Reducing Transducer Equivalence to Register Automata Problems Solved by "Hilbert Method"

Adrien Boiret
University of Warsaw, Poland
a.boiret@mimuw.edu.pl

Radosław Piórkowski
University of Warsaw, Poland
r.piorkowski@mimuw.edu.pl

Janusz Schmude
University of Warsaw, Poland
j.schmude@mimuw.edu.pl

Abstract

In the past decades, classical results from algebra, including Hilbert's Basis Theorem, had various applications in formal languages, including a proof of the Ehrenfeucht Conjecture, decidability of HDT0L sequence equivalence, and decidability of the equivalence problem for functional tree-to-string transducers.

In this paper, we study the scope of the algebraic methods mentioned above, particularily as applied to the functionality problem for register automata, and equivalence for functional register automata. We provide two results, one positive, one negative. The positive result is that functionality and equivalence are decidable for MSO transformations on unordered forests. The negative result comes from a try to extend this method to decide functionality and equivalence on macro tree transducers. We reduce macro tree transducers equivalence to an equivalence problem for some class of register automata naturally relevant to our method. We then prove this latter problem to be undecidable.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases formal language, Hilbert's basis theorem, transducers, register automata, equivalence problem, unordered trees, MSO transformations

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.48

Related Version `https://arxiv.org/abs/1806.04361`

Funding This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (ERC consolidator grant LIPA, agreement no. 683080). Furthermore, it was partially supported by the NCN grant 2016/21/B/ST6/01505.

Acknowledgements We would like to thank Mikołaj Bojańczyk, from the University of Warsaw, for introducing us to the topic of using the Hilbert Method to decide equivalence of register automata, and for his active participation in the finding of this result and the writing of this paper.

1 Introduction

The study of finite-state machines, such as transducers [15, 8, 14] or register automata [2, 3], and of logic specifications, such as MSO-definable transformations [9], provides a theoretical ground to study document and data processing.

In this paper, we will consider the equivalence problem of functional transducers. We focus on register automata, i.e. transducers that store values in a finite number of registers that can be updated or combined after reading an input symbol. Streaming String Transducers (SST) [2] and Streaming Tree Transducers (STT) [3] are classes of register automata (see for example [4]) where the equivalence is decidable for the *copyless* restriction, i.e. the case where each register update cannot use the same register twice. This restriction makes SST equivalent to MSO-definable string transformations. Macro tree transducers (MTT) [11], an expressive class of tree transducers for which equivalence decidability remains a challenging open problem, can be seen as register automata, whose registers store tree contexts. Although equivalence is not known to be decidable for the whole class, there exists a linear size increase fragment of decidable equivalence, that is equivalent to MSO-definable tree transformations, and can be characterized by a restriction on MTT quite close to copyless [10].

Some equivalence decidability results have been proven on register automata without copyless restrictions [16, 5], by reducing to algebraic problems such as ideal inclusion and by applying Hilbert's Basis Theorem and other classical results of algebraic geometry. In this paper we will refer to this as the "Hilbert Method". This method was used to prove diverse results, dating back to at least the proof of the Ehrenfeucht Conjecture [1], and the sequence problem for HDT0L [13, 12]. It has recently found new applications in formal languages; for example, equivalence was proven decidable for general tree-to-string transducers by seeing them as copyful register automata on words [16].

In this paper, we use an abstraction of these previous applications of the "Hilbert Method" as presented in [6]. We apply these preexisting results to the study of unordered forest transductions – and notably MSO functions. Note that equivalence of MSO-definable transductions on unordered forests is not a straightforward corollary of the ordered case, as the loss of order makes equivalence more difficult to identify. We also try to apply those methods to obtain decidability of MTT equivalence. For unordered forests, we obtain a positive result, showing that register automata on forest contexts with one hole have decidable functionality and equivalence. For the attempt to study MTT, we prove an undecidability result on register automata using polynomials and composition, which means the natural extension of this approach does not yield a definitive answer for the decidability of MTT equivalence.

