Shirmohammadi, Sandra Kiefer, Sławomir Lasota and Radosław Piórkowski for helpful discussions.

## 1 Introduction

Let us consider the following computation device that computes string-to-string functions: it has two registers  $R, S$  that store strings and reads input words from left to right. It initializes the registers to empty words and if the read letter is  $a$ , it updates the registers by putting simultaneously

$$\begin{cases} R := a \cdot R, \\ S := \cancel{R} \cdot a. \end{cases}$$

After input word is read, it outputs  $R \cdot S$ . For example, a run over input word  $w = abb$  is

$$\begin{array}{ccccccc} R = \varepsilon & \xrightarrow{a} & R = a & \xrightarrow{b} & R = ba & \xrightarrow{b} & R = bba \\ S = \varepsilon & & S = a & & S = ab & & S = abb \end{array} \rightarrow \text{output } bbaabb.$$

This device computes function  $w \mapsto \text{rev}(w) \cdot w$ , where  $\text{rev}$  is the reverse function. Such a device is an instance of a *register transducer* [8] (in general, register transducers can compute tree-to-string functions, however in all the examples we give string-to-string functions, which are a special case where input trees are monadic). Sometimes two register transducers defined in two different ways compute the same function – the *equivalence problem* asks, given two register transducers, if this is the case. For example, the register transducers that compute, respectively, the reverse and identity function are equivalent if and only if the alphabet is unary.

There are several methods of deciding equivalence of register transducers e.g. by use of an equivalent model of two-way transducers [11] or by reduction to non-emptiness of one

counter automata in case of *copyless* register transducers, which are called *Streaming String Transducers* [1]. Method that is close to one we use in this paper is by reduction to *zerness problem* for *polynomial grammars* [8].

Let us describe a variant of polynomial grammars that uses a free monoid instead of a ring and explain how it is related to equivalence problem of register transducers. It is a generalisation of context-free grammar, in which each nonterminal is associated with some dimension  $k$ , and therefore outputs  $k$ -tuples, and right-hand sides of production rules may use function symbols to avoid introducing independent copies of a nonterminal. For example, consider a polynomial grammar over  $(\Sigma^*, \cdot)$  with one nonterminal  $X$  of dimension 2 and production rules

$$\begin{aligned} X &\rightarrow p_a(X), a \in \Sigma \mid (\varepsilon, \varepsilon), \text{ where} \\ p_a &: (\Sigma^*)^2 \rightarrow (\Sigma^*)^2 \\ p_a(w_1, w_2) &= (aw_1, w_2a) \text{ for } a \in \Sigma, w_1, w_2 \in \Sigma^*. \end{aligned}$$

By definition, its language consists of pairs  $p_{a_n}(p_{a_{n-1}}(\dots(p_{a_2}(p_{a_1}(\varepsilon, \varepsilon))\dots))$  for  $a_1, \dots, a_n \in \Sigma$ . We define the *grammar equivalence problem*, in the case the initial nonterminal is 2-dimensional, as follows: is the first coordinate equal to the second coordinate for every produced pair? Observe that equivalence holds for the mentioned grammar if and only if the register transducers that compute, respectively, reverse and identity function, are equivalent.

Register transducers considered in this paper are enriched with a substitution operation, we call them register transducers with substitution. For example, consider the function  $\text{sqrev} : \Sigma^* \rightarrow (\Sigma \cup \{\#\})^*$ , called *squared reverse* (the definition and name are motivated by *iterated reverse* [6]), defined as follows:

$$\text{sqrev} : w \mapsto (\# \cdot \text{rev}(w))^{|w|}.$$

It can be computed by two following register transducers with substitution that initialize their registers on empty strings, output register  $R$ , and have register updates as follows:

$$\begin{cases} R := \# \cdot a \cdot S \cdot (R[\# := \# \cdot a]), \\ S := a \cdot S, \end{cases} \quad \text{for } a \in \Sigma$$

and

$$\begin{cases} R := (R[\# := \# \cdot a]) \cdot \# \cdot a \cdot S, \\ S := a \cdot S, \end{cases} \quad \text{for } a \in \Sigma.$$

To decide their equivalence, one may consider a polynomial grammar with substitution (again, over free monoid instead of a ring) with initial nonterminal  $Y$  of dimension 4:  $Y \rightarrow q_a(Y), a \in \Sigma \mid (\varepsilon, \varepsilon, \varepsilon, \varepsilon)$ , where  $q_a(w_1, w_2, w_3, w_4) = (\# \cdot a \cdot w_2 \cdot (w_1[\# := \# \cdot a]), a \cdot w_2, (w_3[\# := \# \cdot a]) \cdot \# \cdot a \cdot w_4, a \cdot w_4)$  and test for equivalence the grammar  $S_1 \rightarrow \pi_{[1,3]}(Y)$ , where  $\pi_I$  is projection on coordinates from  $I$ .

We are interested in decidability of grammar equivalence for such grammars.

We use the Hilbert Method [8], which relies on encoding objects manipulated by transducers, like strings or trees, into rings e.g. of integers or of polynomials; this is done in order to use algebraic geometry machinery. This method has been used to prove decidability of equivalence for register transducers without substitution, where strings were encoded into ring of integers [14] (see also [8] for a slightly different encoding). In this paper we use an analogous encoding (Definition 4), this time into ring of polynomials, in order to cover substitution, thus reducing problem to equivalence of *polynomial grammars with substitution*

over a ring of polynomials. Equivalence is known to be undecidable for this model, even in a rather restricted special case [5, Theorem 20]. In this paper we prove two decidability results for other special cases and strengthen this undecidability result, clarifying the scope of this “polynomial” approach.

Obtained decidability results transfer immediately to corresponding register transducers. This is stated in Lemma 5 for unrestricted polynomial grammars with substitution, which is a reduction to an undecidable problem, however, as explained in Remark 6, reduction holds also for special cases, which are decidable (Theorem 16 and Lemma 22).

**Main results.** The main problem discussed in this paper is *zeroness* of polynomial grammars with substitution over a ring of polynomials (Definition 3). First we discuss such grammars with an additional fixed bound on the number of transitions per derivation that use substitution. We provide a positive result (Theorem 16) and a negative one (Theorem 19). The negative result is a strengthening of [5, Theorem 20], mentioned in the previous paragraph. Then we move to grammars that use a special case of substitution – evaluation. For this variant we give a positive result (Lemma 22) and describe how it transfers to register transducers with substitution (Theorem 27).

## 2 Preliminaries

**Polynomial functions.** By a *ring* we mean a commutative ring with unity. A ring has *no zero divisors* if for all non-zero  $a, b \in R$ , their multiplication  $a \cdot b$  is non-zero. Let  $R$  be a ring, for example the ring of integers. Following [8], it is convenient to define polynomial functions so that they can return tuples. Tuples inputted by polynomials will be also called *vectors*. A function

$$p : R^n \rightarrow R^k$$

for some  $n, k \geq 0$  is a *polynomial function* if every output coordinate is represented by a polynomial in  $n$  variables. For a set  $V \subseteq R^n$ , by  $p|_V$  we denote function  $p$  restricted to  $V$ .

**Substitution.** In the ring  $R[X]$ , substitution is a  $(|X| + 1)$ -ary operation defined as

$$\begin{aligned} \text{subs} : R[X] \times R[X]^X &\rightarrow R[X] \\ \text{subs}(p, (q_x)_{x \in X}) &= p[x := (q_x)_{x \in X}]. \end{aligned}$$

We consider a vectorized variant of substitution:

$$\begin{aligned} \text{subs}_X^Y : R[X]^Y \times R[X]^X &\rightarrow R[X]^Y \\ \text{subs}_X^Y((p_y)_{y \in Y}, (q_x)_{x \in X}) &= (p_y(x := q_x, x \in X))_{y \in Y}. \end{aligned}$$

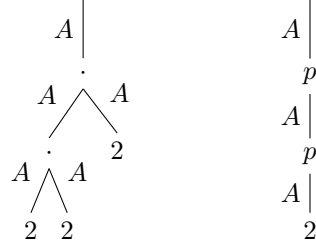
If the arities are clear from the context, we simply write  $\text{subs}(\bar{p}, \bar{q})$ , or even

$$\bar{p}(\bar{q}),$$

where  $\bar{p} = (p_y)_{y \in Y}$  and  $\bar{q} = (q_x)_{x \in X}$  are tuples of polynomials in  $R[X]$ .

