The Hilbert Method for Transducer Equivalence

Mikołaj Bojańczyk[1], University of Warsaw

This Hilbert Method, as discussed in this paper, is the idea to encode combinatorial objects used by automata, such as words or trees, into algebraic objects, such as numbers and polynomials, and to reason about the latter using results about polynomials like Hilbert's Basis Theorem. A typical application of the method is deciding of equivalence for string-to-string transducers that use registers.

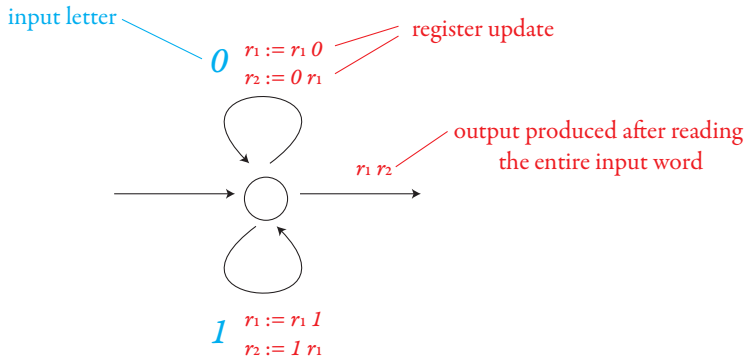
1. INTRODUCTION

The idea to apply Hilbert's Basis Theorem to reason about combinatorial objects like words or trees dates back at least to the solution of the Ehrenfeucht Conjecture by Albert and Lawrence [Albert and Lawrence 1985]. This method has gained popularity in the automata community, especially as a tool for reasoning about transducers, see [Seidl et al. 2015], [Filiot and Reynier 2017] and [Benedikt et al. 2017]. This paper is an introduction to the method, and is based closely on [Bojańczyk and Czerwiński , Chapter 11].

Register transducers

The main application of the Hilbert method is going to be equivalence for transducers which use registers to store parts of the output. We begin with an example of such a transducer, which inputs and outputs strings[2]. The transducer has finitely many states and registers. The idea is that the registers store parts of the output word. Whenever the transducer reads an input letter, it updates its state like a deterministic finite automaton, and updates the registers using concatenation. For the moment, we describe the model only by example. A more formal definition of the model is given in Section 3.

Example 1.1. The following picture shows an example of a register transducer, which inputs a word $a_1 \cdots a_n \in \{0,1\}^*$ and outputs the palindrome $a_1 \cdots a_n a_n \cdots a_1$.



When the transducer reads a letter, it appends it to the first register and prepends it to the second register. The registers start empty, and the output of the transducer is the concatenation of the two registers.

[1]Author supported by the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (ERC consolidator grant LIPA, agreement no. 683080).

[2]In the terminology of [Alur and Cerný 2010] and [Filiot and Reynier 2017], this model is called a *copyful streaming string transducer*.

Equivalence. Suppose that we have two string-to-string functions

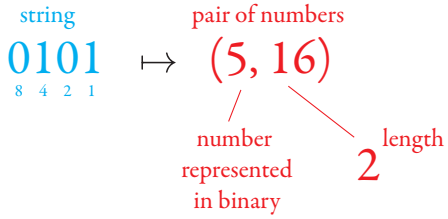
$$f, g : \Sigma^* \to \Gamma^*$$

recognised by register transducers, and we want to decide if they are equal. To solve this problem, we will encode words as integers, yielding two string-to-number functions

$$f', g' : \Sigma^* \to \mathbb{Z},$$

and then we will decide if the function $f' - g'$ produces 0 for all inputs.

The representation of strings as numbers is simply binary representation (or ternary, etc. depending on the alphabet size). Apart from the actual number, we also store the length of the string (in exponential format), as illustrated below:



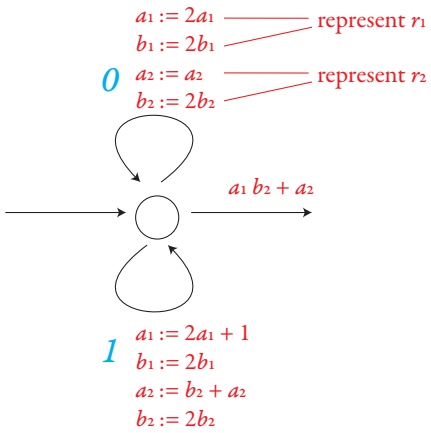
The key point about this representation is that if we know the representations (a_1, b_1) and (a_2, b_2) of strings w_1, w_2, then we can recover the representation (a, b) of the concatenation $w_1 w_2$ by using only addition and multiplication:

$$a = a_1 b_2 + a_2 \qquad b = b_1 b_2.$$

Using this representation, a string-to-string transducer $\mathcal{A}$ can be converted into a string-to-number transducer where each register of $\mathcal{A}$ is represented by two registers storing the corresponding numbers[3].