Layout

Section 2 presents the notions of algebra, register automata, and the notions necessary to use an abstraction of the "Hilbert Method" as presented in [6]. **Section 3** is dedicated to the proof of the positive result that we can apply the "Hilbert Method" to contexts of unordered forests with at most one hole (i.e. the algebra of unordered forests with limited substitution). This provides a class of register automata encompassing MSO functions on unordered forests where functionality is decidable. Finally, **Section 4** describes how applying a method similar as in Section 3 to study MTT equivalence leads to studying register automata on the algebra of polynomials with the substitution operation, a class whose functionality and equivalence we prove to be undecidable.

2 Preliminaries

Algebras. An *algebra* $\mathbf{A} = (A, \rho_1, \dots, \rho_n)$ is a (potentially infinite) set of elements A, and a finite number of operations $\rho_1, \dots, \rho_n$. Each operation is a function $\rho : A^k \to A$ for some $k \in \mathbb{N}$.

Polynomials. For an algebra $\mathbf{A}$ and a set $X = \{x_1, \dots, x_n\}$ of variables, we note $A[X]$ the set of terms over $A \cup X$. A *polynomial function* of $\mathbf{A}$ is a function $f : A^k \to A$. For example on $\mathbf{A} = (\mathbb{Q}, +, \times)$, the term $\times(+(x, 2), +(x, y))$ induces the polynomial function $f : (x, y) \mapsto (x + 2)(x + y)$. The definition of polynomial functions can be extended to functions $f \colon A^k \to A^m$ by product of their output: if $f_1, \dots, f_m$ are polynomial functions from A^k to A, then $f' \colon \overline{a} \in A^k \mapsto (f_1(\overline{a}), \dots, f_m(\overline{a}))$ is a polynomial function from A^k to A^m. Note that polynomial functions are closed under composition.

One can define the *algebra of polynomials over* $\mathbf{A}$ *with variable set* X, denoted $\mathbf{A}[X]$. Its elements are equivalence classes of terms over $A \cup X$ with the operations of $\mathbf{A}$, where two terms are called *equivalent* if they induce identical polynomial functions. $\mathbf{A}[X]$ can be seen as an algebra that subsumes $\mathbf{A}$, with natural definition of operations. A classical example of this construction is the ring of polynomials $(\mathbb{Q}[x], +, \times)$, obtained from the ring $(\mathbb{Q}, +, \times)$.

By adding the substitution operation $(-)[X := (-)]$ to $\mathbf{A}[X]$, we get a new algebra called a *composition algebra of polynomials* and denoted $\mathbf{A}[X]^{\mathsf{subs}}$. Homomorphisms of such algebras are called *composition homomorphisms*. For brevity we write $(-)[x_i := (-)]$ for the substitution of a single $x_i \in X$. Examples of such algebras include well-nested words with a placeholder symbol "?", as used in the registers of Streaming Tree Transducers [3], or tree contexts with variables in their leaves, as used in Macro Tree Transducers [11].

Simulation. Following the abstractions as they are presented in [6] [1], we define simulations between algebras in a way that is relevant to the use of the "Hilbert Method".

▶ **Definition 1.** Let $\mathbf{A}$ and $\mathbf{B}$ be algebras. We say that $\alpha : A \to B^n$ is a *simulation* of $\mathbf{A}$ in $\mathbf{B}$ if for every operation $\rho : A^m \to A$ of $\mathbf{A}$, there is a polynomial function $f : B^{m \times n} \to B^n$ of $\mathbf{B}$ such that $\alpha \circ \rho = f \circ \alpha$, where α is defined from A^m to $B^{m \times n}$ coordinate-wise. If such a simulation α exists, we say that $\mathbf{A}$ is *simulated by* $\mathbf{B}$ ($\mathbf{A} \preceq_{\mathsf{pol}} \mathbf{B}$).

$$\begin{array}{ccc} A^m & \xrightarrow{(\alpha, \cdots, \alpha)} & B^{m \times n} \\ \rho \downarrow & & \downarrow f \\ A & \xrightarrow[\alpha]{} & B^n \end{array}$$

The following lemma states that simulations extend to composition algebras.