**Polynomial grammars.** We define polynomial grammars following [8, Definition 2.1]. As opposed to the Introduction, we define them only for rings. Most of the times it will be the ring of polynomials, although in the proof of Theorem 16 we also use other rings.

- **Definition 1.** A polynomial grammar *over ring  $R$*  consists of
- finite set of nonterminals with a distinguished initial nonterminal, each nonterminal with an associated dimension  $\in \{0, 1, 2, \dots\}$ , and



■ **Figure 1** Example derivation trees for grammars  $A \rightarrow p(A) \mid 2$  with  $p(a) = a \cdot a$  and  $A \rightarrow A \cdot A \mid 2$ . They produce  $8 = 2^3$  and, respectively,  $16 = 2^{2^2}$ .

- *finite set of production rules, each of form  $X \rightarrow p(Y_1, \dots, Y_k)$  for  $k = 0, 1, \dots$  where  $p : R^n \rightarrow R^m$  is a polynomial function,  $Y_i$  for  $i = 1, \dots, k$  are nonterminals, and input and output dimensions of expressions “match”, i.e. sum of dimensions of  $Y_i$ ’s, for  $i = 1 \dots k$ , is equal to  $n$ , and dimension of  $X$  is equal to  $m$ .*

If a nonterminal has dimension  $n$ , then it generates a subset of  $R^n$ , which is defined as follows by induction. (The language generated by the grammar is defined to be the subset generated by its initial nonterminal.) Suppose that

$$X \rightarrow p(Y_1, \dots, Y_k)$$

is a production and we already know that vectors  $v_1, \dots, v_k$  are generated by nonterminals  $Y_1, \dots, Y_k$  respectively. Then the vector  $p(v_1, \dots, v_k)$  is generated by nonterminal  $X$ . The induction base is the special case of  $k = 0$ , where the polynomial  $p$  is a constant.

► **Example 2** (see Fig. 1). <sup>He</sup>Polynomial grammar

$$A \rightarrow p(A) \mid 2$$

with  $p(a) = a \cdot a$  has language

$$\{2^{2^n} : n \in \mathbb{Z}_+\}.$$

A similar polynomial grammar,

$$A \rightarrow A \cdot A \mid 2$$

has language

$$\{2^n : n \in \mathbb{Z}_+\}.$$

We define the dimension of a grammar to be the dimension of its initial nonterminal. For simplicity, the language of a grammar  $G$  with initial nonterminal  $A$  is sometimes denoted by  $A$ . For a polynomial function  $f : R^n \rightarrow R^m$ , for a grammar  $G$  with initial nonterminal  $A$  of dimension  $n$ , by  $f(G)$  or  $f(A)$  we denote a “canonical” grammar with language  $f(A)$ , i.e. grammar build from  $G$  by adding a fresh initial nonterminal  $S$  and a production  $S \rightarrow f(A)$ .

**Polynomial grammars with substitution.** Let  $R$  be a ring. A polynomial grammar with substitution over  $R[X]$  is an extension of polynomial grammar over  $R[X]$  where polynomials in production rules may use substitution (additionally to  $+$  and  $\cdot$ ).

► **Definition 3.** For a polynomial grammar or a polynomial grammar with substitution  $A$  over a ring of polynomials, that has a non-empty language <sup>1</sup>, zeroness problem asks if  $L(A) = \{0\}$ ; we often abbreviate it to  $A = 0$ .

<sup>1</sup> A technical assumption, does not change the essence of the problem. Notice it can be checked easily - it is enough to find productive nonterminals e.g. by fixpoint strategy. In particular, the algorithm does not depend on the output algebra.

**String-to-polynomial encoding.** The principal application of zeroness problem is equivalence of register transducers; the latter can be reduced to the former by the encoding we describe now.

► **Definition 4.** Consider the function  $\Phi$  that maps strings over  $\Sigma$  into pairs of polynomials in variables from set  $\tilde{\Sigma} \cup \bar{\Sigma}$ , which are two disjoint copies of  $\Sigma$ , defined as follows:

*Injective!* 
$$\Phi : a_1 \dots a_n \mapsto \left( \sum_{i=1}^n \bar{a}_1 \dots \bar{a}_{i-1} \tilde{a}_i, \quad \bar{a}_1 \dots \bar{a}_n \right). \quad (1)$$

We denote first and second coordinate of  $\Phi(w)$  by  $\tilde{w}$  and  $\bar{w}$ , i.e.

$$\Phi(w) = (\tilde{w}, \bar{w}). \quad (2)$$

For example,  $\Phi(abbab) = (\tilde{a}\tilde{b}^2\bar{a} + \tilde{b}\tilde{b}^2\bar{a} + \tilde{b}\tilde{b}\bar{a} + \tilde{a}\tilde{b} + \tilde{b}, \bar{a}^2\bar{b}^3)$ .

The function  $\Phi$  is a “generalisation<sup>2</sup>” of a natural encoding of strings into numbers they represent in  $|\Sigma|$ -ary, i.e.  $w \mapsto (\text{val}_{|\Sigma|}(w), |\Sigma|^{|w|})$  where  $\text{val}_{|\Sigma|}(w)$  is the value of  $w$  as a number in  $|\Sigma|$ -ary (assuming some bijection of  $\Sigma$  with  $\{0, 1, \dots, |\Sigma| - 1\}$ ). We use this generalisation to be able to use substitution.

Notice that  $\Phi$  “commutes” with substitution, i.e.

$$\Phi(w[a := v_a, a \in \Sigma]) = \Phi(w)[\tilde{a} := \tilde{v}_a, \bar{a} := \bar{v}_a, a \in \Sigma] \text{ for } w, (v_a)_{a \in \Sigma} \in \Sigma^*.$$

We say that the polynomial substitution  $(\tilde{a} := \tilde{v}_a, \bar{a} := \bar{v}_a)_{a \in \Sigma}$  is induced by the word substitution  $(a := v_a)_{a \in \Sigma}$ .

Using the encoding above, we obtain the following lemma.

► **Lemma 5.** Equivalence of register transducers with substitution over  $\Sigma^*$  can be reduced to zeroness of polynomial grammars with substitution over  $\mathbb{Q}[\tilde{\Sigma} \cup \bar{\Sigma}]$ .

► **Remark 6** (Undecidability in Lemma 5). Let us emphasize that even though in Lemma 5 we have a reduction to an undecidable problem, it remains valid for special cases of this problem that are decidable – Theorem 16 and Lemma 22. As corollaries we obtain, respectively, result from beginning of Section 3.3 and Theorem 27.

**Proof sketch of Lemma 5.** Suppose that we have two register transducers with substitution that compute string-to-string functions  $f, g : \Sigma^* \rightarrow \Sigma^*$ . We can convert them into a polynomial grammar  $G$  over  $\mathbb{Q}[\tilde{\Sigma} \cup \bar{\Sigma}]$  that produces pairs of polynomials  $(\tilde{f(w)}, \bar{g(w)})$  for input words  $w$ . It can be done ~~us~~ using the fact that

$$\Phi(w \cdot v) = p(\Phi(w), \Phi(v))$$

for some polynomial function  $p$  – indeed, for any words  $w, v$  we have  $\tilde{wv} = \tilde{w} \cdot \bar{v} + \tilde{v}$  and  $\bar{wv} = \bar{w} \cdot \bar{v}$  or explicitly

$$p(x_1, x_2, y_1, y_2) = (x_1 y_2 + y_1, x_2 y_2)$$

– and that  $\Phi$  “commutes” with substitution (see Definition 4). Applying polynomial function  $p(x_1, x_2) = x_1 - x_2$  to  $G$  results in grammar  $p(G)$  for which zeroness holds if and only if  $f, g$  are equivalent. ◀

<sup>2</sup> It is a generalisation in a sense, that natural encoding is an evaluation of  $\Phi$  at  $\tilde{a} = \tau(a), \bar{a} = |\Sigma|$  for a bijection  $\tau$  of  $\Sigma$  with  $\{0, 1, \dots, |\Sigma| - 1\}$

## 2.1 Fields, algebraic sets

A *field* is a ring in which every non-zero element has an inverse. Let  $K$  be a field. By  $K[X]$  we denote the ring of polynomials over set of variables  $X$ . By  $K(X)$  we denote the field of rational functions over  $X$ , i.e.  $K(X) = \{\frac{p}{q} \mid p, q \in K[X], q \neq 0\}$ .