Example 1.2. Consider the palindrome string-to-string transducer from Example 1.1. The corresponding string-to-number transducer is illustrated below:

[3]A small issue with the representation of strings by numbers is that the final output does not account for leading zeros. This issue can be solved in three ways: (i) add a leading 1 to every output, which does not affect equivalence for string-to-string transducers; (ii) if the output word w is represented by (a, b), then output the pair (a, b) instead of just a; (iii) output $a + b$ instead of just a.



Using the representation described above, we can reduce the equivalence problem for string-to-string transducers to the equivalence problem for string-to-number transducers. Importantly, the string-to-number transducers use only addition and multiplication (i.e. polynomials) to update their registers. This will allow us to use algebraic methods, e.g. Hilbert's Basis Theorem, to decide equivalence. These methods are described in the next section.

2. POLYNOMIAL GRAMMARS

Let $\mathbb{F}$ be a ring. Later on, we will impose more restrictions on $\mathbb{F}$, in particular it will need to be a field, but for the moment any ring is enough, e.g. the ring of integers. In the definitions below, it will be convenient to use polynomial functions from vectors to vectors. Define a *polynomial function* to be a function of the form

$$p : \mathbb{F}^n \to \mathbb{F}^k \qquad \text{for } n, k \in \{0, 1, 2, \ldots\}$$

which is given by k polynomials representing the coordinates of the output vector, each one with n variables representing the coordinates of the input vector. The coefficients of the polynomials are taken from $\mathbb{F}$.

For the purposes of this section, instead of transducers, it will be more convenient to use grammars. The connection with transducers will be described in Section 3.

Polynomial grammars

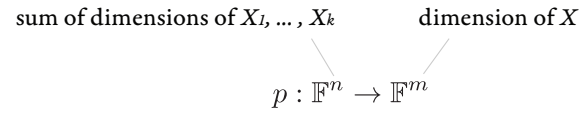
A polynomial grammar is a variant of a context free grammar. It generates numbers, or tuples of numbers, from the fixed ring $\mathbb{F}$ (as opposed to words) and uses polynomial functions in the rules (as opposed to concatenation). Also, nonterminals are allowed to generate tuples of numbers.

Definition 2.1. A *polynomial grammar* over a ring $\mathbb{F}$ consists of

— a set $\mathcal{X}$ of nonterminals, each one with an assigned *dimension* in $\{1, 2, \ldots\}$;
— a designated starting nonterminal;
— a finite set of productions of the form

$$X \to p(X_1, \ldots, X_k)$$

where $k \in \{0, 1, \ldots\}$, $X_1, \ldots, X_k, X$ are nonterminals, and



is a polynomial function.

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

$$X \to p(X_1, \ldots, X_k)$$

is a production and we already know that vectors $v_1, \ldots, v_k$ are generated by the terminals $X_1, \ldots, X_k$ respectively. Then the vector $p(v_1, \ldots, v_k)$ is generated by nonterminal X. The induction base is the special case of $k = 0$, where the polynomial p is a constant.

Example 2.2. In the following examples, the ring is the integers. This grammar (there is only the starting nonterminal, which has dimension one) generates all nonzero even natural numbers

$$X \to 2 \qquad X \to X + 2.$$

If we replace $X+2$ by $X+X$, then we get the powers of two. If we replace $X+2$ by square X^2, the grammar generates numbers of the form 2^{2^n}. We can also generate factorials. Apart from the starting nonterminal X, we have a nonterminal Y of dimension two which generates pairs of the form $(n, n!)$. The crucial rule is this:

$$Y \to p(Y) \qquad \text{where } p \text{ is defined by } (a, b) \mapsto (a + 1, (a + 1) \cdot b).$$

The remaining rules are $Y \to (1, 1)$ and $X \to \pi_1(Y)$ where π_1 is defined by $(a, b) \mapsto a$.

As usual, one can adopt an alternative fix-point view on grammars. We present this view since it will be used in the proof of Theorem 2.3. Define a *solution* to a grammar to be a function η which maps each nonterminal X of dimension n to a subset of $\mathbb{F}^n$, and which is consistent with all of the productions in the sense that

$$\underbrace{X \to p(X_1, \ldots, X_k)}_{\text{is a production}} \qquad \text{implies} \qquad \eta(X) \supseteq p(\eta(X_1) \times \cdots \times (\eta(X_k))$$

It is not difficult to see that the function which maps a nonterminal to the set of vectors generated by it is a solution, and it is the least solution with respect to coordinate-wise inclusion.

Zeroness is decidable

Here are several decision problems that can be studied for polynomial grammars:

— *Nonemptiness.* Is the generated language nonempty?
— *Universality.* Is the generated language equal to $\mathbb{F}$?
— *Membership.* Does the generated language contain a given number $a \in \mathbb{F}$?
— *Zeroness.* Is the generated language contained in $\{0\}$?

The nonemptiness problem is decidable in polynomial time for the usual reasons (search for some derivation tree, the choice of ring plays no role) and the universality problem is undecidable for the usual reasons (assuming e.g. the ring of integers). Membership is undecidable, by a reduction from the problem "does a given weighted automaton produce 0 for some input", see [Gimbert and Oualhadj 2010, Section 2.1] for a

discussion of this problem already in the case of probabilistic automata, or [Bojańczyk and Czerwiński , Chapter 8.3].

