University of Warsaw

Faculty of Mathematics, Informatics and Mechanics

Janusz Schmude

Equivalence problem of transductions: algorithms based on commutative algebra

PhD dissertation

Supervisor:
prof. dr hab. Mikołaj Bojańczyk
Institute of Informatics
University of Warsaw

November 2021

Author's declaration:
I hereby declare that this dissertation is my own work.

November 3, 2021 ....J. Schmude........

Janusz Schmude

Supervisor's declaration:
The dissertation is ready to be reviewed.

November 3, 2021 ....

prof. dr hab. Mikołaj Bojańczyk

Acknowledgments

I thank my supervisor, Mikołaj Bojańczyk, for being a great mentor and an empathetic person at the same time. I thank all my lecturers and colleagues who spent their time on teaching me. I also thank my colleagues, especially Rafał Stefański, for many fruitful discussions (not only the research-oriented ones).

I thank my bachelor and master studies supervisors, Stanisław Kasjan and Grzegorz Bobiński, for taking care of my mathematical development. I also thank all my mathematics teachers, especially Włodzimierz Obremski, Barbara Kot, and my greatest mentor, Henryk Pawłowski (who is not with us anymore, but who we will always remember) for piquing my curiosity and showing me the most interesting aspects of mathematics.

I express gratitude to my parents. You laid the foundations of who I am. I also thank my brother for stimulating my mathematical development during school times.

I thank all my friends for their support during the hard times. Above all, I thank my beloved wife Daria – for who you are, for your great love. Finally, I thank my little daughter Tosia – you are a treasure no one can measure.

Abstract

The main goal of this thesis is to apply the Hilbert Method in order to find decision procedures for the equivalence problem for various classes of transformations of combinatorial objects, like words, trees, or graphs. The Hilbert Method, roughly speaking, amounts to finding a register transducer model that captures the desired class of transformations, and encoding combinatorial objects used by this model into integers, in a way that the basic combinatorial operations (like, for example, word concatenation) can be simulated with the use of algebraic operations of addition and multiplication.

While proving our main results, we extend the Hilbert Method by proving the following lemmas: (1) the combinatorial objects may be encoded into any computable reduced ring, (2) if the combinatorial objects are encoded into a field (note that any field is a reduced ring), then also the division operation can be used to simulate the combinatorial operations, (3) if the combinatorial objects are encoded into a ring of polynomials (note that it can be embedded into a field of rational functions), then also the substitution operation can be used, with restrictions, to simulate the combinatorial operations.

In Chapter 1 we give preliminaries about rings, polynomial ideals, and Gröbner bases.

In Chapter 2 we give a self-contained presentation of the Hilbert Method, and prove decidability of equivalence of register transducers with output in an unordered variant of the free forest algebra introduced by Bojańczyk and Walukiewicz.

In Chapter 3 we obtain a register transducer model that captures MSO_2 transductions of graphs of bounded treewidth, and prove decidability of a (very) restricted variant of their equivalence problem.

In Chapter 4 we investigate the third of our ways of extending the Hilbert Method, finding a strong limitation of this approach, and obtaining two positive results about register transducers with output in a free monoid enriched with the substitution operation.

In Chapter 5 we propose a proof scheme for showing decidability of equivalence of register transducers with output in a given finitely presented monoid, and obtain a criterion for equational Noetherianity of a monoid (which is also called compactness property).

Keywords: register transducer, equivalence problem, Hilbert Method, bounded treewidth, polynomial ideal, word substitution

Streszczenie

Głównym celem tej pracy jest zastosowanie Metody Hilberta do znalezienia procedur decyzyjnych rozstrzygających problem równoważności dla różnych klas transformacji obiektów kombinatorycznych, takich jak słowa, drzewa bądź grafy. Metoda Hilberta, w zarysie, polega na znalezieniu modelu transduktora rejestrowego, który jest w stanie wyrazić daną klasę transformacji, oraz zakodowaniu obiektów kombinatorycznych używanych przez ten model za pomocą liczb całkowitych w taki sposób, że podstawowe operacje kombinatoryczne (jak, na przykład, konkatenacja słów) mogą być symulowane z użyciem operacji algebraicznych dodawania i mnożenia.

W trakcie pracy rozszerzamy Metodę Hilberta dowodząc następujących lematów: (1) obiekty kombinatoryczne mogą być zakodowane w dowolnym obliczalnym pierścieniu zredukowanym, (2) jeżeli obiekty kombinatoryczne są zakodowane w ciele (zauważmy, że każde ciało jest pierścieniem zredukowanym), to również operacja dzielenia może być użyta do symulowania operacji kombinatorycznych, (3) jeżeli obiekty kombinatoryczne są zakodowane w pierścieniu wielomianów (zauważmy, że można go zanurzyć w ciało funkcji wymiernych), to również operacja podstawienia może być, z ograniczeniami, użyta do symulowania operacji kombinatorycznych.

W rozdziale pierwszym, przedstawiamy informacje wstępne dotyczące pierścieni, ideałów oraz baz Gröbnera.

W rozdziale drugim, przedstawiamy Metodę Hilberta oraz dowodzimy rozstrzygalności równoważności transduktorów rejestrowych, których algebra wyjściowa jest wariantem nieuporządkowanym wolnej algebry lasów wprowadzonej przez Bojańczyka i Walukiewicza.

W rozdziale trzecim, wskazujemy model transduktora rejestrowego, który jest w stanie wyrazić MSO$_2$ transdukcje grafów o ograniczonej szerokości drzewiastej, oraz dowodzimy rozstrzygalności (bardzo) ograniczonej wersji jego problemu równoważności.

W rozdziale czwartym, badamy trzeci ze sposobów rozszerzania Metody Hilberta, znajdując spore ograniczenia tego podejścia, a także otrzymując dwa pozytywne wyniki z jego użyciem, dotyczące transduktorów rejestrowych o wyjściu w monoidzie wolnym rozszerzonym o operację podstawienia.

W rozdziale piątym, proponujemy schemat dowodzenia rozstrzygalności równoważności dla transduktorów rejestrowych, których algebra wyjściowa jest skończenie prezentowanym monoidem, oraz otrzymujemy kryterium na równaniową Noetherowskość monoidu (zwaną również własnością zwartości).

Słowa kluczowe: transduktor rejestrowy, problem równoważności, Metoda Hilberta, ograniczona szerokość drzewiasta, ideał w pierścieniu wielomianów, podstawienie słów

Contents

Introduction

This thesis is focused on the problem of decidability of functionality and of equivalence of functional register transducers that input trees. Our general theme is the Hilbert Method, which, roughly speaking, is a reduction of combinatorial objects, like words or trees, to algebraic objects, like numbers or polynomials, such that the operations of the former can be reduced to the operations of the latter. We describe the Hilbert Method later; let us now focus on transducers.

In general, a *tree transducer* is a formal machine model built on a tree automaton that, additionally to accepting trees, returns some output, which for example could be another tree or a number. In consequence, it defines not only a language but also a transformation. The definition of a register transducer is given later in the text. A transducer is called *functional* if the transformation it defines is a function (i.e. admits at most one output for every input), and two transducers are called *equivalent* if they define the same transformation. A decision procedure for testing equivalence of transducers can be used, for example, to verify if a given transducer defines the desired transformation. This can be done by taking another transducer that undoubtedly defines the transformation of our interest (but possibly is suboptimal) and testing both transducers for equivalence. The motivation for tree transducers comes, among others, from theory of document transformation[1], compiler theory[2], and natural language processing[3]. For more information and references regarding tree transducers and their applications we refer to a textbook of Fülöp and Vogler[4].

The field of transducers has numerous models. Let us give several examples. We begin with a word transducer – it is a special case of a tree transducer, since words can be represented as trees in which every node has at most one child. The oldest and simplest model of a word transducer is a Mealy Machine, introduced in 1955 by Mealy[5]. It traverses the input word in the left-to-right manner like a deterministic finite automaton (**DFA**), and in every step outputs one letter, which is determined by both the state and the input letter; the states are used only for determining the output letters – the machine returns output for every input word.

[1] Hosoya, 2010.

[2] For example see Thirunarayan, 2009. Attribute grammars can be seen as tree transducers: attribute grammars and attributed tree transducers have the same expressive power (Maneth, 1998).

[3] Maletti, 2015.

[4] Fülöp and Vogler, 1998.

[5] Mealy, 1955.

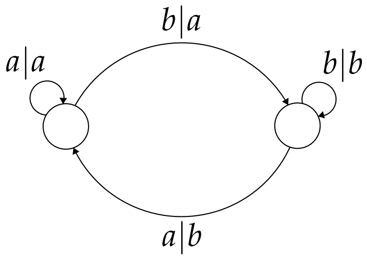


FIGURE 1: A Mealy Machine. For example, it maps *abbba* to *aabbb*.

A transducer model that is slightly more general is a deterministic finite state transducer. It can be described shortly as a deterministic finite automaton with output: in every step it outputs a word (which, in contrast to the case of Mealy Machines, might consist of more than one letter), and it either accepts or reject the input word, depending on the state reached after reading it.

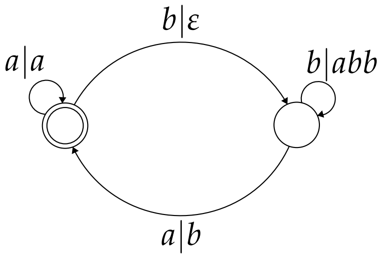


FIGURE 2: A deterministic finite state transducer. It returns output for words of form $(a+b)^*a$; for example, it maps *abbba* to *aabbabbb*.

Analogously, one can construct transducer models from other automata models, and thus obtain the following: nondeterministic finite state transducer, push-down transducer, two-way (non)deterministic finite transducer, pebble transducer[6], and marble transducer[7] (we do not claim the list to be exhaustive). Also, one can apply an analogous construction to tree automata, and thus obtain (non)deterministic top-down (bottom-up) transducers or pebble tree transducers. There are also transducer models of other kinds, let us mention macro tree transducer[8], attribute transducer (derived from attribute grammar)[9], Higher Order deterministic top-down transducer[10], and ground tree transducer[11].

Transformation models of this thesis. Let us proceed to transformation models we use in this thesis. The first one is defined using Monadic Second

[6] Pebble automata were introduced in Globerman and Harel, 1996.

[7] Marble automata were introduced for trees in Engelfriet, Hoogeboom, and Van Best, 1999.

[8] Engelfriet and Vogler, 1985.

[9] Fülöp, 1981.

[10] Gallot, Lemay, and Salvati, 2020.

[11] Dauchet et al., 1990.

Order logic (MSO); it is called MSO *transduction*. MSO is an extension of first order logic that allows quantification over sets of elements of the structure. Connections of MSO and automata theory date back to 1960s when the famous theorem of Büchi, Elgot, and Trakhtenbrot established the equivalence of **DFA** and MSO over words with the successor relation[12]; similar results were proved for trees[13] and graphs[14]. These connections were extended from automata to transducers: Engelfriet and Hoogeboom established the equivalence of deterministic two-way transducers and MSO transductions of words with the successor relation[15], and Engelfriet and Maneth established the equivalence of macro tree transducers with regular look-ahead that are single use restricted and MSO transductions of trees[16]; also, Alur et al. introduced another equivalent model (register transducer, cf. next paragraph) for both words[17] and trees[18]. MSO transductions were studied also for graphs, see a survey of Courcelle[19], although with no relation to tree transducers. (In Chapter 3, we discuss MSO transductions of graphs (of bounded treewidth), introducing connections with register transducers (using the mentioned results of Alur et al.))

The second and the most important transformation model we use is, following the work of Engelfriet, Maneth, Alur, Černý, and D'Antoni[20], a tree transducer that uses registers; we call it a *register transducer*, following Bojańczyk[21]. It outputs elements of a fixed algebra (for example a free monoid or a field of rational numbers), denoted as *output algebra*. (A register transducer with output algebra $\mathbb{A}$ is also called a *register transducer with output in* $\mathbb{A}$.) A register transducer is based on a bottom-up tree automaton, and is enriched with a finite tuple of variables, called registers, that store elements of the output algebra. The tuple of registers is initiated in every leaf of the input tree, then in every non-leaf node it is synthesized by applying a polynomial operation of $\mathbb{A}$ (in the sense of universal algebra) to its children's registers, and finally the returned output is an evaluation of a polynomial operation of $\mathbb{A}$ on the root's registers. The output algebra can be arbitrary, although typically it is either of "combinatorial" kind, like a free monoid or an algebra of ranked trees, or a ring, for example of integers or of polynomials.

[12] Büchi, 1960, Elgot, 1961, and Trakhtenbrot, 1961.

[13] The proof is analogous as for words.

[14] Courcelle and Engelfriet, 2012.

[15] Engelfriet and Hoogeboom, 2001.

[16] Engelfriet and Maneth, 1999.

[17] Alur and Černý, 2010; Alur and Černý, 2011.

[18] Alur and D'Antoni, 2012.

[19] Courcelle, 1994; see also related comments in M. Bojańczyk and Pilipczuk, 2016, Introduction.

[20] Engelfriet and Maneth, 1999; Alur and Černý, 2010; Alur and Černý, 2011; Alur and D'Antoni, 2012.

[21] Mikołaj Bojańczyk, 2019.

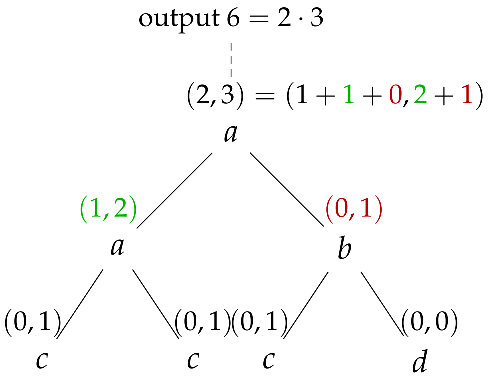


FIGURE 3: A run of a register transducer with output in the ring of integers that computes the product of the numbers of a-labeled and c-labeled nodes of an input tree.

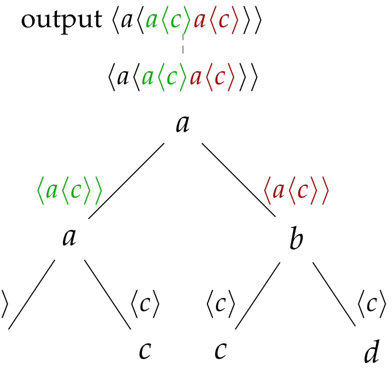

output $\langle a\langle a\langle c\rangle a\langle c\rangle\rangle\rangle$
$\langle a\langle a\langle c\rangle a\langle c\rangle\rangle\rangle$
a
$\langle a\langle c\rangle\rangle$
$\langle a\langle c\rangle\rangle$
a
b
$\langle c\rangle$
$\langle c\rangle$
$\langle c\rangle$
$\langle c\rangle$
c
c
c
d

FIGURE 4: A run of a register transducer with output in a free monoid that computes the string representation of an input tree.

Register transducers are expressive: for the output algebra being a free monoid and input alphabet that consists of unary and 0-ary symbols they strictly subsume MSO string-to-string transductions, and for output algebra being a free forest algebra (introduced by Bojańczyk and Walukiewicz[22]) they strictly subsume MSO tree-to-tree transductions. They were introduced, with so-called *copyless restriction*, by Alur and Černý in the string-to-string case[23], and by Alur and D'Antoni in the tree-to-tree case[24] (with the copyless restriction, in both cases the register transducers and corresponding MSO transductions are equally expressive[25]). Register transducers were also used by Seidl et al.[26] with the output algebra being the field of rational numbers, for which they proved decidability of equivalence[27].

[22] M. Bojańczyk and Walukiewicz, 2008.

[23] Alur and Černý, 2010.

[24] Alur and D'Antoni, 2012.

[25] Alur and Černý, 2011; Alur and D'Antoni, 2012.

[26] Seidl, Maneth, and Kemper, 2018.

[27] The proof is done for deterministic transductions; the general case follows from a rather standard reduction to the deterministic case, cf. the proof of Lemma 2.9.

Hilbert Method. The proof scheme we use in this thesis is analogous to the one used by Seidl et al. They simulated a subclass of register transducers with output in a free monoid (deterministic top-down tree-to-string transducers) by register transducers with output in the ring of integers. Their simulation amounts to simulating the output structures: a free monoid is simulated by the ring of integers, by embedding it into the multiplicative monoid of the matrix ring, see the below figure[28].

$$\begin{aligned} a_1 &\overset{\phi}{\mapsto} \begin{bmatrix} 3 & 1 \\ 0 & 1 \end{bmatrix} \\ a_2 &\overset{\phi}{\mapsto} \begin{bmatrix} 3 & 2 \\ 0 & 1 \end{bmatrix} \\ w \cdot v &\overset{\phi}{\mapsto} \phi(w) \cdot \phi(v) \qquad \text{for } w, v \in \{a_1, a_2\}^*. \end{aligned}$$

FIGURE 5: A simulation of a free monoid $\{a_1, a_2\}^*$ by the ring of integers, using a homomorphism into the multiplicative matrix monoid; a word $w = a_{i_1} a_{i_2} \ldots a_{i_n}$ is mapped to $\begin{bmatrix} 3^n & w_3 \\ 0 & 1 \end{bmatrix}$ where w_3 is the number represented by $i_1 i_2 \ldots i_n \in \{1, 2\}^*$ in ternary.

Seidl et al. obtained decidability of equivalence for the former model by showing decidability of equivalence for the latter model, using polynomial ideals. An analogous proof scheme has been used in formal language theory before[29], having its roots in representation theory, and has recently been called *the Hilbert Method* by Bojańczyk[30]. We use it and expand it in each of the forthcoming chapters – each time with an algebra other than a free monoid and a ring other than the one of integers – and obtain decidability results for other classes of transductions.

Organization of the thesis. In Chapter 1, we give preliminaries about rings, polynomial ideals, and Gröbner bases.

In Chapter 2, we present the proof of decidability of functionality and equivalence of functional register transducers with output in a computable commutative ring with no zero divisors (or, without loss of generality, with output in a computable field[31]) (Theorem 5)[32]. Then, we simulate a certain

[28] This particular injective homomorphism is from Seidl, Maneth, and Kemper, 2018, Example 3.2; analogous homomorphisms can be found in the literature, see for example the, classical, Sanov linear representation of a free group.

[29] Albert and Lawrence, 1985; Honkala, 2000; Benedikt et al., 2017.

[30] Mikołaj Bojańczyk, 2019.

[31] Register transducers with output in fields have the same expressive power as register transducers with output in rings with no zero divisors (cf. Theorem 5). It is because every field is a ring with no zero divisors, and every commutative ring with no zero divisors is contained in some field (for example, its field of fractions).

[32] The proof repeats the arguments of Seidl et al. (in an equivalent formalism of polynomial grammars), but with weaker assumptions on the output ring.

algebra of unordered forests by the ring of univariate polynomials, thus obtaining a decidability result for register transducers with output in this algebra. We also formalize the notion of a simulation of algebras and the used proof scheme.

In Chapter 3, we give a register transducer model that captures MSO transductions of graphs of bounded treewidth. Then, we simulate graphs of bounded treewidth – up to a certain relaxation of isomorphism, which we call *walk-equivalence* – by the field of univariate rational functions enriched with the division operation. We extend the scope of Theorem 5 to register transducers that use the division operation, thus our simulation gives decidability of functionality and equivalence of functional MSO transduction of graphs of bounded treewidth in which the output graphs are identified up to walk-equivalence.

In Chapter 4, we consider register transducers with output in a ring of polynomials enriched with the substitution operation. First, we obtain undecidability of equivalence for this model. Then, we obtain two positive results, using this model's restricted variants. The first result states decidability of satisfaction of a (potentially infinite) system of word equations by every tuple of words from a given language, if both the system and the language are generated by a register transducer (notice that an equation in variable set X can be represented as a pair of words over an alphabet that contains X). The second result states decidability of functionality and equivalence of functional register transducers with output in a free monoid enriched with word homomorphisms that satisfy the following restrictions: (1) the homomorphisms must be applied simultaneously to all registers, and (2) each of these homomorphisms, when mapped by the simulation of a free monoid by a ring of polynomials that we introduce, induces an automorphism of some field that contains the output ring of polynomials (we show this property can be easily decided).

In Chapter 5, we propose a proof scheme for showing decidability of functionality and equivalence of functional register transducers with output in a given finitely presented monoid (a classical example of such a monoid in computer science is a trace monoid). We apply this proof scheme successfully for an example monoid ($\mathbb{M} = \langle a, b, c \mid abc = cba \rangle$), although we do not show it is more useful than simpler proof methods. Finally, we observe that our proof scheme yields a criterion for *equational Noetherianity* (also called *compactness property*) of a given finitely presented monoid, a property that states that every infinite system of equations admits a finite equivalent subsystem.

Chapter 1

Commutative algebra. Ideal Membership

In this chapter, we present the technical background regarding rings, polynomial ideals, and the theory of Gröbner bases. An ideal contained in a polynomial ring $\mathbb{Q}[x_1, \ldots, x_n]$ can represent a set of polynomial relations that hold between all tuples of a given subset C of $\mathbb{Q}^n$. For example, C can be the set of all valuations of variables in a given location of a given program. The program might be of any kind, however we restrict our attention to a *register transducer*, this thesis's main computational model.

We focus on proving decidability of the *Ideal Membership problem*, which asks if a given polynomial belongs to a given ideal. To this end, we use Gröbner bases. Our intended application of Ideal Membership is to decide if a given polynomial relation holds between variables of a register transducer (called *registers*).

1.1 Rings, fields, ideals

A *(commutative) ring* is a set with constants $0, 1$ and operations $+, \cdot$ that satisfy the usual commutative ring axioms (see for example Cox, Little, and O'Shea, 2015, Appendix A, §1, Definition 2). Let us emphasize that every ring that occurs in this thesis, except a matrix ring, is *commutative*, which means that $a \cdot b = b \cdot a$ for all its elements a, b.

A ring R has *no zero divisors* if $a \neq 0, b \neq 0$ implies $a \cdot b \neq 0$ for all $a, b \in R$, and is a *computable ring* if its elements can be enumerated so that the ring operations $+, \cdot$ are computable functions. A *field* K is a ring in which every non-zero element $a \in K$ admits an *inverse element*, i.e. an element b such that $a \cdot b = 1$. A *computable field* is a field that is a computable ring – note that in such a case the function that maps every non-zero element to its inverse is a computable function. The rings we usually consider are *rings of polynomials* with coefficients in some field K, denoted $K[X]$ where X is a variable set. Having said that, in Chapter 4 we also consider quotients of rings of polynomials, which we define in that chapter.

An *ideal in a ring* R is a subset $I \subseteq R$ such that

$$f, g \in I \Rightarrow f + g \in I \text{ for all } f, g \in R$$

and

$$f \in I \Rightarrow f \cdot h \in I \text{ for all } f, h \in R.$$

For a subset $F \subseteq R$, by $\langle F \rangle_R$ we denote the smallest ideal in R that contains the set F; if R is clear from the context we abbreviate $\langle F \rangle_R$ to $\langle F \rangle$. It can be shown that

$$\langle F \rangle_R = \{h_1 f_1 + \ldots + h_m f_m \mid f_1, \ldots, f_m \in F, h_1, \ldots, h_m \in R, m \in \{0, 1, \ldots\}\}.$$

The ideal $\langle F \rangle$ is called the *ideal generated by F*; the set F is called a *generating set* or a *set of generators* of the ideal $\langle F \rangle$ and its elements are called *generators* of $\langle F \rangle$.

Let us invoke Hilbert's Basis Theorem, which implies that every ideal can be represented in a finite way.

Definition 1.1. A ring R is called *Noetherian* if it satisfies either of the following equivalent conditions:

(i) every set of polynomials admits a finite subset that generates the same ideal,

(ii) every strictly ascending chain of ideals of R

$$I_1 \subsetneq I_2 \subsetneq I_3 \subsetneq \ldots$$

is finite.

Theorem 1 (Hilbert's Basis Theorem)**.** *If a ring R is Noetherian then the ring of polynomials $R[x]$ in one variable x also is Noetherian.*

Corollary 1.2. *Let K be a Noetherian ring (for example a field). The polynomial ring $K[x_1, \ldots, x_n]$ is Noetherian for every $n \in \{1, 2, \ldots\}$.*

In consequence, for every field K, every ideal $I \subseteq K[x_1, \ldots, x_n]$ admits a *finite* set of generators. Therefore, for algorithmic purposes we always assume that every ideal is given by (one, out of many possible) finite generating set(s).

1.2 Ideal Membership

Let K be a computable field and X be a finite variable set. Consider the following decision problem.
Name: Ideal Membership for $K[X]$
Input: $f \in K[X]$,
F – an finite subset of $K[X]$,
Question: Is it the case that

$$f \in \langle F \rangle?$$

How to decide Ideal Membership? We take the "division approach", following a textbook of Cox, Little, and O'Shea[1]. Given a polynomial f and

[1] Cox, Little, and O'Shea, 2015, Chapter 2.

an ideal $I = \langle f_1, \dots, f_s \rangle$ it is tempting to "divide" f by $f_1, \dots, f_s$ and obtain a "decomposition" of f as

$$f = g_1 f_1 + \ldots + g_s f_s + r$$

where $g_1, \dots, g_s$ are polynomials and r is a remainder of f divided by $f_1, \dots, f_s$, which hopefully is unique w.r.t. I (that is, is independent of the choice of generators of I and their order as divisors) and is equal to 0 if and only if f belongs to I.

Before we define division for multivariable polynomials, let us demonstrate the difficulties of the above approach to deciding Ideal Membership that occur even in the one-variable case.

Example 1.3. Let $f = x^5$ and let the ideal $I \subseteq \mathbb{Q}[x]$ be defined by $I = \langle f_1, f_2 \rangle$ where $f_1 = x^3 - x$ and $f_2 = x^2 + x$. Then f is in I:

$$x^5 = (-\frac{1}{2}x^4) \cdot (x^3 - x) + (-\frac{1}{2}x^5 + \frac{1}{2}x^4) \cdot (x^2 + x).$$

However, it has a non-zero remainder (x) of division by f_1, f_2:

$$x^5 = (x^2 + 1) \cdot (x^3 - x) + 0 \cdot (x^2 + x) + x.$$

Notice that changing the order of divisors to (f_2, f_1) results in a different decomposition, yet again in a non-zero remainder (x, which happens to be the same as with the previous order of divisors):

$$x^5 = (x^3 - x^2 + x - 1) \cdot (x^2 + x) + 0 \cdot (x^3 - x) + x.$$

We investigate this problem deeper in the, general, multivariable case.

For the division of multivariable polynomials to be properly defined, we *order* the monomials; this way, we can perform a step of division by dividing a monomial of f by the largest monomial of one of f_i's (denoted later as the *leading monomial*); in consequence, the result of such process – a remainder – has all monomials not divisible by any of the largest monomials of f_i's, and is unique for a special choice of the generating set (Lemma 1.14(i)).

Ordering monomials. Let $X = \{x_1, \dots, x_n\}$ be a finite variable set. A *monomial* in variable set X is a product of variables of X, possibly with repetitions; we identify a monomial with an element of $\mathbb{N}^X \simeq \mathbb{N}^n$. We adopt the convention of denoting a monomial $x_1^{\alpha_1} \cdot \ldots \cdot x_n^{\alpha_n}$ as $\mathbf{x}^\alpha$ where $\alpha = (\alpha_1, \dots, \alpha_n) \in \mathbb{N}^n$. The *total degree of a monomial* $\mathbf{x}^\alpha$, denoted $\deg(\mathbf{x}^\alpha)$, is the sum of the exponents of its variables. For a polynomial $f = \sum_{\alpha \in \mathbb{N}^n} a_\alpha \mathbf{x}^\alpha$ (note that $a_\alpha = 0$ for almost all α) we say that a monomial $\mathbf{x}^\alpha$ *occurs* in f if $a_\alpha \neq 0$; we sometimes simply call it a *monomial of* f. The *total degree of a polynomial* $f = \sum_{\alpha \in \mathbb{N}^n} a_\alpha \mathbf{x}^\alpha$, denoted $\deg(f)$, is the maximal total degree of its monomials.

Definition 1.4 (Graded-lexicographic order (grlex))**.** Let X be a finite variable set. The *graded lexicographical order (grlex)* is an order $>$ on the set of monomials in variable set X (or equivalently – on the set $\mathbb{N}^X$) in which $\mathbf{x}^\alpha > \mathbf{x}^\beta$ if

- $\deg(\mathbf{x}^\alpha) > \deg(\mathbf{x}^\beta)$, or
- $\deg(\mathbf{x}^\alpha) = \deg(\mathbf{x}^\beta)$ and $\alpha > \beta$ in lexicographical order,

for $\alpha, \beta \in \mathbb{N}^X$. Intuitively, among two monomials with the same total degree, the one with more occurrences of larger variables is larger.

Example 1.5. Consider an ordered set of variables $X = \{x > y > z\}$. With respect to grlex order, the following inequalities hold:

$$y^2z^3 > x^3y$$

and

$$x^2yz^2 > xy^3z.$$

We use the following properties of grlex order[2].

Definition 1.6 (Properties of grlex)**.** Let X be a finite variable set and let $>$ denote grlex order. Then the following properties hold:

(i) $x > 1$ for every variable $x \in X$,

(ii) $m_1 > m_2$ implies $m_1 n > m_2 n$ for all monomials m_1, m_2, n,

(iii) $>$ is a well-order, i.e. it has no infinite descending chains.

Throughout this chapter, whenever we compare two monomials, it is always by grlex order for some fixed ordering of variables.

In Example 1.3 we saw that the "naive division approach" to deciding Ideal Membership is not correct. Let us investigate that again, this time in a multivariable setting.

Example 1.7. Let $f = x^2y^2$ and $I \subseteq \mathbb{Q}[x,y]$ be an ideal defined by $I = \langle f_1, f_2 \rangle$ where $f_1 = xy^2 + y$ and $f_2 = x^2y - x$. Then f is in I:

$$x^2y^2 = \frac{1}{2}x^2y \cdot (xy^2 + y) + (-\frac{1}{2}xy^2) \cdot (x^2y - x).$$

but it has a non-zero remainder $(-xy)$ when divided by f_1, f_2:

$$x^2y^2 = x \cdot (xy^2 + y) + 0 \cdot (x^2y - x) + (-xy).$$

Notice that changing the order to (f_2, f_1) results in a different remainder, yet again a non-zero one (xy):

$$x^2y^2 = y \cdot (x^2y - x) + 0 \cdot (xy^2 + y) + xy.$$

[2] In fact, these are the only properties we need in this chapter. Every ordering of monomials that satisfies those properties is called a *monomial order*. There are other examples of monomial orders, for example *grevlex (graded reverse lexicographical order)*, which, intuitively, is similar to grlex, but among two monomials with the same total degree, the one with less occurrences of smaller variables is larger.

Notice that in the above sum a monomial of degree larger than $\deg(f)$ occurs in both summands (x^3y^3) and gets canceled out in the sum. For the division algorithm to decide Ideal Membership, we enrich the set of generators of a given ideal with polynomials being results of such cancel-outs (called *S-polynomials*). This new set of generators is called a *Gröbner basis* of I.

Gröbner bases. From now until the end of this chapter let K be a computable field and X be a finite variable set.

Definition 1.8. Let $f = \sum_{\alpha \in \mathbb{N}^n} a_\alpha \mathbf{x}^\alpha \in K[X]$. Then

- the *leading monomial of f*, denoted by $\mathrm{LM}(f)$, is the largest monomial (w.r.t. grlex order) that occurs in f.

Let $\mathbf{x}^\alpha$ be the leading monomial of f. Then

- the *leading term of f*, denoted by $\mathrm{LT}(f)$, is $a_\alpha \mathbf{x}^\alpha$,
- the *leading coefficient of f*, denoted by $\mathrm{LC}(f)$, is a_α.

Observe that the notion of leading monomial is a multivariable analogue of the notion of degree of a one-variable polynomial. This is also reflected in the proofs, as many of them will proceed by induction on the leading monomial of some polynomial.

Remark 1.9 (Membership in $\langle \mathrm{LT}(G) \rangle$)**.** In the forthcoming lemmas we will use ideals of form $\langle \mathrm{LT}(G) \rangle = \langle \{\mathrm{LT}(g) \mid g \in G\} \rangle$ for some set of polynomials $G \subseteq K[X]$. We will use the following property of $\langle \mathrm{LT}(G) \rangle$: a monomial belongs to $\langle \mathrm{LT}(G) \rangle$ if and only if it is divisible by the leading monomial of some polynomial from G.

Definition 1.10 (Division step)**.** Let $f = \sum_{\alpha \in \mathbb{N}^n} a_\alpha \mathbf{x}^\alpha$ and g be polynomials. A *division step* of f by g that uses a monomial $\mathbf{x}^\alpha$ that occurs in f and is divisible by the leading monomial of g produces a polynomial

$$\widetilde{f} := f - a_\alpha \cdot \frac{\mathbf{x}^\alpha}{\mathrm{LM}(g)} \cdot g$$

and results in a decomposition

$$f = a_\alpha \cdot \frac{\mathbf{x}^\alpha}{\mathrm{LM}(g)} \cdot g + \widetilde{f}.$$

In such situation we write

$$f \to_g \widetilde{f}.$$

If G is a finite set of polynomials then we write

$$f \to_G \widetilde{f}$$

if $f \to_g \widetilde{f}$ for some $g \in G$.

Lemma 1.11. *Let G be a finite set of polynomials. Then the following hold.*

(i) *Every sequence of division steps* $f \to_G f_1 \to_G f_2 \to_G \ldots$ *ultimately terminates in some, not necessarily unique, polynomial* r*, and, as a by-product, results in a decomposition*

$$f = \sum_{g \in G} r_g \cdot g + r, \text{ no monomial of } r \text{ is divisible by the leading monomial of some polynomial from } G \tag{1.1}$$

for some polynomials $(r_g)_{g \in G}$ *that satisfy*

$$\max_{g \in G} \mathrm{LM}(r_g \cdot g) = \mathrm{LM}(f).$$

(In particular, all $(r_g \cdot g)$*'s and* r *have smaller or equal leading monomials than* f*.) We call every such* r *a* remainder *of division of* f *by* G*.*

(ii) *If* $f \to_G \widetilde{f}$ *and it was the leading monomial of* f *that was used for this step, then* $\mathrm{LM}(\widetilde{f}) < \mathrm{LM}(f)$*.*

The proof is straightforward and we omit it.

We announced Gröbner basis as an enriched set of generators of an ideal I. However, by definition, a Gröbner basis of an ideal I is a finite subset of I that "completely" represents I in terms of division.

Definition 1.12. Let $I \subseteq K[X]$ be an ideal. A *Gröbner basis* of I is a finite subset $G \subseteq I$ such that

$$\langle \mathrm{LT}(G) \rangle = \langle \mathrm{LT}(I) \rangle, \tag{1.2}$$

i.e. every monomial is divisible by a leading monomial of some polynomial of I if and only if it is divisible by the leading monomial of some polynomial of G[3]. Notice that in such case any polynomial f can be divided by I (with either zero or a non-zero remainder, but necessarily different than f) if and only it can be divided by G (likewise).

Lemma 1.13. *Every ideal* $I \subseteq K[X]$ *has a Gröbner basis* G*. Every Gröbner basis of* I *generates* I*.*

Proof. By Noetherianity of $K[X]$ viewed as in item (i) of Definition 1.1 (Corollary 1.2), there is a finite set $G \subseteq I$ that satisfies (1.2). We prove that G is a generating set of I.

By contradiction, take a polynomial f with the smallest leading monomial that belongs to $I \setminus \langle G \rangle$. Then the leading monomial of f belongs to $\langle \mathrm{LT}(I) \rangle = \langle \mathrm{LT}(G) \rangle$, and hence it can be divided by some polynomial $g \in G$ as in Lemma 1.11(ii). This results in a polynomial $\widetilde{f}$ with a smaller leading monomial such that $f = h \cdot g + \widetilde{f}$ for some polynomial h; observe that $\widetilde{f}$ also belongs to $I \setminus G$, a contradiction. □

Now we prove that our division approach is correct, that is, it decides Ideal Membership if the generating set of a given ideal is its Gröbner basis.

[3] Note that for every ideal I the ideal $\langle \mathrm{LT}(I) \rangle$ is the same as the ideal $\langle \mathrm{LM}(I) \rangle$, nevertheless we use the first notation, following Cox, Little, and O'Shea, 2015.

Lemma 1.14. *Let $f \in K[X]$ be a polynomial, $I \subseteq K[X]$ be an ideal, and G be a Gröbner basis of I. Then*

(i) there is a unique *remainder of division of f by G,*

(ii) $f \in I$ if and only if this remainder is 0.