A field is a *computable field* if its elements can be enumerated such that operations  $+$ ,  $\cdot$  are computable functions.

A field  $K$  is *algebraically closed*, if every univariate polynomial with coefficients in  $K$  has a root in  $K$ . An *algebraic closure* of a field  $K$ , denoted by  $\overline{K}$ , is, roughly speaking, the smallest algebraically closed field that contains  $K$ . Typical examples of algebraically closed fields are (1) the field of complex numbers, which is algebraic closure of field of real numbers and (2) the field of algebraic numbers, which is algebraic closure of field  $\mathbb{Q}$ .

Introducing algebraic closures does not affect effectiveness:

Rabin'60

► **Lemma 7.** ([13, Theorem 7]) Let  $K$  be a given computable field. Then one can compute an embedding  $K \subset \overline{K}$ , which moreover is a computable function. In particular,  $\overline{K}$  is a computable field and it can be computed, given  $K$ .

By an *ideal* we mean a subset  $I$  of  $K[X]$  in which  $f, g \in I$  implies  $f+g \in I$  and  $f \in I, p \in K[X]$  implies  $f \cdot p \in I$ , for all polynomials  $f, g, p$ . For a set of polynomials  $F$ , by  $\langle F \rangle$  we denote set  $\{r_1 f_1 + r_2 f_2 + \dots + r_k f_k \mid f_i \in F, r_i \in K[X], k \geq 0\}$ ; this is the smallest ideal that contains set  $F$  called *ideal generated by  $F$* ; elements of  $F$  are called *generators* of  $\langle F \rangle$ . Ideals can be represented in a finite way, as shown by Hilbert's Basis Theorem.

► **Fact 8.** (Hilbert's Basis Theorem, [10, Chapter 2, §5, Theorem 5]). Let  $K$  be a field. Every ideal in  $K[X]$  can be generated by a finite set of polynomials.

Let  $K$  be a field. An *algebraic set* is a subset of  $K^n$  which is a set of zeros of some set of polynomials in  $n$  variables, i.e. is of form  $\{v \in K^n \mid p(v) = 0 \text{ for all } p \in F\}$  for some set of polynomials  $F \subseteq K[X]$ . For an algebraic set  $V$ , by  $I(V)$  we denote the set of all polynomials that are zero on  $V$ ; this is an ideal. For a ideal  $I$  in  $K[X]$ , by  $V(I)$  we denote the set of common zeros of all polynomials of  $I$ ; this is a variety. We say that  $V(I)$  is *represented* by  $I$ . Every algebraic set can be represented by some ideal, for example by  $I(V)$ ; this, together with Hilbert's Basis Theorem (Fact 8), provides a way to represent algebraic sets in a finite way.

From this we conclude the following.

► **Corollary 9.** Let  $K$  be a computable field. Then algebraic sets can be enumerated.

A coordinate ring of an algebraic set  $V \subseteq K^n$  is the ring of polynomial functions from  $V$  to  $K$ . An algebraic set is called an (irreducible) *variety* if it cannot be represented as a finite union of pairwise distinct algebraic sets.

► **Remark 10** (Notation: does algebraic set = variety?). Different sources introduce different names for algebraic sets. For example, in [10] they are called varieties, and what we call a variety is called an irreducible variety. To avoid confusion, we always precede word "variety" with "(irreducible)".

► **Lemma 11.** ([10, Chapter 4, §6, Theorem 2 and remarks at the end of Chapter 4, §6]) Let  $K$  be a computable field. Then every algebraic set can be effectively decomposed into (irreducible) varieties.

► **Lemma 12.** ([10, Chapter 4, §5, Proposition 3]) An algebraic set is an (irreducible) variety if and only if its coordinate ring has no zero divisors.

### 3 Polynomial grammars with a bounded number of substitutions

In this section we discuss decidability of zeroness problem for polynomial grammars with substitution over a ring of polynomials with the following restriction: substitution operation can be used once per derivation. We call this model *polynomial grammar with one substitution*. Let us recall that zeroness is decidable in case of no substitutions.

► **Lemma 13** ([7], Theorem 15). *Let  $R$  be a computable ring with no zero divisors. Then equivalence is decidable for polynomial grammars over  $R$ .*

Let us define the main decision problem of this section, which generalises zeroness of polynomial grammars without substitution.

#### Zeroness of independent substitution of polynomial grammars

**Input:**

$R$  - a computable ring with no zero divisors,

$X$  - finite set of variables,

$A$  - polynomial grammar over ring  $R[X]$ ,

$B$  - polynomial grammar over ring  $R$

**Question:** *Is it the case that for all  $a \in A, b \in B$   $a(b) = 0$ ?*

We often abbreviate the statement above to

$$A(B) = 0.$$

This problem can be seen as a special case of zeroness of polynomial grammars with one substitution, for grammars of the form  $\text{subs}(A, B)$ , where  $A, B$  are polynomial grammars. We call such substitution *independent* because tuples of substituted values come from a different nonterminal than polynomials to which they are substituted, and therefore they are produced independently; we prove this problem decidable in Theorem 16. As we will see in Section 3.2, in case of *dependent* substitution, the problem is undecidable.

► **Example 14.** Consider a polynomial grammar  $A$  over ring of polynomials in two variables  $\tilde{x}, \tilde{x}$  with coefficients in  $\mathbb{Q}(\tilde{a}, \tilde{a})$ , i.e.  $(\mathbb{Q}(\tilde{a}, \tilde{a}))[\tilde{x}, \tilde{x}]$  defined as  $A \rightarrow m(S), S \rightarrow p(S) \mid (0, 1, 0, 1)$  where  $m(\tilde{r}, \tilde{r}, \tilde{s}, \tilde{s}) = \tilde{r} - \tilde{s}$  and  $p(\tilde{r}, \tilde{r}, \tilde{s}, \tilde{s}) = (\tilde{r} \cdot \tilde{x} \cdot \tilde{a} + \tilde{x} \cdot \tilde{a} + \tilde{a}, \tilde{r} \cdot \tilde{x} \cdot \tilde{a}, \tilde{x} \cdot \tilde{s} \cdot \tilde{a} + \tilde{s} \cdot \tilde{a} + \tilde{a}, \tilde{x} \cdot \tilde{s} \cdot \tilde{a})$  and a polynomial grammar  $B$  over ring of polynomials in two variables  $\tilde{a}, \tilde{a}$  and coefficients in  $\mathbb{Q}$ , i.e.  $\mathbb{Q}[\tilde{a}, \tilde{a}]$  defined as  $B \rightarrow q(B) \mid (0, 1)$  where  $q(\tilde{t}, \tilde{t}) = (\tilde{t} \cdot \tilde{a} + \tilde{a}, \tilde{t} \cdot \tilde{a})$ . Then  $A(B) = 0$ .

► **Remark 15.** The above Example expresses the fact, under our string-to-polynomial encoding, that  $(xa)^n = x^n a^n$  for all words  $x \in \{a\}^*$ , for all  $n \geq 0$ .

**Remark on algebraic closure.** Let us discuss a subtlety before we state and prove Theorem 16. Despite the fact that all polynomials arising from our string-to-polynomial encoding are elements of field  $\mathbb{Q}(\tilde{\Sigma} \cup \tilde{\Sigma})$ , we carry out the proof for algebraically closed fields. It does not affect generality, as every field can be effectively embedded into algebraically closed field (Lemma 7). We do so because it makes the description of the algorithm more readable, as in such case one can compute the coordinate ring of a given algebraic set (Lemma 39); however, we present a version of the proof that does not use the notion of algebraic closure in Section A.3.