The interesting problem is zeroness.

THEOREM 2.3. *Assume the ring of integers. One can decide if the language of a polynomial grammar is contained in* $\{0\}$.

The rest of Section 2 is devoted to proving the above theorem. The proof is a combination of the techniques from [Benedikt et al. 2017] and [Seidl et al. 2015]. Without loss of generality we assume that the initial nonterminal has dimension one. Indeed, the zeroness problem for grammars which generate tuples of integers reduces to the zeroness problem for grammars which generated integers, because one can add a new initial nonterminal, which generates the sum of squares of all coordinates in the old initial nonterminal.

The (complement of the) zeroness problem mentioned in the above theorem is clearly semi-decidable: one can enumerate all derivations of the grammar, and stop when a derivation is found that generates a nonzero number. The interesting part of the theorem is that the problem is also co-semi-decidable, i.e. one can find a finite witness proving that the generated language is contained in $\{0\}$. For this, we use the Hilbert Basis Theorem. The general idea is to show that a grammar is generates only the zero vector if and only if there is a closed solution (to be defined below, using polynomials) which is consistent with the grammar and also generates only the zero vector. Thanks to the Hilbert Basis Theorem, closed solutions can be represented in a finite way, which leads to a semi-algorithm for deciding zeroness, which searches through all possible closed solutions.

Definable reals. Although we are ultimately interested in integers, in the proof we will use a different ring, in fact a field, namely the definable reals[4]. All integers are definable reals, so polynomial grammars over the definable reals generalise polynomial grammars over the integers. The reason why we use the definable reals is that their first-order theory is decidable, unlike for the integers.

Definition 2.4. A real number a is called *definable* if it is the unique real satisfying

$$(\mathbb{R}, +, \times) \models \varphi(a)$$

for some formula φ of first-order logic.

For example 0 is a definable real, because it is the neutral element of addition. The same goes for 1. Definable reals are clearly seen to be a field, i.e. they are closed under addition, multiplication and inverses. In particular, all rational numbers are definable. Some irrational numbers are also definable, for example $a = \sqrt{2}$ is the unique number that satisfies

$$a^2 = 2 \wedge a > 0.$$

Apart from being a field, the key property of the definable reals is that they can be represented in a finite way (by definition), and operations on them (such as addition, multiplication, or testing equality) are computable. For computability of equality tests (since different formulas might define the same number), we use the decidability of the first-order theory of $(\mathbb{R}, +, \times)$, which was proved by Tarski [Tarski 1951].

For the rest of this section, $\mathbb{F}$ is the field of definable reals. We will prove a strengthening of Theorem 2.3, where the ring of interest is not the integers, but the field of definable reals.

[4]The definable reals are the same as the algebraic numbers with no imaginary part, but this observation, or the definition of algebraic numbers, is not used below.

Hilbert's Basis Theorem. The key result in the proof of Theorem 2.3 is Hilbert's Basis Theorem about ideals. Let X be a set of variables. We write $\mathbb{F}[X]$ for the set of polynomials with variables X and coefficients from $\mathbb{F}$. Define an *ideal* (of polynomials) to be a set $I \subseteq \mathbb{F}[X]$ with the following closure properties:

$$\underbrace{p, q \in I \quad \Rightarrow \quad p + q \in I}_{\text{addition inside } I} \qquad \underbrace{p \in I, q \in \mathbb{F}[X] \quad \Rightarrow \quad pq \in I}_{\text{multiplication by arbitrary polynomials}}.$$

If $P \subseteq \mathbb{F}[X]$ is a set of polynomials, then the ideal generated by P, denoted by $\langle P \rangle$, is the set of polynomials of the form

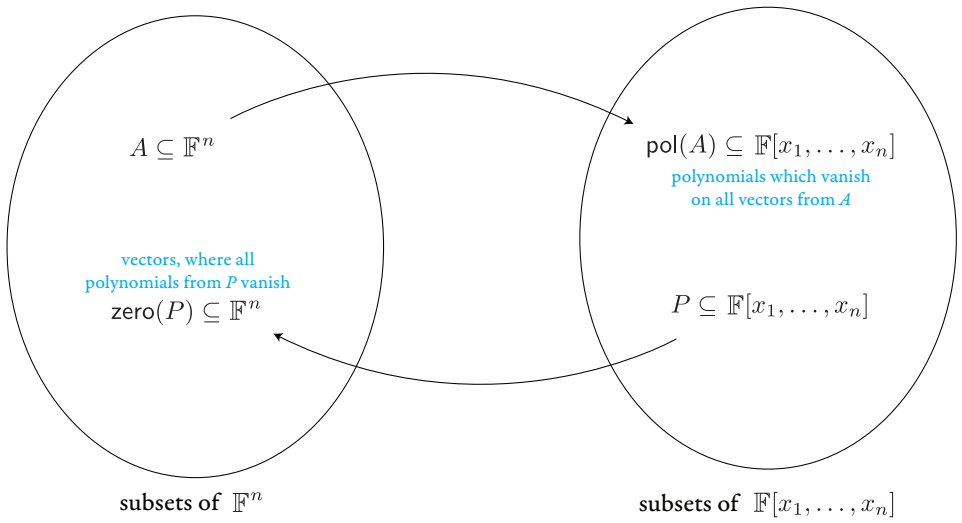
$$p_1 q_1 + \cdots + p_n q_n \qquad \text{where } p_i \in P, q_i \in \mathbb{F}[X].$$