Proof. Take any two remainders r, r' of division of f by G, and let

$$f = \sum_{g \in G} r_g \cdot g + r = \sum_{g \in G} r'_g \cdot g + r' \tag{1.3}$$

be the corresponding decompositions of f. Then

$$r - r' = \sum_{g \in G} (r'_g - r_g) \cdot g \in \langle G \rangle.$$

If $r - r'$ was non-zero, then the leading monomial of $r - r'$ would be in $\langle \mathrm{LT}(G) \rangle$, which is impossible since *no* monomial of either r or r' is in $\langle \mathrm{LT}(G) \rangle$; in consequence $r - r'$ is 0. The proof of item (ii) is analogous: from (1.3) we have that $f \in I$ if and only if $r \in I$, and since no monomial of r is in $\langle \mathrm{LT}(G) \rangle = \langle \mathrm{LT}(I) \rangle$, we have that $r \in I$ if and only if r is 0. □

Corollary 1.15. *Ideal Membership reduces to the problem of finding a Gröbner basis of a given ideal.*

Therefore the rest of this chapter is devoted to the computation of a Gröbner basis of a given ideal.

Computing Gröbner bases. As we saw in Example 1.7, sometimes a polynomial f does belong to the ideal $\langle f_1, \ldots, f_s \rangle$ but in every "decomposition" that certifies this fact – $f = g_1 f_1 + \ldots + g_s f_s$ for some polynomials $g_1, \ldots, g_s \in K[X]$ – there must be a cancel-out of monomials of $g_i f_i$'s that are larger than the leading monomial of f, and in consequence division of f by $(f_1, \ldots, f_s)$ does not capture the fact that f belongs to I. We fix this problem by enlarging the set of generators with polynomials that capture the simplest of such cancel-outs – S-polynomials.

Definition 1.16 (S-polynomial)**.** Let $f, g \in K[X]$ be a pair of polynomials and let the monomial m be the least common multiple of the leading monomials of f and g. Then the *S-polynomial of* (f, g), denoted $S(f, g)$, is defined by

$$S(f,g) := \frac{m}{\mathrm{LT}(f)} \cdot f - \frac{m}{\mathrm{LT}(g)} \cdot g. \tag{1.4}$$

Notice that in $S(f, g)$ we divide by the leading *terms* of f and g, not by their leading monomials; in consequence, in the right-hand side of (1.4) the largest monomial m gets canceled-out. Intuitively, $S(f, g)$ is the "simplest cancel-out" of the largest monomial that can occur in a sum of form $rf + sg$, where r, s are polynomials from $K[X]$.

Let us see two first main properties of an S-polynomial.

Lemma 1.17. *Let $f, g \in K[X]$. Then the following hold.*

(i) if f, g belong to some ideal I, so does $S(f,g)$,

(ii) if additionally f and g have the same leading monomial then

$$\mathrm{LM}(S(f,g)) < \max(\mathrm{LM}(f), \mathrm{LM}(g)).$$

Proof. We omit the proofs. □

It turns out that *S-polynomials* capture *all* cancel-outs, in the sense made precise in the below lemma.

Lemma 1.18. *Let $p, p_1, \dots, p_s \in K[X]$ be polynomials and m be a monomial such that*

$$p = p_1 + \ldots + p_s$$

and

$\mathrm{LM}(p_i) = m$ *for all* $1 \le i \le s$ *(i.e. monomial m occurs in each p_i) but*
$\mathrm{LM}(p) < m$ *(i.e. monomial m gets canceled out in the sum $p_1 + \ldots + p_s$).*

Then p is a linear combination of $S(p_i, p_j), 1 \le i < j \le s$.

Proof. Let d_i be the leading coefficient of p_i for $1 \le i \le s$; then $\mathrm{LT}(p_i) = d_i m$. By assumptions, $d_1 + \ldots + d_s = 0$ and $S(p_i, p_j) = \frac{1}{d_i} p_i - \frac{1}{d_j} p_j$.

To prove the claim, observe that

$$\sum_{i=1}^{s-1} d_i S(p_i, p_s) = p. \tag{1.5}$$

Indeed, $\sum_{i=1}^{s-1} d_i S(p_i, p_s) = \sum_{i=1}^{s-1} p_i - \left(\sum_{i=1}^{s-1} d_i\right) \cdot \frac{1}{d_s} p_s \overset{(1.5)}{=} \sum_{i=1}^{s-1} p_i + p_s = p$. □

Now we are ready to state and prove two main theorems of this chapter.

Theorem 2 (Buchberger's Criterion)**.** *Let $I \subseteq K[X]$ be an ideal. Then a finite generating set G of I is a Gröbner basis of I if and only if every remainder of division of $S(g, \widetilde{g})$ by G is 0, for $g, \widetilde{g} \in G, g \neq \widetilde{g}$.*

Proof. The left-to-right implication is a direct consequence of the fact that $S(g, \widetilde{g}) \in I$ (Lemma 1.17(i)) and uniqueness of the remainder in a division by a Gröbner basis (Lemma 1.14(ii)).

For the right-to-left implication, take any $f \in I$ and a decomposition

$$f = \sum_{g \in G} r_g \cdot g \tag{1.6}$$

with the smallest $m := \max_{g \in G} \mathrm{LM}(r_g \cdot g)$. Now consider cases regarding the leading monomial of f. If it is equal to m, then it clearly belongs to $\langle \mathrm{LM}(G) \rangle$.

Otherwise, i.e. if $\mathrm{LM}(f) < m$ (and hence some monomial larger that $\mathrm{LM}(f)$ got canceled out in the sum $\sum_{g \in G} r_g \cdot g$), we show that f admits another decomposition of form as in (1.6), but with a smaller m, which is a contradiction.

Indeed, take the part of $\sum_{g \in G} r_g \cdot g$ constituted by those $(\mathrm{LM}(r_g) \cdot g)$'s from (1.6) whose leading monomial is m and, by Lemma 1.18, rewrite it as a linear combination of S-polynomials of pairs of those $(\mathrm{LM}(r_g) \cdot g)$'s – observe each of these S-polynomials has a strictly smaller leading monomial than m (Lemma 1.17(ii)). Since the S-polynomial of any pair of $(\mathrm{LM}(r_g) \cdot g)$'s is a multiple of the S-polynomial of the pair of the same g's[4], by the assumption its remainder of division by G is 0, so dividing it by G yields its decomposition into a sum of form $\sum_{g \in G} h_g \cdot g$ whose summands have smaller or equal leading monomials. This way we obtain a representation of f in the same form as in (1.6), but with a smaller m, a contradiction. □

Theorem 3 (Buchberger's algorithm). *Given an ideal $I \subseteq K[x_1, \ldots, x_n]$ by a finite set of generators $F \subseteq K[x_1, \ldots, x_n]$, one can compute a set $G \supseteq F$ that is a Gröbner basis of I by the following algorithm:*

```
$\underline{Input}: F \subseteq K[x_1, \ldots, x_n]$ − a finite set,
$\underline{Output}: G \subseteq K[x_1, \ldots, x_n]$ − a Gröbner basis of $I := \langle F \rangle$.

$G := F$
repeat
    $G_{old} := G$
    for $g, \widetilde{g} \in G, g \neq \widetilde{g}$
        $r :=$ a remainder of $S(g, \widetilde{g})$ of division by $G$
        if $r \neq 0$ then
            add $r$ to $G$
until $G = G_{old}$
return $G$
```

Proof. If the algorithm terminates, it clearly returns a finite generating set of the input ideal that satisfies the Buchberger's Criterion, and hence is a Gröbner basis of I (Theorem 2). The algorithm always terminates, because after each loop execution the ideal $\langle \mathrm{LT}(G) \rangle$ strictly increases (cf. Corollary 1.2 and Definition 1.1(ii)). □

Corollary 1.19. *Ideal Membership is decidable.*

Proof. Combine Corollary 1.15 and Theorem 3. □

Corollary 1.20 (Ideal Inclusion). *Let K be a computable field and X be a finite variable set. The following problem is decidable:*
Name: *Ideal Inclusion*

[4] With multiplier being the gcd of the leading monomials of $(\mathrm{LM}(r_g) \cdot g)$'s divided by gcd of the leading monomials of g's.

Input: *F, G – finite subsets of $K[X]$,*
Question: *Is it the case that*

$$\langle F\rangle \subseteq \langle G\rangle?$$

Proof. Observe that $\langle F\rangle \subseteq \langle G\rangle$ if and only if $f \in \langle G\rangle$ for all $f \in F$, which is a reduction to Ideal Membership. □

1.3 Summary

In this chapter, we give a self-contained presentation of Buchberger's algorithm (Theorem 3), which can be used to decide if a given polynomial belongs to a given ideal (Ideal Membership). In consequence, we obtain a procedure to decide, given a pair of ideals, if one of them is contained in the other (Ideal Inclusion). To this end, we introduce the well-known notion of a Gröbner basis.

Let us give four remarks. First, the algorithm we describe can be optimized; this involves, among others, avoiding computing the same S-polynomial twice, restricting the computation of S-polynomials to *critical pairs* in G, i.e. pairs of polynomials of G whose leading monomials are not relatively prime[5], and finally, replacing – by the remainders – not only the newly added S-polynomials but also the generators given in the input. Second, if the last optimization is applied, the result of the algorithm is a unique[6] Gröbner basis, called *reduced Gröbner basis*. Third, Buchberger's algorithm works in exponential space[7], and in this sense it is optimal: deciding Ideal Membership requires at least exponential space[8]. The fourth, last, remark is that applications of Gröbner bases techniques reach far beyond deciding Ideal Membership: we refer to a chapter about elimination theory in a textbook of Cox, Little, and O'Shea[9], or to a broader survey of Buchberger and Winkler[10].

References. This chapter is based on a textbook of Cox, Little, and O'Shea[11].

[5] Two monomials m, n are relatively prime if and only if they have disjoint sets of variables.

[6] The uniqueness is up to multiplying generators by scalars from the field.

[7] Kühnle and Mayr, 1996.

[8] Mayr and Meyer, 1982, Main Corollary.

[9] Cox, Little, and O'Shea, 2015, Chapter 3, Elimination Theory.

[10] Buchberger and Winkler, 1998.

[11] Cox, Little, and O'Shea, 2015.

Chapter 2

The Hilbert Method

In this chapter, we define this thesis's main computational model, a register transducer, and this thesis's main decision problems, functionality and equivalence of functional register transducers, for register transducers with output in various algebras. We also present a decision procedure for these problems in the case when register transducers have output in some ring with no zero divisors, for example a ring of integers or of polynomials with integer coefficients (Theorem 5). Finally, we apply *the Hilbert Method* to prove decidability of each of the main problems for register transducers that output unordered trees (i.e. trees in which there is no order on the siblings), that is, we encode the unordered trees into univariate polynomials so that the tree operations can be simulated by addition and multiplication of polynomials (Theorem 7). We also show that the case of the, usual, ordered trees can be reduced to the unordered one (Fact 2.35).

2.1 Register transducers

A register transducer is a formal machine model that describes recursively defined functions that input trees. Let us begin with an informal definition. Register transducers input *ranked trees*, i.e. rooted trees labeled by *ranked symbols*, which are letters with an associated natural number called *arity* (or *rank*) that describes the number of children of a node: a node labeled by a ranked symbol of arity $k \in \{0, 1, \ldots\}$ has k children. A set of ranked symbols is called a *ranked set* or a *ranked alphabet*. For a ranked alphabet Σ, the set of trees labeled by Σ, also called Σ*-labeled trees*, is denoted as $\mathrm{RankedTrees}(\Sigma)$. By $a^{(k)}$ we denote a ranked symbol with letter a and arity k.

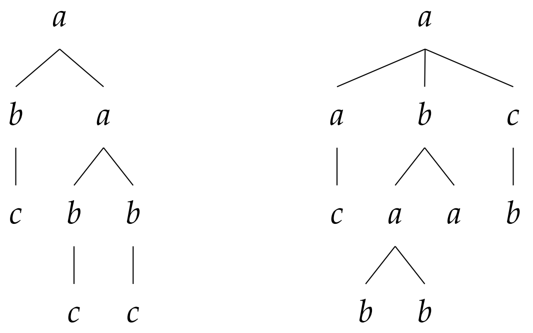


FIGURE 2.1: A ranked tree over the ranked alphabet $\{a^{(2)}, b^{(1)}, c^{(0)}\}$ and an unranked tree over the unranked alphabet $\{a, b, c\}$.

The values output by a register transducer can come from an arbitrary algebra, like a free monoid, the ring of integers etc. Given an input tree, a register transducer runs a bottom-up (nondeterministic, finite) tree automaton that is additionally equipped with a finite number of registers that store and manipulate elements of the chosen algebra. After reading the tree, the transducer outputs the evaluation of a (fixed) function on the tuple of current register values. Now we proceed to the formal definition.

Algebras and polynomial operations. Let $\mathbb{A}$ be an *algebra*, i.e. a pair $\mathbb{A} = (A, \Theta_{\mathbb{A}})$ where A is a set, called the *universe* of $\mathbb{A}$, and $\Theta_{\mathbb{A}}$ is a set of functions, called *basic operations* of $\mathbb{A}$, each of type $A^k \to A$ for some $k \in \{0, 1, \ldots\}$. The basic operations of $\mathbb{A}$ give rise to polynomial operations of $\mathbb{A}$, which can be defined shortly as functions of type $A^n \to A^m$ for some $n \in \{0, 1, \ldots\}, m \in \{1, 2, \ldots\}$ that are compositions of projections, constant functions, and basic operations of $\mathbb{A}$. Formally, a *polynomial (of* $\mathbb{A}$*)* in variable set X is a term built on the union of the signature of $\mathbb{A}$, the set of constants of $\mathbb{A}$, and X; a *polynomial operation (of* $\mathbb{A}$*)* of type $\mathbb{A}^X \to \mathbb{A}^m$ is a function from A^X to A^m defined by an m-tuple of polynomials in the usual way. If the set X is ordered, we naturally identify $\mathbb{A}^X$ with $\mathbb{A}^n$ and A^X with A^n where $n = |X|$.

Let us give a few examples of polynomial operations. In the case when $\mathbb{A}$ is the free monoid $(\{a, b\}^*, \cdot)$ the function

$$(w, v) \mapsto (a \cdot w \cdot w \cdot v \cdot b, b \cdot v)$$

is a polynomial operation of $\mathbb{A}$, and

$$w \mapsto (\text{head (first letter) of } w, \text{tail (remaining part) of } w)$$

is not. In the case when $\mathbb{A}$ is the ring of polynomials $\mathbb{Z}[x]$ the function

$$f \mapsto x \cdot f + 2$$

is a polynomial operation of $\mathbb{A}$, and

$$f \mapsto f[x := x + 1]$$

is not.

Definition 2.1. A *(nondeterministic) register transducer with output algebra* $\mathbb{A}$, or *with output in* $\mathbb{A}$, consists of:

- a finite ranked *input alphabet* Σ,
- a finite set of *states* Q,
- a finite set of *final states* $F \subseteq Q$,
- a finite set of *registers* $\mathcal{R}$,

- a finite set of *transitions* Δ, each of form $(q_1, \ldots, q_k, \sigma, p, q)$, where $k \in \{0, 1, \ldots\}$, $q_1, \ldots, q_k, q \in Q$, $\sigma \in \Sigma$ is a ranked symbol of arity k, and p is a *polynomial operation* of type $(\mathbb{A}^{\mathcal{R}})^k \to \mathbb{A}^{\mathcal{R}}$,
- an *output function* f_{out} that is a polynomial operation of type $\mathbb{A}^{\mathcal{R}} \to \mathbb{A}^m$ for some $m \in \{1, 2, \ldots\}$.

For a transition $(q_1, \ldots, q_k, \sigma, p, q)$, the polynomial operation p is called its *register update*. We denote the class of register transducers with output algebra $\mathbb{A}$ by **RT**-$\mathbb{A}$.

The semantics of a register transducer $T = (\Sigma, Q, F, \mathcal{R}, \Delta_0, \Delta, f_{\text{out}})$ with output algebra $\mathbb{A}$ is a binary relation from RankedTrees(Σ) to $\mathbb{A}$; it is defined as follows. Let $t \in$ RankedTrees(Σ) be an input tree. A *reachable configuration* on t is a pair $(q, \boldsymbol{\alpha}) \in Q \times A^{\mathcal{R}}$ defined inductively as follows: for a tree $t = \sigma(t_1, \ldots, t_k)$ where $k \in \{0, 1, \ldots\}$ it is every pair $(q, p(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_k))$ for every transition $(q_1, \ldots, q_k, \sigma, p, q) \in \Delta$ and reachable configurations on t_i $(q_i, \boldsymbol{\alpha}_i)$ for $i \in \{1, \ldots, k\}$; notice the base of the induction is the case when $t = \sigma$ for some 0-ary symbol σ; in consequence, we call a transition with a 0-ary symbol an *initial configuration*; we denote the set of initial configurations by Δ_0, and the set of transitions where the arity of the symbol is positive by Δ_+. Finally, an output of T on t is $f_{\text{out}}(\boldsymbol{\alpha})$ for any reachable configuration $(q, \boldsymbol{\alpha})$ on t in which the state is q final, i.e. $q \in F$.

Register transducers define (compute) relations (or in an equivalent point of view, multi-valued functions); nevertheless, in this thesis we are mainly interested in transducers that define (compute) functions – every such transducer is called *functional*. A register transducer is *deterministic* if the transition relation is a function in the sense that in every transition $(q_1, \ldots, q_k, \sigma, p, q) \in \Delta$ the states $q_1, \ldots, q_k$ together with the ranked symbol σ determine both the register update p and the state q. Clearly every deterministic register transducer is functional.

We give four examples of register transducers, all of which are functional. We denote the unique output of a functional transducer T on an input tree t as $T(t)$.

We call a register transducer *stateless* if it has only one state and it is a final state; in such case states can be removed from the description of the transducer: a configuration is a tuple of elements of $\mathbb{A}$, and a transition is a pair: (input letter, register update).

Remark 2.2. Register transducers input strings by representing them as monadic trees.

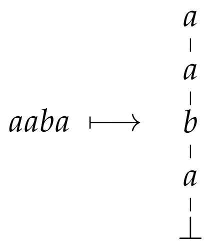


FIGURE 2.2: Strings as monadic trees.

Example 2.3 (Reverse function)**.** Let Σ be a finite alphabet. The function $\texttt{rev} : \Sigma^* \to \Sigma^*$ defined by

$$\texttt{rev}(\sigma_1\sigma_2\dots\sigma_n) := \sigma_n\sigma_{n-1}\dots\sigma_1, \quad \text{for } \sigma_1,\sigma_2,\dots,\sigma_n \in \Sigma, n \in \{0,1,\dots\}$$

can be computed by a stateless deterministic register transducer that inputs trees over the alphabet $\{\sigma^{(1)} \mid \sigma \in \Sigma\} \cup \{\perp^{(0)}\}$ with output in the free monoid $(\Sigma^*,\cdot)$ with one register, transitions $\Delta = \{(\sigma,p_\sigma) \mid \sigma \in \Sigma\} \cup \{(\perp,\varepsilon)\}$ where $p_\sigma : \Sigma^* \to \Sigma^*$ is defined by $p_\sigma(x) := \sigma \cdot x$, and the identity output function.

Example 2.4 (Number of a's below some b)**.** Let Σ be a ranked alphabet and let $a, b \in \Sigma$ be *different* ranked symbols of positive arity. Consider a function $f : \text{RankedTrees}(\Sigma) \to \mathbb{Z}$ defined by

$$f(t) := \text{number of } b\text{-labeled nodes of } t \text{ that have} \\ \text{an } a\text{-labeled ancestor.}$$

This function can be computed by the following nondeterministic register transducer with output in the ring of integers. Intuitively, it has one counter that stores the value of f on the currently read subtree, and the state represents a "guessed in advance" information regarding if some ancestor of the currently read node is labeled by a. Formally, it has one register and two states $Q = \{q_{anc-a}, q_{no-anc-a}\}$, only $q_{no-anc-a}$ being accepting, initial configurations $\Delta_0 = \{(\sigma,q,0) \mid \sigma \in \Sigma, q \in Q\}$, identity output function, and the following transitions with a symbol of positive arity

$$\begin{aligned}
&\{(\bar{q}, a, x \mapsto x, q) \mid \bar{q} \in \{q_{anc-a}\}^{(\text{arity of } a)}, q \in Q\} \cup \\
&\{(\bar{q}, b, x \mapsto x+1, q_{anc-a}) \mid \bar{q} \in \{q_{anc-a}\}^{(\text{arity of } b)}\} \cup \\
&\{(\bar{q}, \sigma, x \mapsto x, q_{anc-a}) \mid \bar{q} \in \{q_{anc-a}\}^{(\text{arity of } \sigma)}, \sigma \in \Sigma \setminus \{a,b\}\} \cup \\
&\{(\bar{q}, \sigma, x \mapsto x, q_{no-anc-a}) \mid \bar{q} \in \{q_{no-anc-a}\}^{(\text{arity of } \sigma)}, \sigma \in \Sigma \setminus \{a\}\}.
\end{aligned}$$

The case when $a = b$ is analogous.

In the next example we describe a register transducer with output in the *algebra of ranked trees*; for a ranked alphabet Σ, ranked trees form an algebra, where every ranked symbol $\sigma \in \Sigma$ of arity $k \in \{0,1,\dots\}$ induces an operation of type $\text{RankedTrees}(\Sigma)^k \to \text{RankedTrees}(\Sigma)$ defined by

$$(t_1,\dots,t_k) \to \sigma(t_1,\dots,t_k).$$

Example 2.5 (Ranked tree to binary tree)**.** Let Σ be a ranked alphabet and let Σ' be a copy of Σ in which the arity of each symbol of non-zero arity is set to 2. Consider the function $\texttt{bin} : \text{RankedTrees}(\Sigma) \to \text{RankedTrees}(\Sigma' \cup \{\bullet^{(2)}\})$ defined by

$$\begin{aligned}
&\texttt{bin}(\sigma(t_1,\dots,t_k)) := \sigma(\texttt{bin}(t_1), \bullet(\texttt{bin}(t_2), \bullet(\dots, \bullet(\texttt{bin}(t_{k-1}), \texttt{bin}(t_k))))), \\
&\texttt{bin}(\sigma) := \sigma, \text{ for } \sigma \text{ of arity } 0,
\end{aligned} \tag{2.1}$$

which converts a ranked tree into a binary tree by introducing a padding symbol •.

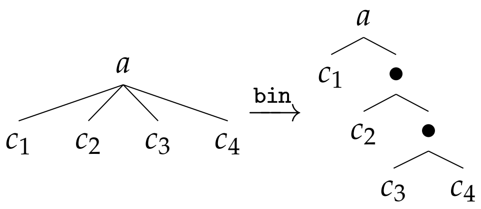


FIGURE 2.3: Function *bin*.

Observe that formula (2.1) can be used to define a stateless deterministic register transducer with output in an algebra of ranked trees with one register and identity output function that computes function bin.

In the last example, Example 2.7, we consider trees labeled by an alphabet that is not ranked. We sometimes call a set, alphabet, or a letter *unranked* to emphasize that it is not ranked. We consider a function unbin which, roughly speaking, is a left inverse of the function bin from Example 2.5. The nodes of trees output by the function unbin can have arbitrarily many children, and therefore this function cannot be captured by a register transducer with output in an algebra of ranked trees over any ranked alphabet. For this reason we introduce an algebra that contains unranked trees, which is similar to the forest algebra introduced by Bojańczyk and Walukiewicz[1], but the trees are *unordered*, i.e. there is no order of the children of the nodes.

Definition 2.6. An *unordered tree* is an unranked tree modulo an equivalence relation that identifies trees that differ only by the order of children of some nodes.

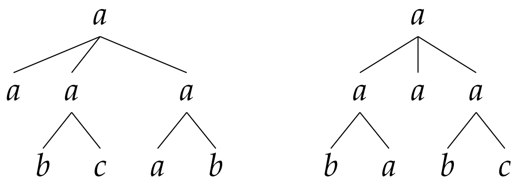


FIGURE 2.4: Two representations of the same unordered tree.

Example 2.7 (Removing padding from binary tree)**.** Let Σ be an unranked alphabet. Let $\mathbb{A}_\Sigma$ be the following algebra:

- the universe consists of *unordered Σ-labeled forests*, which are multisets of Σ-labeled unordered trees,
- there are $|\Sigma| + 2$ basic operations:
  - $\emptyset$ – a constant (0-ary operation), called *empty unordered forest*, that is the empty multiset of unordered trees,
  - root_σ for $\sigma \in \Sigma$ – a unary operation that introduces a new node labeled by σ as a parent of roots of all trees in the forest,

[1] M. Bojańczyk and Walukiewicz, 2008.

- $+$ – a binary operation of multiset addition.

For example, $\mathrm{root}_a(\mathrm{root}_b(\emptyset) + \mathrm{root}_c(\emptyset)) + \mathrm{root}_b(\emptyset)$ is the unordered forest (with two trees) $a(b,c)b$.

Let Σ be a finite ranked alphabet in which every symbol has arity at most 2, and let Σ' be an unranked alphabet that is a copy of Σ with arities removed. Consider the function `unbin` defined by

$$\begin{aligned}
&\mathtt{unbin} : \mathrm{RankedTrees}(\Sigma \cup \{\bullet^{(2)}\}) \to \mathbb{A}_{\Sigma'},\\
&\mathtt{unbin}(\sigma(t_1,t_2)) := \mathrm{root}_\sigma(\mathtt{unbin}(t_1) + \mathtt{unbin}(t_2)),\\
&\mathtt{unbin}(\bullet^{(2)}(t_1,t_2)) := \mathtt{unbin}(t_1) + \mathtt{unbin}(t_2),\\
&\mathtt{unbin}(\sigma(t)) := \mathrm{root}_\sigma(\mathtt{unbin}(t)),\\
&\mathtt{unbin}(\sigma) := \mathrm{root}_\sigma(\emptyset), \text{ for } \sigma \in \Sigma \text{ of arity } 0
\end{aligned} \tag{2.2}$$

whose domain consists of trees in which every $\bullet$-labeled node has a child which is not labeled by $\bullet$ and the root is not labeled by $\bullet$ – it is a regular tree language.

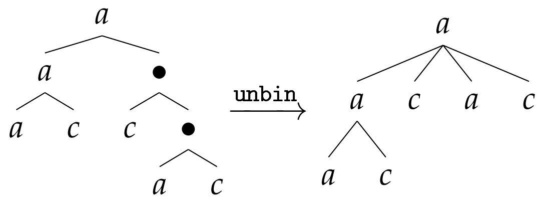


FIGURE 2.5: Function `unbin`.

The function `unbin` can be computed by a deterministic register transducer with output in algebra $\mathbb{A}_{\Sigma'}$ that has one register, whose states verify if the input belongs to the domain, and whose transitions are derived from the formula (2.2).

2.2 The Hilbert Method: equivalence and functionality

Now we define the main problem considered in this thesis, which is equivalence of functional register transducers.

Name: Equivalence of functional register transducers (with output algebra $\mathbb{A}$)

Parameter: $\mathbb{A}$– an algebra

Input: T_1, T_2 – functional **RT**-$\mathbb{A}$ with the same input alphabet.

Question: Is it the case that

$$T_1(t) = T_2(t) \text{ for every input tree } t?$$

In particular, for T_1, T_2 to be equivalent, their domains must be the same. This is a decidable property, as shown in the below lemma.

Lemma 2.8. *Let $\mathbb{A}$ be an algebra. Given two $T_1, T_2 \in$ **RT**-$\mathbb{A}$ with the same input alphabet, one can decide if their domains are the same.*

Proof. First, observe that the domain of a register transducer – over any algebra – is a regular tree language. Indeed, a register transducer admits at least one output on a given input tree if and only if the underlying bottom-up tree automaton of T admits an accepting run on this tree. It is well-known that one can decide if two given regular tree languages are equal. □

Let us emphasize that, in the above problem, if one of the input transducers is not functional, then we do not demand to give the correct answer. The issue of the input correctness gives rise to the *functionality* problem.
Name: Functionality of (nondeterministic) register transducers (with output algebra $\mathbb{A}$)
Parameter: $\mathbb{A}$ – an algebra
Input: T – a (nondeterministic) **RT**-$\mathbb{A}$.
Question: Is it the case that T is functional, i.e.

T has at most one output on every input tree t?

It turns out that equivalence and functionality of register transducers are inter-reducible.

Lemma 2.9. *Let $\mathbb{A}$ be an algebra. Then there are logspace reductions between the decision problems of functionality and equivalence of functional register transducers with output in $\mathbb{A}$.*

Proof. For one direction, let T_1 and T_2 be functional register transducers with output in $\mathbb{A}$. If their domains are not equal, then they are not equivalent (this property can be decided by Lemma 2.8). Otherwise, one can construct a transducer, denote it $T_1 \vee T_2$, in which the image of every input tree is the union of the images of this tree by T_1 and T_2. This is done, roughly speaking, by nondeterministically guessing which of T_1 and T_2 to apply. This can be achieved e.g. by adding a label from $\{1, 2\}$ to every node of the input tree and applying T_i if and only if all the vertices are labeled by i, for $i = 1, 2$. Then T_1 and T_2 are equivalent if and only if the transducer $T_1 \vee T_2$ is functional.

For the converse, let T be a nondeterministic register transducer with output in $\mathbb{A}$ with transition set Δ. Let Δ_1, Δ_2 be two copies of Δ. We construct deterministic register transducers T_1, T_2 that input $(\Delta_1 \times \Delta_2)$-labeled trees, and for every input tree, T_i runs the transducer T (deterministically) with the transitions determined by the labels from the i-th coordinate, for $i = 1, 2$. In this way, for every input tree t and every pair of runs of T on it there exists an input tree for T_1 and T_2 such that its images by T_1, T_2 are the images of t by T with those runs. In consequence, T is functional if and only if T_1 and T_2 are equivalent. □

In the remaining part of this section, we show that equivalence of functional register transducers is decidable when the output algebra is a computable ring with no zero divisors (e.g. the ring of integers) (Theorem 4).

Polynomial grammars. Let us first observe that equivalence of functional register transducers with output in a computable ring with no zero divisors (and hence their functionality) can be reduced to the *zeroness* problem, defined below.
Name: Zeroness of register transducers with output in a ring with no zero divisors
Parameter: R – a ring with no zero divisors
Input: T – a (nondeterministic) **RT**-R.
Question: Is it the case that

$$T(t) = 0 \text{ for every input tree } t?$$

The reduction is as follows. Let T_1, T_2 be functional **RT**-R with the same input alphabet. One checks if T_1 and T_2 have the same domain (Lemma 2.8), and if they do, one constructs a $T \in$ **RT**-R that outputs the difference of outputs of T_1, T_2; sketching the construction, T is essentially a Cartesian product of T_1, T_2, reachable configurations of T on a given input tree are of form $((q_1, q_2), (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2))$, where $(q_i, \boldsymbol{\alpha}_i)$ is some reachable configuration of T_i on this tree for $i = 1, 2$, and the output function of T is the difference of output functions of T_1, T_2. To finish the proof, observe that T_1, T_2 are equivalent if and only if T is equivalent to 0, i.e. zeroness holds for T.

Now let us introduce a grammar model that corresponds to a register transducer.

Definition 2.10. Let $\mathbb{A}$ be an algebra. A *polynomial grammar* with output algebra $\mathbb{A}$ consists of:

- a finite ranked set of *nonterminals* $\mathcal{X}$,
- a distinguished *initial nonterminal* $S \in \mathcal{X}$,
- a finite set of *production rules* P of form $(X, p, X_1, \ldots, X_k)$ where $k \in \{0, 1, \ldots\}$, $X, X_1, \ldots X_k \in \mathcal{X}$ and p is a polynomial operation of type
$$\mathbb{A}^{\operatorname{rank} X_1 + \ldots + \operatorname{rank} X_k} \to \mathbb{A}^{\operatorname{rank} X};$$
a production rule $(X, p, X_1, \ldots, X_k)$ is denoted $X \to p(X_1, \ldots, X_k)$.

The *language of a nonterminal* X of a grammar $G = (\mathcal{X}, S, P)$, denoted $L(X)$, is defined inductively as follows: for every production rule of form (X, p), i.e. with $k = 0$, p – which is constant in such case – belongs to $L(X)$, and for every production rule $(X, p, X_1, \ldots, X_k)$ with $k \in \{1, 2, \ldots\}$, for every $x_i \in L(X_i)$ for $i \in \{1, \ldots, k\}$, $p(x_1, \ldots, x_k)$ belongs to $L(X)$. The *language of a polynomial grammar* G, denoted $L(G)$, is the language of its initial nonterminal. Observe that $L(X) \subseteq \mathbb{A}^{\operatorname{rank} X}$ for a nonterminal X; the *rank of a polynomial grammar* G, denoted rank G, is the rank of its initial nonterminal, hence $L(G) \subseteq \mathbb{A}^{\operatorname{rank} G}$.

Let us justify the correspondence of register transducers with output algebra $\mathbb{A}$ and polynomial grammars with output algebra $\mathbb{A}$, for any algebra $\mathbb{A}$. With a register transducer $T = (\Sigma, Q, F, \mathcal{R}, \Delta_0, \Delta, f_{\text{out}})$ we associate a polynomial grammar G that contains nonterminals $\{X_q \mid q \in Q\}$ for which

$$\boldsymbol{\alpha} \in L(X_q) \text{ if and only if } (q, \boldsymbol{\alpha}) \text{ is a reachable configuration of } T \text{ on some input tree}[2].$$

The grammar G is constructed as follows:

- create nonterminals $\{X_q \mid q \in Q\}$, each of rank $\mathcal{R}$,
- create a production rule

  $$X_q \to p(X_{q_1}, \ldots, X_{q_k}) \text{ for any transition } (q_1, \ldots, q_k, \sigma, p, q) \in \Delta,$$

  in particular, create a production rule

  $$X_q \to \boldsymbol{\alpha} \text{ for any initial configuration } (q, \boldsymbol{\alpha}) \in \Delta_0,$$

- create an initial nonterminal S and a production

  $$S \to f_{\text{out}}(X_q) \text{ for any final state } q \in F.$$

In consequence we have that

$$\text{zeroness holds for } T \text{ if and only if } L(G) \subseteq \{0_R\}^{\text{rank}\, G}. \tag{2.3}$$

From now on we write the right side of the condition (2.3) for polynomial grammars with non-empty language as

$$G = 0.$$

Note that a property of a grammar of having a non-empty language can be decided.

Fact 2.11. *Given a polynomial grammar (with output in any algebra), one can decide if it has a non-empty language.*

Proof. The proof is standard and we omit it. □

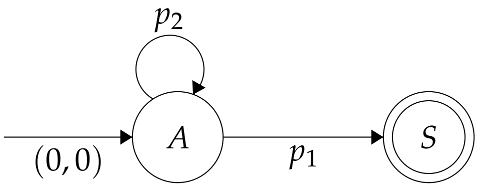
Equivalence (2.3) gives rise to the problem of *zeroness of polynomial grammars with output in a ring with no zero divisors*.
Name: Zeroness of polynomial grammars with output in a ring with no zero divisors
Parameter: R – a computable ring with no zero divisors
Input: G – a polynomial grammar with output in R with non-empty language.
Question: Is it the case that

$$G = 0?$$

Let us give two examples of polynomial grammars and for each of them see if zeroness holds.

[2] In fact, the correspondence is stronger: every derivation tree of G that starts in X_q, denote the tuple it produces by α, corresponds to a run of T that ends in the state q whose register valuation is α.

Example 2.12. Consider the following polynomial grammar G with output in the ring of integers: $S \to p_1(A), A \to p_2(A) \mid (0,0)$ where $p_1(x,y) = x^2 - y^2$ and $p_2(x,y) = (x+1, y-1)$.

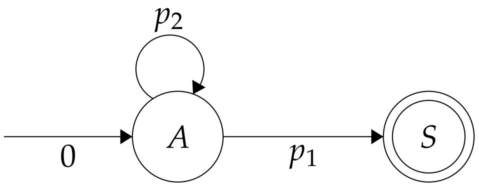


$$p_1(x,y) = x^2 - y^2,$$
$$p_2(x,y) = (x+1, y-1).$$

FIGURE 2.6: An illustration of an example grammar for which zeroness holds.

We prove that zeroness holds for G by showing inductively that a (polynomial) *invariant* $x + y = 0$ holds for all $(x,y) \in L(A)$. Indeed, it holds for a tuple produced by a minimal derivation $A \to (0,0)$ $(0+0=0)$ and moreover it is *inductive* i.e. it is preserved by the production rule $A \to p_2(A)$ – we have $x+y=0 \Rightarrow (x+1)+(y-1)=0$. In consequence, for all $a \in L(S)$ we have $a = x^2 - y^2$ for some $(x,y) \in L(A)$ and hence $a = (x+y)(x-y) = 0 \cdot (x-y) = 0$, which finishes the proof of zeroness of G.