▶ **Lemma 2.** *Let* $\boldsymbol{Q}[X] = (\mathbb{Q}[X], +, \times)$. *If* $\mathbf{A} \preceq_{\mathsf{pol}} \boldsymbol{Q}[X]$ *and* A *is an infinite set, then* $\mathbf{A}[Y]^{\mathsf{subs}} \preceq_{\mathsf{pol}} \boldsymbol{Q}[X][Y]^{\mathsf{subs}}$.

[1] As of this version's redaction, Part 11 of [6] is the part relevant for this paper. This and any theorem or page number can change in future versions of [6].

Proof. If there is a simulation α from $\mathbf{A}$ to $\mathbb{Q}[X]$, then α can be extended into a simulation $\widetilde{\alpha}$ from $\mathbf{A}[Y]$ to $\boldsymbol{Q}[X][Y]$ by setting $\alpha(Y) = Y$, and requiring $\widetilde{\alpha}$ to be a homomorphism. It is important to check that $\widetilde{\alpha}$ indeed is a function (i.e. preserves equivalence of terms): if terms $t_1, t_2 \in A[Y]$ induce the same functions on A, then $\widetilde{\alpha}(t_1)(Y)$ and $\widetilde{\alpha}(t_2)(Y)$ are polynomial functions that are equal on $\alpha(A)$. Since $\alpha(A)$ is an infinite subset of $\mathbb{Q}[X]$, $\alpha(t_1)$ and $\alpha(t_2)$ are equal everywhere. The proof of injectivity is straightforward. Let $P(Y), Q(Y) \in A[Y]$ be any two nonequivalent terms. Then there is a tuple $\overline{a}$ of A such that $P(\overline{a}) \neq Q(\overline{a})$. If $\widetilde{\alpha}(P) = \widetilde{\alpha}(Q)$, we would have $\alpha(P(\overline{a})) = \alpha(P)[Y := \overline{a}] = \alpha(Q)[Y := \overline{a}] = \alpha(Q(\overline{a}))$. This would contradict the injectivity of α on A. ◀

Typed Algebras. Some of the algebras we consider are *multi-sorted*, which is to say that their elements are divided between a finite number of types. A *multi-sorted algebra* is an algebra $\mathbf{A} = (A, \rho_1, \dots, \rho_n)$ such that:

- A can be partitioned into $A_1, \dots, A_m$,
- each operation ρ is a function $\rho : A_{i_0} \times \cdots \times A_{i_k} \to A_j$.

To each $a \in A$ we associate a *type*, which is a unique i such that $a \in A_i$. Note that in a multi-sorted algebra $\mathbf{A}$ polynomial functions are typed $f : A_{i_0} \times \cdots \times A_{i_k} \to A_j$, and simulation and substitutions must be defined type-wise.

Register automata. In this paper we will work on register automata that make a single bottom-up pass on an input ranked tree, use a finite set of states, and a finite set of registers with values in $A$for some algebra $\mathbf{A}$. When the automaton reads an input symbol, it updates its register values as a polynomial function of $\mathbf{A}$ applied to the register values in its subtrees. This formalism is already present in the literature: streaming tree transducers [2], for example, are register automata on input words and register values in the algebra of words on an alphabet Σ, with the concatenation operation.

A *signature* Σ is a finite set of symbols a, each with a corresponding finite rank $\mathrm{rk}(a) \in \mathbb{N}$. A *ranked tree* is a term on this signature Σ: if $a \in \Sigma$, $\mathrm{rk}(a) = n$, and $t_1, \dots, t_n$ are trees, then $a(t_1, \dots, t_n)$ is a tree.

▶ **Definition 3.** Let $\mathbf{A} = (A, \rho_1, \dots, \rho_n)$ be an algebra. A *bottom-up register automaton* with values in $\mathbf{A}$ (or $\mathbf{A}$-RA) is a tuple $M = (\Sigma, n, Q, \delta, f_{\mathrm{out}})$, where:

- Σ is a ranked set
- n is the number of A-registers used by M
- Q is a finite set of states
- δ is a finite set of transitions of form $a(q_1, \dots, q_k) \to q, f$ where $a \in \Sigma$ of rank k, $\{q, q_1, \dots, q_k\} \subset Q$, and $f : A^{n \times k} \to A^n$ a polynomial function of $\mathbf{A}$.
- f_{out} is a partial output function that to some states $q \in Q$ associates $f_q : A^n \to A$ a polynomial function of $\mathbf{A}$.