This is the inclusion-wise least ideal that contains P. Hilbert's Basis Theorem says that every ideal is finitely generated.

THEOREM 2.5 (HILBERT'S BASIS THEOREM). *If X is a finite set of variables, then every ideal in $\mathbb{F}[X]$ is finitely generated, i.e. of the form $\langle P \rangle$ for some finite set of polynomials.*

Hilbert's Basis Theorem is not specific to the definable reals, it holds for any ring $\mathbb{F}$ where ideals are finitely generated.

Algebraic closure. We say that a polynomial function $p : \mathbb{F}^n \to \mathbb{F}$ *vanishes* on a set $X \subseteq \mathbb{F}^n$ if p is the constant zero over this set. For a fixed dimension n, consider the following two operations pol and zero which go from sets of vectors to sets of polynomials, and back again:



The picture above is a special case of what is known as a *Galois connection*. Note that the operation pol produces only ideals. For a set of vectors $A \subseteq \mathbb{F}^n$, define its *closure* as follows:

$$\overline{A} \quad \stackrel{\text{def}}{=} \quad \mathsf{zero}(\mathsf{pol}(A)).$$

In other words $\overline{A}$ is the common zeros of polynomials that vanish on A. The operation $A \mapsto \overline{A}$ is easily seen to be a closure operator, in the sense that it can only add ele-

ments to the set, it is monotone with respect to inclusion, and applying the closure a second time adds nothing. A *closed set* is defined to be any set obtained as the closure, equivalently a closed set is the set of common zeros for some ideal of polynomials. By Hilbert's Basis Theorem, ideals of polynomials can be represented by giving a finite basis. Therefore, closed sets can be represented by giving a finite basis for the corresponding ideal of polynomials.

Example 2.6. Suppose that the dimension is $n = 1$. A univariate polynomial in $\mathbb{F}[x]$ has infinitely many zeros if and only if it is constant zero. Therefore $\mathsf{pol}(A)$ contains only the constant zero polynomial whenever A is infinite. This means that the algebraic closure of any infinite set $A \subseteq \mathbb{F}$ is the whole space $\mathbb{F}$. On the other hand, when A is finite, then there is a polynomial which vanishes exactly on the points from A, and therefore $\overline{A} = A$. For higher dimensions, an example of a closed set is the unit circle, because in this case the ideal $\mathsf{pol}(A)$ is generated by the polynomial $x^2 + y^2 = 1$.

The following lemma is the key to deciding if a grammar generates only zero.

LEMMA 2.7. *Let $\mathcal{G}$ be a polynomial grammar with nonterminals $\mathcal{X}$, and let η be a solution, not necessarily the least solution. Then $\overline{\eta}$, defined by $X \in \mathcal{X} \mapsto \overline{\eta(X)}$ is also a solution.*

PROOF. We first prove two inclusions, (1) and (2), which show how closure interacts with polynomial images and Cartesian products. The first inclusion is about polynomial images:

$$p(\overline{A}) \subseteq \overline{p(A)} \qquad \text{for every } A \subseteq \mathbb{F}^n \text{ and polynomial } p : \mathbb{F}^n \to \mathbb{F}^m. \tag{1}$$

To prove the above inclusion, we need to show that if a polynomial vanishes on $p(A)$, then it also vanishes on $p(\overline{A})$. Suppose that a polynomial q vanishes on $p(A)$. This means that the polynomial $q \circ p$ vanishes on A, which means that $q \circ p$ also vanishes on $\overline{A}$, by definition of closure. Therefore q, vanishes on $p(\overline{A})$.

The second inclusion is about Cartesian products:

$$\overline{A} \times \overline{B} \subseteq \overline{A \times B} \qquad \text{for every } A \subseteq \mathbb{F}^n \text{ and } B \subseteq \mathbb{F}^m. \tag{2}$$

We need to show that if a polynomial vanishes on $A \times B$, then it also vanishes on $\overline{A} \times \overline{B}$. Suppose that q vanishes on $A \times B$. Take some $b \in B$. The polynomial $q(\_, b)$ vanishes on A, and therefore it vanishes on $\overline{A}$ by definition of closure. Therefore, q vanishes on $\overline{A} \times B$. Applying the same reasoning again, we get that q vanishes on $\overline{A} \times \overline{B}$, proving the inclusion (2).