We conjecture that the algorithm from Theorem 16 does *not* have to use elements of  $\overline{K}$  that are not in  $K$  – in consequence, the decision of introducing algebraic closure would affect only the high-level description of the algorithm, not the algorithm itself. This might be important when trying to obtain a feasible complexity for a special case (recall in general zeroness of polynomial grammars over a ring is Ackermann-hard [3, Theorem 1]).



### 3.1 Zeroness of independent substitution of polynomial grammars: decidability

In this section we prove Theorem 16.

► **Theorem 16.** *Let  $R$  be a computable ring with no zero divisors. Then zeroness of independent substitution of polynomial grammars over  $R[X]$  is decidable.*

As stated in Remark 6, the reduction from Lemma 5, restricted to the special case from Theorem 16, yields decidability of the corresponding problem for register transducers (Section 3.3).

The proof is based on the proof of [14, Corollary 8.2] in the presentation of [8].

**Proof of Theorem 16.** Without loss of generality we may assume that grammar  $A$  is of dimension 1 (notice we cannot assume it for  $B$ ). In this proof, by abuse of notation, we identify a polynomial grammar  $A$  with its the language. We treat  $A$  and  $B$  as polynomial grammars over a computable algebraically closed field that contains  $R$ , for example algebraic closure of its field of fractions.

There is a clear semi-procedure for non-zeroness – it essentially amounts to enumerating all derivations. In rest of the proof, we show a semi-procedure for zeroness.

Observe that  $A(B) = 0$  if and only if for some algebraic set  $V$ :

$$\begin{cases} B \subseteq V \text{ (i) and} \\ A|_V = 0 \text{ (ii),} \end{cases}$$

where  $A|_V$  denotes set  $\{f|_V \mid f \in A\}$ . Indeed – for example, as  $V$ , one may take  $V(A)$ . The semi-procedure for equivalence is as follows: guess, by infinite enumeration (see Corollary 9), an algebraic set  $V$  and test conditions (i) and (ii), which can be done the following way.

► **Sublemma 1.** *For a given  $V$ , condition (i) can be tested effectively.*

**Proof.** Condition (i) reduces to zeroness of polynomial grammars in the following way. Denote by  $I$  the ideal that  $V$  is represented by. Observe that  $B \subset V(I) = V$  if and only if  $f(B) = 0$  for all generators  $f$  of  $I$ . ◀

► **Sublemma 2.** *For a given  $V$ , condition (ii) can be tested effectively.*

**Proof sketch.** Decompose  $V$  into (irreducible) varieties  $V = V_1 \cup V_2 \cup \dots \cup V_k, k \geq 1$  (see Lemma 11). Then  $A|_V = 0$  holds iff  $A|_{V_i} = 0$  for  $i = 1, \dots, k$ . Observe that  $A|_{V_i} = 0$  is equivalent to zeroness of  $A$ , treated as a polynomial grammar over coordinate ring of  $V_i$  (see Lemma 39 for effectiveness); it has no zero divisors (Lemma 12) and hence decidability follows from Lemma 13. ◀

**Generalisation.** An analogous problem can be defined for arbitrary number of grammars; for  $k$  grammars, we will abbreviate it to  $A_1(A_2(\dots(A_{n-1}(A_k)))) = 0$ . Let us sketch the proof of its decidability – it is very similar to the proof of case  $n = 2$ . Observe that  $A_1(A_2(A_3(\dots(A_n)\dots))) = 0$  is equivalent to  $A_2(A_3(\dots(A_n)\dots)) \subseteq V$  and  $A_1|_V = 0$  for some algebraic set  $V$ . For a given  $V$ , first condition is equivalent to  $f(A_2(A_3(\dots(A_n)\dots))) = 0$  for generators  $f$  of ideal that  $V$  is represented by – this is decidable by induction assumption for  $n - 1$  polynomial grammars  $f(A_2), A_3, \dots, A_n$ . Second condition was proven decidable in Sublemma 2. ◀



► **Remark 17.** For any polynomial grammar  $A$  there always exists a *finite* set of polynomials  $F$  such that  $A(v) = 0$  if and only if  $F(v) = 0$ , for every vector  $v$ . In particular  $A(B) = 0$  if and only if  $F(B) = 0$ . Given a finite set of polynomials  $F$ ,  $F(B) = 0$  can be decided (Lemma 13). However, it is not clear how to compute such set  $F$  given grammar  $A$ .

► **Remark 18.** Let  $A, B$  be polynomial grammars. If the grammar  $A$  is linear, i.e. there is at most one nonterminal on the right-hand side of every production rule, then grammar  $A(B)$  can be seen as a polynomial grammar too – it is enough to “concatenate” those two grammars by replacing production rules of grammar  $A$  of form *nonterminal*  $\rightarrow$  *constant* with *nonterminal*  $\rightarrow B$ . However in case  $A$  is not linear, it is not clear how to convert  $A(B)$  to a polynomial grammar.

## 3.2 Polynomial grammar with one substitution: undecidability

In previous section we saw that restriction to one *independent* substitution per derivation yields decidability of zeroness. In this section we show that in general case of one *dependent* substitution, this problem is undecidable. We call such substitution dependent because substituted tuples and polynomials to which they are substituted may come from the same nonterminal and hence be generated dependently.

► **Theorem 19.** *Zeroness of polynomial grammars with one substitution over  $\mathbb{Z}[x]$  is undecidable.*

In contrast to Theorem 16, the reduction from Lemma 5 restricted to special case from Theorem 19 does *not* yield decidability nor undecidability of analogous problem for register transducers.

The proof was suggested by Lasota and Piórkowski [12]. It is analogous to proof of [5, Theorem 17], except two-counter machines are replaced by reset VASS-es; used encoding of reset VASS into register transducer is similar to [4, Example 3].

By a VASS we mean a Vector Addition System with States. We call vectors that describe transitions *step vectors*. By a *reset VASS* we mean an extension of VASS where transitions may also reset some of coordinates to 0. By a *unit step vector* we mean a step vector of form  $(0, 0, \dots, 0, 1, 0, \dots, 0)$ . For a reset VASS, by  $\mathbf{0}$  we denote vector of zeros of its dimension.

**Proof sketch.** We show a reduction from reachability in reset VASS, which is known to be undecidable [2, Theorem 5]. Without loss of generality assume it is reachability from  $\mathbf{0}$  to  $\mathbf{0}$  and step vectors are unit – this ensures that sum of coordinates is an integer from interval  $[0, n]$  after  $n$  steps of a valid run. Given a reset VASS  $\mathcal{V}$  we construct a register transducer with substitution *over ring*  $\mathbb{Z}[x]$  that uses substitution only in the output transition (hence once per run) such that  $\mathcal{V}$  can reach  $\mathbf{0}$  from  $\mathbf{0}$  if and only if the register transducer returns a non-zero output on some input (such register transducers are a different formalism for the same model as polynomial grammars with one substitution).