A *configuration* of M is a n-uple $(q, \overline{r})$ where $q \in Q$ is a state and $\overline{r} = (r_1, \dots, r_n) \in A^n$ is a n-uple of register values in A. We define by induction the fact that a tree t can reach a configuration $(q, \overline{r})$, noted $t \to (q, \overline{r})$: If $a \in \Sigma$ of rank k, $a(q_1, \dots, q_k) \to q, f$ a rule of δ, and for $0 \leqslant i \leqslant k$, $t_i \to (q_i, \overline{r}_i)$, then

$$a(t_1, \dots, t_k) \to (q, f(\overline{r}_1, \dots, \overline{r}_k)).$$

M determines a relation $[\![M]\!]$ from trees to values in A. It is defined using f_{out} as a final step: if $f_{\mathrm{out}}(q) = f_q$, and $t \to (q, \overline{r})$, then $f_q(\overline{r}) \in [\![M]\!](t)$.

We say that a **A**-RA is *functional* if $[\![M]\!]$ is a function. We say that a **A**-RA is *deterministic* if for all $a, q_1, \dots, q_k$ there is at most one rule $a(q_1, \dots, q_k) \to q, f$ in δ. Any deterministic **A**-RA is functional.

Note that on a multi-sorted algebra, we further impose that every state q has a certain type $A_{i_1} \times \cdots \times A_{i_n}$, i.e. if $(q, \overline{r})$ is a configuration of M, then $\overline{r} \in A_{i_1} \times \cdots \times A_{i_n}$.

"Hilbert Method". We now describe an abstraction [6] of the classical algebra methods that are used in the literature [16, 5] to decide equivalence of functional register automata over certain algebras (e.g. $(\Sigma^*, \cdot)$) using what we will refer to as the "Hilbert Method". More specifically, as it is always easy to prove the semi-decidability of non-equivalence of functional **A**-RA, by guessing two runs on the same input with different outputs, this method aims to prove that the functionality and equivalence problems over functional **A**-RA are semi-decidable.

This method can be described as a 4-step process:

- Simulate **A** by $\boldsymbol{Q}$, hence reducing **A**-RA equivalence to $\boldsymbol{Q}$-RA equivalence
- Functional $\boldsymbol{Q}$-RA equivalence can be reduced to $\boldsymbol{Q}$-RA zeroness, i.e. checking if a $\boldsymbol{Q}$-RA only outputs 0.
- $\boldsymbol{Q}$-RA zeroness can be reduced to ideal inclusion problem in $\overline{\mathbb{Q}}[X]$, i.e. the ring of polynomials with algebraic numbers as coefficients
- Ideal inclusion problem in $\overline{\mathbb{Q}}[X]$ is decidable

These results exist in the literature. We will provide references as well as an intuition of the main mechanisms in these proofs. For the first point, the reduction from **A**-RA equivalence to $\boldsymbol{Q}$-RA equivalence, an example is provided in [16], where **A** is $(\Sigma^*, \cdot)$. In essence, this part of the method amounts to a simulation as described in Definition 1. If $\mathbf{A} \preceq_{\mathsf{pol}} \mathbf{B}$ with simulation α, then any **A**-RA M can be simulated by a **B**-RA M', in the sense that M outputs $a \in A$ for an input t if and only M' outputs $\alpha(a) \in B^k$ for the same input t. This gives a reduction from **A**-RA equivalence to **B**-RA equivalence.

The second point is presented in the proof of Theorem 11.8 of [6]. If M and M' are two functional $\boldsymbol{Q}$-RA of same domain, using a natural product construction, one can create M'' that runs M and M' in parallel, then computes the difference of outputs between M and M'. Thus M and M' are equivalent iff M'' only outputs 0.