We are now ready to prove the lemma. Let η be some solution to the grammar. Take some rule $X \to p(X_1, \ldots, X_n)$ in the grammar $\mathcal{G}$. The following reasoning shows that $\overline{\eta}$ is consistent with the rule, and therefore $\overline{\eta}$ is a solution to the grammar by arbitrary choice of the rule.

$$\begin{aligned}
p(\overline{\eta}(X_1), \ldots, \overline{\eta}(X_n)) &= \text{by definition of } \overline{\eta} \\
p(\overline{\eta(X_1)}, \ldots, \overline{\eta(X_n)}) &\subseteq \text{repeated application of (2)} \\
\overline{p(\eta(X_1) \times \cdots \times \eta(X_n))} &\subseteq \text{by (1)} \\
\overline{p(\eta(X_1) \times \cdots \times \eta(X_n))} &\subseteq \text{because } \eta \text{ is solution and closure is monotone} \\
\overline{\eta(X)} &= \text{by definition of } \overline{\eta} \\
\overline{\eta}(X)
\end{aligned}$$

This completes the proof of the lemma. Note that the proof is not very specific to polynomials and definable reals, and it would work for more abstract notions of algebra, as will be defined in Section 3. □

We now complete the proof of Theorem 2.3, about deciding zeroness for polynomial grammars. We use two semi-algorithms. By enumerating derivations, there is an algorithm that terminates if and only if the grammar generates some nonzero vector. We now give an algorithm that terminates if and only if the grammar generates a language contained in $\{0\}$. The algorithm simply enumerates through all *closed solutions* to the grammar, i.e. solutions which map each nonterminal to a closed set. By Lemma 2.7, the generated language is contained in $\{0\}$ if and only if there is a an assignment η which maps nonterminals to closed sets such that:

(1) η is a solution to the grammar; and
(2) η maps the starting nonterminal to a subset of $\{0\}$.

We assume that a closed set A is represented by a finite basis of the ideal $\mathsf{pol}(A)$. By Hilbert's Basis Theorem, we can enumerate candidates for η, by using finite sets of polynomials to represent closed sets. It remains to show that, given η, one can check if conditions (1) and (2) above are satisfied. To do this, we use decidability of the first-order theory of the reals. (There are more efficient algorithms that have been developed in the area computational algebraic geometry, see [Bigatti et al. 2017, Theorem 11]). The following lemma shows that inclusion is decidable for closed sets, assuming that closed sets are represented by finite sets of polynomials which vanish on them.

LEMMA 2.8. *Given finite sets of polynomials $P, Q \subseteq \mathbb{F}[x_1, \ldots, x_n]$ of polynomials, one can decide if* $\mathsf{zero}(P) \subseteq \mathsf{zero}(Q)$.

PROOF. Our goal is to decide

$$\forall \overline{a} \in \mathbb{F}^n \quad \bigwedge_{p \in P} p(\overline{a}) = 0 \quad \Rightarrow \quad \bigwedge_{q \in Q} q(\overline{a}) = 0. \tag{3}$$

Tarski's algorithm for the first-order theory of the reals gives an answer to the same question, except that $\overline{a}$ ranges over not necessarily definable reals:

$$\forall \overline{a} \in \mathbb{R}^n \quad \bigwedge_{p \in P} p(\overline{a}) = 0 \quad \Rightarrow \quad \bigwedge_{q \in Q} q(\overline{a}) = 0. \tag{4}$$

However, the answers to the questions (3) and (4) are the same. The reason is that the set of real vectors $\overline{a} \in \mathbb{R}^n$ which violate (4) can be defined in first-order logic. Every nonempty $X \subseteq \mathbb{R}^n$ that is definable in first-order logic contains at least one vector where all coordinates are definable reals; this is easily seen using quantifier elimination. It follows that a counter-example to (4) can always be made definable, thus proving the equivalence of (3) and (4). □

From the above lemma, it follows that inclusion on closed sets is decidable, and hence condition (2) is decidable. Condition (1) boils down to testing a finite number of inclusions of the form

$$p^{-1}(A) \supseteq A_1 \times \cdots \times A_n \tag{5}$$

for closed sets $A, A_1, \ldots, A_n$ and some polynomial p. It is not hard to see that if A and B are closed sets, then the same is true for $A \times B$ and $p^{-1}(A)$, and furthermore the appropriate representations can be computed. Therefore (5) also boils down to an inclusion check on closed sets.

This completes the proof of Theorem 2.3.