We describe the construction. The transducer takes a sequence of transitions of  $\mathcal{V}$  as an input (this is a word over finite alphabet), which may describing a (valid) run of  $\mathcal{V}$  or not. The state of the transducer is the state of  $\mathcal{V}$  and turns to error state if in the input word the states of some pair of consecutive transitions do not match. The transducer holds the counter values of  $\mathcal{V}$  in registers that we call *counter registers*. It has an *error register* which turns 0 when an “error” occurs, i.e. some coordinate of  $\mathcal{V}$  goes below 0 (note that step vectors are unit so such coordinate equals  $-1$ ) and holds 0 for the rest of the run – this is achieved by multiplying the error register by the product of values of counter registers, each value incremented by 1, in each update. There is a *reachability test register* that holds a

polynomial  $(x - 1)(x - 2) \dots (x - n)$  after  $n$  steps (formally, to construct it one needs an *auxiliary register* that holds number  $n$ ); it has a property that, for an integer input from interval  $[0, n]$ , it evaluates to 0 if and only if the input number is non-zero. Finally, if the state is accepting, in output transition the transducer returns error register multiplied by reachability test register evaluated at the sum of counter registers (this involves substitution of a register into another register); observe this is a non-zero number if and only if input word is a (valid) run of  $\mathcal{V}$  and this run reaches  $\mathbf{0}$  from  $\mathbf{0}$ . We give details of the construction in Section A.2.  $\blacktriangleleft$

### 3.3 Interpretation of Theorem 16 and Theorem 19 as testing infinitely many equations on a language generated by a register transducer

Consider a register transducer  $E$  that returns pairs of words over alphabet  $\Delta \cup X$  and a register transducer  $T$  that returns  $X$ -tuples of words over  $\Delta$ . Transducer  $E$  can be interpreted as a generator of equations (constraints) and  $T$  as generator of tuples to be tested. Then Theorem 16 states that it is decidable if all tuples of words generated by  $T$  satisfy all equations generated by  $E$ . On the other hand, Theorem 19 states that, in case when polynomials and integers are generated, it is undecidable for a polynomial grammar that generates “pairs” (equation, tuple of vectors) if the “second” coordinate is always a solution for the “first” one (formally, the numbers of mentioned coordinates differ as e.g. an equation alone requires two coordinates). In such interpretation, words “independent” and “dependent” describe the relationship between equations and tested tuples. We include a formal presentation in Section A.1.

## 4 Polynomial grammars with evaluations

In previous section we considered polynomial grammars with restricted number of uses of substitution per derivation. Another way of restricting substitution is to consider evaluations – instead of adding general  $(1 + |X|)$ -ary substitution operation

$$\text{subs} : K(X) \times K(X)^X \rightarrow K(X)$$

we add evaluations, which are substitutions where the argument from  $K(X)^X$  is fixed, i.e. unary operations

$$\begin{aligned} \text{eval}_v &: K(X) \rightarrow K(X), \\ \text{eval}_v(p) &\mapsto p(v), \end{aligned}$$

for vectors  $v \in K(X)^X$ . We call this restriction of polynomial grammars with substitution *polynomial grammars with evaluations*.

In this section we prove Theorem 27 that states that polynomial grammars with evaluations that satisfy <sup>Bo</sup>simultaneous automorphism condition (Definition 21) have decidable zeroness. The core technical ingredient is Lemma 22, which is a result of discussions with Bojańczyk, Worrell, Shirmohammadi and Kiefer [9], allows to use simultaneous field automorphisms in polynomial grammars over a field. Lemma 25 gives an easy-to-check characterisation of word substitutions that are mapped to field automorphisms by the string-to-polynomial encoding from Definition 4. Theorem 27 concludes those two lemmas.

#### 4.1 Polynomial grammar with simultaneous automorphisms: decidable equivalence.

► **Definition 20.** Let  $R$  be a ring. An *automorphism* of  $R$  is a bijective function  $h : R \rightarrow R$  that satisfies  $h(\eta(a, b)) = \eta(h(a), h(b))$  for  $\eta$  being either addition or multiplication for every  $a, b \in R$ . For example, evaluation  $\text{eval}_v$  for  $v = x + 1$ , i.e.

$$\mathbb{Q}(x) \ni f \mapsto f[x := x + 1] \in \mathbb{Q}(x)$$

is an automorphism of the field  $\mathbb{Q}(x)$ . An automorphism of product of fields  $K^n$  is called *simultaneous* if it is of form  $\alpha^n = \underbrace{(\alpha, \alpha, \dots, \alpha)}_n$  for some automorphism  $\alpha$  of  $K$ .

► **Definition 21.** A *polynomial grammar with simultaneous automorphisms/evaluations* is an extension of polynomial grammar over a field where production rules are of form  $Y \rightarrow p(\alpha^N(Y_1, \dots, Y_n))$  where  $\alpha$  is a field automorphism/an evaluation and  $N = \dim Y_1 + \dots + \dim Y_n$ .

► **Lemma 22.** Let  $K$  be a computable field. Then polynomial grammars with simultaneous automorphisms over  $K$  have decidable equivalence.

Similarly as for Theorem 16, the reduction from Lemma 5, restricted to the special case from Lemma 22, yields decidability of equivalence for certain class of register transducers with substitution (Theorem 27).

**Proof of Lemma 22.** A proof analogous to proof of Lemma 13 can be performed for polynomial grammars with simultaneous automorphisms – it is enough to show that preimage of an algebraic set by simultaneous automorphism of  $K$  is effectively an algebraic set (note that this fact holds for any algebra, with the same proof as for fields).

Let  $\alpha$  be an automorphism of  $K$ ; by abuse of notation, denote the same way coordinate-wise application of it. Then  $\alpha^{-1}(V(p_1, \dots, p_n)) = V(p_1 \circ \alpha, \dots, p_n \circ \alpha)$ . Functions  $p_i \circ \alpha$  might not be polynomial, however for a polynomial  $p(x_1, \dots, x_n) = \sum_{i=0}^n a_i x^i$  equality  $(p \circ \alpha)(x_1, \dots, x_n) = 0$  is equivalent to a polynomial equation  $\sum_{i=0}^n \alpha^{-1}(a_i) x^i = 0$ . ◀

► **Definition 23.** We call a word substitution  $p : \Sigma \rightarrow \Sigma^*$  *com-injective* if  $w \mapsto p(w)$  is injective, when treated as a mapping of commutative words.

By abuse of notation, we also denote as com-injective a polynomial substitution, which is an image of com-injective word substitution by our string-to-polynomial encoding.

► **Lemma 24.** If  $p$  is a substitution for one letter, call it  $\sigma$ , (in other words,  $p$  is an identity function on all letters except  $\sigma$ ) and it does not vanish letter  $\sigma$ , then it is com-injective. Example of such substitution is  $p : a \mapsto bacbaab$ . A non-example of such substitution is  $p : a \mapsto bbc$ .

**Proof.** Straightforward. ◀

In the following lemma we characterise word substitutions that induce field automorphisms via our string-to-polynomial encoding. Finding such word substitutions is challenging, as almost no word substitution induces an automorphism of field  $\mathbb{Q}(\tilde{\Sigma} \cup \bar{\Sigma})$ . However, a reasonably large class of word substitutions induces automorphisms of the field  $\mathbb{Q}(\tilde{a}, \bar{a}^{\mathbb{Q}_{\geq 0}}, a \in \Sigma)$ , an extension of  $\mathbb{Q}(\tilde{\Sigma} \cup \bar{\Sigma})$  that is defined below.

Notions of a polynomial and a rational function, considered as formal expressions, can be extended to situation where some of variables have exponents from an arbitrary infinite monoid, that can be embedded in some group – “ordinary” polynomials use monoid  $(\mathbb{N}, +, 0)$ . We use this extension with monoid  $(\mathbb{Q}_{\geq 0}, +, 0)$ . Let  $K$  be a field. By  $K[\tilde{a}, \bar{a}^{\mathbb{Q}_{\geq 0}}, a \in \Sigma]$  we denote the ring of polynomials over variables  $\tilde{\Sigma} \cup \bar{\Sigma}$  where variables  $\bar{\Sigma}$  have exponents in  $(\mathbb{Q}_{\geq 0}, +, 0)$  and by  $K(\tilde{a}, \bar{a}^{\mathbb{Q}_{\geq 0}}, a \in \Sigma) = \{ \frac{p}{q} \mid p, q \in K[\tilde{a}, \bar{a}^{\mathbb{Q}_{\geq 0}}, a \in \Sigma], q \neq 0 \}$  the corresponding field of rational functions; let us emphasize that the latter contains elements with negative exponents as well, e.g.  $\bar{a}^{-1/2} = \frac{1}{\bar{a}^{1/2}}$ .

**A subtlety.** There is a subtlety regarding defining evaluations. In field  $K(\tilde{a}, \bar{a}^{\mathbb{Q}_{\geq 0}}, a \in \Sigma)$ , as opposed to  $K(\tilde{\Sigma} \cup \bar{\Sigma})$ , evaluation is *not* defined for every vector: for it to do so, it must evaluate variables from  $\bar{\Sigma}$  on monomials in variables from  $\bar{\Sigma}$ ; however polynomial substitutions induced by word substitutions via our word-to-polynomial encoding have that property.