The third point can be found in the proof of Theorem 11.8 of [6]. The idea is to express $\boldsymbol{Q}$-RA zeroness as a set problem (with polynomial grammars as an intermediary in [6]). We want to find for each state q the set X_q of register values that M can hold in state q. These states obey to some inclusion equations: if $a(q_1, \dots, q_k) \to q, f$ is a rule of M, then $f(X_{q_1} \times ... \times X_{q_n}) \subseteq X_q$. Furthermore, if zeroness is true for M, then for every q such that final output function f_q is defined, $f_q(X_q) \subseteq \{0\}$. Interestingly, if such a family of sets of $\mathbb{Q}^n$ $(X_q)_{q \text{ state of } M}$ exists to satisfy those inclusions, then there exists a family of ideals of $\overline{\mathbb{Q}}[X]$, $(S_q)_{q \text{ state of } M}$, that satisfy opposite inclusions (Lemma 11.5 of [6]).

The fourth point uses classical algebra results to find such a family of ideal sets. The proof, as it is presented in Theorem 11.3 of [6], works as follows: Hilbert's Basis Theorem ensures that all families of ideals $(S_q)_{q \text{ state of } M}$ can be enumerated. For each of these families, it can be checked using Groebner Basis whether it respects a set of inclusions or not. Eventually, if a solution exists, it will be found, making this ideal problem, and thus $\boldsymbol{Q}$-RA zeroness, semi-decidable.

For this paper, we point out a few natural extensions to those methods and establish Theorem 4 and Corollary 5 as a basis for our work.

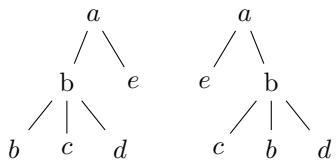


Figure 1 Two representations of the same unordered tree.

The first remark is that one can consider more problems than functional equivalence. Functionality itself can be studied with these methods. It is preserved by simulations, and $\boldsymbol{Q}$-RA functionality can be reduced to zeroness: instead of comparing two functional $\boldsymbol{Q}$-RA in the second point, M'' can run two copies of the same $\boldsymbol{Q}$-RA M and compute the output difference. M is functional iff M'' only outputs 0.

The second remark is that the classical algebra results (Hilbert Basis Theorem, Groebner Basis, algebraic closure of a field...) used in the fourth point extend to any computable field $\mathcal{K}$. In consequence, Theorem 11.8 of [6] holds for any $\mathcal{K}$-RA. Since the polynomial ring $\mathcal{K}[X]$ is a subring of a computable field $\mathcal{K}(X)$ (rational functions over $\mathcal{K}$), it holds for $\mathcal{K}[X]$-RA as well. We therefore state the following theorem.

▶ **Theorem 4.** *Let $\boldsymbol{Q}[X] = (\mathbb{Q}[X], +, \times)$. Functionality of $\boldsymbol{Q}[X]$-RA and equivalence of functional $\boldsymbol{Q}[X]$-RA are decidable.*

The result of Theorem 4 can be extended to other algebras using simulations from algebra to algebra. Indeed, if $\mathbf{A} \preceq_{\mathsf{pol}} \mathbf{B}$, then any $\mathbf{A}$-RA can be simulated by a $\mathbf{B}$-RA, and problems of functionality and equivalence reduce from $\mathbf{A}$-RA to $\mathbf{B}$-RA.

▶ **Corollary 5.** *Let $\mathbf{A}$ be an algebra. If $\mathbf{A} \preceq_{\mathsf{pol}} \boldsymbol{Q}[X]$, then functionality of $\mathbf{A}$-RA and equivalence of functional $\mathbf{A}$-RA are decidable.*

3 Unordered forests are simulated by polynomials

In this section we will show that the unordered tree forests (and more generally – the unordered forest algebra [7] that contains both forests and contexts with one hole) can be simulated in the sense of Definition 1 by polynomials with rational coefficients over a variable x (noted $\mathbb{Q}[x]$) with the operations $+, \times$. This, combined with Corollary 5, implies the decidability of functionality and equivalence for a class of Forests-RA. We then prove that this class can express all MSO-transformations on unordered forests.