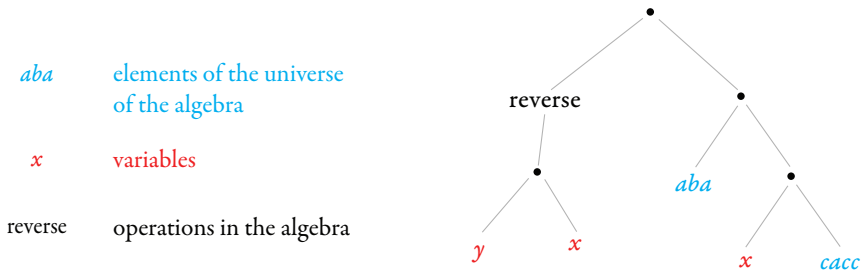
3. APPLICATION TO EQUIVALENCE OF REGISTER TRANSDUCERS

In this section, we use Theorem 2.3 to decide equivalence for transducers as discussed in the beginning of the paper. Instead of considering only string-to-string transducers (and the string-to-number transducers that appear in the proof), we consider a more general model where the inputs are strings, but the outputs are elements of some abstract algebra. By algebra, we mean the notion from universal algebra, i.e. an *algebra* is defined to be a set equipped with some operations. Examples include

$$(\mathbb{Z}, +, \times) \qquad (\{a, b\}^*, \cdot).$$

We adopt the convention that boldface letters like $\mathbf{A}$ and $\mathbf{B}$ range over algebras, and the universe (the underlying set) of an algebra $\mathbf{A}$ is denoted by A.

Abstract algebras also have a notion of polynomial, which instantiates to the usual notion of polynomials when the algebra in question is a ring. Define a *polynomial* with variables X in an algebra $\mathbf{A}$ to be a term built out of operations from the algebra, variables from X and elements of the universe. Here is a picture of a polynomial with variables $\{x, y\}$ in an algebra where the universe is $\{a, b, c\}^*$ and the operations are concatenation (binary) and reverse (unary):



We write $\mathbf{A}[X]$ for the set of polynomials over variables X in the algebra $\mathbf{A}$. Such a polynomial represents a function of type $A^X \to A$ in the natural way. We extend this notion to function of type $A^n \to A^m$ by using m-tuples of polynomials with n variables.

Example 3.1. If the algebra is $(\mathbb{F}, +, \times)$, then the polynomials are the polynomials in the usual sense, e.g. a binary polynomial is

$$x^2 + 3x^3y^4 + 7.$$

If we remove multiplication, but we add the infinite family of scalar multiplications $\{x \mapsto ax\}_{a \in \mathbb{F}}$, then the polynomials are exactly the affine functions.

Register transducers over an algebra

We are now ready to define the abstract model of string-to-algebra transducers.

Definition 3.2 (*Register transducer*). Let $\mathbf{A}$ be an algebra. The syntax of a *register transducer over* $\mathbf{A}$ consists of:

— a finite input alphabet Σ;
— a finite set R of registers;
— a finite set Q of states;
— an initial configuration in $Q \times A^R$;
— a transition function $\delta : Q \times \Sigma \to Q \times (\mathbf{A}[R])^R$;
— an output dimension n and an *output function* $F : Q \to (\mathbf{A}[R])^n$.

The semantics of the transducer is a function of type $\Sigma^* \to A^n$ defined as follows. The transducer begins in the initial configuration. After reading each letter, the configuration is updated according to

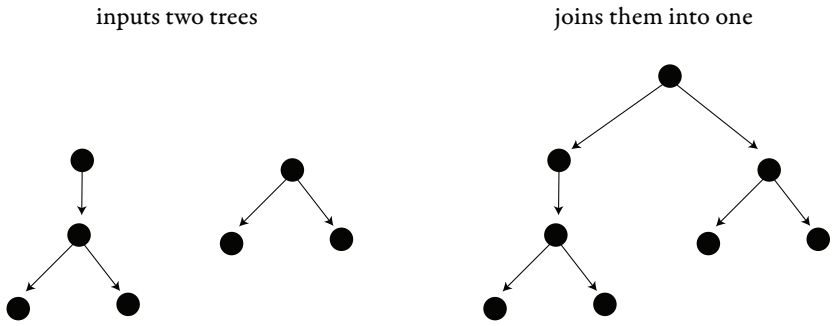
$$(q, v) \overset{a}{\mapsto} (p, f(v)) \qquad \text{where } \delta(q, a) = (p, f).$$

If the configuration after reading the entire input is (q, v), then the output of the transducer is obtained by applying $F(q)$ to v.

Example 3.3. Consider the algebra whose universe is $\{0, 1\}$ and which has no operations. Register transducers over this algebra are the same thing as regular languages.

Example 3.4. Consider the algebra whose universe is Σ^* for some finite set Σ, and which has one binary operation for concatenation. Register transducers over this algebra are the same thing as the (copyful) string-to-string transducers described in Section 1.

Example 3.5. Consider the algebra where the universe is binary trees (viewed as directed graphs, with edges directed away from the root), and which has one binary operation depicted as follows:



Register transducers over this algebra are a form of string-to-tree mso transductions, see [Boiret et al. 2018, Section 3]. For example, one can write a register transducer over this algebra, with input alphabet $\{a\}$, which maps a word a^n to a balanced binary tree of depth n.

Equivalence for register transducers. The purpose of the polynomial grammars and their zeroness problem is to decide equivalence of register transducers over the integers, and for algebras that reduce to them. We begin with the result for the integers, and then discuss how it can be extended to other algebras.