► **Lemma 25.** *Com-injective word substitutions induce automorphisms of field  $\mathbb{Q}(\tilde{a}, \bar{a}^{\mathbb{Q}_{\geq 0}}, a \in \Sigma)$ .*

*Non-com-injective word substitutions do not induce automorphisms of any field that contains  $\mathbb{Q}(\tilde{\Sigma} \cup \bar{\Sigma})$ .*

► **Corollary 26.** *Grammar  $Y$  from Introduction is a polynomial grammar with simultaneous automorphisms via our string-to-polynomial encoding.*

**Proof.** Substitutions in the grammar are simultaneous – both register transducers apply the same word substitutions at the same steps on all of their registers (notice that update of register  $S$  does not use substitution explicitly, but it could with no difference, as strings stored in  $S$  do not contain symbol  $\#$ , for which the substitution is performed). They are com-injective 24 and hence induce automorphisms (Lemma 25). ◀

From Lemma 25 one can conclude Theorem 27.

► **Theorem 27.** Register transducers with simultaneous evaluations of com-injective substitutions have decidable equivalence.

**Proof.** Due to Lemma 25, the corresponding polynomial grammar with evaluations is a polynomial grammar with simultaneous automorphisms, when considered over the field  $\mathbb{Q}(\tilde{a}, \bar{a}^{\mathbb{Q}_{\geq 0}}, a \in \Sigma)$ . Its zeroness is decidable due to Lemma 22. ◀

Now we give a proof of Lemma 25 for an example substitution, in order to avoid unnecessary formalism; the proof idea is the same in the general case, for which we give a proof in Section A.4.

**Proof of Lemma 25 for an example substitution.** Let  $\Sigma = \{a, b\}$ . Consider a substitution of words  $p : a^* \rightarrow a^*$  defined as  $p : a \mapsto aa$ . Then induced substitution of polynomials is  $p' : \tilde{a} \mapsto \tilde{a}\bar{a} + \tilde{a}, \bar{a} \mapsto \bar{a}^2$ . This substitution is not invertible when considered in the field  $\mathbb{Q}(\tilde{\Sigma} \cup \bar{\Sigma})$ , however we show it is invertible when considered in the field  $\mathbb{Q}(\tilde{a}, \bar{a}^{\mathbb{Q}_{\geq 0}}, a \in \Sigma)$ . To find the inverse substitution, we consider a system of equations:

$$\begin{cases} \tilde{a}' = \tilde{a} \cdot (\bar{a} + 1), \\ \bar{a}' = \bar{a}^2. \end{cases} \quad (3)$$

Second equation is purely in variable  $\bar{a}$ , hence we solve it first; it is a linear equation, where the exponents play the role of coefficients from the field  $\mathbb{Q}$ . We get  $\bar{a} = \bar{a}'^{1/2}$ . We substitute this into the first equation and get  $\tilde{a}' = \tilde{a} \cdot (\bar{a}'^{1/2} + 1)$ , This is a linear equation in variable

$\tilde{a}$  with coefficients in the field  $\mathbb{Q}(\tilde{a}', \tilde{a}'^{\mathbb{Q}_{\geq 0}})$ . Solving it, we obtain  $\tilde{a} = \frac{\tilde{a}'}{1+\tilde{a}'^{1/2}}$ . This shows that substitution  $\tilde{a} \mapsto \frac{\tilde{a}'}{1+\tilde{a}'^{1/2}}, \tilde{a} \mapsto \tilde{a}'^{1/2}$  is the right-inverse of  $p$ . It is a two-sided inverse because the equations are linear (in the mentioned sense). This finishes the proof for this example. ◀

► **Remark 28.** Let us notice that for some grammars, Theorem 27 can be reduced to zeroness of polynomial grammars *without* substitution by simply removing the automorphisms from production rules – for example this is the case when polynomial functions used in productions have integer coefficients. However this is not true in general when “new” occurrences of substitutable constants are introduced in the productions. For example, consider polynomial grammar with substitution over  $\mathbb{Q}[a, b]$  with production rules

$$S \rightarrow p(X), X \rightarrow q(Y), Y \rightarrow a - b,$$

where  $p, q : \mathbb{Q}[a, b] \rightarrow \mathbb{Q}[a, b]$  are defined as  $p(f) = f + b, q(f) = f[b := a + b]$ . Then  $S = 0$ , but if automorphisms were removed, we would have  $S = a$ .

## 5 Summary, future work

Theorem 19 draws a rather pessimistic view on polynomial grammars with substitution over ring of polynomials – even almost the simplest model has undecidable zeroness. In this paper we show positive results when the model is restricted either to independent substitution (Theorem 16) or to evaluations (Theorem 27). Theorem 16 can be interpreted as decidability of testing infinite systems of equations on languages, where both the system and the language can be generated by register transducers (Section 3.3). Theorem 27 seems to have large limitations: first, every evaluation must be “simultaneous” i.e. in each production the same evaluation has to be applied to all coordinates of all nonterminals at once, second, this evaluations needs to induce a field automorphism, which excludes evaluations that vanish letters as they are not an injective. Having said that, in Introduction we define grammar  $Y$  that satisfies both these conditions and can be used to decide equivalence of two register transducers (also defined in Introduction) that compute function **sqrev**.

Finally, we ask open questions. First one is about expressiveness of register transducers with evaluations (no automorphism condition).

1. What are other interesting string-to-string functions computed by register transducers with evaluations whose evaluations are not allowed to vanish letters?

A Function similar to **sqrev**, that maps each word  $w$  to  $\text{rev}(w)^{|w|}$  seems not to be an example, as it seems to require introducing symbol  $\#$  in a way analogous to **sqrev** and then vanishing it.

Also, it is interesting what problems about register transducers are decidable, in particular:

2. Is equivalence undecidable?
3. Is equivalence decidable if we restrict to register transducers with evaluations (this question could be asked also for polynomial grammars)?

---

## References

- 1 Rajeev Alur. Streaming string transducers. In Lev D. Beklemishev and Ruy de Queiroz, editors, *Logic, Language, Information and Computation*, pages 1–1, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

- 2 Toshiro Araki and Tadao Kasami. Some decision problems related to the reachability problem for petri nets. *Theoretical Computer Science*, 3(1):85–104, 1976. URL: <https://www.sciencedirect.com/science/article/pii/0304397576900670>, doi:[https://doi.org/10.1016/0304-3975\(76\)90067-0](https://doi.org/10.1016/0304-3975(76)90067-0).
- 3 M. Benedikt, T. Duff, A. Sharad, and J. Worrell. Polynomial automata: Zeroness and applications. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005101.
- 4 M. Benedikt, T. Duff, A. Sharad, and J. Worrell. Polynomial automata: Zeroness and applications. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005101.
- 5 Adrien Boiret, Radosław Piórkowski, and Janusz Schmude. Reducing Transducer Equivalence to Register Automata Problems Solved by "Hilbert Method". In Sumit Ganguly and Paritosh Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*, volume 122 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 48:1–48:16, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: <http://drops.dagstuhl.de/opus/volltexte/2018/9947>, doi:10.4230/LIPIcs.FSTTCS.2018.48.
- 6 Mikołaj Bojańczyk. Polyregular functions, 2018. [arXiv:1810.08760](https://arxiv.org/abs/1810.08760).
- 7 Mikołaj Bojańczyk and Janusz Schmude. Some Remarks on Deciding Equivalence for Graph-To-Graph Transducers. In Javier Esparza and Daniel Kráľ, editors, *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*, volume 170 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:14, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. URL: <https://drops.dagstuhl.de/opus/volltexte/2020/12752>, doi:10.4230/LIPIcs.MFCS.2020.19.
- 8 Mikołaj Bojańczyk. The hilbert method for transducer equivalence. *ACM SIGLOG News*, 6(1):5–17, February 2019. URL: <https://doi.org/10.1145/3313909.3313911>, doi:10.1145/3313909.3313911.
- 9 Mikołaj Bojańczyk, James Worrell, Mahsa Shirmohammadi, and Sandra Kiefer. Personal communication.
- 10 David Cox, John Little, and Donal O’Shea. *Ideals, Varieties, and Algorithms. An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Fourth edition edition, 2015. doi:10.1007/978-3-319-16721-3.
- 11 Eitan M. Gurari. The Equivalence Problem for Deterministic Two-Way Sequential Transducers is Decidable. *SIAM J. Comput.*, 11(3):448–452, 1982.
- 12 S. Lasota and R. Piórkowski. Personal communication.
- 13 Michael O. Rabin. Computable algebra, general theory and theory of computable fields. *Transactions of the American Mathematical Society*, 95(2):341–360, 1960. URL: <http://www.jstor.org/stable/1993295>.
- 14 Helmut Seidl, Sebastian Maneth, and Gregor Kemper. Equivalence of deterministic top-down tree-to-string transducers is decidable. *J. ACM*, 65(4), April 2018. doi:10.1145/3182653.