An *unordered tree* on a finite signature Σ is an unranked tree (i.e. every node can have arbitrarily many children), but the children of a node form an unordered multiset, rather than an ordered list. For example, the following figure displays two representations of the same unordered tree. An *unordered forest* is a multiset of unordered trees.

Unordered forests can thus be defined as an algebra $\mathbf{UF} = (\mathsf{UF}, +, \{\mathsf{root}_a\})$:

1. UF is the set of unordered forests, including $\emptyset$ – the empty forest;
2. the operations are:
   - binary operation $+$ is the multiset addition,
   - for each letter $a \in \Sigma$, unary operation root_a: if $h = t_1 + \cdots + t_n$, then $\mathsf{root}_a(h) = a(t_1, \ldots, t_n)$.

In the rest of this paper, we will reason with a unary signature (and thus a unique root operation). This is done without loss of generality, as unordered forests on a finite signature

$\Sigma = \{a_1, \dots, a_n\}$ can easily be encoded by forests on a unary signature. To express it as a polynomial simulation, we can say that $\alpha(\emptyset) = \emptyset$, and that for all $1 \leqslant i \leqslant n$

$$\alpha(\mathsf{root}_{a_i}(h)) = \mathsf{root}^i(\mathsf{root}(\emptyset) + \mathsf{root}(\alpha(h)))$$

3.1 Encoding forests into polynomials

This subsection's aim is to prove the following result:

▶ **Proposition 6.** $(\mathsf{UF}, +, \mathsf{root})$ *is simulated by* $(\mathbb{Q}[x], +, \times)$.

To this end we construct an injective homomorphism $\phi : \mathsf{UF} \to \mathbb{Q}[x]$. This ϕ associates injectively to each forest a rational polynomial p. It is important to check that two identical forests with different representations (as in Figure 1) will not obtain different value by ϕ. Furthermore, the operations $+, \mathsf{root}$ must be encoded as ψ_+, ψ_r, two polynomial functions in $(\mathbb{Q}[x], +, \times)$, such that $\phi(h + h') = \psi_+(\phi(h), \phi(h'))$ and $\phi(\mathsf{root}(h)) = \psi_r(\phi(h))$.

Note that the term "polynomial" suffers here from semantic overload. We will take care to differentiate, on one hand, rational polynomials (i.e. the elements of $\mathbb{Q}[x]$, e.g. $2x - 7$), denoted by variants on letters p, q, and on the other hand, polynomial functions on the algebra $(\mathbb{Q}[x], +, \times)$ (e.g. $\psi : (p, q) \mapsto q \times q + 2p$), denoted by variants on the letter ψ.

Since $+$ in **UF** is both associative and commutative, we choose ψ_+ to be multiplication between rational polynomials: $\psi_+ : (p, q) \mapsto p \times q$. This leaves root to encode. To ensure that ϕ is injective, we would like to pick ψ_r so that ϕ sends all $\mathsf{root}(h)$ to pairwise different irreducible polynomials. This is done by picking $\psi_r : p \mapsto 2 + x \times p$ and using the Eisenstein's criterion with prime number 2: if a monic polynomial has all its nonleading coefficients divisible by 2, and the constant coefficient not divisible by 4, then this polynomial is irreducible over $\mathbb{Q}$. From there we define ϕ inductively: $\phi(\emptyset) = 1$, $\phi(h + h') = \phi(h) \times \phi(h')$, and $\phi(\mathsf{root}(h)) = 2 + x \times \phi(h)$. It is clear that ϕ respects the condition of polynomial simulation that any operation of **UF** must be encoded as polynomial operation in $(\mathbb{Q}[x], +, \times)$.

This leads directly to the proof of Proposition 6: ϕ is a simulation from **UF** to $\mathbb{Q}[x]$ as defined in Definition 1. It is injective, the operation $+$ is encoded by the polynomial function $\psi_+ : (p, q) \mapsto p \times q$, and the operation root is encoded by the polynomial function $\psi_r : p \mapsto 2 + x \times p$.