Example 2.13. Consider another example polynomial grammar H with output in the ring of integers: $S \to p_1(A), A \to p_2(A) \mid 0$, where $p_1(x) = x^2 - x$ and $p_2(x) = x+1$.



$$p_1(x) = x^2 - x,$$
$$p_2(x) = x+1.$$

FIGURE 2.7: An illustration of an example grammar for which zeroness does not hold.

Zeroness does not hold for H; the derivation

$$S \to p_1(A) \to p_1(p_2(A)) \to p_1(p_2(p_2(A))) \to p_1(p_2(p_2(0))) = 3$$

produces $3 \neq 0$.

These two examples together illustrate the algorithm for zeroness we now propose (provided the grammar's language is non-empty): search in parallel, (1) for an invariant that witnesses zeroness (as in the first example) and (2) for a derivation that witnesses non-zeroness (as in the second example). In the rest of this section we define the notion of invariant formally, prove there exists one that witnesses zeroness in the case when zeroness holds, and show that candidates for an invariant that witnesses zeroness can be effectively enumerated and verified.

Decidability of zeroness. Let K be a Noetherian ring. The observation is as follows: fix some set $C \subseteq K^n$; if polynomials $f, g \in K[x_1, \ldots, x_n]$ satisfy the property that $f(\overline{x}) = 0$ and $g(\overline{x}) = 0$ for all tuples $\overline{x} \in C$, then the same holds for $f + g$ and $f \cdot h$ for any polynomial $h \in K[x_1, \ldots, x_n]$ – in other words, polynomials $f \in K[x_1, \ldots, x_n]$ such that $f(\overline{x}) = 0$ for $\overline{x} \in C$ form an *ideal* in $K[x_1, \ldots, x_n]$. Let us recall that due to Hilbert's Basis Theorem every ideal $I \subseteq K[x_1, \ldots, x_n]$ admits a finite set of generators, i.e. a set F such that $\langle F \rangle = I$. In consequence, for algorithmic purposes we assume that every ideal is represented by a finite generating set.

Definition 2.14. In the proof we will use the *field of fractions* of a ring with no zero divisors R, which is roughly speaking the smallest field that contains R (for example, the field of fractions of $\mathbb{Z}$ is $\mathbb{Q}$; it exists only for rings with no zero divisors); formally, the field of fractions of R is the set of pairs $(r, s) \in R^2$ modulo equivalence relation $(r_1, s_1) \sim (r_2, s_2) \Leftrightarrow r_1 s_2 = r_2 s_1$ – a pair (r, s) intuitively represents the fraction $\frac{r}{s}$, and the ring operations are defined as for usual fractions.

Definition 2.15. An ideal I in a ring R is *proper* if $I \neq R$.

Definition 2.16 (Inductive invariant)**.** Let $G = (\mathcal{X}, S, P)$ be a polynomial grammar with output in R in which every nonterminal has a non-empty language (this is without loss of generality) and let K be the field of fractions of R (it is a computable field because R is a computable ring). For a production rule $X \to p(X_1, \ldots, X_k)$, we represent the polynomial operation p by a (rank X)-tuple of polynomials from the ring $K[t_{X_1, i_1}, \ldots, t_{X_k, i_k} \mid i_j \in \{1, \ldots, \operatorname{rank} X_j\}, j \in \{1, \ldots, k\}]$. An *inductive invariant (for G)* is an $\mathcal{X}$-tuple of *proper*[3] ideals $\mathcal{I} = (I_X)_{X \in \mathcal{X}}$ where $I_X \subseteq K[t_{X,i} \mid i \in \{1, \ldots, \operatorname{rank} X\}]$ that satisfies the following conditions:

(a) $I_X(a) = 0$ for any production rule $X \to a$, where a is constant,

(b) $\langle I_{X_1} \cup \ldots \cup I_{X_k} \rangle \supseteq I_X \circ p$ for all production rules $(X, p, X_1, \ldots, X_k)$ (notice that X can be initial).

If additionally

(c) $I_S = \langle t_{S,i} \mid i \in \{1, \ldots, \operatorname{rank} S\} \rangle$,

then we say that $\mathcal{I}$ *witnesses zeroness (of G)*.

In this subsection, by abuse of notation we write X instead of $L(X)$, for a nonterminal X.

[3] If an invariant consisted of all polynomials (i.e. was defined by a non-proper ideal), it would describe an empty set; we choose not to include such possibility.

Lemma 2.17. *For a polynomial grammar with output in a ring in which every nonterminal has a non-empty language and an inductive invariant $\mathcal{I}$ we have*

$$I_X(X) = 0 \tag{2.4}$$

i.e. every tuple $\overline{x} \in X$ satisfies invariants $f = 0$ for all $f \in I_X$.

Proof. Observe that condition (a) of an inductive invariant states that (2.4) holds for any a that can be derived from a production rule of form $X \to a$ for some nonterminal X, and hence is an induction base. Condition (b) yields an inductive step of the proof: for $\overline{x}_1, \ldots, \overline{x}_k$ derivable from, respectively, $X_1, \ldots, X_k$, for which $I_{X_i}(\overline{x}_i) = 0$ for $i \in \{1, \ldots, k\}$ and a production rule $X \to p(X_1, \ldots, X_k)$ we have that $\overline{x}$ satisfies $I_X(\overline{x}) = 0$ – indeed, $I_X(\overline{x}) = I_X(p(\overline{x}_1, \ldots, \overline{x}_k)) = (I_X \circ p)(\overline{x}_1, \ldots, \overline{x}_k) \subseteq (\langle I_{X_1} \cup \ldots \cup I_{X_k} \rangle)(\overline{x}_1, \ldots, \overline{x}_k) = I_{X_1}(\overline{x}_1) + \ldots + I_{X_k}(\overline{x}_k) = 0.$ □

Lemma 2.18. *For a polynomial grammar with output in a ring with no zero divisors in which every nonterminal has a non-empty language, zeroness holds if and only if there exists an inductive invariant that witnesses zeroness.*

Proof. For the right-to-left direction, see that condition (2.4), in presence of (c), clearly witnesses zeroness of G.

For the converse, assume that zeroness holds for G. Let $\mathcal{I} = (I_X)_{X \in \mathcal{X}}$ be the strongest invariant that holds for G, that is

$$I_X := \{f \in K[t_{X,i} \mid i \in \{1, \ldots, \operatorname{rank} X\}] \mid f(X) = 0\} \text{ for } X \in \mathcal{X}.$$

It is an immediate consequence of an exercise from a textbook of Cox, Little, and O'Shea[4] that $\mathcal{I}$ witnesses zeroness provided zeroness holds for G. It remains to prove that it is an inductive invariant. Condition (a) of an inductive invariant holds for $\mathcal{I}$ – for a production rule $X \to a$ we have $I_X(a) \subseteq I_X(X) = 0$. For condition (b), take a production rule $X \to p(X_1, \ldots, X_k)$. Due to a lemma from a paper of Seidl et al.[5] (which is a classical fact) for any nonterminals $X_1, \ldots, X_k$ we have

$$\langle I_{X_1} \cup \ldots \cup I_{X_k} \rangle = \{f \in K[t_{X_i,j} \mid i \in \{1, \ldots, k\}, j = 1, \ldots, \operatorname{rank} X_i] \mid f(X_1 \times \ldots \times X_k) = 0\}.$$

In consequence, (b) is equivalent to $(I_X \circ p)(X_1 \times \ldots \times X_k) = 0$; this holds, as $(I_X \circ p)(X_1 \times \ldots \times X_k) = I_X(p(X_1 \times \ldots \times X_k)) \subseteq I_X(X) = 0.$ □

A candidate for an invariant is an $\mathcal{X}$-tuple of proper ideals (it is decidable if a given ideal I is proper by checking $1 \notin I$), hence candidates can be enumerated. For effectiveness of the proposed algorithm, it remains to prove that one can verify if a given candidate is an inductive invariant that witnesses zeroness.

Lemma 2.19. *Given a tuple of proper ideals $\mathcal{I} = (I_X)_{X \in \mathcal{X}}$, $I_X \subseteq K[t_{X,i} \mid i \in \{1, \ldots, \operatorname{rank} X\}]$ one can verify if it is an inductive invariant that witnesses zeroness.*

[4] Cox, Little, and O'Shea, 2015, Exercise 18a., §4, Chapter 1.

[5] Seidl, Maneth, and Kemper, 2018, Lemma 6.3.

Proof. Regarding condition (a) of an inductive invariant, observe that for a proper ideal I of an arbitrary ring of polynomials $K[x_1, \ldots, x_n]$ and a constant $a = (a_1, \ldots, a_n) \in K^n$ we have $I(a) = 0$ if and only if $I = \langle x_i - a_i \mid i \in \{1, \ldots, n\}\rangle$ (this is a shifted variant of an exercise from a textbook of Cox, Little, and O'Shea[6]). Regarding condition (b), observe that for ideals $I_1, \ldots, I_k$ ideal $\langle I_1 \cup \ldots \cup I_k\rangle$ can be computed from $I_1, \ldots, I_k$ (take the union of sets of generators as the generating set), and for ideal I and a polynomial operation p the set $I \circ p$ is an ideal and can be computed from I and p (take the set of $f \circ p$ for generators f of I as the generating set).

In consequence, all conditions of an inductive invariant (i.e. (a), (b), and (c)) can be reduced to Ideal Inclusion, which is decidable (1.20). □

In consequence, the proposed algorithm is indeed effective, which finishes the proof of the following theorem[7].

Theorem 4. *Zeroness of polynomial grammars with output in a computable ring with no zero divisors is decidable.*

Theorem 5. *Functionality and equivalence of functional register transducers with output in a computable ring with no zero divisors is decidable.*

2.3 First application: unordered trees

Denote the algebra of unordered forests from Example 2.7 with a unary alphabet Σ by $\mathbb{UF}$. The aim of this section is to prove that functionality and equivalence of functional register transducers with output algebra $\mathbb{UF}$ is decidable (Theorem 6). To this end, we "simulate" RT-$\mathbb{UF}$ with RT-$\mathbb{Z}[x]$ (the notion of simulation will be formalized in Section 2.4), for which this problem is decidable due to Theorem 5 (as, clearly, the ring $\mathbb{Z}[x]$ has no zero divisors).

The use of the ring $\mathbb{Z}[x]$ instead of $\mathbb{Z}$ is, to our best knowledge, new. To avoid confusion, let us emphasize that the register updates of a register transducer with output in $\mathbb{Z}[x]$ are polynomial functions with coefficients in $\mathbb{Z}[x]$; an example of such polynomial function is $f(t) = (x^2 + x)t^2 + xt + (x^2 - x) \in (\mathbb{Z}[x])[t]$.

Consider the following polynomial $\phi_f \in \mathbb{Z}[x]$ associated to any unordered forest $f \in \mathbb{UF}$. It is defined inductively as:

$$\begin{aligned}
&\phi : \mathbb{UF} \to \mathbb{Z}[x], \\
&\phi_\varnothing := 1, \\
&\phi_{\mathrm{root}(f)} := x \cdot \phi_f + 2, \\
&\phi_{f+g} := \phi_f \cdot \phi_g.
\end{aligned} \tag{2.5}$$

[6] Cox, Little, and O'Shea, 2015, Exercise 18a., §4, Chapter 1.

[7] Let us remark that the algorithms that decide problems from Theorem 4 and Theorem 5 are uniform in R, i.e. R could be a part of the input.

The function ϕ describes how to, given a functional $T \in \textsf{RT-}\mathbb{UF}$, construct a $\phi T \in \textsf{RT-}\mathbb{Z}[x]$ that returns $\phi_{T(t)}$ on an input tree t. Explicitly, ϕT differs from T only in register updates – in particular, it has the same registers as T – and whenever T applies a basic operation of unordered forests $\theta \in \{\emptyset, \text{root}, +\}$, the transducer ϕT applies the corresponding polynomial function with coefficients in $\mathbb{Z}[x]$ from the corresponding right-hand side of (2.5).

Corollary 2.20. *Let ϕ be the function defined in* (2.5). *Then, for every functional $T \in \textsf{RT-}\mathbb{UF}$ there is a canonical functional $\phi T \in \textsf{RT-}\mathbb{Z}[x]$ such that*

$$(\phi T)(t) = \phi_{T(t)} \textit{ for every input tree } t.$$

To finish the reduction, we claim that any $T_1, T_2 \in \textsf{RT-}\mathbb{UF}$ are equivalent if and only if $\phi T_1, \phi T_2 \in \textsf{RT-}\mathbb{Z}[x]$ are equivalent – this is a consequence of the following lemma.

Lemma 2.21. *The function ϕ defined in* (2.5) *is injective.*

Proof. We prove inductively that (1) ϕ_f is *monic*, i.e. has leading coefficient 1, for all $f \in \mathbb{UF}$, and (2) if $f = t_1 + \ldots + t_n$ is the decomposition of a forest f into a multiset of trees, then

$$\begin{aligned}\phi_f = \ \phi_{t_1} \cdot \ldots \cdot \phi_{t_n} &\text{ is the decomposition of a monic polynomial } \phi_f \\ &\text{into monic } \textit{irreducible} \text{ polynomials (i.e. ones that cannot be further} \\ &\text{decomposed into product of polynomials of } \mathbb{Z}[x] \text{ of positive degree).}\end{aligned} \tag{2.6}$$

Using (1) and (2), the proof of injectivity of ϕ is as follows: for any $f, g \in \mathbb{UF}$ we have that

- (induction base) $\phi_f = \phi_\emptyset = 1$ if and only if $f = \emptyset$,
- (injectivity on trees) $\phi_{\text{root}(f)} = \phi_{\text{root}(g)} \Leftrightarrow x \cdot \phi_f + 2 = x \cdot \phi_g + 2 \Leftrightarrow \phi_f = \phi_g$, hence injectivity on trees reduces to injectivity on *smaller* forests,
- (injectivity on forests) if $f = t_1 + \ldots + t_n$ and $g = s_1 + \ldots + s_m$, $n \in \{2, 3, \ldots\}, m \in \{1, 2, \ldots\}$ are the decompositions of forests f, g into multisets of trees, then due to (2.6) we have that $\phi_f = \phi_g$ if and only if multisets of polynomials $\{\phi_{t_i} \mid i \in \{1, \ldots, n\}\}$ and $\{\phi_{s_j} \mid j \in \{1, \ldots, m\}\}$ are equal, hence injectivity on forests reduces to injectivity on *smaller* trees.

In the proof of (2), we use Eisenstein's Criterion for irreducibility of a polynomial in $\mathbb{Z}[x]$.

Eisenstein's Criterion. Let $r(x) = a_n x^n + \ldots + a_1 x^1 + a_0 x^0 \in \mathbb{Z}[x]$ and let p be a prime number. If $p \mid a_i$ for $i \in \{0, \ldots, n-1\}$, $p \nmid a_n$, $p^2 \nmid a_0$, then r is an irreducible polynomial of $\mathbb{Z}[x]$.

Now we prove (1) and (2). We prove inductively that for every forest $f \in \mathbb{UF}$ (1) ϕ_f is monic (2′) all the non-leading coefficients of ϕ_f are divisible by 2, (2″) the coefficient of ϕ_t standing by x^0 is not divisible by 4, for any tree $t \in \mathbb{UF}$ (conditions (2′) and (2″) imply condition (2) by Eisenstein's Criterion).

Condition (2″) can be proved without using induction, as if $t = \mathrm{root}(f)$ then the coefficient of $\phi_t = x \cdot \phi_f + 2$ standing by x^0 is 2, which is not divisible by 4. Now we prove (1) and (2′). For the induction base, $\phi_\emptyset$ is monic and $\phi_{\mathrm{root}(\emptyset)} = x + 2$ satisfies (2′). For the induction step, if ϕ_f, ϕ_g are monic and satisfy (2′), then the same holds for $x \cdot \phi_f + 2 = \phi_{\mathrm{root}(f)}$ and $\phi_f \cdot \phi_g = \phi_{f+g}$. □

Corollary 2.22. *Any functional $T_1, T_2 \in$ **RT**-$\mathbb{UF}$ are equivalent if and only if $\phi T_1, \phi T_2 \in$ **RT**-$\mathbb{Z}[x]$ (which are functional) are equivalent.*

Theorem 6. *Functionality and equivalence of functional **RT**-$\mathbb{UF}$ are decidable.*

Proof. Combine Corollary 2.22 and Theorem 5. □

2.4 The proof scheme: polynomial simulation

In the forthcoming sections (and chapters) we will apply the proof scheme used in the previous section but for different algebras: for an algebra $\mathbb{A}$ of our interest, we will prove decidability of functionality and equivalence of functional **RT**-$\mathbb{A}$ by "simulating" $\mathbb{A}$ with a ring R with no zero divisors, thus "simulating" **RT**-$\mathbb{A}$ with **RT**-R, for whom the problem is decidable. In this section, we formalize this proof scheme, in particular the notion of simulation.

Definition 2.23 (Compositional function)**.** Let $\mathbb{A} = (A, \Theta_\mathbb{A}), \mathbb{B} = (B, \Theta_\mathbb{B})$ be algebras. Let $\phi : A \to B^m$ be a function for some $m \in \{1, 2, \ldots\}$ and $(\eta_\theta)_{\theta \in \Theta_\mathbb{A}}$ be polynomial operations of type $(\mathbb{B}^m)^{(\text{arity of } \theta)} \to \mathbb{B}^m$ that interpret the operations from $\Theta_\mathbb{A}$ in the image of ϕ in the sense that

$$\phi(\theta(\bar{a})) = \eta_\theta((\phi(a))_{a \in \bar{a}}) \text{ for all } \bar{a} \in A^{(\text{arity of } \theta)}. \tag{2.7}$$

The above formula is equivalent to the following diagram being commutative.

$$\begin{tikzcd}
A^{(\text{arity of } \theta)} \arrow[r, "\theta"] \arrow[d, hook, "\phi \times \ldots \times \phi"] & A \arrow[d, hook, "\phi"] \\
(B^m)^{(\text{arity of } \theta)} \arrow[r, "\eta_\theta"] & B^m
\end{tikzcd}$$

FIGURE 2.8: A basic operation θ of $\mathbb{A}$ is "simulated" by a polynomial operation η_θ of $\mathbb{B}$ in the image of $\mathbb{A}$ by ϕ.

We call every such function ϕ *compositional* (with respect to $(\eta_\theta)_{\theta \in \Theta_\mathbb{A}}$).

Observation 2.24. *Let $\mathbb{A} = (A, \Theta_\mathbb{A}), \mathbb{B} = (B, \Theta_\mathbb{B})$ be algebras. Let $\phi : A \to B^m$ be a compositional function for some $m \in \{1, 2, \ldots\}$. Then, for every functional $T \in$ **RT**-$\mathbb{A}$ there is a canonical functional $\phi T \in$ **RT**-$\mathbb{B}$ such that*

$$(\phi T)(t) = \phi(T(t)) \textit{ for every input tree } t.$$

(Technically speaking, the definition of ϕT depends also on the particular $(\eta_\theta)_{\theta \in \Theta_{\mathbb{A}}}$ that certifies compositionality of ϕ, nevertheless both the register values and the output of ϕT depend only on ϕ.)

Definition 2.25 (Polynomial simulation)**.** Let $\mathbb{A} = (A, \Theta_{\mathbb{A}}), \mathbb{B} = (B, \Theta_{\mathbb{B}})$ be algebras. A *polynomial simulation* of $\mathbb{A}$ by $\mathbb{B}$ is a function $\phi : A \to B^m$ for some $m \in \{1, 2, \dots\}$ that is

1. injective, and
2. compositional.

The related mapping $T \mapsto \phi T$ is called a *polynomial simulation* of RT-$\mathbb{A}$ by RT-$\mathbb{B}$; we say that $\phi(a)$ simulates an element $a \in A$, η_θ simulates a basic operation $\theta \in \Theta_{\mathbb{A}}$, and ϕT polynomially simulates a register transducer T.

Lemma 2.26. *If ϕ is a polynomial simulation of $\mathbb{A}$ by $\mathbb{B}$, then for functional $T_1, T_2 \in$ RT-$\mathbb{A}$, T_1 is equivalent to T_2 if and only if ϕT_1 is equivalent to ϕT_2; in other words, then equivalence of functional* RT-$\mathbb{A}$ *reduces to equivalence of functional* RT-$\mathbb{B}$.

Corollary 2.27 (Proof scheme)**.** *Let $\mathbb{A}$ be an algebra and R be a computable ring with no zero divisors. If there is a polynomial simulation of $\mathbb{A}$ by R, then equivalence of functional* RT-$\mathbb{A}$ *is decidable.*

2.5 From trees to contexts

In this section, we introduce contexts, which are forests with one distinguished leaf for which one can substitute other forests (Figure 2.9). In the next section we will extend the algebra of unordered forests from Example 2.7 by contexts, thus obtaining an unordered variant of the free forest algebra introduced by Bojańczyk and Walukiewicz[8], but first, in this section we consider the, original, ordered variant. We show that adding contexts increases the expressiveness of register transducers (Example 2.32), whilst preserving decidability of their functionality (Corollary 2.34).

Definition 2.28 (Context)**.** Let Σ be an unranked alphabet. An (ordered) Σ-*forest* is a list of sibling ordered Σ-labeled trees. Let ? be a fresh symbol. An (ordered) Σ-*context (with one hole)* is a $(\Sigma \cup \{?\})$-forest with exactly one ?-labeled node which necessarily is a leaf.

Intuitively, the (unique) ?-labeled node of a context represents the part of a forest that has not yet been defined: the ?-labeled node may substituted with a forest or a context, as shown in Figure 2.9.

Definition 2.29 (Context composition)**.** Let C be a context and F be a forest (context). The *composition* of C and F, denoted $C \cdot F$, is the forest (context) resulting from substituting F for the ?-labeled leaf in C. If the substituted forest (context) consists of more than one tree, its trees become children of

[8] M. Bojańczyk and Walukiewicz, 2008.

the parent of ?-node in the context to which we substitute. For example, consider the $\{a, b, c\}$-context $C := a(a, b(?))b(b)$. Then, the composition of C and the tree $b(c)$ results in the forest $a(a, b(b(c)))b(b)$, and the composition of C and the context $b(c)?a$ results in the context $a(a, b(b(c), ?, a))b(b)$.

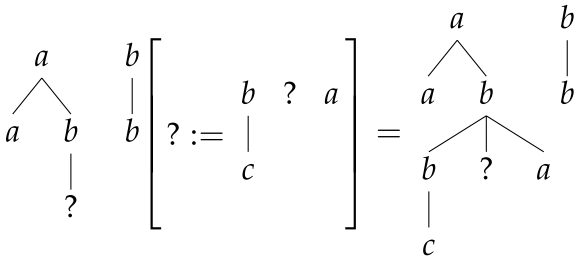


FIGURE 2.9: Composition of contexts.

Definition 2.30. [Free (unordered) forest algebra] We consider the following 2-sorted algebra[9].
Name: free forest algebra (over Σ)
Universe: Σ-forests (sort 0) and Σ-contexts (sort 1)
Basic operations:

- $\emptyset$ – a constant (0-ary operation), called the *empty forest*, that is the empty list of trees; is of sort 0,
- ? – a constant (0-ary operation) that is a tree that consists of one ?-labeled node; is of sort 1,
- $\sigma(?)$ for $\sigma \in \Sigma$ – a constant (0-ary operation); is of sort 1,
- $+$ – a binary operation of list concatenation; formally, there are three operations $+$ of types $0 \times 0 \to 0, 0 \times 1 \to 1$, and $1 \times 0 \to 1$,
- $\cdot$ (also denoted by $(-)[? := (-)]$) – a binary operation of context composition; formally, there are two operations $\cdot$ of types $1 \times 0 \to 0$ and $1 \times 1 \to 1$.

We define the *free unordered forest algebra (over an alphabet* Σ*)* analogously: it consists of unordered Σ-forests (which are multisets of unordered Σ-labeled trees) and *unordered* Σ*-contexts (with one hole)*, which are unordered $(\Sigma \cup \{?\})$-forests with exactly one ?-labeled node which necessarily is a leaf; we omit the description of the basic operations; if the alphabet is unary, we denote this algebra by $\mathbb{CUF}$.

Let us emphasize that every forest can be built using constants from $\{\emptyset\} \cup \{\sigma(?) \mid \sigma \in \Sigma\}$ and basic operations from $\{+, \cdot\}$ – for example, the forest $a(a, b(c), b)b$ is obtained by $a(?) \cdot (a(?) \cdot \emptyset + b(?) \cdot (c(?) \cdot \emptyset) + b(?) \cdot \emptyset) + b(?) \cdot \emptyset$.

Remark 2.31 (Multisorted output algebras, multisorted polynomial simulations)**.** Consider a situation where the output algebra of a register transducer

[9] Ibid.

is a multisorted algebra (for example the free forest algebra). Such a register transducer can clearly, at the cost of extra states, keep the information about the sort of each of the elements currently kept in the registers. In consequence, the construction from Observation 2.24 (cf. Corollary 2.20) can be applied to *multisorted* algebras as well: if $\phi : \mathbb{A} \to \mathbb{B}$ is a multisorted compositional function, then from a $T \in \textsf{RT}\text{-}\mathbb{A}$ one can construct a $\phi T \in \textsf{RT}\text{-}\mathbb{B}$ such that $(\phi T)(t) = \phi(T(t))$ for every input tree t.

Also, we define the notion of polynomial simulation in the case when $\mathbb{A}$ is a multisorted algebra, but $\mathbb{B} = (B, \Theta_{\mathbb{B}})$ is a (single-sorted) algebra. In such case, we require that for each sort of $\mathbb{A}$ there is an injective mapping to a set of form B^m for some $m \in \{1, 2, \ldots\}$, and each basic operation of $\mathbb{A}$ is simulated by a polynomial operation of $\mathbb{B}$.

Example 2.32 (Adding contexts increases expressiveness)**.** The function that reverses the order of the unary nodes of a monadic tree can be defined by a register transducer with output either in the free forest algebra or in the free unordered forest algebra. Moreover, if the alphabet has at least two unary symbols, the use of contexts during the run is necessary.

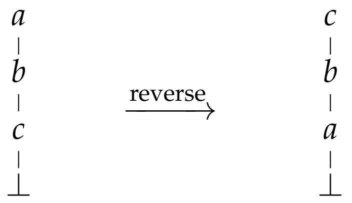


FIGURE 2.10: Reverse of a monadic tree.

Indeed, this function can be computed by the following stateless deterministic register transducer with one register R: the register R is updated over a unary symbol σ by $R := R \cdot \sigma(?)$, and the output is $R \cdot \bot$. We omit the proof of the second claim, as it is straightforward.

Now we proceed to the proof of Corollary 2.34.

Fact 2.33. *Register transducers with output in the free forest algebra can be polynomially simulated, in the sense of Remark 2.31, by register transducers with output in the free monoid.*

Proof. Let Σ be a finite alphabet. We polynomially simulate, in the sense of Remark 2.31, the free forest algebra over Σ by a free monoid.

A forest represented by a word w, over the alphabet Σ enriched with two symbols that represent the brackets, is mapped to the same word w, and a context represented by a word $w?v$ is mapped to the pair of words (w, v). It is straightforward that the basic operations of the free forest algebra can be simulated using concatenation: for example, the composition of a pair of contexts that are mapped to (w_1, v_1) and (w_2, v_2) is mapped to (w_1w_2, v_2v_1); we omit the rest of the details. □

Corollary 2.34. *Functionality and equivalence of functional of register transducers with output in the free forest algebra is decidable.*

Proof. We use the well-known fact that the free monoid over Σ can be polynomially simulated by a ring of integers, for example by embedding it into the free monoid over the binary alphabet, which is then embedded into the multiplicative monoid of 2×2 integer-entry matrices by mapping the letters of the alphabet to the generators of the Sanov group (Figure 2.11). The claim follows from Corollary 2.27.

$$\begin{aligned} \{a, b\}^* &\to \mathbb{M}_{2\times 2}(\mathbb{Z}), \\ a &\mapsto \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}, \\ b &\mapsto \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}. \end{aligned}$$

FIGURE 2.11: Mapping the letters of the alphabet to the generators of the Sanov group.

□

2.6 Unordered trees are more expressive than ordered trees

In this section, we compare the expressiveness of register transducers with output in either free forest algebra or free unordered forest algebra. We show that the unordered variant is at least as expressive as the, usual, ordered one, by constructing a polynomial simulation from the latter to the former (Fact 2.35). Then, we show that the unordered variant is decidable (Theorem 7).

Before we prove Fact 2.35, let us show how ordered *ranked* trees (and contexts) can be polynomially simulated by unordered *ranked* trees (and contexts).

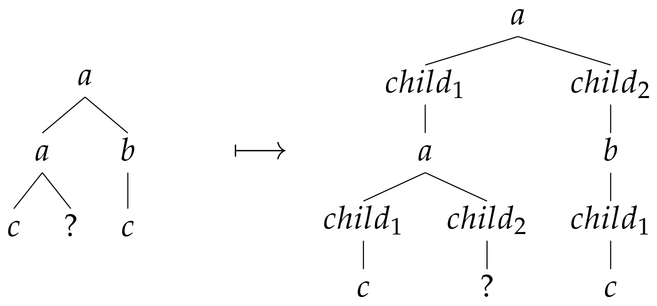


FIGURE 2.12: An ordered ranked context over the alphabet $\{a^{(2)}, b^{(1)}, c^{(0)}\}$ mapped to an unordered ranked context over the alphabet $\{a^{(2)}, b^{(1)}, c^{(0)}\} \cup \{child_1^{(1)}, child_2^{(1)}\}$.

Fact 2.35. *Register transducers with output in the free forest algebra over a finite alphabet can be polynomially simulated by register transducers with output in the free unordered forest algebra over a unary alphabet ($\mathbb{CUF}$).*

Proof. The proof consists of three polynomial simulations, whose composition is a polynomial simulation of the free forest algebra over a finite alphabet by $\mathbb{CUF}$.

First, we take a polynomial simulation, in the sense of Remark 2.31, of a free forest algebra by a free monoid (Fact 2.33). Second, we polynomially simulate a free monoid by a free unordered forest algebra: the word $w = \sigma_1\sigma_2\ldots\sigma_n$ is mapped to the context $\sigma_1(\sigma_2(\ldots(\sigma_n(?))))$; the only basic operation – concatenation – is simulated by (context) composition. Finally, we define a "delabeling" of unordered Σ-contexts, for a finite alphabet Σ, i.e. a polynomial simulation of a free unordered forest algebra over Σ by the algebra $\mathbb{CUF}$. To define it, we put some order on the alphabet Σ, thus obtaining $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$. Then, $\emptyset$ is mapped to $\emptyset$, ? is mapped to ?, σ_i is mapped to $\bullet(\bullet(\underbrace{\bullet, \bullet, \ldots, \bullet}_{i+2}, ?))$, and the basic operations $+, \cdot$ remain unchanged.

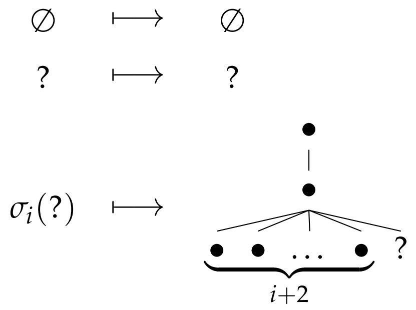


FIGURE 2.13: "Delabeling" unordered forests.

It has the following effect on a forest: every σ_i-labeled node is mapped to a node with $i+2$ additional children for $i \in \{1, \ldots, n\}$, and every edge is subdivided (the root of the image of $\sigma_i(?)$ is the division node); it is hence clear it is injective. □

Theorem 7. *Functionality and equivalence of functional of register transducers with output in the free unordered forest algebra over a unary alphabet ($\mathbb{CUF}$) is decidable.*

Proof. We polynomially simulate the algebra $\mathbb{CUF}$ with the ring $\mathbb{Z}[x]$; the claim follows from Corollary 2.27.

Observe that the polynomial simulation ϕ from (2.5) naturally extends to a compositional mapping from $\mathbb{CUF}$ to $(\mathbb{Z}[x])[?]$ which maps ? to ?. By a standard reasoning, this extension is well-defined, compositional with respect to operations of forests $\emptyset, ?, \sigma(?), +, \cdot$, and injective. It also is compositional with respect to the substitution operation, even in the stronger sense:

$$\phi_{f[?:=g]} = \phi_f[? := g] \text{ for } f, g \in \mathbb{CUF}.$$

In consequence, this extension is a polynomial simulation of the 2-sorted algebra $\mathbb{CUF}$ by the 2-sorted algebra of "$\mathbb{Z}[x]$ and $\mathbb{Z}[x]$-contexts", which we denote $\mathbb{C}$, defined naturally as follows: the universe of $\mathbb{C}$ consists of polynomials from $(\mathbb{Z}[x])[?]$ of form $a(x) + b(x)?$ for $a(x), b(x) \in \mathbb{Z}[x]$; the ones with $b = 0$ are of sort 0 and the remaining ones are of sort 1; informally, the operations are $+, \cdot$, and $(-)[? := (-)]$ – formally, for every $i, j \in \{0,1\}$ there is an operation $+$ of type $i \times j \to \max(i,j)$, there are three $\cdot$ operations of types $0 \times 0 \to 0, 0 \times 1 \to 1, 1 \times 0 \to 1$, and there are two $(-)[? := (-)]$ operations of types $1 \times 0 \to 0, 1 \times 1 \to 1$. Now observe that the algebra $\mathbb{C}$ can be polynomially simulated by the ring $\mathbb{Z}[x]$, in the sense of Remark 2.31. The construction is similar to the one from the proof of Fact 2.35, but now with the ring $\mathbb{Z}[x]$ instead of a free monoid. A polynomial $a(x)$ (of sort 0) is mapped to itself, and a polynomial $a(x) + b(x)?$ where $b(x) \neq 0$ (of sort 1) is mapped to the pair $(a(x), b(x))$. It is straightforward that the basic operations of the algebra $\mathbb{C}$ can be polynomially simulated using the operations $+, \cdot$ of the ring $\mathbb{Z}[x]$: for example, the substitution of a pair of contexts that are mapped to $(a_1(x), b_1(x))$ and $(a_2(x), b_2(x))$ is mapped to $(a_1(x) + a_2(x) \cdot b_1(x), b_1(x) \cdot b_2(x))$; we omit the rest of the details. □

2.7 Summary

In this chapter, we define register transducers, which transform ranked trees into elements of an arbitrary algebra. We also define grammars that correspond to register transducers (polynomial grammars). Using the grammar model, we show that for the output algebra being a computable ring with no zero divisors (like the ring of integers, or of polynomials), functionality and equivalence of functional register transducers are decidable (Theorem 5). As the first application of Theorem 5, we prove decidability of functionality and equivalence of functional register transducers with output in the algebra $\mathbb{UF}$ which consists of unordered forests (Theorem 6). We also compare the expressiveness of register transducers that output either ordered or unordered trees and use contexts during the run, and show, by a reduction to Theorem 6, that functionality and equivalence of functional register transducers is decidable in both cases (Theorem 7). We formalize the scheme of the proof of Theorem 6 (Section 2.4), introducing the notion of polynomial simulation; we use this proof scheme in each of the forthcoming chapters.

Let us mention the complexity of the problem we reduce to, i.e. equivalence of functional register transducers with output in a ring with no zero divisors[10]: there is an Ackermann-hard lower bound in general[11] and the problem is Ackermann-complete in the simplest case[12] when the output ring is the ring of integers[13]. Nevertheless, there are notable subclasses of register transducers with output in the ring of integers for which zeroness is primitive recursive[14], or even PSPACE[15].

[10] Technically, we reduce to the inter-reducible problem of zeroness.

[11] Benedikt et al., 2017, Theorem 1.

[12] Every ring in which for every $n \in \{1, 2, \ldots\}$ $\underbrace{1 + 1 + \ldots + 1}_{n} \neq 0$ (i.e. a ring of characteristic zero) has a subring isomorphic to $\mathbb{Z}$.

[13] Benedikt et al., 2017, Theorem 1 and Corollary 1.

[14] Ibid., Theorem 5.

[15] Ibid., Theorem 8.

Let us conclude with the following diagram of this chapter's polynomial simulations[16] .

$$\begin{tikzcd}
\text{free forest algebra} \arrow[r, leftrightarrow, "\text{Fact 2.33}"] & \text{free monoid} \arrow[r, "\text{Figure 2.11}"] \arrow[dl, "\text{Fact 2.35}"'] & \mathbb{Z} \arrow[dl] \\
\text{free unordered forest algebra} \arrow[r, "\text{Theorem 7}"] & \mathbb{Z}[x] &
\end{tikzcd}$$

FIGURE 2.14: Polynomial simulations between this chapter's algebras.

References. The essence of the proof of Theorem 4 comes from a paper of Seidl et al.[17] – the difference is that we do not assume the output ring to be the ring of integers. The proof of Theorem 6 can be found in a paper of Boiret, Piórkowski and S.[18].