## **A** Appendix

### **A.1** Formal presentation of Section 3.3

In this section we give a formal presentation of the problem presented in Section 3.3. For a polynomial grammar  $A$ , by abuse of notation we identify it with its language. We represent equations as a pair of polynomials, denoted as  $e = (e_1, e_2)$ ; a potential solution  $v$  is a vector of polynomials;  $v$  is a solution of  $e$  iff  $e_1(v) = e_2(v)$ .

#### **Independent equation satisfiability**

**Input:**

$\Delta$  - finite alphabet,

$X$  - finite set of variables,

$E \subseteq ((\mathbb{Q}[\Delta])[X])^2$  - set of equations in variables  $X$ , given by polynomial grammar  $E$  of dimension 2.

$T \subseteq (\mathbb{Q}[\Delta])^{|X|}$  - tested language, given by polynomial grammar  $T$  of dimension  $|X|$ ,

**Question:** Is it the case that for all inputs  $(e_1, e_2) \in E, v \in T$ :

$$e_1(v) = e_2(v)?$$

### Dependent equation satisfiability

**Input:**

$\Delta$  - finite alphabet,

$X$  - finite set of variables,

$G$  - polynomial grammar over  $(\mathbb{Q}[\Delta])[X]$  of dimension  $(|X| + 2)$

**Question:** Is it the case that for all inputs  $((e_1, e_2), v) \in G$ :

$$e_1(v) = e_2(v)?$$

## A.2 Detailed proof of Theorem 19

In this section we give a details of the construction of register transducer from proof of Theorem 19.

► **Theorem 19.** *Zeroneess of polynomial grammars with one substitution over  $\mathbb{Z}[x]$  is undecidable.*

**Details of construction from 19.** Given a reset VASS  $\mathcal{V}$  we construct a register transducer with substitution over ring  $\mathbb{Z}[x]$  that uses substitution once per run such that  $\mathcal{V}$  can reach  $\mathbf{0}$  from  $\mathbf{0}$  if and only if the register transducer returns a non-zero output on some input. We omit description of the states as it is clear.

**Description** of register transducer:

*Registers:*

$R_1$ : will store  $\mathbb{Z}[x]$  (*reachability test register*),

$R_1^{(aux)}$ : will store  $\mathbb{Z}$  (*auxiliary register*),

$R_2$ : will store  $\mathbb{Z}$  (*error register*),

$S_1, S_2, \dots, S_{dim}$ : will store  $\mathbb{Z}$  (*counter registers*).

*Input alphabet:*

$\Sigma$  = set of transitions of  $\mathcal{V}$ .

*Output function:*

$$\text{output} = \begin{cases} 0, & \text{if state is rejecting,} \\ R_1[x := S_1 + \dots + S_{dim}] \cdot R_2, & \text{else.} \end{cases}$$

**Invariants.** Observe, that run of reset VASS can be identified with a word over an alphabet consisting of its transitions. Registers after reading a run of  $\mathcal{V}$  of length  $n$  satisfy the following:

1.  $S_1, \dots, S_{dim} \in \mathbb{Z}$  are current coordinates of  $\mathcal{V}$ , viewed as  $\mathbb{Z}$  - VASS (i.e. can go below 0),
2.  $R_2 = 0$  iff some coordinate of  $\mathcal{V}$  went below 0 (hence  $R_2$  checks correctness of the run),
3.  $R_1[x := S_1 + \dots + S_{dim}] \neq 0$  iff current point is  $\mathbf{0}$ ,
  - a.  $R_1[x := i] = 0$  iff  $i \neq 0$  for integer  $i$  from range  $[0, n]$ ,
  - b.  $R_1^{(aux)} = n$ .



**Construction** of polynomial grammar with one substitution:

**Initial values** of registers:

1.  $S_1, \dots, S_{dim} = 0$
2.  $R_2 = 1$ :
3.  $R_1$  :
  - a.  $R_1 = 1$ ,
  - b.  $R_1^{(aux)} = 0$ .

**Update of registers:**

1.  $S_1, \dots, S_{dim}$  are incremented, decremented or reset to 0, according to read transition,
2.  $R_2$ :
  - a.  $R_2 = R_2 \cdot (S'_1 + 1) \cdot \dots \cdot (S'_{dim} + 1)$ , where  $S'_i$  is the value of  $S_i$  after the update.
3.  $R_1$  :
  - a.  $R_1 = R_1 \cdot (x - R_1^{(aux)})$  (recall  $x$  is a constant in  $\mathbb{Z}[x]$ ),
  - b.  $R_1^{(aux)} = R_1^{(aux)} + 1$ .

◀

### A.3 Proof of Theorem 16 without algebraic closure

In this section we present the proof of Theorem 16 without introducing the notion of algebraic closure of a field.

► **Observation 29** (Representing algebraic sets – not unique). There is a subtlety regarding representing algebraic sets. Algebraic sets are *not* represented by ideals of polynomials that zero on them ( $I(V)$ ) but by ideals that they are zeros of ( $I$  such that  $V = V(I)$ ) – these two notions coincide in case field is algebraically closed and radical of  $I$  is taken (taking radical of  $I$  does not change  $V(I)$ ) due to Fact 35. The former is a unique representation, but we do not choose it because of computability issues. It is not clear how to decide for a given ideal, even for a radical one, if it is of form  $I(V)$  (it might be merely contained in  $I(V)$  for  $V = V(I)$ ). Therefore, it is not clear how to enumerate such ideals and, if we chose it as the representation, algebraic sets.

This subtlety is relevant to Sublemma 2. Assume that  $V$  was given by an ideal that is smaller than  $I(V)$ . Then there are two options:

- test for condition (2) is passed, which gives a true positive (we show that in a moment), or
- test for condition (2) is failed, potentially giving false negative. Such event fortunately will be covered either by a true positive when enumerating  $V$  (again) by  $I(V)$  or by some previously enumerated true positive.

► **Observation 30.** Let algebraic set  $V$  be given by ideal  $I$ , i.e.  $V = V(I)$ . Then coordinate ring of  $V$  is a quotient of  $K[X]/I$ .

**Proof.** Straightforward from  $I \subseteq I(V)$ , which follows from definition of  $I(V)$ . ◀

Before we prove the main result of this section, let us state the following lemma.

► **Lemma 31.** ([10, Chapter 4, §8, Corollary 10 and remarks at the end of Chapter 4, §8]) *Let  $K$  be a computable field. Then every radical ideal  $I$  effectively admits a prime decomposition.*

Now we are ready to present the main proof.

**Proof of first bullet point.** Let us go through the algorithm for Sublemma 2 and see if indeed positives are true, even for  $V$  being represented by radical ideal  $I$  contained in  $I(V)$ . If  $A$  is zero, treated as a polynomial grammar over  $K[X]/I$ , the more it is when zero treated as a polynomial grammar over coordinate ring (Observation 30). Is crucial that when decomposing  $V$  into irreducibles, we use algorithm for prime decomposition (Lemma 31) of the ideal – then this ring has no zero divisors (Lemma 12) and hence decidability follows from Lemma 13. ◀

## A.4 Proof of Lemma 25

In this section we give the proof of Lemma 25 used in the proof of Theorem 27.