THEOREM 3.6. *The following problem is decidable:*

— **Input.** *Two functions $\Sigma^* \to \mathbb{Z}^n$ recognised by register transducers over $(\mathbb{Z}, +, \times)$;*
— **Question.** *Are the functions equal?*

The result was first shown in [Benedikt et al. 2017, Theorem 4], where the proof was also based on the Hilbert Basis Theorem. Similar ideas appeared also in [Filiot and Reynier 2017], which in turn was based on the decision procedure for HDT0L systems from [Culik II and Karhumäki 1986]. The proof in [Benedikt et al. 2017] uses the linear structure of transducers with string inputs, while the proof below, which is inspired by [Seidl et al. 2015], does not need string inputs and readily generalises to tree inputs. A lower bound of Ackermann time is given in [Benedikt et al. 2017, Theorem 1].

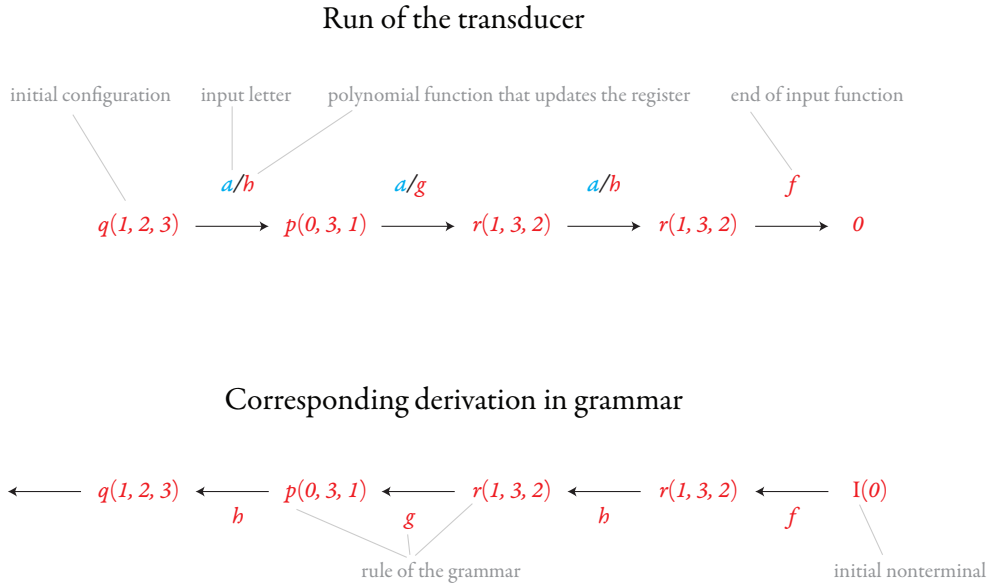


Fig. 1. A grammar derivation corresponding to a run of a transducer. Note that since the transducer inputs words, the derivation of the grammar is not branching (i.e. the grammar is a *linear* grammar).

PROOF. Suppose that the input functions are f, g. By doing a natural product construction, we can compute a register transducer that recognises the difference function $f - g$. Therefore, the problem boils down to deciding if a function h, e.g. the difference, is constant equal to zero. For this we use a grammar. Suppose that h is recognised by a register transducer with states Q and n registers. Define a grammar where the nonterminals are

$$\underbrace{Q}_{\text{dimension } n} + \underbrace{I}_{\text{dimension } 1} .$$

The starting nonterminal is I. By copying the transitions of the transducer, we can write the rules of the grammar so that nonterminal q generates exactly those tuples $\overline{a} \in \mathbb{F}^n$ such that configuration $(q, \overline{a})$ can be reached over some input. By using the output function of the transducer, we can ensure that the starting nonterminal produces exactly the outputs of the transducer. This construction is illustrated in Figure 1. Therefore, the language defined by the grammar is contained in $\{0\}$ if and only if the transducer can only produce 0, and the former is decidable by Theorem 2.3.

The same proof would work if, instead of the ring of integers, we would use the field of definable reals. □

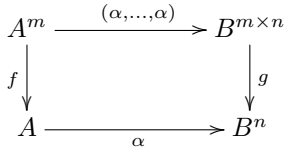
Other algebras. In Theorem 3.6, we showed that equivalence is decidable for register transducers over the ring of integers (or the field of definable reals). From this we can infer decidability for other algebras, by coding them into numbers according to the following definition.

Definition 3.7. Let $\mathbf{A}$ and $\mathbf{B}$ be algebras. We say that $\mathbf{A}$ can be simulated by polynomials of $\mathbf{B}$ (no relation to polynomial time computation) if there is some dimension

n and an injective function

$$\alpha : A \to B^n$$

with the following property. For every operation $f : A^m \to A$ in the algebra $\mathbf{A}$, there is a polynomial $g : B^{m \times n} \to B^n$ in the algebra $\mathbf{B}$ which makes the following diagram commute

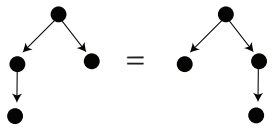


It is easy to see that if $\mathbf{A}$ can be simulated by polynomials of $\mathbf{B}$, then decidability of equivalence of register transducers over $\mathbf{B}$ implies decidability equivalence of register transducers over $\mathbf{A}$.