[16] Most likely there are no polynomial simulations in the opposite directions, nevertheless we do not prove that.

[17] Seidl, Maneth, and Kemper, 2018.

[18] Boiret, Piórkowski, and Schmude, 2018.

Chapter 3

Transductions of graphs of bounded treewidth

In this chapter, we present an application of register transducers with output in the ring of polynomials to transductions of graphs of bounded treewidth. The treewidth of a graph is a positive integer parameter that describes its resemblance to a tree – roughly speaking, the lower the treewidth, the higher the resemblance. Diestel attributes the introduction of treewidth (under a different name) to Halin, 1976[1]. The notion of treewidth is well-known for its role in the proof of the famous Robertson and Seymour's Graph Minor Theorem, which states that the relation of being a minor is a well-quasi-ordering of finite graphs – the proof is performed first for trees, then for graphs of bounded treewidth, and finally for all graphs; for more details, we refer to a Diestel's textbook's chapter[2]. Also, it has numerous applications to graph algorithms, especially fixed-parameter tractable algorithms parameterized by the treewidth; for more details, we refer to a Cygan et al.'s textbook's chapter[3].

We consider the problems of functionality and equivalence of functional MSO_2 transductions of graphs of bounded treewidth. To the author's best knowledge, these problems are not known to be decidable or not, even in the simplest case of treewidth 1, i.e. when both the input and the output graphs are trees (unrooted trees, cf. Section 3.1). We do not decide these problems, but we give an algorithm for their (very) restricted variant in which the output graphs are considered up to a certain relaxation of isomorphism (Theorem 8).

3.1 Introduction: different kinds of trees

Let us begin with a discussion about MSO transductions of trees. Originally, MSO transductions input and output trees that are rooted, ordered, and unranked (see for example a survey of Courcelle[4]). In this section, we consider

[1] Diestel, 2017, Notes.

[2] Ibid.

[3] Cygan et al., 2015.

[4] Courcelle, 1994.

three kinds of trees that are not ranked: (rooted ordered) trees, (rooted) unordered trees, and unrooted (unordered) trees[5] (cf. Figure 3.1). We assume the input trees to be rooted and ordered. This case is the most general of the three, regardless of the chosen kind of the output trees, as there are surjective MSO transductions from trees through unordered trees to unrooted trees (which can be used for pre-composition, cf. Fact 3.9). Indeed, one can take MSO interpretations that merely forget structure, cf. Example 3.7. It seems that there are no surjective MSO transductions in the opposite directions, although we do not attempt to prove so. On the other hand, regarding the kind of the output trees, removing the structure *increases* the expressiveness of MSO transductions. It is because there are injective MSO transductions from trees through unordered trees to unrooted trees (which can be used for post-composition, cf. Fact 3.9).

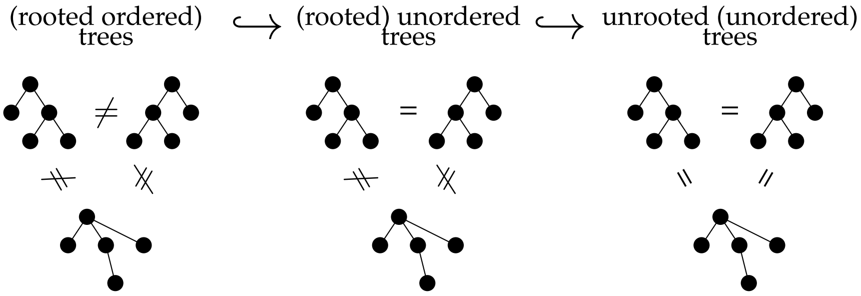


FIGURE 3.1: In MSO transductions that output trees, more structure in the output reduces to less structure in the output.

Indeed, the injective mapping from trees to unordered trees from the proof of Fact 2.35 is an MSO transduction, and unordered trees can be injectively mapped to unrooted trees by treating the root node as a vertex labeled by a fresh label. Again, it seems that there are no injective MSO transductions in the opposite direction, although again we do not attempt to prove so.

Returning to our main point – functionality and equivalence of functional MSO transductions – these problems are decidable for output trees being either trees or unordered trees (the latter model is at least as expressive as the former, as we showed in the previous paragraph). The case of trees was proved in 2005 by Engelfriet and Maneth[6]. The case of unordered trees is decidable due to Theorem 7: MSO transductions that output unordered trees are captured by register transducers with output in a free unordered forest algebra[7], and the latter model has decidable functionality and equivalence

[5] Unrooted tree is the same thing as undirected acyclic graphs.

[6] Engelfriet and Maneth, 2005; the proof is done for deterministic transductions; the general case follows from a rather standard reduction to the deterministic case, cf. the proof of Lemma 3.16.

[7] The proof of this fact is as follows: given an MSO transduction that outputs unordered trees, consider an enrichment of this transduction that induces some ordering of the siblings in the output tree (the ordering can be derived from the ordering of the input tree, cf. the construction from item 3 of the proof of Lemma 3.24). This is an MSO transduction that outputs trees, hence we can take a register transducer with output in a free (ordered) forest algebra that computes it (Alur and D'Antoni, 2012); this register transducer, with semantics taken in a free unordered forest algebra, computes the given MSO transduction.

of functional transductions (Theorem 7). For unrooted trees, the question of decidability of these problems is, to the author's best knowledge, open.

Finally, let us note that the class of unrooted trees coincides with the class of graphs of treewidth 1; it is the smallest output class of this chapter's model, MSO transductions of graphs of bounded treewidth.

3.2 Treewidth

Sourced graphs and treewidth. Treewidth can be defined in many ways (for example by tree decompositions[8]). For this notion to be suitable for register transducers, we define it algebraically, following Courcelle[9].

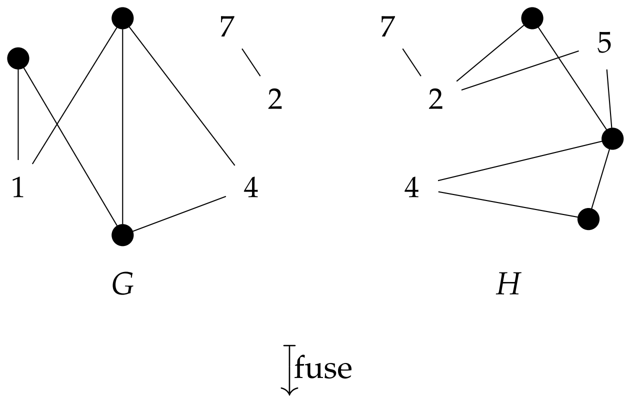
Definition 3.1 (k-sourced graphs)**.** A (finite, undirected) *graph* is a pair (V, E) where V is a finite set of *vertices* and E is a set of *edges* which is a subset of the set of unordered pairs of elements of V. A *k-sourced graph* is a graph with a tuple of at most k distinguished vertices, called *sources*. Formally, a k-sourced graph is a pair (G, source) where $G = (V, E)$ is a graph and $\text{source} : \{1, \ldots, k\} \to V$ is an injective partial function.

We define the following operations on k-sourced graphs:

- fuse – a partial binary operation that connects the graphs along the common sources (see Figure 3.2), provided their subgraphs induced by the common sources are identical,

- forget_i, for a source $i \in \{1, \ldots, k\}$ – a partial unary operation that turns the i-th source into a non-source vertex (see Figure 3.3), provided the i-th source is defined,

- rename_π, for a permutation π of the set of source names $\{1, \ldots, k\}$ – a unary operation that renames the i-th source to $\pi(i)$ for $i \in \{1, \ldots, k\}$ (see Figure 3.4).



[8] Numerous papers, for example Diestel, 2017.

[9] Courcelle, 1990.

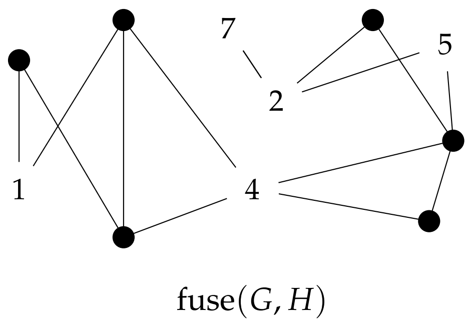


FIGURE 3.2: Fuse operation of 7-sourced graphs.

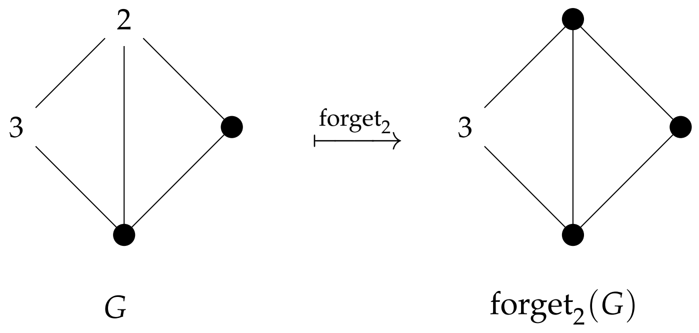


FIGURE 3.3: Forget operation of 7-sourced graphs.

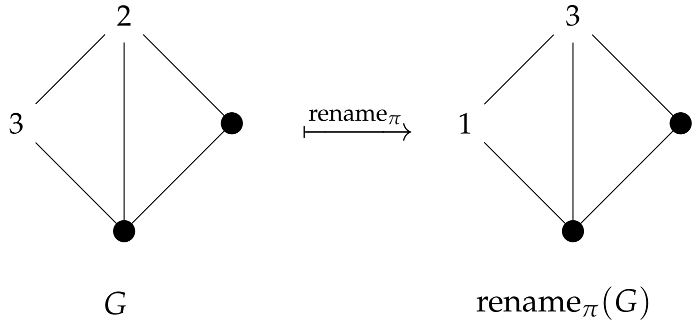


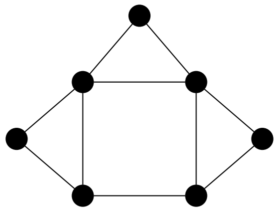
FIGURE 3.4: Rename by $\pi = \{1 \mapsto 2, 2 \mapsto 3, 3 \mapsto 1\}$.

Let us now define treewidth.

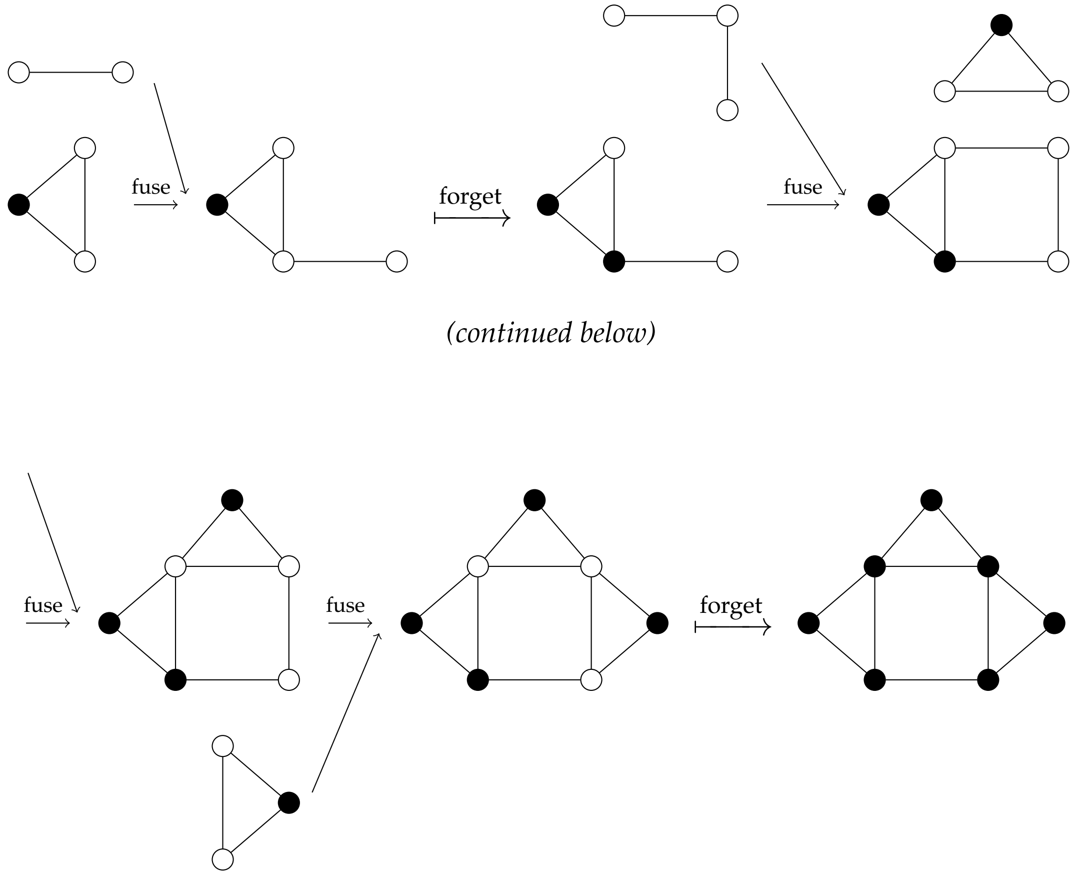
Definition 3.2 (Treewidth)**.** A graph has *treewidth* $\leq k$ if it can be obtained using constants and operations of $(k+1)$-sourced graphs fuse, forget$_i$ for $i \in \{1, \ldots, k\}$, and rename$_\pi$ for permutations π of $\{1, \ldots, k\}$. A graph has *treewidth k*, if it has treewidth $\leq k$ and does not have treewidth $\leq k-1$.

Example 3.3. Graphs of treewidth 1 are exactly the unrooted trees (undirected acyclic graphs). We omit the proof.

Example 3.4. The following graph has treewidth 2.



To verify this, we show how to obtain this graph using constants and operations of 3-sourced graphs. We start with a graph and fuse it with other graphs, forgetting some vertices along the way; all mentioned graphs have three vertices, hence clearly have treewidth ≤ 2. The source names are omitted from the notation.



(continued below)

FIGURE 3.5: Constructing a graph of treewidth 2 using 3-sourced graphs with three vertices and basic operations of the 3-sourced graph algebra.

The k-sourced graph algebra.

Definition 3.5. The *(multisorted) k-sourced graph algebra* is defined as follows:

- the sorts are graphs built on k-sources (with no non-source vertices; in particular, there are $2^{\binom{k}{2}}$ sorts),
- the universe consists of all k-sourced graphs that can be obtained using the constants and the operations of this algebra; the sort of a graph is the k-sourced subgraph induced by the set of sources,
- the operations are as follows:
  - there is a constant (0-ary operation) for every graph whose all vertices are sources; there are $2^{\binom{k}{2}}$ constants,

- for every pair of sorts (I, J) whose subgraphs induced by the common sources are identical, there is an operation fuse (we omit the sort names from the notation) of type $I \times J \to \mathrm{fuse}(I, J)$,
- for every source $i \in \{1, \ldots, k\}$, forget_i for every sort I that has an i-th source, there is an operation forget_i (we omit the sort names from the notation) of type $I \to I \setminus \{i\}$,
- for every permutation π of $\{1, \ldots, k\}$ for every sort I, there is an operation rename_π (we omit the sort names from the notation) of type $I \to \mathrm{rename}_\pi(I)$.

Sorts of the k-sourced graph algebra might seem not to be of any use (possibly except the fuse operation, in which they restrict the domain of the operation). However, equipping an algebra $\mathbb{A}$ with sorts is useful, because it allows to define more compositional functions from $\mathbb{A}$ (to some algebra $\mathbb{B}$), and in consequence, more simulations. This is because in such case the polynomial on the right-hand side of the formula (2.7) of the definition of a compositional function may additionally depend on the sorts of the elements on the left-hand side; we use this observation in the proof of Lemma 3.26. Sorts can be captured by register transducers in the sense of Remark 2.31.

3.3 Monadic second order logic (MSO)

MSO transductions of relational structures. A *relation* on a set D is a subset of D^k for some $k \in \{1, 2, \ldots\}$; the number k is called the *arity* of this relation. Let Σ be a ranked set. A *relational* Σ*-structure* is a pair $\mathbb{D} = (D, \Theta_\mathbb{D})$ where D is a set, called the *universe* of $\mathbb{D}$, and $\Theta_\mathbb{D}$ is a Σ-tuple of relations on D, where the arity of the σ-th relation is the arity of σ for $\sigma \in \Sigma$. For a relational Σ-structure, Σ is called its *signature*, elements of Σ are called *relational symbols*, and $\Theta_\mathbb{D}$ is called its *interpretation of the signature* or *interpretation of relational symbols*. Later on we call a ranked set a signature to designate its intended use. We assume all relational structure to be finite. We denote the set of all (finite) relational structures with signature Σ by $\mathrm{Rel}(\Sigma)$.

Let Σ be a signature. *Monadic second-order logic (*MSO*)* on signature Σ is an extension of first-order logic on the same signature obtained by adding variables that range over *sets* of elements, called *monadic (second-order) variables*, and by adding a binary symbol $\in$ that denotes set membership. We denote the first-order variables by lowercase letters like $x, y, z, \ldots$ and the monadic variables by uppercase letters like $X, Y, Z, \ldots$, and write $x \in X$ to say that an element x belongs to the set X – formally, first-order variables are 0-ary symbols, monadic variables are unary symbols, and the expression $x \in X$ is rewritten as $X(x)$. And so, $\forall x$ means "for every element x" and $\forall Y$ means "for every set of elements Y" (likewise for the existential quantifier): for example, if $\Sigma = \{a^{(1)}, b^{(1)}, c^{(1)}, \mathrm{parent}^{(2)}\}$ is the signature of $\{a, b, c\}$-labeled trees, then the MSO sentence

$$\phi := \forall x\, a(x) \to \exists Y\, \underbrace{\Big(x \in Y \land \forall y\, (y \in Y \land \neg b(y)) \to \exists z\, z \in Y \land \mathrm{parent}(z, y)\Big)}_{\text{the parent of a not } b\text{-labeled node from } Y \text{ is in } Y}$$

holds on those trees in which every a-labeled node has a b-labeled ancestor.

Definition 3.6 (MSO transduction)**.** Now we define MSO transductions following Bojańczyk and Pilipczuk Mi.[10]. There is an equivalent presentation by Courcelle, see the survey for more details[11]. An MSO *transduction* is a mapping or a relation on relational structures that are considered *up to isomorphism* that is a composition of a finite number of the following mappings or relations:

- *copying* from $\mathrm{Rel}(\Sigma)$ to $\mathrm{Rel}(\Sigma \cup \{\mathrm{layer}_1, \mathrm{layer}_2, \ldots, \mathrm{layer}_m, \mathrm{copy}\})$ for every signature Σ and $m \in \{1, 2, \ldots\}$; a function that returns the disjoint union of m copies of an input structure, called *layers*, enriched with unary predicates $\mathrm{layer}_1(x), \mathrm{layer}_2(x), \ldots, \mathrm{layer}_m(x)$ that state that the element x comes from respectively, first, second, ..., m-th copy of the universe of the input structure, and a binary predicate $\mathrm{copy}(x_1, x_2)$ that states that the elements x_1, x_2 (possibly from different layers) are copies of the same element of the input structure,

- *coloring* from $\mathrm{Rel}(\Sigma)$ to $\mathrm{Rel}(\Sigma \cup \{X_1, \ldots, X_k\})$ for every signature Σ and unary predicates $X_1, \ldots, X_k$ (called *colors*); a relation that relates an input structure with each of its enrichments by an interpretation of $X_1, \ldots, X_k$,

- *interpreting* from a subset of $\mathrm{Rel}(\Sigma)$ to $\mathrm{Rel}(\Gamma)$ for every signatures Σ and Γ, MSO sentence ϕ_{dom}, MSO formula in one free variable ϕ_{univ}, and a Γ-tuple of MSO formulas $(\psi_\gamma)_{\gamma \in \Gamma}$ where the number of free variables of ψ_γ is the arity of γ for $\gamma \in \Gamma$; a function that is the composition of the following three functions, in that order:

  1. *domain restriction* that restricts the domain to Σ-structures that satisfy ϕ_{dom},
  2. *universe restriction* that restricts the universe of an input structure to elements that satisfy ϕ_{univ},
  3. MSO *interpretation* that does not modify the universe of an input structure and interprets every symbol $\gamma \in \Gamma$ via the formula ψ_γ.

  If any of the above three functions is an identity, we omit it from the description of a given interpreting transduction.

An MSO transduction is *deterministic* if it can be obtained as a composition of copying and interpreting transductions (without coloring transductions), and is *inherently nondeterministic* otherwise. An MSO transduction is *functional* if it is a function; notice that deterministic transductions are functional.

[10] M. Bojańczyk and Pilipczuk, 2016.

[11] Courcelle, 1994.

Now we give two examples of MSO transductions.

Example 3.7. Our first example is a deterministic MSO transduction that is the “identity” function from ordered to unordered trees that merely forgets the order of the siblings (and does not change the parent relation). The input signature Σ consists of two binary symbols $FC^{(2)}, NS^{(2)}$, which are interpreted as the relations “is first child of” and “is next sibling of”, and the output signature Γ consists of one binary symbol $child^{(2)}$, which is interpreted as “is a child of” relation. This transduction is an interpreting transduction that consists only of the following MSO interpretation:

$$\psi_{child}(x_1, x_2) :=$$
$$\underbrace{\forall Y \Big[(\forall y_1\ FC(x_1, y_1) \to y_1 \in Y) \land (\forall y\ y \in Y\ \forall y'\ NS(y, y') \to y' \in Y) \Big] \to x_2 \in Y}_{\text{if a set contains the first child of } x_1 \text{ and is closed under siblings, then it contains } x_2}.$$

Example 3.8. Another example is a deterministic MSO transduction of words. A word, as a relational structure, is a monadic tree where a “node” is called a “position” and a “child” is called a “successor”. Consider a function that doubles every word over alphabet $\{a, b\}$, i.e. maps every $w \in \{a, b\}^*$ to ww. The input signature is $\Sigma = \{a^{(1)}, b^{(1)}, s^{(2)}\}$, where $a^{(1)}, b^{(1)}$ denote the label of a position and $s^{(2)}$ denotes the “is a successor of” relation, and the output signature is $\Gamma = \Sigma$. This transduction is the composition of the following transductions:

1. copying with two layers ($m = 2$),
2. interpreting that consists only of the following MSO interpretation:

$$\psi_a^{(i)}(x) := a(x), \psi_b^{(i)}(x) := b(x), i = 1, 2, \text{ and}$$
$$\begin{cases} \psi_s^{(i,i)}(x_1, x_2) = s(x_1, x_2), i = 1, 2, \\ \psi_s^{(2,1)}(x_1, x_2) = \bot, \\ \psi_s^{(1,2)}(x_1, x_2) = last(x_1) \land first(x_2), \end{cases}$$

where $last(x)$ is an abbreviation for $\neg \exists y\ s(x, y)$ and $first(x)$ is an abbreviation for $\neg \exists y\ s(y, x)$.

In each of the two above examples the MSO transductions are deterministic. However, in Example 3.11 and Example 3.13 we give functional MSO transductions that are not deterministic.

Following Bojańczyk and Pilipczuk Mi.[12], we state two fundamental properties of MSO transductions.

Fact 3.9. MSO *transductions are closed under composition.*

Fact 3.10 (Canonical form of an MSO transduction)**.** *Every* MSO *transduction is a composition of* MSO *transductions of the following kinds, in that order: at most one coloring transduction, at most one copying transduction, and at most one interpreting transduction.*

[12] M. Bojańczyk and Pilipczuk, 2016.

The proof of Fact 3.9 follows straightforwardly from our definition of an MSO transduction. The proof of Fact 3.10 is rather straightforward yet technical. (For a complete proof, one may combine the following results: Fact 3.9 was proved by Courcelle for his model of an MSO transduction[13], from which one can infer equivalence of his and our model, and this equivalence immediately implies Fact 3.10.)

MSO transductions of graphs. To define MSO transductions of graphs, one must see them as relational structures. One way of doing so is to take the set of the vertices as the universe and consider the *incidence* relation. The corresponding MSO logic is called MSO_1[14]. The second way is to take the union of the set of vertices and the set of edges as the universe and consider the *incidence* relation. The corresponding MSO logic is called MSO_2[15]. In this thesis we are interested in MSO transductions of graphs seen as relational structures in the second way, both in the input and the output – they are called MSO_2 transductions. Formally, an MSO_2 *transduction* is a relation on relational structures over one binary symbol – up to isomorphism – that are graphs when this symbol is interpreted as the incidence relation (i.e. the universe can be partitioned into sets V and E such that the relation is a subset of $V \times E$ in which every element of E is related to exactly two elements of V).

Analogously as in the case of register transducers, we are mainly interested in MSO transductions that are functional. Let us consider two following examples of functional MSO transductions.

Example 3.11 (Cycle-to-path)**.** The mapping of graphs that maps a cycle to a path with the same number of vertices (and is undefined for input graphs that are not cycles) is an MSO_2 transduction.

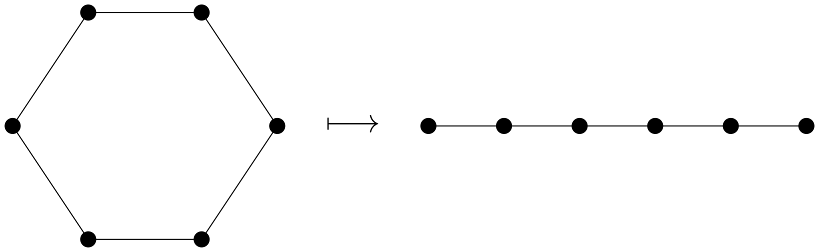


FIGURE 3.6: Mapping a cycle to a path with the same number of vertices.

It can be defined as a composition of the following: domain restriction to graphs that are cycles, coloring with one color (i.e. $k = 1$), domain restriction to graphs in which exactly one edge is colored, and universe restriction to non-colored edges (the vertices remain unchanged). Intuitively speaking, this transduction checks if the input graph is a cycle, and if it is, guesses an edge and removes it.

[13] Courcelle, 1994, Proposition 3.2(2).

[14] Ibid.

[15] Ibid.

Remark 3.12. Notice that, in the above transduction, if the input graph is a cycle of length ≥ 2, then there are multiple choices of an edge to be removed, yet they all yield the same output graph *up to isomorphism*. In other words, the above transduction is *functional*.

On the other hand, this transduction is *inherently nondeterministic*. Indeed, by contradiction assume it could be defined without coloring. There clearly exists an MSO$_2$ formula that selects the set of endpoints of a given path (they have exactly one neighbor) and therefore, by pre-image, there would exist an MSO$_2$ formula that could *select* some set of at most two vertices in a given cycle. But this is impossible, as for cycles of length ≥ 3 every set of at most two vertices can be mapped to a different set by some automorphism of a cycle. This finishes the proof's sketch.

Example 3.13 (Subdivide every edge)**.** The mapping of graphs of bounded treewidth that subdivides every edge is an MSO$_2$ transduction.

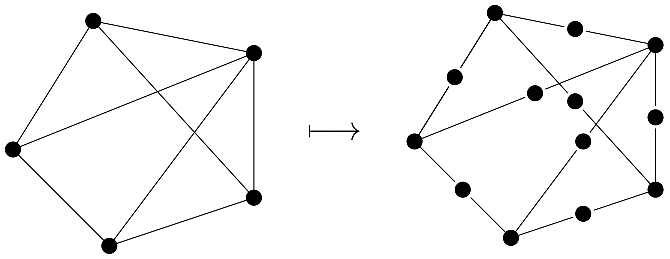


FIGURE 3.7: Subdivision of every edge.

Subdividing requires, for a graph of treewidth $\leq k$, guessing nondeterministically a $(k+1)$-coloring of this graph[16]– this way, every edge has the "first" and the "second" vertex, which are defined by comparing their, necessarily different, colors. Then, every edge E is mapped to three copies: two edges $E^{(1)}, E^{(2)}$ and a vertex $E^{(3)}$, where $E^{(1)}$ is set incident with the "first" vertex of E and $E^{(3)}$, and $E^{(2)}$ is set incident with the "second" vertex of E and $E^{(3)}$.

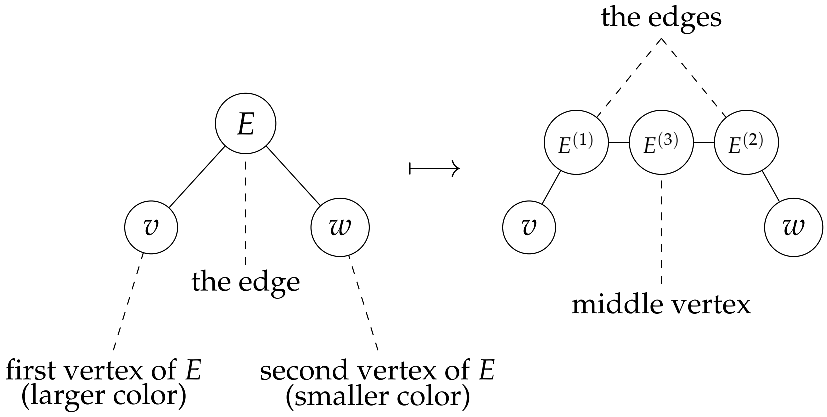


FIGURE 3.8: Subdividing an edge using MSO$_2$ and a finite coloring of a graph; the edges of this picture represent incidence.

[16] Every graph of treewidth $\leq k$ admits a $(k+1)$-coloring, due to the following recursive algorithm: a coloring is constructed successively, starting from the root of the term of the $(k+1)$-sourced graph algebra that defines the graph, and going down, either coloring the forgotten sources or, by recursion, coloring the subgraphs begin fused; we omit the proof's details.

Notice that guessing a $(k+1)$-coloring of a graph can not be removed from the above reasoning, as when setting the incidences, we use MSO_2-definability of the notion of the first vertex of a given edge E.

As we saw in Example 3.11, for graphs represented as relational structures with the incidence relation, functional MSO transductions are strictly more expressive that deterministic ones. We finish this section with a remark that for some classes of relational structures this is not the case.

Fact 3.14. *(Engelfriet and Hoogeboom, 2001, Theorem 21). Every functional* MSO *transduction that inputs words with successor relation is deterministic.*

The idea of the above fact's proof is to represent a given MSO transduction in a normal form as in Fact 3.10, and use an MSO formula that selects a unique coloring of positions that passes the domain restriction (and hence for which an output is returned) – then the colors can be replaced by MSO formulas; there exists such a formula, for example one that describes the lexicographically smallest coloring that passes the domain restriction.

3.4 Equivalence problem

The central problems of this chapter are functionality and equivalence of functional MSO_2 transductions. We consider the restricted variants of the problem in which both the input and the output graphs have some fixed bounds on the treewidth. We do it, because if the input graphs have unbounded treewidth, then both problems are immediately seen to be undecidable – in fact, in such case even non-emptiness of the domain is undecidable[17]. In consequence, we are interested in the two following problems.

Name: Functionality of MSO_2 transductions of graphs of bounded treewidth
Input: $k, \ell \in \{1, 2, \ldots\}$,
f – an MSO_2 transduction such that all input graphs have treewidth $\leq k$ and all output graphs have treewidth $\leq \ell$
Question: Is it the case that for every graph G of treewidth $\leq k$ there is at most one graph H (up to isomorphism) such that $(G, H) \in f$?

Name: Equivalence of functional MSO_2 transductions of graphs of bounded treewidth
Input: $k, \ell \in \{1, 2, \ldots\}$,
f_1, f_2 – functional MSO_2 transductions such that all input graphs have treewidth $\leq k$ and all output graphs have treewidth $\leq \ell$
Question: Is it the case that

$$f_1(G) = f_2(G)$$

for every graph G of treewidth $\leq k$?

[17] Seese, 1991.

Notice that for every MSO_2 transduction one can restrict its domain (codomain) to graphs with a given bound on the treewidth; this can be done by pre-(post-)composing with an appropriate domain restriction transduction[18].

Let us prove a couple inter-reductions between the above problems, and some their variants. A *labeled variant* of the problem is when the *vertices* of the graphs (both input and output) are labeled by letters from a finite alphabet.

Fact 3.15. *There are polynomial-time reduction between the decision problem of functionality of* MSO_2 *transductions of graphs of bounded treewidth and its labeled variant. Likewise for the problem of equivalence of functional transductions.*

Proof. It is enough to show a reduction from the labeled to the unlabeled variant. For both problems, this reduction follows from the following encoding of labeled graphs into unlabeled graphs.

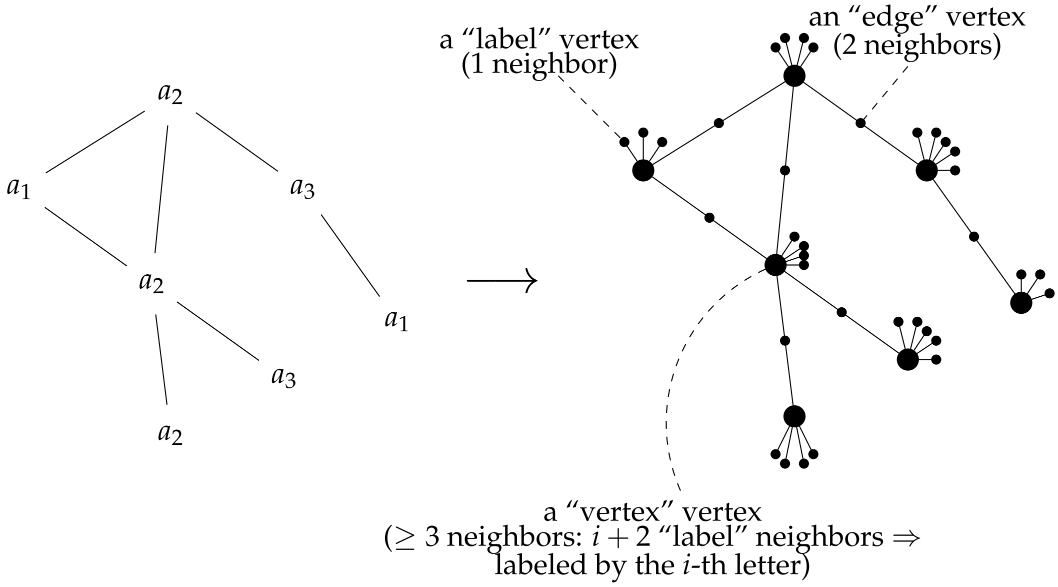


FIGURE 3.9: Encoding labeled graphs into unlabeled graphs.

The above encoding can be defined by an MSO_2 transduction that, intuitively speaking, maps every vertex labeled by the i-th symbol of the alphabet to a vertex connected with $i+2$ vertices of degree one, for $i \in \{1,\ldots,|\Sigma|\}$, and subdivides every edge, as described in Example 3.13. Also, observe that the inverse mapping can be defined by an MSO_2 transduction too; we omit the details.

Reductions in both directions follow by pre-composing given transduction(s) with the inverse of the above encoding and post-composing it (them) with the above encoding, which results in an MSO_2 transduction (cf. Fact 3.9). □

[18] Given a natural number k, one can obtain an MSO_2 formula that is satisfied exactly by graphs of treewidth $\leq k$. We give a proof sketch, using the Backwards Translation Theorem (Courcelle and Engelfriet, 2012, Theorem 1.40, see also M. Bojańczyk and Pilipczuk, 2016, Section 2.1) that allows to identify an MSO formula with an MSO transduction that returns either 0 or 1: such an MSO_2 formula (transduction that returns either 0 or 1) can be obtained by composing an MSO_2 transduction that returns a tree decomposition of optimal width (M. Bojańczyk and Pilipczuk, 2017) and a MSO_2 formula (transduction that returns either 0 or 1) that tests if a given decomposition has the desired width.

Lemma 3.16. *There are polynomial-time reductions between the decision problem of functionality and equivalence of functional* MSO_2 *transductions.*

Proof. For each of the above problems we prove it is reducible to the labeled variant of the other problem. This will be sufficient as each of those problems is inter-reducible with its labeled variant (Fact 3.15).

For one direction, let f_1 and f_2 be functional MSO_2 transductions. If their domains are not equal, then they are not equivalent (this property can be decided[19]). Otherwise, one can construct the transduction $f_1 \cup f_2$, i.e. a relation in which the image of a graph is the union of the images of this graph by f_1 and f_2. This is done, roughly speaking, by nondeterministically guessing which of f_1 and f_2 to apply. This can be achieved e.g. by adding a label from $\Gamma = \{1, 2\}$ to every vertex of the input graphs and applying f_i if and only if all the vertices are labeled by i, for $i = 1, 2$. Then f_1 and f_2 are equivalent if and only if the transduction $f_1 \cup f_2$ is functional.