► **Lemma 25.** *Com-injective word substitutions induce automorphisms of field  $\mathbb{Q}(\tilde{a}, \tilde{a}^{\mathbb{Q}_{\geq 0}}, a \in \Sigma)$ .*

*Non-com-injective word substitutions do not induce automorphisms of any field that contains  $\mathbb{Q}(\tilde{\Sigma} \cup \bar{\Sigma})$ .*

► **Lemma 32.** *Let  $p : \Sigma \rightarrow \Sigma^*$  be a com-injective word substitution. The following two linear mappings of  $|\Sigma|$ -dimensional linear spaces are invertible:*

- (i)  $\tilde{w} \mapsto \widetilde{p(w)}$  of linear space of formal linear combinations of variables  $\tilde{\Sigma}$  with coefficients in the field  $\mathbb{Q}(\tilde{\Sigma})$ ,
- (ii)  $w \mapsto p(w)$  treated as a mapping of commutative words, extended to  $\mathbb{Q}$ -linear space  $\mathbb{Q}^{\Sigma}$

We illustrate the statement of above lemma with the following example.

► **Example 33.** Let  $\Sigma = \{a, b\}$  and  $p : a \mapsto ab, b \mapsto babb$ . Values of  $\mathbb{Q}(\tilde{\Sigma} \cup \bar{\Sigma})$ -linear mapping  $p$  in basis  $\tilde{\Sigma}$  are:

$$\begin{aligned} p(\tilde{a}) &= \tilde{a}\tilde{b} = \tilde{a}\tilde{b} + \tilde{b}, \\ p(\tilde{b}) &= \widetilde{babb} = \tilde{b}\tilde{a}\tilde{b}^2 + \tilde{a}\tilde{b}^2 + \tilde{b}\tilde{b} + \tilde{b}. \end{aligned}$$

Values of  $\mathbb{Q}$ -linear mapping  $p$  on basis  $\Sigma$  (coefficients are in exponents) are:

$$\begin{aligned} p(a) &= ab = a^1b^1, \\ p(b) &= babb = a^1b^3, \end{aligned}$$

(recall in (ii) outputted words are considered commutative) hence the corresponding matrices are  $\begin{bmatrix} \tilde{b} & \tilde{b}^2 \\ 1 & \tilde{a}\tilde{b}^2 + \tilde{b} + 1 \end{bmatrix}$  and  $\begin{bmatrix} 1 & 1 \\ 1 & 3 \end{bmatrix}$ . They are invertible: determinants are equal to, respectively,  $\tilde{a}\tilde{b}^3 + \tilde{b}$  and 2.

**Proof of Lemma 32.** (ii) is straightforward from definition of com-injectivity. Observe that mapping from (ii) is the same as mapping from (i) when 1 is substituted for variables from  $\bar{\Sigma}$ ; the same holds for their determinants, and hence if determinant for (ii) is a non-zero number, determinant for (i) is a non-zero polynomial. ◀

No we proceed towards the proof of Lemma 25. By *explicit form* of a homomorphism we mean a substitution that defines it; a homomorphism given in this form is said to be *given explicitly*. We write systems of equations as (name of equation: equation); we define subsystems by giving equation names.

**Proof of Lemma 25.** Assume substitution  $p$  is com-injective. To find the inverse substitution (if exists) we consider system of equations  $S = \{\tilde{\sigma} : \tilde{\sigma}' = \widetilde{p(\sigma)}\} \cup \{\bar{\sigma} : \bar{\sigma}' = \overline{p(\sigma)}\}$ .  $\bar{\Sigma}$ -subsystem of  $S$  is purely in variables  $\bar{\Sigma}$  hence we solve it first. Right hand sides are monomials and it can be seen as system of linear equations, where exponents play the role of coefficients from field  $\mathbb{Q}$ ; we solve it (solution exists – its existence is equivalent to (ii) of Lemma 32). This solution gives explicitly an automorphism of  $K(\bar{a}^{\mathbb{Q}}, a \in \Sigma)$  which naturally is also an automorphism of  $K(\tilde{a}, \bar{a}^{\mathbb{Q}_{\geq 0}}, a \in \Sigma)$ . We substitute this to  $\tilde{\Sigma}$ -subsystem and obtain a system of linear equations in variables  $\tilde{\Sigma}'$  and coefficients in  $K(\tilde{a}, \bar{a}^{\mathbb{Q}_{\geq 0}}, a \in \Sigma)$ ; this system is automorphic to  $\tilde{\Sigma}$ -subsystem of  $S$  and hence has a solution (due to (i) of Lemma 32). Its solution, together with solution of  $\bar{\Sigma}$ -subsystem, gives the right-inverse substitution explicitly. It is a two-sided inverse because both systems of equations were linear (in the mentioned sense).

The proof of the converse is analogous, which finishes the proof.  $\blacktriangleleft$

## A.5 Proof of Lemma 39

In this section we prove Lemma 39 used in the proof of Theorem 16. It is a well known result (e.g. implicit in [10, Chapter 5, §2 and §3 and §4]), but we did not find an explicit reference.

**Preliminaries.** By  $K[X]$  we denote over some field  $K$  and a finite set of variables  $X$ .

Every ideal  $I$  in ring  $K[X]$  induces a congruence  $\sim_I$  defined as  $f \sim_I g$  iff  $f - g \in I$ , for  $f, g \in K[X]$ . By  $K[X]/I$  we denote *quotient ring*  $K[X]/\sim_I$ . For  $f \in K[X]$ . A *radical of an ideal*  $I$  is the set  $\sqrt{I} = \{f \in K[X] \mid f^n \in I \text{ for some } n \geq 0\}$ ; it is an ideal. An ideal  $I$  is *radical* if  $f^n \in I$  implies  $f \in I$  for all polynomials  $f$  and  $n \geq 0$ , or in other word,  $I$  is equal to its radical. Let  $V$  be an algebraic set. A *coordinate ring* of  $V$  is the ring of polynomial functions from  $V$  to  $K$ . A ring is called a *computable ring* if its elements can be enumerated in a way that ring operations are computable functions.

► **Fact 34.** ([10, Remarks at the end of §2 of Chapter 4] ) *Given ideal  $I$  in  $K[X]$ , one can compute its radical.*

► **Fact 35** ([10, Chapter 4, §2, Theorem 7 (iii)]). *Let  $K$  be an algebraically closed field. Then mappings  $V \mapsto I(V)$ ,  $I \mapsto V(I)$  form a bijective correspondence between algebraic sets and radical ideals.*

**Representation.** For effectiveness of results, we assume that  $K$  is a *computable field*, i.e. its elements can be enumerated such that field operations are computable functions. Then we represent objects used by algorithms as follows:

- ideals are represented by finite sets of generators; every ideal admits such a representation due to Hilbert's Basis Theorem (Fact 8), and
- *algebraic sets* are represented by ideals that they are zeroes of, i.e. an algebraic set  $V$  is represented by  $I$  such that  $V = V(I)$ .

**Proof of Lemma 39.**

► **Fact 36.** *Let  $K$  be a computable, algebraically closed field. Given algebraic set  $V$ , one can compute  $I(V)$ .*

**Proof.** Let  $V$  be represented by ideal  $I$ . Then  $I(V)$  is the radical of  $I$  (Fact 35); it can be computed (Fact 34).  $\blacktriangleleft$

► **Fact 37.** ([10, Chapter 5, §3, Proposition 5]) *Quotient rings are computable rings. More precisely, given ideal  $I$ , ring  $K[X]/I$  is a computable ring.*

► **Fact 38.** ([10, Chapter 5, §2, remark about Theorem 7 after Definition 8]) For an algebraic set  $V$ , its coordinate ring is isomorphic to  $K[X]/I(V)$ .

► **Lemma 39.** Let  $K$  be an algebraically closed field. Then, given algebraic set  $V$ , one can compute its coordinate ring and it is a computable ring.

**Proof.** Coordinate ring is isomorphic to  $K[X]/I(V)$  (Fact 38). It can be computed from  $V$ , as we can compute  $I(V)$  (Fact 36). It is a computable ring (Fact 37). ◀