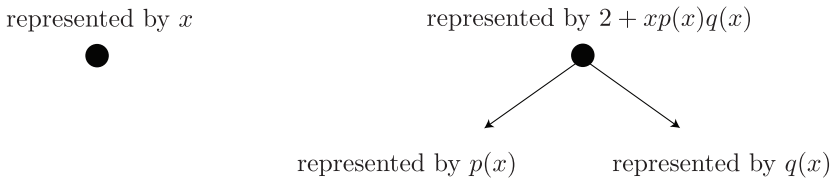
Example 3.8. In Section 1, we show how words can be simulated by pairs of numbers. In the language of Definition 3.7, we have shown that the algebra $(\Sigma^*, \cdot)$ is simulated by polynomials of $(\mathbb{Z}, +, \times)$. By Theorem 3.6 it follows that equivalence is decidable for register transducers over $(\Sigma^*, \cdot)$, which are the same as copyful streaming string transducers as mentioned in Example 3.4. This result was first shown in [Filiot and Reynier 2017, Theorem 3], using a reduction to equivalence of HDT0L systems. Since decidability for HDT0L systems was proved using Hilbert's Basis Theorem, the argument described in this paper cannot be seen as anything new.

Example 3.9. As mentioned just after Theorem 3.6 and before its proof, the theorem also would works for register transducers with tree inputs (the derivation tree in a context-free grammar can be seen as a tree input). Combining this the observations from Example 3.8, we see that equivalence is decidable for deterministic tree-to-string register transducers, which process the input bottom-up, storing parts of the output string in their registers. This result was proved in [Seidl et al. 2015]. The proof from [Seidl et al. 2015] is the foundation of the presentation in this paper.

Example 3.10. Let $\mathbf{A}$ be the algebra of trees discussed in Example 3.5. The difficulty is that the trees are unordered, in the sense that there is no order on siblings. For example, the following trees are equal



Ordered trees can be encoded as strings (e.g. using XML encoding), and these can be encoded as numbers. For unordered trees, the question is harder, since the encoding should be invariant under reordering the siblings. A solution, which comes from [Boiret et al. 2018, Section 3.1], is to code unordered trees as univariate polynomials with integer coefficients, as described by the following inductive definition:



It turns out that the above encoding is injective (the proof uses Eisenstein's criterion). Furthermore, the target of the encoding – the polynomial ring $\mathbb{Z}[x]$ – is subject to the same methods as were used in the proof of Theorem 2.3, in particular equivalence is decidable for register transducers over this ring. It follows that equivalence is decidable for register transducers that manipulate unordered trees. Extending these techniques, one can prove that equivalence is decidable for MSO transductions from unordered trees to unordered trees, see [Boiret et al. 2018, Section 3.3].

ACKNOWLEDGMENTS

I would like to thank Wojciech Czerwiński for helpful comments about a draft of this paper.

REFERENCES

M H Albert and J Lawrence. 1985. A proof of Ehrenfeucht's Conjecture. *Theoretical Computer Science* 41 (1985), 121–123.

Rajeev Alur and Pavol Cerný. 2010. Expressiveness of streaming string transducers. *FSTTCS* (2010).

Michael Benedikt, Timothy Duff, Aditya Sharad, and James Worrell. 2017. Polynomial automata - Zeroness and applications. *LICS* (2017), 1–12.

Anna M Bigatti, Philippe Gimenez, and Eduardo Sáenz-de Cabezón (Eds.). 2017. *Computations and Combinatorics in Commutative Algebra*. Lecture Notes in Mathematics, Vol. 2176. Springer International Publishing, Cham.

Adrien Boiret, Radoslaw Piórkowski, and Janusz Schmude. 2018. Reducing Transducer Equivalence to Register Automata Problems Solved by" Hilbert Method". In *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Mikołaj Bojańczyk and Wojciech Czerwiński. Automata Toolbox. https://www.mimuw.edu.pl/~bojan/upload/reduced-may-25.pdf

Karel Culik II and Juhani Karhumäki. 1986. The equivalence of finite valued transducers (on HDT0L languages) is decidable. *Theoretical Computer Science* 47 (1986), 71–84.

Emmanuel Filiot and Pierre-Alain Reynier. 2017. Copyful Streaming String Transducers. In *International Workshop on Reachability Problems*. Springer, 75–86.

Hugo Gimbert and Youssouf Oualhadj. 2010. Probabilistic automata on finite words: Decidable and undecidable problems. In *International Colloquium on Automata, Languages, and Programming*. Springer, 527–538.

Helmut Seidl, Sebastian Maneth, and Gregor Kemper. 2015. Equivalence of Deterministic Top-Down Tree-to-String Transducers is Decidable. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 943–962.

Alfred Tarski. 1951. A Decision Method for Elementary Algebra and Geometry.