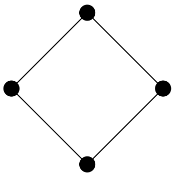
For the converse, let T be an MSO_2 transduction given in a canonical form as in 3.10 with the set of colors $\mathcal{X}$. Let $\mathcal{X}_1, \mathcal{X}_2$ be two copies of $\mathcal{X}$. We construct deterministic MSO_2 transductions f_1, f_2 that input graphs whose vertices are labeled by $\mathcal{P}(\mathcal{X}_1) \times \mathcal{P}(\mathcal{X}_2)$, and for every input graph, f_i applies the transduction T (deterministically) with the interpretation of $\mathcal{X}$ determined by the labels from the i-th coordinate, for $i = 1, 2$. In this way, for every graph G and every pair of interpretations of $\mathcal{X}$ in G there exists an input graph for f_1 and f_2 such that its images by f_1, f_2 are the images of G by T with those interpretations of $\mathcal{X}$. In consequence, T is functional if and only if f_1 and f_2 are equivalent. □

Remark 3.17. By an analogous proof, there exist inter-reductions for the variants where both the input and output graphs have fixed bounds on the treewidth and the relation used for comparing the output graphs, which originally is isomorphism, is changed to a relation that we call walk-equivalence (Definition 3.20), as in Theorem 8.

Let us now proceed to the main result of this chapter, Theorem 8, which states that functionality and equivalence of functional MSO_2 transductions of graphs of bounded treewidth are decidable in the restricted variant when the output graphs are considered up to walk-equivalence (Definition 3.20)

Definition 3.18. A *walk* in a graph G is a sequence of vertices of G interleaved with edges that connect them; the *length* of a walk is the number of edges it contains (counting multiplicities).

Example 3.19. The following graph has 16 $(= 4 \cdot 2^2)$ length-2 walks: there are 4 vertices to start a walk in, and there are 2 choices for the next vertex in each of the 2 steps of a walk.



[19] Courcelle, 1989.

FIGURE 3.10: A graph with 16 length-2 walks.

Definition 3.20 (Walk-equivalence)**.** We call two graphs *walk-equivalent* if they have the same number of length-n walks for every $n \in \{0, 1, \ldots\}$.

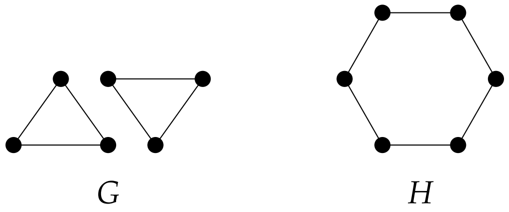


FIGURE 3.11: Two walk-equivalent non-isomorphic graphs.

We proceed to Fact 3.22, which describes how walk-equivalence is related to the isomorphism relation.

Definition 3.21. The *adjacency matrix* of a graph $G = (V, E)$, denoted A_G, is the $V \times V$ matrix defined by

$$(A_G)_{v,w} := \begin{cases} 1 & \text{if there is an edge between } v \text{ and } w, \\ 0 & \text{otherwise.} \end{cases}$$

Let G, H be graphs. Consider the following property of a pair of graphs G and H:

$$\begin{cases} XA_G = A_G X, \text{ and} \\ \text{the entries of every row of } X \text{ sum up to } 1, \text{ and} \\ \text{the entries of every column of } X \text{ sum up to } 1, \end{cases} \tag{3.1}$$

for some real-entry matrix X (notice that X might have negative entries).

Dell, Grohe, and Rattan noted that if we additionally require X to have *non-negative* entries, then the relation described by (3.1) is the *fractional isomorphism*, and if we additionally require X to have *natural* entries, then (3.1) describes the *isomorphism* relation[20]. They also characterized walk-equivalence in the following way.

Fact 3.22 (Dell, Grohe, and Rattan, 2018)**.** *Two graphs are walk-equivalent if and only if the formula* (3.1) *holds for some real-entry matrix* X[21].

[20] Dell, Grohe, and Rattan, 2018.

[21] Observe that a walk in a graph G can be equivalently seen as a homomorphism from a path (with a distinguished endpoint) to G. Every class of graphs $\mathcal{H}$ induces the following equivalence relation: two graphs are related if they have the same number of homomorphisms from H for every H from $\mathcal{H}$. Fact 3.22 states that by taking as $\mathcal{H}$ the class of all paths one obtains an equivalence relation similar to fractional isomorphism but in which negative coefficients are allowed. Let us also mention the classical result of Lovász that states that by taking as $\mathcal{H}$ the class of all graphs one obtains the isomorphism relation. The distinguishing power of relations obtained this way, which obviously varies for different choices of the class $\mathcal{H}$, has been further studied (see e.g. Dvořák, 2010): for example, each of the classes of 2-degenerate graphs and of non-bipartite graphs suffices to give the isomorphism relation, and each of the classes of graphs of treewidth $\leq k$ for $k \in \{1, 2, \ldots\}$ gives a weaker relation, which admits several descriptions, i.a. in terms of k-dimensional Weisfeiler-Lehman (color refinement) algorithm and of k-variable first-order logic with counting.

Remark 3.23. Let us remark that we can reformulate the property of a pair of graphs G and H given by the formula (3.1) as follows: there is a mapping X of vertices of G to (formal) linear combinations of vertices of H that, when treated as a linear mapping, has the following properties:

- every vertex of G is mapped to one vertex of H in total
- every vertex of H is an image of one vertex of G in total
- the neighborhood of every vertex of G (seen as a formal sum of its neighbors) is mapped to the neighborhood of its image in total ($XA_G = A_GX$).

Theorem 8. *The following problems, which can be reduced to each other in polynomial time (Lemma 3.16), are decidable:*
Name: *Walk-functionality of* MSO$_2$ *transductions of graphs of bounded treewidth*
Input: $k, \ell \in \{1, 2, \ldots\}$,
f *– an* MSO$_2$ *transduction such that all input graphs have treewidth* $\leq k$ *and all output graphs have treewidth* $\leq \ell$
Question: *Is it the case that for every graph* G *of treewidth* $\leq k$ *there is* *at most one* *graph* H up to walk-equivalence *such that* $(G, H) \in f$*?*

and the problem of walk-equivalence of walk-functional MSO$_2$ transductions of graphs of bounded treewidth*, defined analogously, i.e. that inputs positive integers* $k, \ell \in \{1, 2, \ldots\}$ *and walk-functional* MSO$_2$ *transductions* f_1, f_2 *whose input graphs have treewidth* $\leq k$ *and output graphs have treewidth* $\leq l$*, and asks if they output walk-equivalent graphs for every input graph.*

The proof of Theorem 8 proceeds as follows:

(1) we show how register transducers with output in the k-sourced graph algebra can express MSO$_2$ transductions of graphs of bounded treewidth (Section 3.5),

(2) we find a "relaxed" polynomial simulation of that algebra with a ring of polynomials that is injective if the graphs are considered up to walk-equivalence (Section 3.6).

3.5 From MSO$_2$ transductions to register transducers

In this section we introduce a register transducer model that can express MSO$_2$ transductions of graphs of bounded treewidth. Since register transducers input trees, not graphs, the input graphs are encoded as terms over a k-sourced graph algebra (which are finitely labeled graphs for a fixed k). Also, the output graphs are computed using a k-sourced graph algebra.

Lemma 3.24. *Let* $k, \ell \in \{1, 2, \ldots\}$ *and* f *be an* MSO$_2$ *transduction (not necessarily functional) that inputs graphs of treewidth* $\leq k$ *and outputs graphs of treewidth* $\leq \ell$*.*

$$\begin{tikzcd} \begin{array}{c}\textit{graphs of}\\ \textit{treewidth} \le k\end{array} \arrow[r, "f"] & \begin{array}{c}\textit{graphs of}\\ \textit{treewidth} \le \ell\end{array} \end{tikzcd}$$

Then there exists a register transducer T with output in the $(\ell+1)$-sourced graph algebra such that the following diagram is commutative:

$$\begin{tikzcd} \begin{array}{c}\textit{terms of the}\\ \textit{(k+1)-sourced graph algebra}\end{array} \arrow[d, "eval"'] \arrow[dr, "T"] & \\ \begin{array}{c}\textit{graphs of}\\ \textit{treewidth} \le k\end{array} \arrow[r, "f"] & \begin{array}{c}\textit{graphs of}\\ \textit{treewidth} \le \ell\end{array} \end{tikzcd}$$

Proof of Lemma 3.24. We combine the following facts. For each of them, we either provide a reference or give a proof. Recall that an *ordered graph* is a graph with a binary relation that is an ordering of the vertices.

(1) Every MSO transduction of (rooted, ordered) trees can be defined by a register transducer with output in the free forest algebra (Definition 2.30)[22].

(2) One can construct a deterministic MSO transduction that maps a term over the k-sourced graph algebra to the graph it represents enriched with some ordering of vertices; we denote this transduction as $eval_{ord}$.

(3) Every MSO_2 transduction (of graphs) can be extended to an MSO_2 transduction of ordered graphs; more precisely, for every MSO_2 transduction (of graphs) T there is an MSO_2 transduction of ordered graphs T_{ord} such that if T maps a graph (V, E) to a graph (V', E'), then T_{ord} maps every ordered graph of form $(V, E, <)$ for some ordering $<$ of V to an ordered graph of form $(V', E', <')$ for some ordering $<'$ of V'.

(4) One can construct a non-functional MSO_2 transduction that maps an ordered graph of treewidth $\le k$ to a term over a $(k+1)$-sourced graph algebra that represents it, for $k \in \{1, 2, \ldots\}$.

(5) The composition of two MSO transductions (of relational structures over arbitrary signatures) is an MSO transduction (Fact 3.9).

Proof of item (3). The ordering of the output vertices can be inherited from the ordering of the input vertices, in the following way: for two vertices of the output graph, first order them by the vertices of the input graph they come from, and if these vertices are the same, order them by the layers they belong to. □

Proof of item (2). Evaluating a term to a graph can be done by an MSO_2 transduction in a straightforward way. By proceeding the same as in the proof of item (3), the ordering of the output vertices can be inherited from any MSO-definable ordering of the nodes of the input tree – for example, one can pick the depth-first search ordering. □

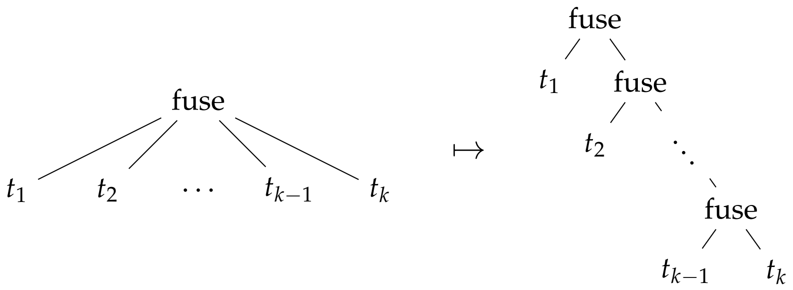
[22] Alur and D'Antoni, 2012, Theorem 1.

Proof of item (4). First, by a result of Bojańczyk and Pilipczuk Mi., an analogous claim holds for (unordered) graphs and "unranked unordered terms"[23]. We describe their transduction in more detail, and use it to construct the desired transduction.

By an *unranked unordered term* over the k-sourced graph algebra we mean an unranked unordered tree labeled by the names of the basic operations of this algebra in which only the fuse symbol does not respect its original arity, i.e. it can have arbitrarily many children – it is not a major change as the fuse operation is associative; despite the lack of order on the siblings in this term, its semantics is well-defined, as the non-unary nodes are labeled by a commutative basic operation (fuse).

Technically, the cited result uses a representation of graphs of bounded treewidth, called a *tree decomposition*, that differs from the unordered unranked term. Nevertheless, it is straightforward (e.g. from the proof of Bojańczyk[24]) that there are MSO transductions that map each of these representations to the other. In consequence of this, and of closure of MSO transductions under composition, both representations can be used interchangeably.

Now we construct the desired MSO$_2$ transduction. By item (3), we turn the mentioned "graph-to-(unranked unordered term)" MSO$_2$ transduction into "(ordered graph)-to-(unranked term)" MSO$_2$ transduction. Then, we compose it with the transduction that turns the fuse-labeled nodes into binary nodes without changing the decomposition defined below.



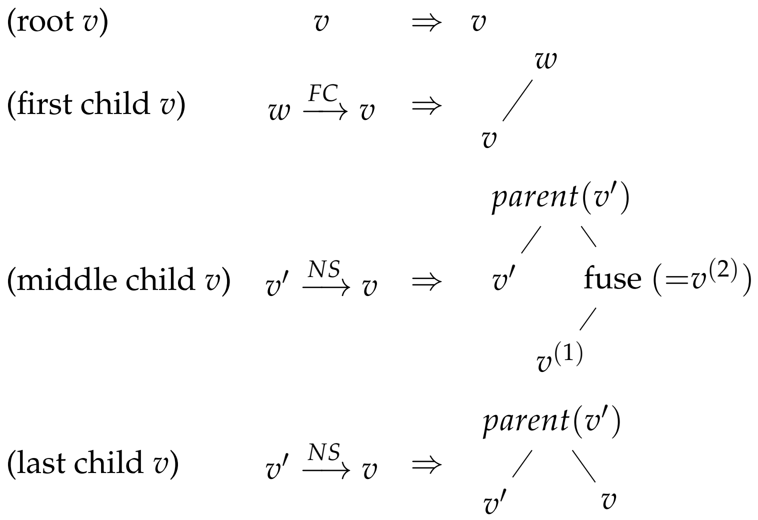
Turning the fuse-labeled nodes into binary nodes without changing the decomposition; $t_1, \ldots, t_k$ are trees.

To finish the proof, we prove that the above is an MSO transduction; it is the composition of the following. Recall that the input signature consists of unary symbols fuse, forget$_i$ for $i \in \{1, \ldots, k+1\}$, and rename$_\pi$ for a permutation $\pi : \{1, \ldots, k\} \to \{1, \ldots, k\}$, that are labels of vertices, and two binary symbols FC, NS, which are interpreted as the relations "is first child of" and "is next sibling of", and the output signature consists of the same set of unary symbols, and two binary symbols $child_1, child_2$, which are interpreted as the relations "is first child of" and "is second child of". The first transduction colors the nodes using labels $root, first\_child, middle\_child$, and $last\_child$ so that every node is labeled by its property (a last child is assumed

[23] M. Bojańczyk and Pilipczuk, 2017, Corollary 3.

[24] M. Bojańczyk, 2015.

not to be a first child), i.e. the root is labeled by the label *root* and so on – formally, this is done by first coloring and then restricting the domain. The next transductions, and the last ones at the same time, are copying and interpreting transductions described in the below figure; only some nodes are copied (precisely, the middle children), which formally is achieved by first copying all vertices and then restricting the universe.



Copying the vertices, labeling the new copies, and interpreting relations $child_1, child_2$; for a middle child v, the vertex $v^{(i)}$ is its i-th copy, for $i = 1, 2$; for the remaining nodes, we identify the first (and only) copy of v with v itself.

□

Let us consider the MSO transduction that is the composition of the MSO transductions from items (2), (3), and (4), and denote by T'' the register transducer with output in the free forest algebra that computes it.

$$\begin{tikzcd}
\text{terms of the } (k+1)\text{-sourced graph algebra} \arrow[r, dashed, "T''"] \arrow[d, "eval_{ord}"'] & \text{terms of the } (\ell+1)\text{-sourced graph algebra} \\
\text{ordered graphs of treewidth } \leq k \arrow[r, "f"] & \text{ordered graphs of treewidth} \leq \ell \arrow[u]
\end{tikzcd}$$

We would like to compose the register transducer T'' with the function that evaluates terms of the $(\ell + 1)$-sourced graph algebra to graphs of treewidth $\leq \ell$. There are two issues that we encounter when trying to do so.

The first issue is that T'' might store a forest in some of its registers, which in this case is a *part* of a term and hence cannot be directly evaluated in the $(\ell + 1)$-sourced graph algebra. Nevertheless, such part can only be a list of trees to which the fuse operation has not been applied yet. In consequence, T'' can be easily transformed to an equivalent transducer that applies the fuse operation in advance to any such list, and hence always stores a term in each of its registers.

After solving the first issue, the second issue is that, in its registers, T'' might store not only terms (trees), which evaluate to $(\ell+1)$-sourced graphs, but also *contexts*, which evaluate to contexts in the $(\ell+1)$-sourced graph algebra, defined below.

A *context (with one hole) in a k-sourced graph algebra* is a multisorted polynomial operation in the k-sourced graph algebra defined by a polynomial with one occurrence of variable '?'; since the operations we consider are multisorted, every context has a fixed sort of the inputs and of the outputs, which we call the *input sort* and the *output sort*; we define the *sort of a context* to be the pair: (input sort, output sort). By adding contexts one obtains the *multisorted algebra of k-sourced graphs and contexts* in which, like in the case of forest-contexts (cf. Definition 2.30), there is an additional basic operation of substitution for symbol ?. There is a related issue that the symbol ? (and in consequence, every context) can come with various sorts assigned to it; however, a register transducer can, by the use of nondeterminism, guess the sort of each occurrence of the symbol ? in advance, and later verify the correctness of that guess at the moment of substitution for that occurrence (notice that it is possible due to the fact that the contexts have at most one occurence of ?).

And so, by evaluating the terms stored in the registers of T'' we obtain a register transducer T' with output in the algebra of $(\ell+1)$-sourced graphs *and contexts*.

$$T' : \begin{array}{c}\text{terms of the}\\ (k+1)\text{-sourced graph algebra}\end{array} \to \begin{array}{c}\text{graphs of}\\ \text{treewidth} \le \ell\end{array} .$$

One can remove contexts from T' using the following claim. This finishes the proof.

Claim. *The algebra of ℓ-sourced graphs and contexts can be polynomially simulated by an ℓ'-sourced graph algebra (without contexts) for some $\ell' \in \{1,2,\ldots\}$.*

In the proof of the above claim we use the following remark.

Remark 3.25. Let $\mathbb{A}$ and $\mathbb{B}$ be multisorted algebras. Then in a polynomial simulation of $\mathbb{A}$ by $\mathbb{B}$, a basic operation θ of $\mathbb{A}$ may be simulated by a collection of polynomial operations of $\mathbb{B}$, one for each combination of the sorts of the input elements.

Proof of Claim. We pick $\ell' = 3\ell$. For the simplicity of notation we denote the sources by $1,\ldots,\ell,\widehat{1},\ldots,\widehat{\ell},\widehat{\widehat{1}},\ldots,\widehat{\widehat{\ell}}$. Intuitively speaking, we use sources $1,\ldots,\ell$ to simulate the sources of the graph (including the ones that are to appear after substitution for the ? symbol), sources $\widehat{1},\ldots,\widehat{\ell}$ to simulate the forgotten sources of the graph that will be substituted for ? symbol, and sources $\widehat{\widehat{1}},\ldots,\widehat{\widehat{\ell}}$ temporarily when simulating the basic operation of substitution.

The polynomial simulation is defined inductively on the term structure as follows. A sourced graph is mapped to itself, and the symbol ? of input and output sort I is mapped to the sourced graph I. Each of the basic operations fuse and rename is simulated by itself (the remaining one-dashed and two-dashed sources are not renamed). Now we move to the basic operations of forget and substitution.

Every time the source that originates from (the graph to be substituted for) ? is forgotten, it is renamed in the image to its dashed counterpart (i.e. i-th source is renamed to $\widehat{i}$, where $i \in \{1, \ldots, \ell\}$); the remaining sources are simply forgotten. In consequence, a dashed source $\widehat{i}$ of the image of a context represents the forgotten i-th source of the graph to be substituted for ? and is kept in the image until the moment of substitution.

When the substitution $G[? := H]$ is performed (recall H can be a graph or a context), the image of the substituted graph (or context) H is "fused" with the image of G in the following way: the sources of the image of H that are dashed are not being fused, and those that are not dashed are fused, in the image of G, as follows: the i-th source of H is fused with the $\widehat{i}$-th source of G if it exists, and with the i-th source otherwise for $i \in \{1, \ldots, l\}$, and, additionally, in the result of the fusion, the fused dashed sources are forgotten (notice the set of sources to be forgotten can be inferred from the sorts of the *images* of G and H, which is admissible, cf. Remark 3.25). The operation we just described cannot be performed directly (we cannot fuse sources with different names), however its result can be achieved using renaming: it can be done by applying, in the image of H, a renaming that adds one dash to every zero-dashed source that does not occur in G and to every one-dashed source – denote the resulting sourced graph by $\widehat{H}$ – then fusing $\widehat{H}$ and the image of G, then forgetting the one-dashed sources, and finally applying a renaming that removes one dash from every two-dashed source (Figure 3.12).

Finally, notice that when simulating both fuse and the substitution operation, the fused graphs have the same subgraphs induced by the common sources, and so their sorts are as required.

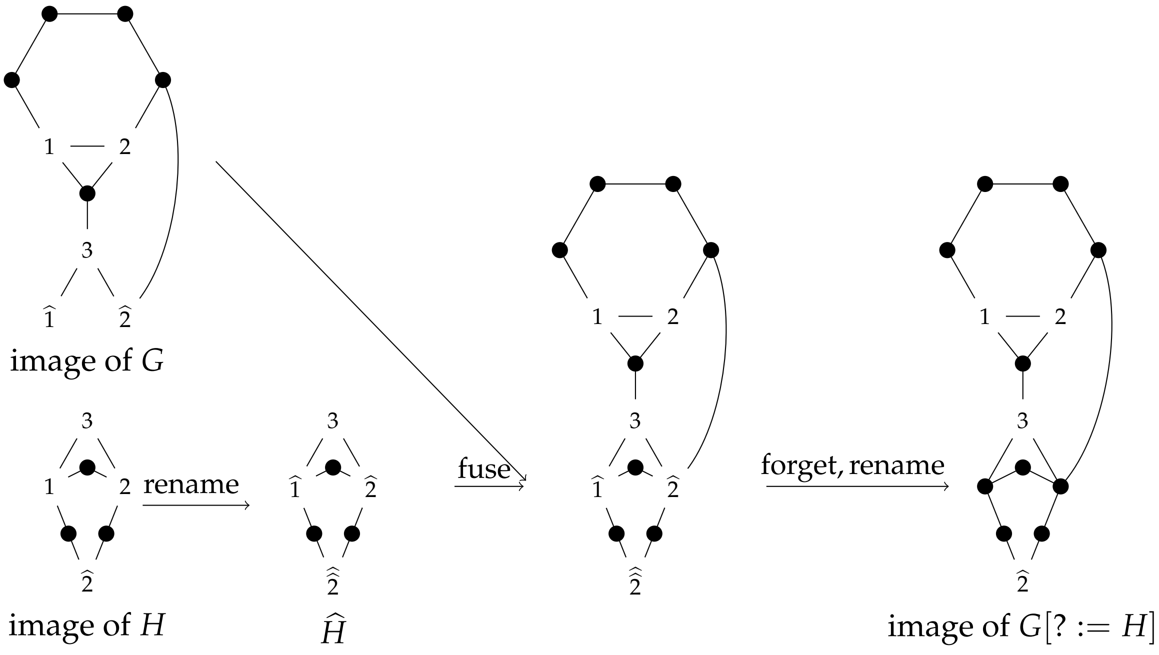


FIGURE 3.12: Constructing the image of the substitution of a context (H) into a context (G) using the basic operations of the $3l$-sourced graph algebra.

□

□

3.6 Rational power series

In this section we prove the following lemma.

Lemma 3.26. *Let $k \in \{1, 2, \ldots\}$. The following problem can be reduced to zeroness of register transducers with output in the ring $\mathbb{Z}[x]$: walk-equivalence of walk-functional register transducers with output in the k-sourced graph algebra (defined in Theorem 8 for* MSO_2 *transductions) .*

Now we proceed to the definition of the walk series, which is a power series in which we will keep the numbers of length-n walks of a graph for $n \in \{1, 2, \ldots\}$.

Definition 3.27. A *power series* in variable x is an "infinite polynomial", i.e. an expression of form $\sum_{n=0}^{\infty} a_n x^n$ where $(a_n)_{n \in \mathbb{N}}$ is a sequence of coefficients. Let us emphasize that it might happen that all the coefficients are non-zero, e.g. there is a power series $\sum_{n=0}^{\infty} n^2 x^n$. The operations of addition and multiplication are defined for power series in the same way as for polynomials. For more information on power series we refer to a textbook of Salomaa and Soittola[25].

By $\mathbb{Z}[[x]]$ we denote the *ring of power series* in variable x with integer coefficients. A power series is called *quasiregular* if it is of form $\sum_{n=1}^{\infty} a_n x^n$, i.e. its coefficient standing by x^0 is 0; clearly the set of quasiregular power series is equal to $x\mathbb{Z}[[x]]$ (as $\sum_{n=1}^{\infty} a_n x^n = x \sum_{n=0}^{\infty} a_{n+1} x^n$). A partial operation of *Kleene plus*, denoted $^+$, is defined for quasiregular power series by[26]

$$\begin{aligned} ^+ : x\mathbb{Z}[[x]] &\to x\mathbb{Z}[[x]], \\ f^+ &:= f + f \cdot f + f \cdot f \cdot f + \ldots. \end{aligned}$$

A power series is *rational* if it either:

- is a polynomial,

- is a sum of two rational power series,

- is a product of two rational power series,

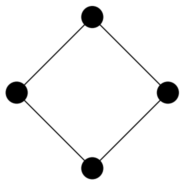
- is a Kleene plus of a quasiregular rational power series.

[25] Salomaa and Soittola, 1978.

[26] Notice that the formula that defines Kleene plus has no meaning for some non-quasiregular power series, for example 2^+ would give $2 + 2^2 + \ldots$ which does not converge.

The set of rational power series is denoted by $\mathbb{Z}^{\text{rat}}[[x]]$. In the proof of Lemma 3.26 we will use the algebra $(x\mathbb{Z}^{\text{rat}}[[x]], +, \cdot, ^+)$, to which we refer shortly as "the algebra $x\mathbb{Z}^{\text{rat}}[[x]]$".

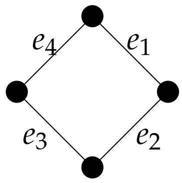
Definition 3.28 (Walk series)**.** Let W be a subset of the set of walks of positive length in a graph G. We define the *walk series* of W as $\sum_{n=1}^{\infty} a_n x^n$ where a_n is the number of length-n walks in W for $n \in \{1, 2, \ldots\}$. We define the *walk series* of a graph G to be the walk series of the set of walks in G of positive length, i.e. $\sum_{n=1}^{\infty} a_n x^n$ where a_n is the number of length-n walks in G.



walk series $=$ $\sum_{n=1}^{\infty}(4 \cdot 2^n)x^n$

FIGURE 3.13: A graph and its walk series (cf. Example 3.19).

Remark 3.29. Observe that the walk series of a set of walks of positive length W can also be obtained in the following way. A set of walks W can be seen as a multivariable non-commutative power series, as a formal sum of its walks, where a single walk is a product of its edges (which are the variables of this series). Then the walk series of W is the power series obtained from W by replacing every edge (variable) by x.



$$\begin{aligned} w_{non-com} = {} & 2 \cdot (e_1 + e_2 + e_3 + e_4) + \\ & ((e_4e_3 + e_4e_4 + e_1e_1 + e_1e_2) + (e_1e_4 + e_1e_1 + e_2e_2 + e_2e_3) + \\ & (e_2e_1 + e_2e_2 + e_3e_3 + e_3e_4) + (e_3e_2 + e_3e_3 + e_4e_4 + e_4e_1)) + \ldots \end{aligned}$$

$$\text{walk series} = w_{non-com}[e_1, e_2, e_3, e_4 := x] = \sum_{n=1}^{\infty}(4 \cdot 2^n)x^n$$

FIGURE 3.14: Representation of the set of walks as a multivariable non-commutative power series and evaluation to the walk series.

Observation 3.30. *Two graphs are walk-equivalent if and only if they have the same number of vertices and their walk series are equal.*

3.6.1 The proof of Lemma 3.26

We split the proof of Lemma 3.26 into three following steps.

Step 1. *There is a multisorted compositional function ϕ from the k-sourced graph algebra to the algebra $x\mathbb{Z}^{\text{rat}}[[x]]$ that is "injective up to walk-equivalence", i.e.*

$$\phi(G) = \phi(H) \textit{ if and only if } G, H \textit{ are walk-equivalent.} \tag{3.2}$$

Step 2. *The algebra $x\mathbb{Z}^{\text{rat}}[[x]]$ can be polynomially simulated by the enrichment of the field $\mathbb{Z}(x)$ by the division operation.*

Step 3. *Zeroness of register transducers with output in the enrichment of the field $\mathbb{Z}(x)$ by the division operation can be reduced to zeroness of register transducers with output in the ring $\mathbb{Z}[x]$.*

The above steps are composed as follows: due to Lemma 2.26 and Remark 2.31 composing Step 1 with Step 2 gives a reduction to zeroness of register transducers with output in the field $\mathbb{Z}(x)$ enriched with the division operation, and Step 3 reduces this problem to zeroness of register transducers with output in the ring $\mathbb{Z}[x]$ (without the division operation), which is shown decidable in Theorem 4 for an equivalent model of polynomial grammars.

Proof of Step 1. We will construct ϕ that satisfies the property (3.2) using Observation 3.30. For a sort I by $G : I$ we denote a k-sourced graph G of sort I. As the first component of ϕ we put the mapping

$$G = (V, E) \overset{\#_V}{\longmapsto} |V|.$$

Observe it is compositional (the proof is straightforward; for example, $\#_V(\text{fuse}(G : I, H : J)) = \#_V(G) + \#_V(H) - |I \cap J|$; we omit the rest of the details). The second component of ϕ is a compositional mapping $\widetilde{\phi}$ from the k-sourced graph algebra to $x\mathbb{Z}^{\text{rat}}[[x]]$ such that $\widetilde{\phi}(G)$ is the walk series of G for a graph G with no sources. After defining $\widetilde{\phi}$, we define ϕ by

$$\phi : G \mapsto (\#_V(G), \widetilde{\phi}(G)).$$

It satisfies the property (3.2) due to Observation 3.30, which finishes the proof.

In the rest of the proof we define $\widetilde{\phi}$, and show that it is compositional.

Let $G : I$ be a k-sourced graph and $\bullet$ be a fresh symbol. Let $G_{i,j}$, for i, j being either a number of a source of I or the $\bullet$ symbol, be the set of walks of positive length from the i-th source to the j-th source of G whose *internal* vertices are not sources, where the symbol $\bullet$ denotes the set of all non-source vertices (and hence by an abuse of notation, $G_{i,\bullet}$ denotes the set of walks of positive length from the i-th source to a non-source vertex, and $G_{\bullet,\bullet}$ denotes the set of walks of positive length from a non-source vertex to a non-source vertex). Let $g_{i,j}$ denote the walk series of $G_{i,j}$ (define $h_{i,j}, k_{i,j}$ analogously for a graph denoted H, K etc.). In particular, if G is a graph with no sources, then

$g_{\bullet,\bullet}$ is the walk series of G. We have the following:

$$k_{i,j} = \begin{cases} g_{i,j} + h_{i,j} - x & \text{if there is an edge from } i\text{-th to } j\text{-th source in both } G \text{ and } H \\ g_{i,j} + h_{i,j} & \text{otherwise} \end{cases},$$

where $K = \text{fuse}(G,H), i,j \in \{1,\dots,k\} \cup \{\bullet\}$, and

$$k'_{i,j} = \begin{cases} g_{i,j} + g_{i,l} \cdot (1 + g^+_{l,l}) \cdot g_{l,j} & \text{if } i \neq \bullet \text{ and } j \neq \bullet \\ g_{i,j} + g_{i,l} \cdot (1 + g^+_{l,l}) \cdot (1 + g_{l,j}) & \text{if } i \neq \bullet \text{ and } j = \bullet \\ g_{i,j} + (1 + g_{i,l}) \cdot (1 + g^+_{l,l}) \cdot g_{l,j} & \text{if } i = \bullet \text{ and } j \neq \bullet \\ g_{i,j} + (1 + g_{i,l}) \cdot (1 + g^+_{l,l}) \cdot (1 + g_{l,j}) & \text{if } i = \bullet \text{ and } j = \bullet \end{cases},$$

where $K' = \text{forget}_l(G)$, $l \in \{1,\dots,k\}$, $i,j \in \{1,\dots,k\} \cup \{\bullet\}$,

$$k''_{i,j} = k''_{\pi^{-1}(i),\pi^{-1}(j)} \text{ where } \pi(\bullet) = \bullet, \text{ where } K'' = \text{rename}_\pi(G),\ i,j \in \{1,\dots,k\} \cup \{\bullet\}. \tag{3.3}$$

The above formulas can be either proved by hand, or by substituting x for every edge in the analogous, self-evident, formulas for sets of walks (with the mentioned view of sets of walks as multivariable non-commutative power series, see Remark 3.29). The formula (3.3) proves that the mapping $\widetilde{\phi}$ from the multisorted variant of k-sourced graph algebra to the algebra of rational power series defined by

$$\widetilde{\phi}(G\!:\!I) := (g_{i,j})_{i,j \in \text{sources}(I) \cup \{\bullet\}}$$

is compositional. This finishes the proof. □

Proof of Step 2. Let us observe that

$$f \cdot f^+ + f = f^+,$$

and hence

$$f^+ = \frac{f}{1-f}. \tag{3.4}$$

The formula (3.4) has two following consequences: (1) every rational power series can be seen as a rational function from $\mathbb{Z}(x)$ (as it is obtained from polynomials by operations $+, \cdot$, and $^+$), and (2) operation $^+$ can be simulated by the ring operations and *division*. This finishes the proof. □

Proof of Step 3. We will mimic the classical embedding of a field into the projective line, in which the division operation (and, in fact, every rational function of this field) becomes a polynomial function. Precisely, we will construct a mapping ϕ from $\mathbb{Z}(x)$ into the set $\mathbb{Z}[x] \times \mathbb{Z}[x]$ such that an element $a \in \mathbb{Z}(x)$ is mapped to a pair (numerator, denominator) of a fraction that represents a; the counterparts of operations $+, \cdot, /$ will be defined in $\mathbb{Z}[x] \times \mathbb{Z}[x]$ as for usual fractions (see (3.6)). Finally, we will use the fact that for a fraction $\frac{p}{q}$ we have

$$\frac{p}{q} = 0 \text{ if and only if } p = 0. \tag{3.5}$$

We proceed as follows. Consider the multi-valued mapping

$$\phi : \mathbb{Z}(x) \to \mathbb{Z}[x] \times \mathbb{Z}[x]$$

defined by

$$\frac{p}{q} \mapsto (p,q), \text{ for } p,q \in \mathbb{Z}[x]$$

i.e. that sends every element $a \in \mathbb{Z}(x)$ to *all* pairs (p,q) such that $a = \frac{p}{q}$. Define operations $\psi_+, \psi_\cdot, \psi_/$ on $\mathbb{Z}[x] \times \mathbb{Z}[x]$ by

$$\begin{aligned}
\text{(addition)} \quad & \psi_+((p_1,q_1),(p_2,q_2)) = (p_1q_2 + q_1p_2, q_1q_2), \\
\text{(multiplication)} \quad & \psi_\cdot((p_1,q_1),(p_2,q_2)) = (p_1p_2, q_1q_2), \\
\text{(division)} \quad & \psi_/((p_1,q_1),(p_2,q_2)) = (p_1q_2, q_1p_2).
\end{aligned} \tag{3.6}$$

Let T be a register transducer with output in the enrichment of $\mathbb{Z}(x)$ by the division operation. As in Observation 2.24, by applying the mapping ϕ (with any choice of its values) and the operations $\psi_+, \psi_\cdot, \psi_/$ one can construct a register transducer ϕT with output in $\mathbb{Z}[x]$ whose output on a given input can be described as follows: for every run of T that outputs $a \in \mathbb{Z}(x)$ there is a corresponding run of ϕT that outputs $(p,q) \in \mathbb{Z}[x] \times \mathbb{Z}[x]$ such that $\frac{p}{q} = a$. The reduction is hence as follows: zeroness holds for T if and only if zeroness holds for the register transducer that outputs the first coordinate of the output of ϕT (cf. (3.5)). □

3.7 Summary

In this chapter, we consider transductions of graphs of bounded treewidth defined using MSO logic. We characterize them by register transducers with output in a k-sourced graph algebra (Lemma 3.24). Using this characterization, we show that functionality and equivalence of functional transductions are decidable when the output graphs are considered up to walk-equivalence (Theorem 8).

One can apply the proof scheme of Theorem 8 with a different compositional graph polynomial or power series than the walk series, and thus obtain a different, possibly stronger, decidability result. This idea seems particularly promising, as there are many known graph polynomials and power series. However, none of the ones we found were relevant, as they either were not compositional, or had low distinguishing power: for example, Tutte polynomial does not distinguish (unrooted) trees of the same size[27], and characteristic polynomial does not distinguish (unrooted) trees almost surely[28].

We do not know if functionality (or an inter-reducible problem of equivalence of functional) MSO_2 transductions of graphs of bounded treewidth is decidable. However, if it is, the proof could follow the following proof schemes that use our result.

[27] This fact follows straightforwardly from the definition of Tutte polynomial.

[28] Schwenk, 1973, Theorem 8.

1. Prove an analogous result to Theorem 8 but for a different equivalence relation that, together with walk-equivalence, characterize graphs uniquely up to isomorphism.

2. Find an MSO_2 transduction that maps every pair of different graphs to a pair of walk-*inequivalent* graphs; then functionality could be decided by post-composing a given transduction with this transduction and applying Theorem 8.

References. This chapter is based on a paper of Bojańczyk and S.[29]. The introductory section (Section 3.1) is also partially based on a paper of Boiret, Piórkowski, and S.[30].

[29] M. Bojańczyk and Schmude, 2020.

[30] Boiret, Piórkowski, and Schmude, 2018.

Chapter 4

Adding substitution

In this chapter, we consider register transducers with output in the ring of polynomials enriched with the substitution operation. This model can simulate analogous models in which a ring of polynomials is replaced with a free monoid, an algebra of trees, or any other algebra that can be polynomially simulated by a ring with no zero divisors. We show that this model has undecidable zeroness, even in the case when the substitution is applied at most once during the run (Theorem 9). Then we find two restrictions on the use of substitution under each of which zeroness is decidable (Theorem 10 and Lemma 4.22). We also discuss the consequences these results have on tree-to-string transductions (second paragraph of Summary section; Example 4.8 and Corollary 4.9; Theorem 11 and Example 4.21).

Notation. In this chapter we denote vector coordinates by parenthesized superscripts, not subscripts like in the previous chapters; for example, for a vector $\boldsymbol{\alpha}$ of length $n \in \{1, 2, \ldots\}$ we have

$$\boldsymbol{\alpha} = (\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(n)}).$$

4.1 Substitution and reset VASS

For a variable set X, by the substitution operation we mean an $(|X| + 1)$-ary operation that maps elements (like polynomials, words, etc.) $f, (g_x)_{x \in X} =: \overline{g}$ to the element $f[x := g_x, x \in X]$, which we sometimes denote by $f(\overline{g})$. In this section we consider the univariate ring of polynomials; in consequence, substitution is simply composition of univariate polynomials, which we denote by $f \circ g$ for $f, g \in \mathbb{Z}[x]$. The main result of this section is the following.

Theorem 9. *The following problem is undecidable.*
Name: *Zeroness of register composition*
Input: $T = (\Sigma, Q, F, \mathcal{R}, \Delta_0, \Delta, f_{\text{out}})$ *– a register transducer with output in the ring* $\mathbb{Z}[x]$,
$R, S \in \mathcal{R}$ *– registers of* T
Question: *Is it the case that*

$$R \circ S = 0 \textit{ on every accepting run of } T?$$

The rest of this section is devoted to the proof of the above theorem. We proceed to the statement of the undecidability result from which we give the reduction (Corollary 4.4).

Definition 4.1 (Reset VASS)**.** A *reset Vector Addition System with States (reset VASS)* consists of

- a *dimension* $n \in \mathbb{N}$,
- a finite set of *states* Q,
- an *initial state* $q_0 \in Q$,
- a set of *final states* $F \subseteq Q$,
- a set of *transitions* $\Delta \subseteq Q \times \{inc, leave, dec, reset\}^n \times Q$.

The symbols $inc, leave, dec, reset$ represent the counter operations "increment by one, leave as it is, decrement by one, reset to 0". A *configuration* of a reset VASS (n, Q, q_0, F, Δ) is a pair (state, vector) from $Q \times \mathbb{N}^n$. The transitions are applied coordinate-wise: a configuration $(r, u) \in Q \times \mathbb{N}^n$ is *one-step reachable* from (q, v) – a property we denote by $(q, v) {\to}_{\mathcal{V}} (r, u)$ – if there is a transition $(q, \boldsymbol{\alpha}, r) \in \Delta$ such that $u^{(i)} = \boldsymbol{\alpha}^{(i)}(v^{(i)})$ for $i = 1 \ldots, n$.

The *reachability* relation is the transitive closure of $\to_{\mathcal{V}}$. A *run (of $\mathcal{V}$)* from $s \in \mathbb{N}^n$ to $t \in \mathbb{N}^n$ is a sequence

$$(q_0, v_0 = s) \to_{\mathcal{V}} (q_1, v_1) \to_{\mathcal{V}} \ldots \to_{\mathcal{V}} (q_l, v_l = t)$$

where q_0 is the initial state of for some states $q_1, \ldots, q_{l-1} \in Q$ and some final state $q_l \in F$. By abuse of notation, we say that a vector $t \in \mathbb{N}^n$ is *reachable (by $\mathcal{V}$)* from a vector $s \in \mathbb{N}^n$ if there is a run of $\mathcal{V}$ from s to t.

Let us note that the states are a syntactic sugar that can be eliminated without the loss of expressiveness at a cost of a larger dimension.

Example 4.2. In this example, for brevity of notation, we use an equivalent model of reset VASS in which the transitions can increment and decrement coordinates by arbitrary numbers (simulating this model with our reset VASS model is straightforward and requires extra states); we denote the "increment by i" operation by $inc\!:\!i$ for $i \in \{1, 2, \ldots\}$.

Let $n = 2, Q = \{q_0, q_1\}, F = \{q_1\}$, and $\Delta = \{\delta_1, \delta_1', \delta_2, \delta_3\}$ where $\delta_1 = (q_0, (inc : 3, dec : 1), q_0)$, $\delta_1' = (q_1, (inc : 3, dec : 1), q_1)$, $\delta_2 = (q_0, (dec : 5, inc : 3), q_1)$, and $\delta_3 = (q_1, (reset, inc : 2), q_1)$. Consider a VASS $\mathcal{V} = (n, Q, q_0, F, \Delta)$. Then the vector $t_1 = (3, 4)$ is reachable from the vector $s_1 = t_1 - (1, 2) = (2, 2)$. Indeed,

$$(q_0, (2, 2)) \to_{\delta_1} (q_0, (5, 1)) \to_{\delta_1} (q_0, (8, 0)) \to_{\delta_2} (q_1, (3, 3)) \to_{\delta_3} (q_1, (0, 5)) \to_{\delta_1'} (q_1, (3, 4)).$$

On the other hand, the vector $t_2 = (2, 3)$ is not reachable from the vector $s_2 = t_2 - (1, 2) = (1, 1)$. Indeed, the only choice of a transition $(q_0, (1, 1)) \to_{\delta_1} (q_0, (4, 0))$ reaches a dead end.

Fact 4.3. *(Araki and Kasami, 1976, Theorem 5). The following problem is undecidable.*
Name: *Reachability in a reset VASS*
Input: $\mathcal{V} = (n, Q, q_0, F, \Delta)$ *– a reset VASS,*
$s, t \in \mathbb{N}^n$.
Question: *Is t reachable by $\mathcal{V}$ from s?*

The following special case of the above problem also is undecidable. By 0^n we denote the vector $(0, \ldots, 0) \in \mathbb{N}^n$.

Corollary 4.4. *The following problem is undecidable.*
Name: *Reachability from 0 to 0 in a reset VASS*
Input: $\mathcal{V} = (n, Q, q_0, F, \Delta)$ *– a reset VASS.*
Question: *Is 0^n reachable by $\mathcal{V}$ from 0^n?*

Proof. We reduce from reachability in a reset VASS. Intuitively speaking, by introducing additional states we force a reset VASS to add the vector s before the run and subtract the vector t after the run.

More formally, let $\mathcal{V} = (n, Q, q_0, F, \Delta)$ be a reset VASS and $s, t \in \mathbb{N}^n$. We construct a reset VASS $\mathcal{V}' = (n, Q', q_0', \{q_f'\}, \Delta')$ such that (q_f, t) is reachable by $\mathcal{V}$ from (q_0, s) for some final state $q_f \in F$ if and only if $(q_f', 0^n)$ is reachable by $\mathcal{V}'$ from $(q_0', 0^n)$. The reset VASS $\mathcal{V}'$ is constructed from $\mathcal{V}$ by adding a new initial state q_0' (and turning q_0 to a non-initial state), adding a sequence of transitions that changes the state from q_0' to q_0 and translates the vector of the configuration by s (this can be achieved by adding $|s^{(1)}| + |s^{(2)}| + \ldots + |s^{(n)}|$ new states), adding a new final state q_f' and turning all states from F to non-final states, and adding a sequence of transitions that changes the state from q_f to q_f' and translates the vector of the configuration by $-t$ (likewise) for each state $q_f \in F$. □

4.1.1 Proof of Theorem 9

Now we proceed to the proof of Theorem 9. The crucial ingredient of this proof is a polynomial, which we call an *inverting polynomial*, that maps non-zero integers from a fixed interval to zero and vice versa.

Proof of Theorem 9. For the simplicity of notation, we consider an inter-reducible problem of zeroness of register composition problem in which not only registers, but also polynomial expressions of registers can be composed (for a register transducer, one can always introduce a register that stores a polynomial expression of the remaining registers). We reduce from reachability from 0 to 0 in a reset VASS, as in the following claim.

Claim 4.5. *Let $\mathcal{V} = (n, Q, q_0, F, \Delta)$ be a reset VASS. One can construct a deterministic register transducer T with output in the ring $\mathbb{Z}[x]$ with a distinguished tuple of registers $R_1, \ldots, R_n$ and a distinguished register R^{val} that inputs sequences of transitions of $\mathcal{V}$, accepts only those sequences in which the consecutive transitions' states are matched, and has the following properties:*

(i) *if the read input sequence yields a run of $\mathcal{V}$ from 0^n to 0^n then*

- *R^{val} stores a non-zero number, and*
- *all R_i's store 0,*

(ii) *if the read input sequence does not yield a run of $\mathcal{V}$ from 0^n to 0^n then* either

- *R^{val} stores a non-zero number, but some of R_i's are non-zero (this is in case the input sequence yields a run from 0^n, which however does not end in 0^n), or*
- *R^{val} stores 0 (this is in case the input sequence does not yield a run from 0^n: since the matching of the states is being checked by the states of T, this happens if and only if some of the read transitions tried to decrement a coordinate that had value 0)*

Proof. Let us sketch the construction of T. The input alphabet is Δ. The states are $Q \cup \{\perp\}$, where $\perp$ denotes a new state which we call the *error state*; the state of T is the current state of $\mathcal{V}$ if the states of the consecutive read transitions match, and the error state $\perp$ otherwise. There are *n coordinate registers* $R_1, \ldots, R_n$, one for each of n coordinates of $\mathcal{V}$, that intend to store the current vector of $\mathcal{V}$. They are updated by adding integers and resetting to 0 accordingly to the read transitions. However, this implies that some coordinate registers might go below 0, specifically, on those input sequences that do *not* yield a run of $\mathcal{V}$ from the vector 0^n. To prevent "accepting" such an input sequence we introduce a "validity" register. The *validity register* R^{val} stores a non-zero number if and only if none of the coordinate registers went below 0 during the run. It is updated by multiplying it with

$$\Pi_{i=1}^{n}(R_i + 1);$$

it turns 0 at the first time some coordinate register goes below 0.

It is clear that T satisfies the desired properties. □

Observe that the expression

$$R^{\text{val}} \cdot \left(\sum_{i=1}^{n} R_i\right)$$

is non-zero only on those input sequences that yield a run of $\mathcal{V}$ from the vector 0^n that does *not* end in the vector 0^n. To achieve the opposite behavior, we use the *inverting polynomial* $p_N(x)$ that for a fixed $N \in \{0, 1, \ldots\}$ maps a number x from $\{0, 1, \ldots, N\}$ to 0 if and only if x is *non-zero*:

$$p_N(x) := (x - 1) \cdot \ldots \cdot (x - N).$$

We define an *inverting register* R^{inv} and an *auxiliary register* R^{aux} that, after reading an input sequence of length $l \in \{0, 1, \ldots\}$, store the inverting polynomial p_l and the number l. The auxiliary register is updated by adding 1, and the inverting register is updated by multiplying it by $(x - (R^{\mathrm{aux}} + 1))$. Finally, in the problem we ask if the following expression is zero:

$$\left(R^{\mathrm{val}} \cdot R^{\mathrm{inv}}\right) \circ \left(\sum_{i=1}^{n} R_i\right); \tag{4.1}$$

note that R^{val} always stores an integer, hence the above expression is equivalent to, more natural, $R^{\mathrm{val}} \cdot \left(R^{\mathrm{inv}} \circ \left(\sum_{i=1}^{n} R_i\right)\right)$. It is now clear that the above expression is non-zero for those input sequences that yield a run of $\mathcal{V}$ from the vector 0^n to the vector 0^n; in other words, 0^n is reachable by $\mathcal{V}$ from 0^n if and only if zeroness of register composition (in the variant described by (4.1)) does not hold for T. □

References. Theorem 9 is due to Lasota and Piórkowski[1]; a similar proof concept can be found in a paper by Benedikt et al.[2], and the idea to use the inverting polynomial can be found in a paper by Boiret, Piórkowski, and S.[3].

4.2 The independent substitution

In the problem of zeroness of register composition (Theorem 9) the polynomials being composed are generated *dependently*, by the same register transducer. In this section's main result, Theorem 10, we show that if the composed polynomials are generated *independently*, by two separate register transducers (or equivalently, polynomial grammars), then the problem is decidable. In addition, the polynomials may have arbitrarily many variables.

Theorem 10. *Let R be a computable ring with no zero divisors, and X be a variable set. Let G, H be polynomial grammars with output in the ring $R[X]$ where the grammar H is of rank $|X|$. Then one can decide if*

$$L(G)(L(H)) = 0.$$

Now we proceed to Example 4.8 and Corollary 4.9, which show an application of Theorem 10.

Definition 4.6. A *word equation over an alphabet Σ in variable set X* is a pair of words over the alphabet $\Sigma \cup X$. An X-tuple of words $(w_x)_{x \in X} \in (\Sigma^*)^X$ is a solution to an equation $e = (e_1, e_2) \in (\Sigma \cup X)^* \times (\Sigma \cup X)^*$ if

$$e_1[x := w_x, x \in X] = e_2[x := w_x, x \in X].$$

A *system of equations* is simply a set of equations .

[1] Lasota and Piórkowski, 2019.

[2] Benedikt et al., 2017, Example 3.

[3] Boiret, Piórkowski, and Schmude, 2018, Theorem 17.

Example 4.7. Let $\Sigma = \{a, b\}$ and $X = \{x_1, x_2\}$. Consider the word equation $e = (e_1, e_2) = (x_1ax_1ab, bx_2x_2)$, i.e.

$$x_1ax_1ab = bx_2x_2.$$

The X-tuple of words $\{x_1 = b, x_2 = ab\}$ is a solution to the equation e. The set of solutions of e is $\{\{x_1 = bw, x_2 = wab\} \mid w \in \{a, b\}^*\}$.

Example 4.8. Suppose we have a language of k-tuples of words

$$L \subseteq \underbrace{\Sigma^* \times \ldots \times \Sigma^*}_{k}$$

for some $k \in \{1, 2, \ldots\}$ and we want to test if it satisfies a given property ψ that we can describe by an infinite system of word equations. Moreover, suppose that we can generate both L and the system for ψ by polynomial grammars (of rank k and 2, respectively). For example, say that $k = 3$,

$$L = \{(w, v, u) \in (\Sigma^* \times \Sigma^* \times \Sigma^*) \mid w, v \in u^*\},$$

and that the property ψ of triples of words (x, y, z) is defined by the following system of equations E:

$$E := \{x^n z^{k_1} y^m z^{k_2} = x^{n'} z^{k'_1} y^{m'} z^{k'_2} \mid k_1 + k_2 = k'_1 + k'_2 \text{ and } n + m = n' + m'\}.$$

Both L and E can be generated by polynomial grammars. Indeed, L can be generated by the grammar

$$\begin{array}{l} G := \{ \\ S \to X, \\ X \to p_{z,\varepsilon,\varepsilon}(X) \mid Y, \\ Y \to p_{\varepsilon,z,\varepsilon}(Y) \mid Z, \\ Z \to p_{\varepsilon,\varepsilon,\sigma}(Z) \text{ for } \sigma \in \Sigma \mid (\varepsilon, \varepsilon, \sigma) \text{ for } \sigma \in \Sigma\} \end{array}$$

where

$$\begin{array}{l} p_{z,\varepsilon,\varepsilon}(x, y, z) := (x \cdot z, y, z), \\ p_{\varepsilon,z,\varepsilon}(x, y, z) := (x, y \cdot z, z), \\ p_{\varepsilon,\varepsilon,\sigma}(x, y, z) := (x, y, \sigma \cdot z) \quad \text{for } \sigma \in \Sigma. \end{array}$$

The system of equations E (for the property ψ) in variable set $\{x, y, z\}$ can be generated by a polynomial grammar H that generates 8-tuples $(x^n, z^{k_1}, y^m, z^{k_2}, x^{n'}, z^{k'_1}, y^{m'}, z^{k'_2})$ of words over the alphabet $\{x, y, z\}$ such that $n_1 + n_2 = n'_1 + n'_2$ and $n + m = n' + m'$ and concatenates them appropriately – the construction is analogous as for the grammar G, and hence we omit the details. In consequence, the set $L(G)(L(H))$ consists of pairs $(e_1[(x, y, z) := (w, v, u)], e_2[(x, y, z) := (w, v, u)])$ for $(w, v, u) \in L$ for $(e_1, e_2) \in E$.

Now, one can decide if L satisfies ψ as follows. Observe that L satisfies ψ if and only if $L(G)(L(H))$ satisfies an equivalence-like problem that asks if the first and the second coordinate of every element of a given language of pairs of words are equal. Hence, using the construction from Observation 2.24, one can apply the polynomial simulation of a free monoid by a ring of polynomials from Definition 4.23 to both grammars L and E; under this simulation, word substitution corresponds to polynomial substitution (Lemma 4.25), thus this is a reduction to the problem from Theorem 10. Notice that the mentioned polynomial simulations by the ring of integers (Figure 5 and Figure 2.11) do not support substitution in the sense of Lemma 4.25, and hence are not suitable for such applications.

Corollary 4.9. *Let languages*

$$L \subseteq \underbrace{\Sigma^* \times \ldots \times \Sigma^*)}_{k},$$
$$E \subseteq (\Sigma \cup \{?_1, \ldots, ?_k\})^* \times (\Sigma \cup \{?_1, \ldots, ?_k\})^*$$

be generated by polynomial grammars over free monoids for some $k \in \{1, 2, \ldots\}$. Then, it can be decided if every k-tuple from L satisfies every equation in variable set $\{?_1, \ldots, ?_k\}$ from E, i.e.

$$C_1[?_1 := w_1, \ldots, ?_k := w_k] = C_2[?_1 := w_1, \ldots, ?_k := w_k] \text{ for } (C_1, C_2) \in E$$
$$\text{for } (w_1, \ldots, w_k) \in L.$$

Remark 4.10. Let us now discuss if the composition of languages of every two polynomial grammars G, H can be simply obtained by one polynomial grammar, specifically, the one in which the initial nonterminal of H is passed to the terminals of G (which are polynomials) as the tuple of arguments. This idea is correct in the case when the grammar G is linear (this is the case in Example 4.8), however is not correct in general. This is because the values of H passed to different occurrences of terminals of G might return different values (cf. Example 4.11).

Example 4.11. Let $G = \{S \to q(A, A), A \to p\}, H = \{S' \to 0 \mid 1\}$ be polynomial grammars with output in $\mathbb{Z}[x]$, where $q(x_1, x_2) = x_1 + x_2$ and $p = x \in \mathbb{Z}[x]$ (notice p is a constant). Then $L(G) = \{2x\}, L(H) = \{0, 1\}$, and $L(G)(L(H)) = \{0, 2\}$. However, by applying the construction from Remark 4.10 one obtains a grammar

$$\{S \to q(A, A), A \to p(S'), S' \to 0 \mid 1\}.$$

The language of the above grammar is not equal to $L(G)(L(H)$: a derivation

$$S \to q(A, A) \to q(p(S'), p(S')) \to q(p(0), p(1)) = 1,$$

returns a value outside $L(G)(L(H))$.

Now we proceed to the proof of Theorem 10, which relies on additional concepts from commutative algebra that we now describe.

More commutative algebra. Let R be a ring. A *quotient ring* of R by an ideal I of R, denoted R/I, is defined as the set of equivalence classes of the relation

$\sim_I$ defined by

$$a \sim_I b \text{ if } a - b \in I$$

with operations $+, \cdot$ defined by

$$[a]_{\sim_I} + [b]_{\sim_I} := [a+b]_{\sim_I},$$
$$[a]_{\sim_I} \cdot [b]_{\sim_I} := [a \cdot b]_{\sim_I}$$

where $[a]_{\sim_I}$ denotes the equivalence class of an element $a \in R$ with respect to $\sim_I$. An ideal I of R is a *prime ideal* if $a, b \in I$ implies that either $a \in I$ or $b \in I$ for any $a, b \in R$. Observe that an ideal I is a prime ideal if and only if the quotient ring R/I is a ring with no zero divisors. A *radical* of an ideal I of R, denoted $\sqrt{I}$, is a subset of R defined by

$$\sqrt{I} := \{f \in R \mid f^n \in I \text{ for some } n \in \{1, 2, \ldots\}\};$$

it is an ideal of R that contains I. An ideal I is a *radical ideal* if it is equal to its radical.

We will use the following lemma.

Lemma 4.12. *Let K be a computable field, X be a variable set, and $I \subseteq K[X]$ be a radical ideal. Then the quotient ring $K[X]/I$ can be effectively embedded into a finite product of rings with no zero divisors.*

Proof. Compute prime ideals $P_1, \ldots, P_n$ such that

$$I = \bigcap_{i=1}^{n} P_i.$$[4]

It can be checked straightforwardly that the mapping ϕ defined by

$$\phi : K[X]/I \to K[X]/P_1 \times \ldots \times K[X]/P_n,$$
$$\phi([a]_{\sim_I}) := ([a]_{\sim_{P_1}}, \ldots, [a]_{\sim_{P_n}})$$

is an injective ring homomorphism. □

Now we are ready to prove Theorem 10.

Proof of Theorem 10. In this proof, for brevity of notation, we denote the language of a grammar G by G, instead of, formally correct, $L(G)$. Without the loss of generality we assume that R is a field (by embedding R into the field of fractions), and from now on we denote it by K.

Observe that $G(H) = 0$ if and only if there exists a radical ideal I in the ring $K[X]$ such that the following conditions hold:

$$\begin{cases} G \subseteq I, & (1) \\ I(H) = 0 & (2) \end{cases}$$

[4] Such family of ideals $\{P_1, \ldots, P_n\}$ is called a *prime decomposition* of a radical ideal I. It is well-known that it exists for every radical ideal and can be obtained effectively, see for example a remark after Theorem 7 of §6 of Chapter 4 of Cox, Little, and O'Shea, 2015.

(for example for I one can take the radical of the smallest ideal in the ring $K[X]$ that contains G). We show the following claim.

Claim. *Given a radical ideal $I \subseteq K[X]$ one can effectively test conditions (1) and (2).*

Using the above claim, one can decide zeroness of $G(H)$ by the following algorithm. It is the composition of two semi-algorithms: one that enumerates derivations of grammars G, H to find a counterexample to $G(H) = 0$, and one that enumerates radical ideals in the ring $K[X]$ and tests conditions (1) and (2) (radical ideals can be enumerated by enumerating finite sets of polynomials and testing if the ideals they generate are radical[5]). This finishes the proof.

Proof of Claim. We reduce each of conditions (1) and (2) to zeroness of a polynomial grammar with output in a ring with no zero divisors.

Condition (2), i.e. $I(H) = 0$, is equivalent to

$$f(H) = 0 \text{ for all generators } f \text{ of } I.$$

This is a conjunction of instances of the zeroness problem of a polynomial grammar with output in K.

To decide condition (1), first, intuitively speaking, we take the semantics of the grammar G in the ring $K[X]/I$ – then this condition becomes an instance of zeroness, however with the ring possibly having zero divisors. Formally, we consider a polynomial grammar $G_{K[X]/I}$ obtained from G by replacing every coefficient of every polynomial used in the definition of G by its $\sim_I$-equivalence class. Then the condition (1) is equivalent to

$$G_{K[X]/I} = 0. \tag{4.2}$$

Using Lemma 4.12, we embed the ring $K[X]/I$ into a product of rings with no zero divisors, thus reducing (4.2) to a finite number of instances of zeroness of a polynomial grammar, each over a (possibly different) ring with no zero divisors (cf. Lemma 2.26). This finishes the proof. □

□

4.3 The simultaneous substitution

This section is motivated by the following example.

Example 4.13. Let Σ be an alphabet and $\#$ be a fresh symbol. Consider the following stateless deterministic register transducer T_1 that inputs trees over the alphabet $\{\sigma^{(1)} \mid \sigma \in \Sigma\} \cup \{\perp^{(0)}\}$ with output in the enrichment of the free monoid $(\Sigma \cup \{\#\})^*$ with the substitution operation; it has two registers R and S, transitions $\Delta = \{(\sigma, p_\sigma) \mid \sigma \in \Sigma\} \cup \{(\perp, (R \mapsto \varepsilon, S \mapsto \varepsilon))\}$, where the register update p_σ for $\sigma \in \Sigma$ is the composition of the operations

[5] It is well-known that one can effectively test if a given ideal is a radical ideal, see for example ibid., Second bullet point (denoted "Radical Ideal") in remarks after the proof of Theorem 7 of §2 of Chapter 4.

1. substitute $[\# := \sigma \cdot \#]$ in *all* registers, and

2. append $R \cdot \sigma \cdot \#$ to S and σ to R.

(notice that the symbol # never occurs in the register R; nevertheless, we apply the substitution simultaneously to all registers for this example to be consistent with the model we introduce later; let us also remark that in that model not all substitutions are admitted), and the output function is

$$f_{\text{out}}(R, S) = S.$$

Transducer T_1 computes the function $\texttt{sqrev} : \Sigma^* \to (\Sigma \cup \{\#\})^*$ defined by

$$\begin{aligned}
&\texttt{sqrev}(\varepsilon) := \varepsilon, \\
&\texttt{sqrev}(\sigma_1 \ldots \sigma_n) := (\sigma_n \ldots \sigma_1 \#)^n \quad \text{for } \sigma_1, \ldots, \sigma_n \in \Sigma, n \in \{1, 2, \ldots\}.
\end{aligned}$$

Consider a register transducer T_2 defined similarly to T_1 except that the register update over $\sigma \in \Sigma$ is the composition of the following:

1. substitute $[\# := \sigma \cdot \#]$ in all registers, and

2. *prepend* $R \cdot \sigma \cdot \#$ to S and *append* σ to R.

In this section we show that equivalence can be decided for register transducers like T_1 and T_2 (Theorem 11).

Remark 4.14. The transducers T_1, T_2 from Example 4.13 *are* equivalent: they both compute the function $\texttt{sqrev}$ (we omit the proof). Let us also add that the function $\texttt{sqrev}$ is inspired by the function $\texttt{squaring}$ defined by Bojańczyk[6].

Transducer models. Now we proceed to the definition of this section's register transducer models, i.e. *register transducer with output in a field with simultaneous automorphisms* and *register transducer with output in a free monoid with simultaneous com-injective substitutions*. Each of these models is a register transducer whose output algebra is enriched with homomorphisms, and for each of them we additionally require that (1) it always applies a given homomorphism *simultaneously* to all registers, and (2) each of the homomorphisms can be extended to an automorphism of some (fixed) algebra that contains the output algebra (notice it can be the output algebra itself).

Definition 4.15 (Simultaneous homomorphism)**.** Let $\mathbb{A}$ be an algebra and $\tau : \mathbb{A} \to \mathbb{A}$ be a homomorphism. By $\tau \times \ldots \times \tau$ we denote the homomorphism defined by

$$\underbrace{(\tau \times \ldots \times \tau)}_{n} : \mathbb{A}^n \to \mathbb{A}^n,$$
$$(a_1, \ldots, a_n) \in \mathbb{A}^n \to (\tau(a_1), \ldots, \tau(a_n)).$$

We call it a *simultaneous homomorphism of $\mathbb{A}^n$ (defined by τ)*.

[6] M. Bojańczyk, 2018.

Definition 4.16 (Register transducer with simultaneous homomorphisms)**.** Let $\mathbb{A}$ be an algebra. A *register transducer with output in $\mathbb{A}$ with simultaneous homomorphisms* is a register transducer with output in the enrichment of $\mathbb{A}$ with homomorphisms that satisfies the following property: its register updates (which, recall, are polynomial operations of type $\mathbb{A}^{\mathcal{R}} \to \mathbb{A}^{\mathcal{R}}$, where $\mathcal{R}$ denotes the set of the transducer's registers) are compositions of polynomial operations of $\mathbb{A}$ and simultaneous homomorphisms of $\mathbb{A}^{\mathcal{R}}$. When defining the register transducer model, one can use a subclass of homomorphisms: for example, in Lemma 4.22 we take the class of automorphisms, and hence define a *register transducers with output in (a field) K with simultaneous automorphisms*.

Com-injectivity.

Definition 4.17 (Com-injectivity)**.** Let Σ be a finite alphabet and $\tau : \Sigma \to \Sigma^*$ be a word substitution. Observe that the word homomorphism defined by τ induces a homomorphism of *commutative* words. We call a substitution τ (or a word homomorphism it defines) *com-injective* if the induced homomorphism of commutative words is injective.

Example 4.18. Let $\Sigma = \{a, b\}$ and τ, τ' be the word substitutions defined by $\tau := \{a \mapsto aab, b \mapsto ab\}$ and $\tau' := \{a \mapsto ab, b \mapsto ba\}$. Then τ is com-injective: if $\tau(w)$ has n a's and m b's then necessarily w has $n - m$ a's and $2m - n$ b's. On the other hand, τ', despite being an injective homomorphism of words, is not com-injective: for example, a and b are mapped to the same commutative word.

Com-injectivity of a given word substitution can be easily decided, as stated in the following lemma.

Lemma 4.19. *Let Σ be a finite alphabet and $\tau : \Sigma \to \Sigma^*$ be a word substitution. It can be decided in polynomial time if τ is com-injective.*

Proof. Observe that com-injectivity of τ is equivalent to invertibility of the $\Sigma \times \Sigma$ integer-entry matrix that represents the homomorphism of commutative words defined by τ. Indeed, a $\mathbb{Q}$-linear mapping is injective on integer-entry vectors if and only if it is injective (on all, rational-entry, vectors), and when the domain and codomain are of the same dimension, then injectivity is equivalent to invertibility. Finally, it is well-known that invertibility of a matrix can be decided in polynomial time. □

Example 4.20. Let Σ and τ be as in Example 4.18. Then the homomorphism of commutative words defined by τ is represented by the following integer-entry matrix.

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}.$$

This matrix is invertible. Its inverse, whose entries happen to be integers (which is not required for com-injectivity), is equal to

$$\begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}.$$

We conclude this section with the following theorem.

Theorem 11. *Functionality and equivalence of functional register transducers with output in a free monoid with simultaneous com-injective substitutions is decidable.*

Before we prove the above theorem, we show its example application.

Example 4.21. Consider the substitution $[\# := \sigma \cdot \#]$ where $\sigma \in \Sigma$ from Example 4.13. It clearly is com-injective, as the corresponding matrix is triangular with ones on the diagonal, and so is invertible. In consequence, equivalence of the transducers T_1 and T_2 from that example can be decided by the algorithm from Theorem 11 (cf. Remark 4.14).

4.3.1 Proof of Theorem 11

The proof of Theorem 11 is based on the following lemma.

Lemma 4.22. *Let K be a computable field. Zeroness is decidable for register transducers with output in K with simultaneous automorphisms.*

In the proof of Theorem 11, Lemma 4.22 is combined with the fact that, via our polynomial simulation of a free monoid by a ring of polynomials (Definition 4.23), com-injective word homomorphisms induce automorphisms of a (fixed) computable field that contains the corresponding ring of polynomials (Lemma 4.26).

Proof of Lemma 4.22. The general idea behind this proof is that "applying an automorphism to the elements does not change the algebraic properties of the whole", in particular, applying an automorphism to the registers preserves existence of inductive invariants that witness zeroness of a register transducer, as we now prove. The proof proceeds analogously to the one of Theorem 4. The only difference is in the condition (b) of the definition of an inductive invariant: the function p might not be a polynomial function, but a composition of polynomial functions and simultaneous automorphisms of K. In consequence, the set $I \circ p$ might not consist of polynomial functions. However, we show that $I \circ p$ can be replaced by an equivalent set of polynomials I' that is equivalent in the sense that it maps the same tuples of elements of K to the zero element of K, i.e.

$$I'(\mathbf{a}) = 0 \text{ if and only if } (I \circ p)(\mathbf{a}) = 0 \quad \text{for every tuple } \mathbf{a}. \tag{4.3}$$

We give the proof for the case when p is a simultaneous automorphism; the proof of the general case follows by composition.

Let $i = \sum_{\alpha \in \mathbb{N}^n} a_\alpha \mathbf{x}^\alpha$ be a polynomial (implicitly: from the ideal I) from a polynomial ring $K[x_1, \ldots, x_n]$. Let τ be an automorphism of K, and consider a simultaneous automorphism $p = \underbrace{\tau \times \ldots \times \tau}_{n}$. Then, for any tuple $\mathbf{a} = (a_1, \ldots, a_n) \in K^n$ the formula $(i \circ p)(\mathbf{a}) = 0$ can be rewritten as

$$\sum_{\alpha=(\alpha_1,\ldots,\alpha_n)\in\mathbb{N}^n} a_\alpha \tau(a_1)^{\alpha_1} \ldots \tau(a_n)^{\alpha_n} = 0.$$

By applying τ^{-1} to both sides, this is equivalent to

$$\sum_{\alpha \in \mathbb{N}^n} \tau^{-1}(a_\alpha)\mathbf{a}^\alpha = 0.$$

Therefore, for an ideal $I \subseteq K[x_1, \dots, x_n]$ the set of polynomials

$$\tau^{-1}I := \left\{ \sum_{\alpha \in \mathbb{N}^n} \tau^{-1}(a_\alpha)\mathbf{x}^\alpha \mid \sum_{\alpha \in \mathbb{N}^n} a_\alpha \mathbf{x}^\alpha \in I \right\}$$

satisfies the property (4.3), i.e.

$$(\tau^{-1}I)(\mathbf{a}) = 0 \text{ if and only if } (I \circ p)(\mathbf{a}) = 0 \quad \text{for } \mathbf{a} \in K^n.$$

In consequence, we can replace the set of functions $I \circ (\tau \times \ldots \times \tau)$ by the set of polynomial functions $\tau^{-1}I$ for every automorphism τ of K.

Finally, let us see that testing inclusion of $\tau^{-1}I$ in other ideals, as required by the condition (b) of an inductive invariant, can be done effectively. Indeed, observe that $\tau^{-1}I$ is an ideal generated by the set $\tau^{-1}F$ where F is a generating set of I. This finishes the proof. □

Word-to-polynomial simulation. Now we define a polynomial simulation of a free monoid by a ring of polynomials. It is a generalization of the polynomial simulation of a free monoid by the ring of integers from Figure 5, which was used by Seidl et al. to prove decidability of equivalence for a subclass of register transducers with output in a free monoid (deterministic top-down tree-to-string transducers)[7]. The new feature of our polynomial simulation is that it supports the substitution operation (Lemma 4.25), and hence is applicable to a larger class of transducers.

Definition 4.23 (Word-to-polynomial simulation)**.** Let Σ be an alphabet. Let $\widetilde{\Sigma} = \{\widetilde{\sigma} \mid \sigma \in \Sigma\}$ and $\overline{\Sigma} = \{\overline{\sigma} \mid \sigma \in \Sigma\}$ be two copies of Σ. Consider the function ϕ defined by

$$\begin{aligned} &\phi = (\phi_1, \phi_2) : \Sigma^* \to \mathbb{Z}[\widetilde{\Sigma} \cup \overline{\Sigma}] \times \mathbb{Z}[\overline{\Sigma}], \\ &\phi(\sigma) := (\widetilde{\sigma}, \overline{\sigma}) \quad \text{for } \sigma \in \Sigma, \\ &\phi(\sigma w) := (\widetilde{\sigma} \cdot \phi_2(w) + \phi_1(w), \overline{\sigma} \cdot \phi_2(w)) \quad \text{for } \sigma \in \Sigma, w \in \Sigma^*. \end{aligned} \tag{4.4}$$

We introduce the notation

$$\begin{aligned} \widetilde{w} &:= \phi_1(w), \\ \overline{w} &:= \phi_2(w), \end{aligned}$$

and so the second line of (4.4) extends to all words, i.e.

$$\phi(w) = (\widetilde{w}, \overline{w}) \quad \text{for } w \in \Sigma^*. \tag{4.5}$$

[7] Seidl, Maneth, and Kemper, 2018.

In particular, the third line of (4.4) can be rewritten as

$$\begin{cases} \widetilde{\sigma w} = \widetilde{\sigma} \cdot \overline{w} + \widetilde{w}, \\ \overline{\sigma w} = \overline{\sigma} \cdot \overline{w} \end{cases} \quad \text{for } \sigma \in \Sigma, w \in \Sigma^*. \tag{4.4'}$$

Formula (4.4′) says that the mapping ϕ is essentially a homomorphism from the free monoid Σ^* to the monoid of $\mathbb{Z}[\widetilde{\Sigma} \cup \overline{\Sigma}]$-entry 2×2 matrices defined, for the generators of Σ^*, by

$$\sigma \overset{\phi}{\mapsto} \begin{bmatrix} \overline{\sigma} & 0 \\ \widetilde{\sigma} & 1 \end{bmatrix} \quad \text{for } \sigma \in \Sigma,$$

and so for an arbitrary word the following holds

$$w \overset{\phi}{\mapsto} \begin{bmatrix} \overline{w} & 0 \\ \widetilde{w} & 1 \end{bmatrix} \quad \text{for } w \in \Sigma^*. \tag{4.6}$$

We call the above mapping the *matrix form* of ϕ. As a consequence of (4.6), we obtain

$$w \cdot v \mapsto \begin{bmatrix} \overline{w} & 0 \\ \widetilde{w} & 1 \end{bmatrix} \cdot \begin{bmatrix} \overline{v} & 0 \\ \widetilde{v} & 1 \end{bmatrix} = \begin{bmatrix} \overline{w} \cdot \overline{v} & 0 \\ \widetilde{w} \cdot \overline{v} + \widetilde{v} & 1 \end{bmatrix} \quad \text{for } w, v \in \Sigma^*,$$

and hence

$$\begin{cases} \widetilde{wv} = \widetilde{w} \cdot \overline{v} + \widetilde{v}, \\ \overline{wv} = \overline{w} \cdot \overline{v} \end{cases} \quad \text{for } w, v \in \Sigma^*. \tag{4.4''}$$

Also, by unfolding the recursive formula (4.4′) one obtains that

$$\begin{cases} \widetilde{\sigma_n \ldots \sigma_1} = \sum_{i=1}^n \widetilde{\sigma}_i \cdot \overline{\sigma_{i-1}} \cdot \ldots \cdot \overline{\sigma_1}, \\ \overline{\sigma_n \ldots \sigma_1} = \overline{\sigma_n} \cdot \ldots \cdot \overline{\sigma_1} \end{cases} \quad \text{for } \sigma_1, \ldots, \sigma_n \in \Sigma, n \in \{0, 1, \ldots\}. \tag{4.4'''}$$

Fact 4.24. *Mapping ϕ is a polynomial simulation of a free monoid by a ring of polynomials.*

Proof. The formula (4.4″) proves that ϕ is compositional. Injectivity of ϕ follows from (4.4‴): the (unique) monomial of $\widetilde{w}$ of total degree i is divisible by $\widetilde{\sigma}$ if and only if the i-th letter of a word $w \in \Sigma^*$ is $\sigma \in \Sigma$ for $i = 1, \ldots, |w|$. □

Now we show that if the substitution operation is added as a basic operation, ϕ still is a polynomial simulation.

Lemma 4.25. *Mapping ϕ is a polynomial simulation of the enrichment of a free monoid with the substitution operation by the enrichment of a ring of polynomials with the substitution operation. More precisely,*

$$\phi(w[\sigma := w_\sigma, \sigma \in \Sigma]) = \phi(w)[\widetilde{\sigma} := \widetilde{w_\sigma}, \overline{\sigma} := \overline{w_\sigma}, \sigma \in \Sigma] \quad \textit{for } w, w_\sigma \in \Sigma^*, \sigma \in \Sigma. \tag{4.7}$$

Proof. For the simplicity of notation we rewrite the substitution expression $[\widetilde{\sigma} := \widetilde{w_\sigma}, \overline{\sigma} := \overline{w_\sigma}]$ as $[\phi(\sigma) := \phi(w_\sigma)]$[8] for $\sigma \in \Sigma$. In particular, we rewrite the formula (4.7) as

$$\phi(w[\sigma := w_\sigma, \sigma \in \Sigma]) = \phi(w)[\phi(\sigma) := \phi(w_\sigma), \sigma \in \Sigma] \text{ for } w, w_\sigma \in \Sigma^*, \sigma \in \Sigma.$$

The proof proceeds by induction. The desired formula (4.7) clearly holds in the base case when $w = a \in \Sigma$:

$$\phi(a[\sigma := w_\sigma, \sigma \in \Sigma]) = \phi(w_a) = \phi(a)[\phi(a) := \phi(w_a)] = \phi(a)[\phi(\sigma) := \phi(w_\sigma), \sigma \in \Sigma].$$

The inductive step is as follows: (in the below formula we assume that ϕ returns a 2×2 matrix as in (4.6))

$$\begin{aligned}
&\phi((a \cdot w)[\sigma := w_\sigma, \sigma \in \Sigma]) = \\
&\phi(a[\sigma := w_\sigma, \sigma \in \Sigma] \cdot w[\sigma := w_\sigma, \sigma \in \Sigma]) = \\
&\phi(a[\sigma := w_\sigma, \sigma \in \Sigma]) \cdot \phi(w[\sigma := w_\sigma, \sigma \in \Sigma]) = \\
&\phi(a)[\phi(\sigma) := \phi(w_\sigma), \sigma \in \Sigma] \cdot \phi(w)[\phi(\sigma) := \phi(w_\sigma), \sigma \in \Sigma] = \\
&(\phi(a) \cdot \phi(w))[\phi(\sigma) := \phi(w_\sigma), \sigma \in \Sigma] = \\
&\phi(a \cdot w)[\phi(\sigma) := \phi(w_\sigma), \sigma \in \Sigma].
\end{aligned}$$

This finishes the proof. □

Com-injective homomorphisms induce field automorphisms. In the next lemma we prove that, via our polynomial simulation from Definition 4.23, word homomorphisms that are com-injective, and only such, induce automorphisms of some field that contains a ring of polynomials.

This field is fixed for a given alphabet. To define it, we use an extension of a ring of polynomials in which the variables may have non-negative rational exponents. Such an extension is well-defined, as, in general, one can construct a ring of polynomials whose variables' exponents come from an arbitrary monoid: in the usual polynomials, it is the additive monoid of natural numbers $(\mathbb{N}, +)$, and in the below lemma, some variables use the monoid $(\mathbb{Q}_{\geq 0}, +)$ where $\mathbb{Q}_{\geq 0}$ denotes the set of non-negative rational numbers. We introduce the following notation: by $\mathbb{Z}[\widetilde{\Sigma}, \overline{\Sigma}^{\mathbb{Q}_{\geq 0}}]$ (resp. $\mathbb{Z}(\widetilde{\Sigma}, \overline{\Sigma}^{\mathbb{Q}_{\geq 0}})$) we denote the ring of polynomials (resp. the field of rational functions) with integer coefficients in which the variables from the set $\widetilde{\Sigma}$ have natural exponents (from the monoid $(\mathbb{N}, +)$), and the variables from the set $\overline{\Sigma}$ have non-negative rational exponents (from the monoid $(\mathbb{Q}_{\geq 0}, +)$).

Lemma 4.26. *Let Σ be an alphabet and $\tau = \{\sigma \mapsto w_\sigma \mid \sigma \in \Sigma\}$ be a word substitution. Consider the homomorphism from the ring $\mathbb{Z}[\widetilde{\Sigma} \cup \overline{\Sigma}]$ to itself defined by*

$$\mathbb{Z}[\widetilde{\Sigma} \cup \overline{\Sigma}] \ni f \mapsto f[\phi(\sigma) := \phi(w_\sigma), \sigma \in \Sigma]. \tag{4.8}$$

Then

[8] For this notation to be formally correct, the pair $\phi(\sigma)$ should consist of *variables*, not polynomials.

(i) if τ is com-injective, then the unique homomorphism from the field $\mathbb{Z}(\widetilde{\Sigma} \cup \overline{\Sigma}^{\mathbb{Q}_{\geq 0}})$ to itself that is an extension of (4.8) *is an automorphism,*

(ii) if τ is not com-injective, then the homomorphism (4.8) *is not injective; in consequence, there is no field that contains the ring $\mathbb{Z}[\widetilde{\Sigma} \cup \overline{\Sigma}^{\mathbb{Q}_{\geq 0}}]$ for which there is an extension of the homomorphism* (4.8) *that is an automorphism of that field.*

Proof. (i) First we give a proof for an example word substitution; then we give a proof in full generality.

Let $\Sigma = \{a\}$ and $\tau = \{a \mapsto aa\}$. Then the formula (4.8) defines the following homomorphism from the field $\mathbb{Z}(\widetilde{a}, \overline{a}^{\mathbb{Q}_{\geq 0}})$ to itself:

$$\mathbb{Z}(\widetilde{a}, \overline{a}^{\mathbb{Q}_{\geq 0}}) \ni f \mapsto f[(\widetilde{a}, \overline{a}) := (\widetilde{a} \cdot \overline{a} + \widetilde{a}, \overline{a}^2)]. \tag{4.8$'$}$$

We explicitly find the inverse homomorphism of (4.8′), thus proving it is an automorphism. Consider the system of equations in the field $\mathbb{Z}(\widetilde{a}', \overline{a}'^{\mathbb{Q}_{\geq 0}})$

$$\begin{cases} \widetilde{a}' = \widetilde{a} \cdot \overline{a} + \widetilde{a}, \\ \overline{a}' = \overline{a}^2 \end{cases} \tag{4.9}$$

in variables $\widetilde{a}, \overline{a}$ where $\widetilde{a}', \overline{a}'$ are constants. From the second equation we get $\overline{a} = \overline{a}'^{\frac{1}{2}}$. Substituting this to the first equation we get $\widetilde{a} = \widetilde{a}' \cdot \frac{1}{1+\overline{a}'^{\frac{1}{2}}}$. In consequence, the homomorphism from $\mathbb{Z}(\widetilde{a}, \overline{a}^{\mathbb{Q}_{\geq 0}})$ to itself defined by

$$\begin{cases} \widetilde{a} \mapsto \widetilde{a} \cdot \frac{1}{1+\overline{a}^{\frac{1}{2}}}, \\ \overline{a} \mapsto \overline{a}^{\frac{1}{2}} \end{cases}$$

is a left inverse of (4.8′). Moreover, it is a two-sided inverse, as the mappings from the non-primed variables to the primed variables are of kinds for which a one-sided inverse is always a two-sided inverse: the mapping of the barred variable is an exponentiation by a constant, and the mapping of the tilded variable is a multiplication by an expression that depends solely on the barred variable.

Now we perform the above reasoning in full generality, this time having no assumptions on Σ and τ. Let $\widetilde{\Sigma}' = \{\widetilde{\sigma}' \mid \sigma \in \Sigma\}, \overline{\Sigma}' = \{\overline{\sigma}' \mid \sigma \in \Sigma\}$ be copies of $\widetilde{\Sigma}$ and $\overline{\Sigma}$. Consider a system of equations in the field $\mathbb{Z}(\widetilde{\Sigma}', \overline{\Sigma}'^{\mathbb{Q}_{\geq 0}})$

$$\{\widetilde{\sigma}' = \widetilde{w_\sigma} \mid \sigma \in \Sigma\} \cup \{\overline{\sigma}' = \overline{w_\sigma} \mid \sigma \in \Sigma\}$$

in variable set $\widetilde{\Sigma} \cup \overline{\Sigma}$ where $\widetilde{\sigma}', \overline{\sigma}'$ are constants for $\sigma \in \Sigma$. It can be solved in the following way. First solve the subsystem of $|\Sigma|$ equations concerning only the barred variables: it can be seen as a system of linear equations when the exponents are treated as coefficients, and, in this view, it clearly has a solution because τ is com-injective. Substitute the obtained results to the remaining $|\Sigma|$ equations. Now observe that this remaining subsystem (that now concerns only tilded variables) is linear with coefficients in the field $\mathbb{Z}(\widetilde{\Sigma}' \cup \overline{\Sigma}'^{\mathbb{Q}_{\geq 0}})$. We prove it has a solution. First, observe that it has a solution if and only if the original subsystem, before substituting for the barred variables, had a solution when considered in variable set $\widetilde{\Sigma}$ and with coefficients from $\mathbb{Z}(\widetilde{\Sigma}' \cup \overline{\Sigma}^{\mathbb{Q}_{\geq 0}})$ – it is so because our mapping of $\overline{\sigma}$'s to the primed $\overline{\sigma}$'s is an invertible linear change of exponents and hence defines an isomorphism from the field $\mathbb{Z}(\overline{\Sigma}^{\mathbb{Q}_{\geq 0}})$ to $\mathbb{Z}(\overline{\Sigma}'^{\mathbb{Q}_{\geq 0}})$ (and modifying the coefficients of a linear system of equations by an isomorphism does not change the property of this system of having a solution or not). Second, observe that the original subsystem – again considered in variable set $\widetilde{\Sigma}$ and with coefficients from $\mathbb{Z}(\widetilde{\Sigma}' \cup \overline{\Sigma}^{\mathbb{Q}_{\geq 0}})$ – which is a linear system, would have a solution if one substituted $[\overline{\sigma} := 1, \sigma \in \Sigma]$ in the coefficients – this follows from the com-injectivity of τ. In consequence, the determinant of this subsystem, which is a polynomial in variable set $\overline{\Sigma}$, evaluates to a non-zero value at $(\overline{\sigma} = 1)_{\sigma \in \Sigma}$, and hence is a non-zero polynomial, and so this system has a solution.

The obtained solution of the whole system defines a left inverse of the homomorphism from the field $\mathbb{Z}(\widetilde{\Sigma} \cup \overline{\Sigma}^{\mathbb{Q}_{\geq 0}})$ to itself that extends (4.8). (Notice that this solution indeed defines a mapping from $\mathbb{Z}(\widetilde{\Sigma} \cup \overline{\Sigma}^{\mathbb{Q}_{\geq 0}})$ to itself, as every variable from $\overline{\Sigma}$ is mapped to a product of some rational powers of variables from $\overline{\Sigma}$.) This left inverse is a two-sided inverse, because it can be seen as a linear change of variables, and for every linear mapping a one-sided inverse is a two-sided inverse – indeed, the barred variables have linearly changed exponents, and the tilded variables are changed by a linear mapping with coefficients in $\mathbb{Z}(\overline{\Sigma}^{\mathbb{Q}_{\geq 0}})$. This finishes the proof.

(ii) Let u, v be different *commutative* words that are mapped by the homomorphism τ to the same commutative word, denote it s, i.e.

$$u[\sigma := w_\sigma, \sigma \in \Sigma] = v[\sigma := w_\sigma, \sigma \in \Sigma] = s.$$

Then both polynomials $\overline{u}$ and $\overline{v}$ (which are necessarily different) are mapped by the homomorphism (4.8) to the same polynomial of $\mathbb{Z}[\widetilde{\Sigma} \cup \overline{\Sigma}]$, namely $\overline{s}$. In consequence, this homomorphism is not injective.

□

References. Lemma 4.22 is due to Worrell et al.[9].

4.4 Summary

In this chapter, we discuss register transducers with output in a ring of polynomials enriched with the substitution operation.

[9] M. Bojańczyk, Kiefer, et al., 2019.

First we show that this model has undecidable zeroness (Theorem 9). The meta-corollary of this fact is the following: we still do not know if register transducers with output in a free monoid enriched with the substitution operation have decidable functionality or not, but we know that, in contrast to the case of register transducers *without* substitution, the reduction via our polynomial simulation of words with polynomials is a reduction to an *undecidable* problem.

Then we prove two positive results, each using a restricted variant of this model (Theorem 10 and Theorem 11); we also present an example application for each of them (Example 4.8 and Corollary 4.9; Example 4.21). Theorem 10 describes grammars with languages of form $L(G)(L(H))$, where G, H are polynomial grammars with output in a ring of polynomials with coefficients in a ring with no zero divisors (like $\mathbb{Z}$ or $\mathbb{Z}[x_1, \ldots, x_n]$), and Theorem 11 describes register transducers with output in a free monoid whose register updates can use word homomorphisms (which are instances of the substitution operation), however with two restrictions: the homomorphisms must satisfy a certain, easily verifiable, condition, which we call com-injectivity, and they can only be applied simultaneously to all registers. Both results do not give a satisfactory answer to the question "how much substitution can be added to register transducers with output in a free monoid for the functionality and equivalence of functional register transducers to remain decidable", but hopefully they are a step towards an answer.

References. This chapter is based on an author's unpublished paper[10]. The references of both Section 4.1 and Section 4.3 are included at their ends.

[10] Schmude, 2021.

Chapter 5

Output in finitely presented monoids

In this chapter, we consider register transducers with output in a finitely presented monoid[1]. We focus on finitely presented monoids whose relations merely permute the letters of a word, which we call trace-like monoids. Trace-like monoids are motivated by trace monoids, which are a fundamental concept in concurrency theory. Intuitively, a trace monoid consists of actions on some (implicit) set some pairs of which can be executed in any order (this can happen for example when the actions act on disjoint sets of resources). For more information on trace monoids we refer to a survey of Diekert and Métivier[2].

Finitely presented monoids arise in computer science in a similar way, as the monoids of actions on a data structure. For example, Huschenbett, Kuske, and Zetzsche consider the monoid of queue actions; they represent it as a finitely presented monoid, which happens to be trace-like[3]. For illustration, consider a queue over alphabet Σ and two different symbols $a, b \in \Sigma$, and take two sequences w_1, w_2 of queue actions, which are words over alphabet $\Gamma := \{\text{push}_\sigma \mid \sigma \in \Sigma\} \cup \{\text{pop}_\sigma \mid \sigma \in \Sigma\}$, defined by $w_1 := \text{push}_a \text{push}_b \text{pop}_b$ and $w_2 := \text{push}_a \text{pop}_b \text{push}_b$. They give the same effect on every queue, and in consequence represent the same queue action (notice that words $w'_1 := \text{push}_b \text{pop}_b$ and $w'_2 := \text{pop}_b \text{push}_b$ do not represent the same queue action, as only the second one gets stuck on the empty queue); notice that w_2 is a permutation of w_1.

We are interested in decidability of functionality and equivalence of functional register transducers with output in a finitely presented monoid (which are inter-reducible due to Lemma 2.9). For a trace monoid the solution is simple: embed it into a product of free monoids[4], thus reducing to the free

[1] Let us give two remarks. First, formally we consider register transducers with output in a free monoid but take the semantics in this monoid's quotient (although both choices are formally correct). Second, an analogous model, but with one register and for monoids that are embeddable in a group, was considered by Zakharov, for which he proved decidability of equivalence (Zakharov, 2015).

[2] Diekert and Métivier, 1997.

[3] Huschenbett, Kuske, and Zetzsche, 2014, Theorem 4.1, see also Lemma 3.5.

[4] Such an embedding exists for every trace monoid, see e.g. Diekert and Métivier, 1997, Corollary 2.2.

monoid case, which is decidable[5]. For an arbitrary monoid, we consider a proof scheme, to which we refer as "this chapter's proof scheme" (Lemma 5.7), which roughly speaking is as follows: apply our polynomial simulation of a free monoid by a ring of polynomials (Definition 4.23) to the output monoid, thus obtaining a candidate for a polynomial simulation of the output monoid by a quotient of a ring of polynomials, and try to prove that it is injective. The proof attempt of injectivity is done by an exhaustive search for a minimal counter-example, and uses Gröbner bases for computations in the target quotient ring. Additionally, one must prove that the target ring is embeddable into a product of rings with no zero divisors – this property of a ring is decidable, by a rather standard argument (Fact 5.4 and Corollary 5.6).

We apply the above proof scheme to the monoid $\langle a, b, c \mid abc = cba \rangle$ and obtain a decidability result (Corollary 5.11). We also apply it to the monoid of queue actions, but without success (Corollary 5.13). Finally, we analyze this proof scheme in terms of its automation (in the general case).

As a byproduct of this chapter's proof scheme, we obtain a criterion for *equational Noetherianity* of a finitely presented monoid[6].

5.1 Monoids

A *monoid* is a set with an associative binary operation. A *finitely presented monoid* is given by a *finite presentation*, which consists of a finite set Σ of *generators* and a finite set $S \subseteq \Sigma^* \times \Sigma^*$ of *relations*; a relation $(u, v) \in S$ is written as "$u = v$". The presentation with generators Σ and relations S defines the monoid $\langle \Sigma \mid S \rangle$ whose universe is $\Sigma^* / \sim_S$ where $\sim_S \subseteq \Sigma^* \times \Sigma^*$ is the smallest equivalence relation on Σ^* such that $u \sim_S v$ implies $sut \sim_S svt$ for $s, t \in \Sigma^*$, and the multiplication operation is defined by

$$[w_1]_{\sim_S} \cdot [w_2]_{\sim_S} := [w_1 \cdot w_2]_{\sim_S} \quad \text{for } w_1, w_2 \in \Sigma^*$$

where $[w]_{\sim_S}$ denotes the $\sim_S$-equivalence class of an element $w \in \Sigma^*$. In this chapter, we assume all monoids to be given by a finite presentation.

We call a finitely presented monoid $\langle \Sigma \mid S \rangle$ *trace-like* if for every "$u = v$" $\in S$, the word v is a permutation of u.

Let $\mathbb{M}$ be a finitely presented monoid with generating set Σ. By taking the semantics of register transducers with output monoid Σ^* to be in $\mathbb{M}$, one obtains the following variants of the problems of functionality and equivalence of functional register transducers.

Name: $\mathbb{M}$-functionality of register transducers
Parameter: $\mathbb{M} = \langle \Sigma \mid S \rangle$ - a finitely presented monoid

[5] Seidl, Maneth, and Kemper, 2018; they prove this result for a slightly weaker model, but the proof can be extended to all register transducers with output in a free monoid.

[6] We denote this property by equational Noetherianity following Shevlyakov (e.g. Shevlyakov, 2016). Nevertheless, it is often called *compactness property* (see e.g. Harju, Karhumäki, and Plandowski, 1995).

Input: T - a register transducer with output in the monoid Σ^*
Question: Is it the case that

for every input tree t all outputs of T on t are $\sim_S$-equivalent?

We denote the $\sim_S$-equivalence class of elements output by an $\mathbb{M}$-functional register transducer T on an input tree t by $[T(t)]_{\sim_S}$.

Name: $\mathbb{M}$-equivalence of $\mathbb{M}$-functional register transducers
Parameter: $\mathbb{M} = \langle \Sigma \mid S \rangle$- a finitely presented monoid
Input: T_1, T_2 - register transducers with output in the monoid Σ^* with the same input alphabet
Question: Is it the case that

$$[T_1(t)]_{\sim_S} = [T_2(t)]_{\sim_S} \text{ for every input tree } t?$$

Also, one obtains the following variant of our word-to-polynomial simulation (Definition 4.23).

Definition 5.1 ($\mathbb{M}$-to-$R_{\mathbb{M}}$ mapping)**.** Let $\mathbb{M} = \langle \Sigma \mid S \rangle$. Recall that our word-to-polynomial simulation ϕ is defined (in (4.4)) by

$$\begin{aligned}
&\phi = (\phi_1, \phi_2) : \Sigma^* \to \mathbb{Z}[\widetilde{\Sigma} \cup \overline{\Sigma}] \times \mathbb{Z}[\overline{\Sigma}],\\
&\phi(\sigma) := (\widetilde{\sigma}, \overline{\sigma}) \quad \text{ for } \sigma \in \Sigma,\\
&\phi(\sigma w) := (\widetilde{\sigma} \cdot \phi_2(w) + \phi_1(w), \overline{\sigma} \cdot \phi_2(w)) \quad \text{ for } \sigma \in \Sigma, w \in \Sigma^*.
\end{aligned}$$

Let us also recall that ϕ can be equivalently seen as the following monoid homomorphism (cf. (4.6))[7].

$$\begin{aligned}
\phi : \Sigma^* &\to \mathbb{M}_{2\times 2}\left(\mathbb{Q}[\widetilde{\Sigma} \cup \overline{\Sigma}]\right),\\
w &\overset{\phi}{\mapsto} \begin{bmatrix} \overline{w} & 0 \\ \widetilde{w} & 1 \end{bmatrix} \quad \text{ for } w \in \Sigma^*.
\end{aligned}$$

We refer to the above form of ϕ as the *matrix form*.

For a ring R, denote by $\mathbb{LT}_{n\times m}(R)$ the ring of lower-triangular matrices with entries from R. By quotienting the mapping ϕ by $\sim_S$, we naturally obtain the following mapping of $\mathbb{M}$:

$$\phi_{\mathbb{M}} : \mathbb{M} \to \mathbb{LT}_{2\times 2}\left(\mathbb{Q}[\widetilde{\Sigma} \cup \overline{\Sigma}]\right) / \langle \phi(w_1) - \phi(w_2) \mid w_1 \sim_S w_2,\ w_1, w_2 \in \Sigma^* \rangle_{\mathbb{LT}_{2\times 2}(\mathbb{Q}[\widetilde{\Sigma} \cup \overline{\Sigma}])}, \tag{5.1}$$
$$\phi_{\mathbb{M}} : [w] \mapsto [\phi(w)] \quad \text{ for } w \in \Sigma^*.$$

Notice that we consider a quotient of a *lower-triangular* matrix ring. Let us describe it explicitly by computing the defining ideal.

[7] Formally, (4.6) defines the codomain of our word-to-polynomial simulation to be a ring of polynomials with coefficients in the ring $\mathbb{Z}$, and here we use the field $\mathbb{Q}$. We extend to $\mathbb{Q}$ to manage the ideals obtained by the quotient construction.

Claim 5.2 ($\mathbb{M}$-to-$R_{\mathbb{M}}$ mapping for a trace-like monoid $\mathbb{M}$)**.** *Assume that $\mathbb{M}$ is trace-like. Then $\phi_{\mathbb{M}}$, when the matrix form is dropped, is as follows*

$$\begin{aligned} &\phi_{\mathbb{M}} : \mathbb{M} \to \mathbb{Q}[\widetilde{\Sigma} \cup \overline{\Sigma}] / I_{\mathbb{M}} \times \mathbb{Z}[\overline{\Sigma}], \\ &\phi_{\mathbb{M}}([w]) := ([\widetilde{w}], \overline{w}) \quad \textit{for } w \in \Sigma^* \end{aligned} \tag{5.2}$$

where

$$I_{\mathbb{M}} := \langle \widetilde{u} - \widetilde{v} \mid (u, v) \in S \rangle. \tag{5.3}$$

Proof. Observe that

$$\phi(w_1) - \phi(w_2) = \begin{bmatrix} 0 & 0 \\ \widetilde{w_1} - \widetilde{w_2} & 0 \end{bmatrix} \quad \text{for } w_1 {\sim_S} w_2,\ w_1, w_2 \in \Sigma^*.$$

Also,

$$\begin{bmatrix} p_1 & 0 \\ p_2 & p_3 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 \\ \widetilde{w_1} - \widetilde{w_2} & 0 \end{bmatrix} \cdot \begin{bmatrix} p'_1 & 0 \\ p'_2 & p'_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ p_1 \cdot p'_1 \cdot (\widetilde{w_1} - \widetilde{w_2}) & 0 \end{bmatrix},$$

for $w_1, w_2 \in \Sigma^*$, $p_i, p'_i \in \mathbb{Q}[\widetilde{\Sigma} \cup \overline{\Sigma}]$ for $i \in \{1, 2, 3\}$ hence the defining ideal (of $\mathbb{LT}_{2\times 2}\left(\mathbb{Q}[\widetilde{\Sigma} \cup \overline{\Sigma}]\right)$) is generated by

$$\left\{ \begin{bmatrix} 0 & 0 \\ p & 0 \end{bmatrix} \mid p \in I_{\mathbb{M}} \right\}$$

where the ideal $I_{\mathbb{M}} \subseteq \mathbb{Q}[\widetilde{\Sigma} \cup \overline{\Sigma}]$ is defined by

$$I_{\mathbb{M}} := \langle \widetilde{w_1} - \widetilde{w_2} \mid w_1 {\sim_S} w_2,\ w_1, w_2 \in \Sigma^* \rangle_{\mathbb{Q}[\widetilde{\Sigma} \cup \overline{\Sigma}]}. \tag{5.4}$$

This proves (5.2) provided the above-defined $I_{\mathbb{M}}$ is equal to the one in (5.3); it indeed is and we prove it now. By unfolding the definition of $\sim_S$ we obtain that

$$I_{\mathbb{M}} = \langle \widetilde{sut} - \widetilde{svt} \mid (u, v) \in S, s, t \in \Sigma^* \rangle,$$

and since

$$\begin{aligned} \widetilde{sut} - \widetilde{svt} &= (\widetilde{s}\overline{ut} - \widetilde{s}\overline{vt}) + (\widetilde{u}\overline{t} - \widetilde{v}\overline{t}) + (\overline{t} - \overline{t}) = \\ &= \widetilde{s}\widetilde{t}(\overline{u} - \overline{v}) + (\widetilde{u} - \widetilde{v})\overline{t} = \\ &= (\widetilde{u} - \widetilde{v})\overline{t} \quad \text{for } (u, v) \in S \text{ for } s, t \in \Sigma^*, \end{aligned}$$

hence

$$\begin{aligned} I_{\mathbb{M}} &= \langle (\widetilde{u} - \widetilde{v})\overline{t} \mid t \in \Sigma^* \rangle = \\ &= \langle \widetilde{u} - \widetilde{v} \mid (u, v) \in S \rangle. \end{aligned}$$

□

A similar claim holds for an arbitrary finitely presented monoid. The proof is analogous and we omit it.

Claim (M-to-$R_{\mathbb{M}}$ mapping for a monoid $\mathbb{M}$)**.** *Let $\mathbb{M} = \langle \Sigma \mid S \rangle$. When the matrix form is dropped, $\phi_{\mathbb{M}}$ is as follows*

$$\begin{aligned} &\phi_{\mathbb{M}} : \mathbb{M} \to \mathbb{Q}[\widetilde{\Sigma} \cup \overline{\Sigma}]/I_{\mathbb{M}} \times \mathbb{Z}[\overline{\Sigma}]/J_{\mathbb{M}}, \\ &\phi_{\mathbb{M}}([w]) := ([\widetilde{w}], [\overline{w}]) \quad \textit{for } w \in \Sigma^* \end{aligned}$$

where

$$\begin{aligned} &I_{\mathbb{M}} := \langle \{\widetilde{u} - \widetilde{v} \mid (u,v) \in S\} \cup \{\overline{u} - \overline{v} \mid (u,v) \in S\} \rangle, \textit{ and} \\ &J_{\mathbb{M}} := \langle \overline{u} - \overline{v} \mid (u,v) \in S \rangle. \end{aligned}$$

Now we observe that the mapping $\phi_{\mathbb{M}}$ is compositional.

Fact 5.3. *Let $\mathbb{M}$ be a finitely presented monoid. Then the mapping $\phi_{\mathbb{M}}$ is compositional.*

Proof. It follows immediately from compositionality of ϕ. We omit the details of the proof. □

Let us now proceed to the formulation of this chapter's main lemma, Lemma 5.7. It is based on the following observation. Let $\mathbb{M}$ be a finitely presented monoid. If the mapping $\phi_{\mathbb{M}}$ is injective, it yields a polynomial simulation of $\mathbb{M}$ by a ring, namely $R_{\mathbb{M}}$. If additionally the ring $R_{\mathbb{M}}$ can be embedded into a product of rings with no zero divisors, then we are done – the mapping $\phi_{\mathbb{M}}$ yields a reduction like in Corollary 2.27.

Reduced rings. Now we show that rings that are embeddable into a product of rings with no zero divisors are exactly reduced rings, then show that one can decide if a given ring is reduced, and finally state and prove Lemma 5.7.

A ring R is *reduced* if $r^m = 0$ implies $r = 0$ for all $r \in R$ and $m \in \{1, 2, \ldots\}$.

Fact 5.4. *Let X be a variable set and K be a (computable) field. For an ideal $I \subseteq K[X]$, the quotient ring $K[X]/I$ can be (effectively) embedded into a product of (computable) rings with no zero divisors if and only if it is reduced.*

Proof. The right-to-left implication follows from Lemma 4.12. The left-to-right implication is straightforward from the definition. □

The property of a ring being reduced can be decided due to the following fact.

Fact 5.5. *Let X be a variable set and K be a field. For an ideal $I \subseteq K[X]$, where X is a variable set and K is a field (e.g. $\mathbb{Q}$), the quotient ring $K[X]/I$ is reduced if and only if the ideal I is radical.*

Corollary 5.6. *Let X be a variable set and K be a computable field. One can decide if the ring $K[X]/I$ is reduced.*

Proof. The ring $K[X]/I$ is reduced if and only if the ideal I is radical; it is known that the latter can be decided[8]. □

We conclude with the below lemma.

[8] Cox, Little, and O'Shea, 2015, Remarks at the end of §2 of Chapter 4.

Lemma 5.7 (This chapter's proof scheme)**.** *Let $\mathbb{M} = \langle \Sigma \mid S \rangle$ be a finitely presented monoid. If the mapping $\phi_{\mathbb{M}}$ is injective and the ring $R_{\mathbb{M}}$ is reduced, then $\mathbb{M}$-functionality and $\mathbb{M}$-equivalence of $\mathbb{M}$-functional register transducers with output in the monoid Σ^* are decidable.*

Proof. By applying Fact 5.4, the mapping $\phi_{\mathbb{M}}$ is a polynomial simulation by a product of rings with no zero divisors. The claim follows from Corollary 2.27. □

5.2 A proof for an example monoid

In this section, we apply this chapter's proof scheme (Lemma 5.7) to an example finitely presented monoid. We choose $\mathbb{M} = \langle \Sigma \mid S \rangle$ with generators $\Sigma = \{a, b, c\}$ and a singleton set of relations $S = \{abc = cba\}$[9]. We obtain

$$I_{\mathbb{M}} = \langle (\widetilde{a}\overline{b}\overline{c} - \widetilde{c}\overline{b}\overline{a}) + (\widetilde{b}\overline{c} - \widetilde{b}\overline{a}) + (\widetilde{c} - \widetilde{a}) \rangle.$$

We order the variables as follows:

$$\widetilde{a} > \widetilde{b} > \widetilde{c} > \overline{a} > \overline{b} > \overline{c}.$$

The generating set of $I_{\mathbb{M}}$ is of size one, hence it is a Gröbner basis of $I_{\mathbb{M}}$; the leading monomial of the generator of $I_{\mathbb{M}}$ is $\widetilde{a}\overline{b}\overline{c}$. The ideal $I_{\mathbb{M}}$ is radical, due to the following fact.

Fact 5.8. *Let I be a polynomial ideal. If the leading monomials of all polynomials of some Gröbner basis of I are square-free (a monomial is* square-free *if the exponent of each of its variable is 1), then I is a radical ideal.*

Proof. Let G be a Gröbner basis of I whose all polynomials have square-free leading monomials. By contradiction, assume there is a polynomial f and a natural number $n > 1$ such that $f^n \in I$ but $f \notin I$. Let r be the remainder of division of f by G; we have that $r^n \in I$ and $r \notin I$. This means that the leading monomial of r^n is divisible by the leading monomial of some polynomial from G, but r is not – a contradiction with the assumption that the leading monomials of polynomials from G are square-free. □

Corollary 5.9. *The ring $R_{\mathbb{M}}$ is reduced.*

Proof. $I_{\mathbb{M}}$ is radical due to Fact 5.8. In consequence, $R_{\mathbb{M}}$ is reduced (cf. Fact 5.5). □

Now it remains to prove that $\phi_{\mathbb{M}}$ is injective.

Lemma 5.10. *$\phi_{\mathbb{M}}$ is injective.*

[9] We choose such $\mathbb{M}$ because it is the simplest cancellative trace-like monoid that is not a trace monoid (a monoid is right/left cancellative if $xz = yz/zx = zy$ implies $x = y$ for all its elements x, y, z, and is cancellative if it is both left and right cancellative) – it is straightforward that this chapter's proof scheme surely fails on non-left-cancellative monoids, and it seems to surely fail on non-right-cancellative ones.

Proof. By contradiction, let $(w,v) \in \Sigma^* \times \Sigma^*$ be a minimal counterexample to injectivity of $\phi_{\mathbb{M}}$, i.e. $w \not\sim_S v$, $\widetilde{w} = \widetilde{v}, \overline{w} = \overline{v}$, and the total length of w and v is smallest possible; we also assume that both w and v do not contain a substring abc, as it could be replaced with cba without the change of $\mathbb{M}$-equivalence class (for a word w at most $|w|^2$ replacements are needed). Let us consider cases regarding the first letters of w and v (and second, third letters etc. if necessary). For words $w', v' \in \Sigma^*$, by writing $w = w' \ldots, v = v' \ldots$ we mean that w has a prefix w' and v has a prefix v'. We denote the suffix of w (resp. v) that skips the first i letters by w_i (resp. v_i). Finally, if we denote a polynomial by r_i, for $i = 1, 2, \ldots, |w|$, then it is a monomial in variable set $\overline{\Sigma}$ of degree $|w| - i$.

We omit the cases where w and v start with the same letter, as any such case contradicts the minimality of (w,v): if $w = \sigma w_1$ and $v = \sigma v_1$ for some $\sigma \in \Sigma$, then $\overline{w_1} = \overline{w}/\overline{\sigma} = \overline{v}/\overline{\sigma} = \overline{v_1}$ and $\widetilde{w_1} = \widetilde{w} - \widetilde{\sigma}\overline{w_1} = \widetilde{v} - \sigma\overline{v_1} = \widetilde{v_1}$. Each of the remaining cases follows the following scheme. The polynomial $f = \widetilde{w} - \widetilde{v}$ is non-zero and represent the zero element in $R_{\mathbb{M}}$; equivalently, the remainder of the division of f by $G = \{(\widetilde{a}\overline{b}\overline{c} - \widetilde{c}\overline{b}\overline{a}) + (\widetilde{b}\overline{c} - \widetilde{b}\overline{a}) + (\widetilde{c} - \widetilde{a})\}$ is 0. In consequence, after any sequence of division steps $f = f_0 \to_G f_1 \to_G f_2 \to_G \ldots \to_G f_l$ that has not yet reached 0 (including the empty sequence, for which $l = 0$) it is possible to apply a division step to the leading monomial of f_l. We apply consecutive division steps in this way, and in each of the cases we reach either a smaller counter-example or a contradiction.

Case 1° Neither of w, v starts with a. In such case, the leading monomial of $\widetilde{w} - \widetilde{v}$ is not divisible by $\widetilde{a}\overline{b}\overline{c}$, and hence it is not possible to apply a division step to it. A contradiction.

Case 2° $w = a \ldots, v = \sigma \ldots$ for some $\sigma \in \{b, c\}$. In such case, using the divisibility of $\mathrm{LM}(\widetilde{w})$ by $\widetilde{a}\overline{b}\overline{c}$, we get that

$$f = \widetilde{w} - \widetilde{v} = \underbrace{\widetilde{a}\overline{b}\overline{c}r_3 - \widetilde{\sigma}r_1}_{\text{largest degree}} + (\widetilde{w_1} - \widetilde{v_1}) \text{ for some monomials } r_1, r_3 \in \mathbb{Z}[\overline{\Sigma}].$$

By applying a division step to $\widetilde{a}\overline{b}\overline{c}r_3$ we obtain

$$f \to_G f_1 = \underbrace{\widetilde{c}\overline{b}\overline{a}r_3 - \widetilde{\sigma}r_1}_{\text{largest degree or 0}} + \text{smaller degree monomials.}$$

If σ was b, then the polynomial $\widetilde{c}\overline{b}\overline{a}r_3 - \widetilde{\sigma}r_1$ could not be reduced by division steps by G to a smaller degree polynomial; in consequence, $\widetilde{c}\overline{b}\overline{a}r_3 - \widetilde{\sigma}r_1$ is equal to 0, and hence $\sigma = c$.

Case 2°(updated) $w = a \ldots, v = c \ldots$. Observe we have

$$f = \widetilde{w} - \widetilde{v} \to_G f_1 = \underbrace{(\widetilde{b}\overline{a} - \widetilde{c}\overline{a})r_3 + \mathrm{LM}(\widetilde{w_1}) - \mathrm{LM}(\widetilde{v_1})}_{\text{largest degree or 0}} + (\widetilde{a} - \widetilde{c})r_3 + (\widetilde{w_2} - \widetilde{v_2}).$$

The monomials $\widetilde{b}\overline{a}r_3$ and $\widetilde{b}\overline{c}r_3$ neither are divisible by $\widetilde{a}\overline{b}\overline{c}$ nor can be reduced to 0 by division steps by G of the remaining monomials of the largest degree (as the largest degree monomials of the remainders of such division steps are not divisible by $\widetilde{b}$), hence they both necessarily are canceled-out by $\mathrm{LM}(\widetilde{w_1})$ and $\mathrm{LM}(\widetilde{v_1})$. In consequence, the second letter of both w and v is b.

Case 2°(second update) $w = ab \ldots . v = cb \ldots$,

$$f = \widetilde{w} - \widetilde{v} \to_G f_1 = \underbrace{(\widetilde{a} - \widetilde{c})r_3 + \mathrm{LM}(\widetilde{w_2}) - \mathrm{LM}(\widetilde{v_2})}_{\text{largest degree or 0}} + (\widetilde{w_3} - \widetilde{v_3}).$$

By reasoning analogously *ad infinitum*, taking into account that neither w nor v contains a subword abc, one can show that w and v begin with, respectively, $(ab)(ab)^n$ and $(cb)(cb)^k(ab)^{n-k}$ for *every* $n \in \{1, 2, \ldots\}$ for some $k \in \{0, 1, \ldots, n\}$ – a contradiction with finiteness of w and v. □

Corollary 5.11. *For $\mathbb{M} = \langle a, b, c \mid abc = cba \rangle$, the problems of $\mathbb{M}$-functionality and $\mathbb{M}$-equivalence of $\mathbb{M}$-functional register transducers with output in the monoid $\{a, b, c\}^*$ are decidable.*

5.3 A proof attempt for the monoid of queue actions

Let Σ be a finite alphabet. Huschenbett, Kuske, and Zetzsche[10] showed that the monoid of queue actions over alphabet Σ is isomorphic to $\langle \Gamma \mid S \rangle$ where $\Gamma := \{\mathrm{push}_\sigma \mid \sigma \in \Sigma\} \cup \{\mathrm{pop}_\sigma \mid \sigma \in \Sigma\}$ and

$$\begin{aligned} S := &\{\mathrm{push}_a \mathrm{push}_b \mathrm{pop}_b = \mathrm{push}_a \mathrm{pop}_b \mathrm{push}_b \mid a, b \in \Gamma\} \cup \\ &\{\mathrm{push}_a \mathrm{pop}_a \mathrm{pop}_b = \mathrm{pop}_a \mathrm{push}_a \mathrm{pop}_b \mid a, b \in \Gamma\} \cup \\ &\{\mathrm{push}_a \mathrm{pop}_b = \mathrm{pop}_b \mathrm{push}_a \mid a, b \in \Gamma, a \neq b\}. \end{aligned}$$

Notice it is a trace-like monoid. Unfortunately, this chapter's proof method (Lemma 5.7) does not succeed on it, due to the following fact.

Fact 5.12. *Let $\mathbb{M} = \langle \Sigma \mid S \rangle$ be a finitely presented monoid. If $\mathbb{M}$ is not left-cancellative (i.e. $mm_1 = mm_2$ but $m_1 \neq m_2$ for some $m, m_1, m_2 \in \mathbb{M}$.), then $\phi_{\mathbb{M}}$ is not injective.*

Proof. Let $x, y, z \in \Sigma^*$ be representatives of a counter-example to left-cancellativity of $\mathbb{M}$, i.e. $[x]_{\mathbb{M}} \cdot [y]_{\mathbb{M}} = [x]_{\mathbb{M}} \cdot [z]_{\mathbb{M}}$ but $[y]_{\mathbb{M}} \neq [z]_{\mathbb{M}}$. Then $\phi_{\mathbb{M}}([y]_{\mathbb{M}}) = \phi_{\mathbb{M}}([z]_{\mathbb{M}})$, which contradicts injectivity of $\phi_{\mathbb{M}}$. Indeed, using the equality $\phi_{\mathbb{M}}(xz) = \phi_{\mathbb{M}}(yz)$ we obtain the following equalities *in the ring $R_{\mathbb{M}}$*: $\overline{y} = \overline{xy}/\overline{x} = \overline{xz}/\overline{x} = \overline{z}$, and $\widetilde{y} = \widetilde{xy} - \widetilde{xy} = \widetilde{xz} - \widetilde{xz} = \widetilde{z}$. □

Corollary 5.13. *Let $\mathbb{M}$ be the monoid of queue actions. Then $\phi_{\mathbb{M}}$ is not injective.*

Proof. Due to Fact 5.12, it suffices to show that $\mathbb{M}$ is not left-cancellative. Indeed, observe that for a queue symbol σ we have $\mathrm{push}_\sigma \mathrm{push}_\sigma \mathrm{pop}_\sigma = \mathrm{push}_\sigma \mathrm{pop}_\sigma \mathrm{push}_\sigma$, but $\mathrm{push}_\sigma \mathrm{pop}_\sigma \neq \mathrm{pop}_\sigma \mathrm{push}_\sigma$, as only the second sequence of queue actions gets stuck on the empty queue. □

[10] Huschenbett, Kuske, and Zetzsche, 2014, Theorem 4.1, see also Lemma 3.5.

5.4 Can this chapter's proof scheme be automated?

In the case when $\mathbb{M}$ is an arbitrary monoid, a proof analogous to the one from Section 5.2 is unlikely to be approachable by hand-calculations (e.g. due to the necessarily large size of a Gröbner basis of ideal $I_{\mathbb{M}}$). This raises the necessity to automate at least some parts of the proof. Let us describe this proof in general terms and see which parts can be automated.

Let $\mathbb{M} = \langle \Sigma \mid S \rangle$ be a finitely presented monoid. First we need a unique representative of every $\sim_S$-equivalence class. It can be defined using string rewriting systems.

Definition 5.14. A *string rewriting system (over alphabet* Σ) is a finite set $\mathcal{S} \subseteq \Sigma^* \times \Sigma^*$. It defines a relation of *one rewriting step* $\rightarrow_{\mathcal{S}} \subseteq \Sigma^* \times \Sigma^*$ by

$$w_1 u w_2 \rightarrow_{\mathcal{S}} w_1 v w_2 \quad \text{for } (u,v) \in S \text{ for } w_1, w_2 \in \Sigma^*.$$

By $\rightarrow_{\mathcal{S}}^*$ we denote the reflexive transitive closure of $\rightarrow_{\mathcal{S}}$, and by $\leftrightarrow_{\mathcal{S}}^*$ we denote the smallest equivalence relation that contains $\rightarrow_{\mathcal{S}}$.

A string rewriting system $\mathcal{S}$ is *complete* if for every word w every sequence of rewriting steps $w = w_0 \rightarrow_{\mathcal{S}} w_1 \rightarrow_{\mathcal{S}} \ldots$ ultimately ends, and moreover every such sequences ends in the same word, called the *$\mathcal{S}$-normal form of w*.

Fact 5.15. *Let $\mathcal{S}$ be a complete string rewriting system. Then if $w_1 \leftrightarrow_{\mathcal{S}}^* w_2$, then w_1 and w_2 have the same $\mathcal{S}$-normal form.*

Proof. The proof is standard and we omit it. □

Due to the above fact, we define the unique representative of a $\sim_S$-equivalence class of some word to be its $\mathcal{S}$-normal forms for some complete string rewriting system $\mathcal{S}$ that contains S (up to reversing the order of words in some relations). Observe that a word is an $\mathcal{S}$-normal form of some word if and only if it does not contain a substring from $\{u \mid (u,v) \in \mathcal{S}\}$. Hence, if $\mathcal{S}$ is finite, then the language of normal forms is regular and can be obtained effectively. However, it is not clear how to effectively complete a string rewriting system in general. In some cases this task can be performed by the Knuth-Bendixson completion algorithm, which is a word analogue of the Buchberger's algorithm. If it terminates on the set of relations S, it returns such a finite system.

If we manage to effectively obtain the language of unique representatives, there are further steps that can be done automatically: we can check if $R_{\mathbb{M}}$ is reduced, compute a Gröbner basis of the ideal $I_{\mathbb{M}}$, and start performing the steps of the proof, i.e. take an arbitrary minimal counter-example pair (w, v), consider cases regarding prefixes of (w, v), perform division steps on $f := \widetilde{w} - \widetilde{v}$, while analyzing the largest degree monomials of the current value of f. Despite the fact that each proof step can be performed automatically and the branches of the proof seem to always lead to either a counter-example or some cyclical behavior of both the polynomial f and the prefixes of w and v, the proof is not fully automated yet, as it is not clear under what conditions the proof branches should be terminated.

5.5 Equational Noetherianity

In this section, we show how this chapter's proof scheme gives a criterion for equational Noetherianity of a finitely presented monoid. We begin by defining this notion.

Two systems of equations (in an arbitrary algebra, for example a monoid or a ring) are *equivalent* if they have the same sets of solutions. A monoid is *equationally Noetherian*[11] if every infinite system of equations in this monoid admits a finite equivalent subsystem. Equational Noetherianity has been used in formal language theory. For example, a proof of decidability of equivalence of HDT0L sequences uses the fact that a free monoid is equationally Noetherian[12]. For more information and references on equationally Noetherian monoids we refer to lecture notes of Shevlyakov[13], in particular, to the references made in Section 7.1.

Example 5.16. Every free monoid is equationally Noetherian[14]. For example, the system of equations $\mathcal{S} := \{x^n(ab)^n = (ab)^n x^n \mid n \text{ is a prime number}\}$ in the monoid $\{a, b\}^*$, i.e.

$$\begin{aligned} x^2(ab)^2 &= (ab)^2x^2 \\ x^3(ab)^3 &= (ab)^3x^3 \\ x^5(ab)^5 &= (ab)^5x^5 \\ &\vdots \end{aligned}$$

admits a finite equivalent subsystem $\mathcal{S}' := \{x^2(ab)^2 = (ab)^2x^2\}$; the set of solutions of both $\mathcal{S}$ and $\mathcal{S}'$ is $(ab)^*$.

Example 5.17 (Bicyclic monoid)**.** The finitely presented monoid $\mathbb{M} = \langle a, b \mid ab = \varepsilon\rangle$, called the *bicyclic monoid*, is not equationally Noetherian. Indeed, Lothaire shows[15] that the system of equations $\mathcal{S} := \{x_1^n x_2^n x_3 = x_3 \mid n \in \{1, 2, \dots\}\}$ does not admit a finite equivalent subsystem.

Now we state and prove the main result of this section.

Theorem 12 (Criterion for equational Noetherianity)**.** *Let $\mathbb{M}$ be a finitely presented monoid. If $\phi_{\mathbb{M}}$ is injective, then $\mathbb{M}$ is equationally Noetherian.*

Corollary 5.18. *The monoid $\langle a, b, c \mid abc = cba\rangle$ is equationally Noetherian.*

Proof. Combine Lemma 5.10 and Theorem 12. □

The proof of Theorem 12. Before we prove Theorem 12, we formally introduce the notions related to systems of equations.

Let $\mathbb{M} = \langle \Sigma \mid S\rangle$ be a finitely presented monoid. For a pair of words $w_1, w_2 \in \Sigma^*$ we write $w_1 =_{\mathbb{M}} w_2$ if $w_1 {\sim_S} w_2$. We define an *equation* in $\mathbb{M}$

[11] Equational Noetherianity is also called *compactness property* by some authors.

[12] Honkala, 2000.

[13] Shevlyakov, 2016.

[14] Albert and Lawrence, 1985 and Guba, 1986.

[15] Lothaire, 2002 (Lothaire denotes equational Noetherianity as *compactness property*).

in variable set X to be a pair of words over alphabet $\Sigma \cup X$[16]. We write an equation $(s_1, s_2) \in (\Sigma \cup X)^*$ as $s_1 = s_2$. We say that an X-tuple $\mathbf{m} = (m_x)_{x \in X}$ of elements of $\mathbb{M}$ *satisfies*, or is a *solution* to, an equation $s_1 = s_2$, and write $\mathbf{m} \models s_1 = s_2$, if

$$s_1[x := m_x, x \in X] =_{\mathbb{M}} s_2[x := m_x, x \in X].$$

We generalize this notation to *systems of equations*, which by definition are sets of equations; and so, if $\mathbf{m}$ satisfies every equation from a system of equations $\mathcal{S}$, then we say that $\mathbf{m}$ *satisfies*, or is a *solution* to, $\mathcal{S}$ and write $\mathbf{m} \models \mathcal{S}$.

Example 5.19. Consider the bicyclic monoid $\mathbb{M} = \langle \Sigma \mid S \rangle$ where $\Sigma = \{a, b\}$ and $S = \{ab = \varepsilon\}$. Let $X = \{x_1, x_2\}$ and let $s_1, s_2 \in (\Sigma \cup X)^*$ be defined by

$$\begin{aligned} s_1 &:= a x_1 x_1, \\ s_2 &:= a x_2 a x_1 x_1. \end{aligned}$$

Then $\mathbf{m} = \{x_1 \mapsto ba, x_2 \mapsto b\}$ is a solution to the equation $s_1 = s_2$, or in other words, $\mathbf{m} \models s_1 = s_2$. Indeed, $s_1[x_1 := ba, x_2 := b] = ababa =_{\mathbb{M}} a$ and $s_2[x_1 := ba, x_2 := b] = abababa =_{\mathbb{M}} a$.

Also, analogously we define each of the above notions for quotients of rings of polynomials; in particular, for a ring $\mathbb{Q}[X]/I$, where $I \subsetneq \mathbb{Q}[X]$ is an ideal, an equation in variable set Y is a pair of polynomials in variable set $X \cup Y$ (where the variables from Y are supposed to be substituted by elements from $\mathbb{Q}[X]/I$).

Proof of Theorem 12. Let $\mathcal{S}$ be a system of equations in $\mathbb{M}$ in variable set X. Consider a system of equations $\phi\mathcal{S}$ in $R_{\mathbb{M}}$ defined by

$$\phi\mathcal{S} := \{\phi(s_1) = \phi(s_2) \mid s_1 = s_2 \in \mathcal{S}\}.$$

The ring $R_{\mathbb{M}}$ is Noetherian, hence it is equationally Noetherian; in consequence, we can take an equivalent finite subsystem $\phi\mathcal{S}' := \{\phi(s_1) = \phi(s_2) \mid s_1 = s_2 \in \mathcal{S}'\}$ of $\phi\mathcal{S}$, for some finite $\mathcal{S}' \subseteq \mathcal{S}$. Then $\mathcal{S}'$ is equivalent to $\mathcal{S}$, due to the fact that every X-tuple $\mathbf{m} = (m_x)_{x \in X}$ of elements of $\mathbb{M}$ is a solution to $\mathcal{S}$ if and only if $\phi\mathbf{m} := (\phi(m_x))_{x \in X}$ is a solution to $\phi\mathcal{S}$. Indeed, due to Lemma 4.25 we have that

$$\begin{aligned} &\mathbf{m} \models s_1 =_{\mathbb{M}} s_2 \iff \\ &s_1[x := m_x, x \in X] =_{\mathbb{M}} s_2[x := m_x, x \in X] \iff \\ &\phi(s_1[x := m_x, x \in X]) =_{R_{\mathbb{M}}} \phi(s_2[x := m_x, x \in X]) \overset{\text{Lemma 4.25}}{\iff} \\ &\phi(s_1)[\phi(x) := \phi(m_x), x \in X] =_{R_{\mathbb{M}}} \phi(s_2)[\phi(x) := \phi(m_x), x \in X] \iff \\ &\phi\mathbf{m} \models \phi(s_1) =_{R_{\mathbb{M}}} \phi(s_2). \end{aligned}$$

[16] Another, equivalent, approach would be to define an equation in $\mathbb{M}$ as a pair of terms of $\mathbb{M}$ in variable set X; our approach however makes the notation in the proof of Theorem 12 lighter.

In consequence, for every X-tuple m of elements of $\mathbb{M}$

$$\mathbf{m} \models \mathcal{S} \Leftrightarrow \phi\mathbf{m} \models \phi\mathcal{S} \Leftrightarrow \phi\mathbf{m} \models \phi\mathcal{S}' \Leftrightarrow \mathbf{m} \models \mathcal{S}',$$

i.e. $\mathcal{S}$ is equivalent to $\mathcal{S}'$. □

5.6 Summary

By applying the quotient construction to our word-to-polynomial simulation we obtain a proof scheme for deciding functionality and equivalence of functional register transducers with output in a finitely presented monoid (Lemma 5.7). We illustrate this scheme on the monoid $\mathbb{M} = \langle a, b, c \mid abc = cba \rangle$, thus obtaining a decidability result for it (Theorem 5.11). However, let us add that this monoid admits a simpler proof: there is a polynomial simulation of $\mathbb{M}$ by a free monoid (the reasoning is standard: consider normal forms of elements of $\mathbb{M}$ w.r.t. the Semi-Thue system $\{abc \to cba\}$, or in other words, their lexicographically smallest representatives); in fact, we do not know of any monoid $\mathbb{M}$ for which there is no simple polynomial simulation by a free monoid but $\phi_{\mathbb{M}}$ is injective. We also remark that some parts of the proof, e.g. checking if the ring $R_{\mathbb{M}}$ is reduced, or performing a single step of the proof of injectivity of $\phi_{\mathbb{M}}$, can be automated, although we do not know if the crucial steps of terminating proof branches are automatable or not (Section 5.4). In addition to deciding functionality and equivalence of functional register transducers, this chapter's proof scheme yields a criterion for equational Noetherianity of a finitely presented monoid (Theorem 12). In consequence, from the reasoning we perform for the monoid $\mathbb{M} = \langle a, b, c \mid abc = cba \rangle$ we conclude that this monoid is equationally Noetherian (Corollary 5.18).

Bibliography

Albert, M. H. and J. Lawrence (Dec. 1985). “A Proof of Ehrenfeucht’s Conjecture”. In: *Theor. Comput. Sci.* 41.1, pp. 121–123. ISSN: 0304-3975.

Alur, R. and P. Černý (2010). “Expressiveness of streaming string transducers”. In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*. Ed. by Kamal Lodaya and Meena Mahajan. Vol. 8. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 1–12. ISBN: 978-3-939897-23-1. DOI: 10.4230/LIPIcs.FSTTCS.2010.1. URL: http://drops.dagstuhl.de/opus/volltexte/2010/2853.

— (2011). “Streaming Transducers for Algorithmic Verification of Single-Pass List-Processing Programs”. In: *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL ’11. Austin, Texas, USA: Association for Computing Machinery, pp. 599–610. ISBN: 9781450304900. DOI: 10.1145/1926385.1926454. URL: https://doi.org/10.1145/1926385.1926454.

Alur, R. and L. D’Antoni (2012). “Streaming Tree Transducers”. In: *Automata, Languages, and Programming*. Ed. by Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 42–53. ISBN: 978-3-642-31585-5.

Araki, T. and T. Kasami (1976). “Some decision problems related to the reachability problem for Petri nets”. In: *Theoretical Computer Science* 3.1, pp. 85–104. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(76)90067-0. URL: https://www.sciencedirect.com/science/article/pii/0304397576900670.

Benedikt, M., T. Duff, A. Sharad, and J. Worrell (2017). “Polynomial automata: Zeroness and applications”. In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pp. 1–12. DOI: 10.1109/LICS.2017.8005101.

Boiret, A., R. Piórkowski, and J. 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)*. Ed. by Sumit Ganguly and Paritosh Pandya. Vol. 122. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 48:1–48:16. ISBN: 978-3-95977-093-4. DOI: 10.4230/LIPIcs.FSTTCS.2018.48. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9947.

Bojańczyk, M. (2015). *Lecture Notes*. URL: https://www.mimuw.edu.pl/~bojan/20152016-2/jezyki-automaty-i-obliczenia-2/monadic-second-order-logic-and-courcelles-theorem/tree-width.

— (2018). "Polyregular Functions". In: *CoRR* abs/1810.08760. arXiv: 1810.08760. URL: http://arxiv.org/abs/1810.08760.

Bojańczyk, M., S. Kiefer, M. Shirmohammadi, and J. Worrell (2019). *Personal communication*.

Bojańczyk, M. and Mi. Pilipczuk (2016). "Definability Equals Recognizability for Graphs of Bounded Treewidth". In: *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '16. New York, NY, USA: Association for Computing Machinery, pp. 407–416. ISBN: 9781450343916. DOI: 10.1145/2933575.2934508. URL: https://doi.org/10.1145/2933575.2934508.

— (2017). "Optimizing Tree Decompositions in MSO". In: *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*. Ed. by Heribert Vollmer and Brigitte Vallée. Vol. 66. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 15:1–15:13. DOI: 10.4230/LIPIcs.STACS.2017.15. URL: https://doi.org/10.4230/LIPIcs.STACS.2017.15.

Bojańczyk, M. and J. Schmude (2020). "Some Remarks on Deciding Equivalence for Graph-To-Graph Transducers". In: *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*. Ed. by Javier Esparza and Daniel Kráľ. Vol. 170. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 19:1–19:14. ISBN: 978-3-95977-159-7. DOI: 10.4230/LIPIcs.MFCS.2020.19. URL: https://drops.dagstuhl.de/opus/volltexte/2020/12752.

Bojańczyk, M. and I. Walukiewicz (2008). "Forest algebras". In: *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*. Ed. by Jörg Flum, Erich Grädel, and Thomas Wilke. Vol. 2. Texts in Logic and Games. Amsterdam University Press, pp. 107–132.

Bojańczyk, Mikołaj (Feb. 2019). "The Hilbert Method for Transducer Equivalence". In: *ACM SIGLOG News* 6.1, pp. 5–17. DOI: 10.1145/3313909.3313911. URL: https://doi.org/10.1145/3313909.3313911.

Buchberger, B. and F. (Eds.) Winkler (1998). *Gröbner Bases and Applications*. London Mathematical Society Lecture Note Series. Cambridge University Press. DOI: 10.1017/CBO9780511565847.

Büchi, J. (1960). "Weak Second-Order Arithmetic and Finite Automata". In: *Mathematical Logic Quarterly* 6, pp. 66–92.

Courcelle, B. (1989). "The monadic second-order logic of graphs : Definable sets of finite graphs". In: *Graph-Theoretic Concepts in Computer Science*. Ed. by J. van Leeuwen. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 30–53. ISBN: 978-3-540-46076-3.

— (1990). "CHAPTER 5 - Graph Rewriting: An Algebraic and Logic Approach". In: *Formal Models and Semantics*. Ed. by Jan van Leeuwen. Handbook of Theoretical Computer Science. Amsterdam: Elsevier, pp. 193–242. ISBN: 978-0-444-88074-1. DOI: https://doi.org/10.1016/B978-0-444-

88074-1.50010-X. URL: https://www.sciencedirect.com/science/article/pii/B978044488074150010X.

— (1994). "Monadic second-order definable graph transductions: a survey". In: *Theoretical Computer Science* 126.1, pp. 53–75. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(94)90268-2. URL: https://www.sciencedirect.com/science/article/pii/0304397594902682.

Courcelle, B. and J. Engelfriet (2012). *Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach*. Encyclopedia of Mathematics and its Applications. Cambridge University Press. DOI: 10.1017/CBO9780511977619.

Cox, D. A., J. Little, and D. O'Shea (2015). *Ideals, Varieties, and Algorithms*. Springer International Publishing. DOI: 10.1007/978-3-319-16721-3. URL: https://doi.org/10.1007%2F978-3-319-16721-3.

Cygan, M., F.V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, Ma. Pilipczuk, Mi. Pilipczuk, and S. Saurabh (2015). "Treewidth". In: *Parameterized Algorithms*. Cham: Springer International Publishing, pp. 151–244. ISBN: 978-3-319-21275-3. DOI: 10.1007/978-3-319-21275-3_7. URL: https://doi.org/10.1007/978-3-319-21275-3_7.

Dauchet, M., T. Heuillard, P. Lescanne, and S. Tison (1990). "Decidability of the confluence of finite ground term rewrite systems and of other related term rewrite systems". In: *Information and Computation* 88.2, pp. 187–201. ISSN: 0890-5401. DOI: https://doi.org/10.1016/0890-5401(90)90015-A. URL: https://www.sciencedirect.com/science/article/pii/089054019090015A.

Dell, H., M. Grohe, and G. Rattan (2018). "Lovász Meets Weisfeiler and Leman". In: *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Ed. by Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella. Vol. 107. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 40:1–40:14. ISBN: 978-3-95977-076-7. DOI: 10.4230/LIPIcs.ICALP.2018.40. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9044.

Diekert, V. and Y. Métivier (1997). "Partial Commutation and Traces". In: *Handbook of Formal Languages: Volume 3 Beyond Words*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 457–533. ISBN: 978-3-642-59126-6. DOI: 10.1007/978-3-642-59126-6_8. URL: https://doi.org/10.1007/978-3-642-59126-6_8.

Diestel, R. (2017). "Graph Minors". In: *Graph Theory*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 347–391. ISBN: 978-3-662-53622-3. DOI: 10.1007/978-3-662-53622-3_12. URL: https://doi.org/10.1007/978-3-662-53622-3_12.

Dvořák, Z. (2010). "On recognizing graphs by numbers of homomorphisms". In: *Journal of Graph Theory* 64.4, pp. 330–342. DOI: 10.1002/jgt.20461. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jgt.20461. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/jgt.20461.

Elgot, C. C. (1961). “Decision Problems of Finite Automata Design and Related Arithmetics”. In: *Transactions of the American Mathematical Society* 98.1, pp. 21–51. ISSN: 00029947. URL: http://www.jstor.org/stable/1993511.

Engelfriet, J. and H. J. Hoogeboom (Apr. 2001). “MSO Definable String Transductions and Two-Way Finite-State Transducers”. In: *ACM Trans. Comput. Logic* 2.2, pp. 216–254. ISSN: 1529-3785. DOI: 10.1145/371316.371512. URL: https://doi.org/10.1145/371316.371512.

Engelfriet, J., H. J. Hoogeboom, and J.-P. Van Best (1999). “Trips on Trees”. In: *ACTA CYBERNETICA* 14, pp. 51–64.

Engelfriet, J. and S. Maneth (1999). “Macro Tree Transducers, Attribute Grammars, and MSO Definable Tree Translations”. In: *Information and Computation* 154.1, pp. 34–91. ISSN: 0890-5401. DOI: 10.1006/inco.1999.2807. URL: https://www.sciencedirect.com/science/article/pii/S0890540199928079.

— (2005). “The Equivalence Problem for Deterministic MSO Tree Transducers Is Decidable”. In: *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science*. Ed. by Sundar Sarukkai and Sandeep Sen. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 495–504. ISBN: 978-3-540-32419-5.

Engelfriet, J. and H. Vogler (1985). “Macro tree transducers”. In: *Journal of Computer and System Sciences* 31.1, pp. 71–146. ISSN: 0022-0000. DOI: 10.1016/0022-0000(85)90066-2. URL: https://www.sciencedirect.com/science/article/pii/0022000085900662.

Fülöp, Z. (Jan. 1981). “On attributed tree transducers”. In: *Acta Cybern.* 5, pp. 261–279. DOI: 10.1007/978-3-642-72248-6_5.

Fülöp, Z. and H. Vogler (1998). *Syntax-Directed Semantics: Formal Models Based on Tree Transducers*. Monographs in Theoretical Computer Science An EATCS Series. Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-642-72248-6.

Gallot, P.D., A. Lemay, and S. Salvati (2020). “Linear High-Order Deterministic Tree Transducers with Regular Look-Ahead”. In: *45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*. Ed. by Javier Esparza and Daniel Kráľ. Vol. 170. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 38:1–38:13. ISBN: 978-3-95977-159-7. DOI: 10.4230/LIPIcs.MFCS.2020.38. URL: https://drops.dagstuhl.de/opus/volltexte/2020/12705.

Globerman, N. and D. Harel (1996). “Complexity Results for Two-Way and Multi-Pebble Automata and their Logics”. In: *Theoretical Computer Science* 169, pp. 161–184.

Guba, V. S. (Sept. 1986). “Equivalence of infinite systems of equations in free groups and semigroups to finite subsystems”. In: *Mathematical notes of the Academy of Sciences of the USSR* 40.3, pp. 688–690. ISSN: 1573-8876. DOI: 10.1007/BF01142470. URL: https://doi.org/10.1007/BF01142470.

Halin, R. (Mar. 1976). “S-functions for graphs”. In: *Journal of Geometry* 8.1, pp. 171–186. ISSN: 1420-8997. DOI: 10.1007/BF01917434. URL: https://doi.org/10.1007/BF01917434.

Harju, T., J. Karhumäki, and W. Plandowski (1995). “Compactness of systems of equations in semigroups”. In: *Automata, Languages and Programming*. Ed. by Zoltán Fülöp and Ferenc Gécseg. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 444–454. ISBN: 978-3-540-49425-6.

Honkala, J. (2000). “A short solution for the HDT0L sequence equivalence problem”. In: *Theoretical Computer Science* 244.1, pp. 267–270. ISSN: 0304-3975. DOI: 10.1016/S0304-3975(00)00158-4. URL: https://www.sciencedirect.com/science/article/pii/S0304397500001584.

Hosoya, H. (2010). “Tree transducers”. In: *Foundations of XML Processing: The Tree-Automata Approach*. Cambridge University Press, pp. 127–137. DOI: 10.1017/CBO9780511762093.011.

Huschenbett, M., D. Kuske, and G. Zetzsche (2014). “The Monoid of Queue Actions”. In: *Mathematical Foundations of Computer Science 2014*. Ed. by Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 340–351. ISBN: 978-3-662-44522-8.

Kühnle, K. and E. W. Mayr (1996). “Exponential Space Computation of GröBner Bases”. In: *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation*. ISSAC ’96. Zurich, Switzerland: Association for Computing Machinery, pp. 63–71. ISBN: 0897917960. DOI: 10.1145/236869.236900. URL: https://doi.org/10.1145/236869.236900.

Lasota, S. and R. Piórkowski (2019). *Personal communication*.

Lothaire, M. (2002). *Algebraic Combinatorics on Words*. Encyclopedia of Mathematics and its Applications. Cambridge University Press. DOI: 10.1017/CBO9781107326019.

Maletti, A. (2015). “Extended Tree Transducers in Natural Language Processing”. In: *Proceedings of the 12th International Conference on Finite-State Methods and Natural Language Processing 2015 (FSMNLP 2015 Düsseldorf)*. Association for Computational Linguistics. URL: https://aclanthology.org/W15-4801.

Maneth, S. (Dec. 1998). “The Generating Power of Total Deterministic Tree Transducer”. In: *Inf. Comput.* 147.2, pp. 111–144. ISSN: 0890-5401. DOI: 10.1006/inco.1998.2736. URL: https://doi.org/10.1006/inco.1998.2736.

Mayr, E. W. and A. R. Meyer (1982). “The complexity of the word problems for commutative semigroups and polynomial ideals”. In: *Advances in Mathematics* 46.3, pp. 305–329. ISSN: 0001-8708. DOI: 10.1016/0001-8708(82)90048-2. URL: https://www.sciencedirect.com/science/article/pii/0001870882900482.

Mealy, George H. (1955). “A method for synthesizing sequential circuits”. In: *The Bell System Technical Journal* 34.5, pp. 1045–1079. DOI: 10.1002/j.1538-7305.1955.tb03788.x.

Salomaa, A. and M. Soittola (1978). “Automata-Theoretic Aspects of Formal Power Series”. In: *Texts and Monographs in Computer Science*.

Schmude, J. (2021). *On polynomial grammars extended with substitution*. arXiv: 2102.08705 [cs.FL].

Schwenk, A. J. (1973). *Almost all trees are cospectral, New directions in the theory of graphs (Proc. Third Ann Arbor Conf., Univ. Michigan, Ann Arbor, Mich., 1971)*.

Seese, D. (1991). "The structure of the models of decidable monadic theories of graphs". In: *Annals of Pure and Applied Logic* 53.2, pp. 169–195. ISSN: 0168-0072. DOI: 10.1016/0168-0072(91)90054-P. URL: https://www.sciencedirect.com/science/article/pii/016800729190054P.

Seidl, H., S. Maneth, and G. Kemper (Apr. 2018). "Equivalence of Deterministic Top-Down Tree-to-String Transducers Is Decidable". In: *J. ACM* 65.4. ISSN: 0004-5411. DOI: 10.1145/3182653. URL: https://doi.org/10.1145/3182653.

Shevlyakov, A. (2016). *Lectures notes in universal algebraic geometry*. arXiv: 1601.02743 [math.AG].

Thirunarayan, Krishnaprasad (Jan. 2009). "Attribute Grammars and Their Applications". In: DOI: 10.4018/978-1-60566-026-4.ch046.

Trakhtenbrot, B. A. (1961). "Finite automata and logic of monadic predicates". In: *Doklady Akademii Nauk SSSR*, 149:326–329.

Zakharov, V. A. (2015). "Equivalence Checking Problem for Finite State Transducers over Semigroups". In: *Algebraic Informatics*. Ed. by Andreas Maletti. Cham: Springer International Publishing, pp. 208–221. ISBN: 978-3-319-23021-4.