3.2 Extension to contexts

The combination of Corollary 5 and Proposition 6 gives decidability results on the class of **UF**-RA. The transducers of this class read a ranked input, and manipulate registers with values in UF. As an example, an **UF**-RA can read a binary input, and output the unordered forests that it encodes in a "First Child Next Sibling" manner, that is to say the left child in the input corresponds to the child in the output, and the right child in the input corresponds to the brother in the output. Note that this is an adaptation of classical FCNS encoding of unranked **ordered** trees in binary trees, but where the order is forgotten.

This can be described by a one-state one-register **UF**-RA that uses rules of form

$$(a, q, q) \to (q, (x, y) \mapsto \mathsf{root}_a(x) + y).$$

However, **UF**-RA have their restriction: since root and $+$ are the only two operations allowed, registers can only store subtrees to be placed at the bottom of the output. This leaves the class without the ability to combine subtrees of its output as freely as the MSO

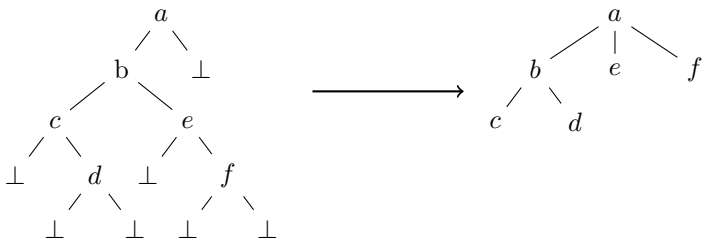


Figure 2 "FCNS" decoding.

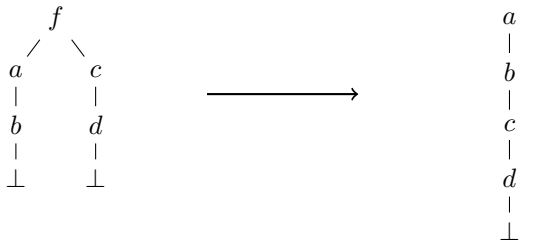
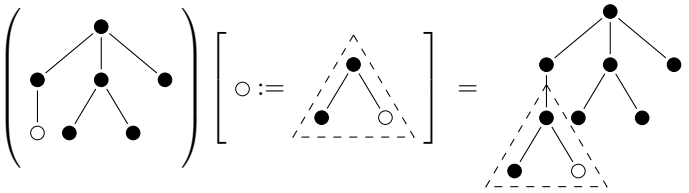


Figure 3 Subtree concatenation.

logic does. As an example, it is impossible to create an **UF**-RA that, if given an input $f(u, v)$ where u and v are two unary subtrees, outputs the subtree u above the subtree v as shown in Figure 3.

To get a more general class of register automata, that can perform such superpositions, we need to allow registers to store contexts, rather than forests. While the use of the Hilbert Methods for algebras of general contexts remains a difficult and interesting open problem, we will show that forest contexts with at most one hole are simulated by polynomials of $\mathbb{Z}[x]$.

We use the unordered version of 2-sorted Forest Algebra [7], consisting of unordered forests of trees and contexts with at most one hole. Since the previous subsection deals with an algebra of forests, to avoid confusion, we will call this the *Unordered Context and Forest* algebra (noted **UCF**). Using the definition of composition algebras, **UCF** is a subset of $\mathbf{UF}[\circ]^{\mathsf{subs}}$, where we impose that the replacable variable $\circ$ occurs at most once.



On this algebra, we will show the following results:

▶ **Theorem 7.** *UCF is simulated by* $(\mathbb{Q}[x], +, \times)$.

▶ **Corollary 8.** *Functionality of UCF-RA and equivalence of functional UCF-RA are decidable.*

Lemma 2 ensures that since $\mathbf{UF} \preceq_{\mathsf{pol}} (\mathbb{Q}[x], +, \times)$, then $\mathbf{UF}[\circ]^{\mathsf{subs}} \preceq_{\mathsf{pol}} \mathbb{Q}[x][y]^{\mathsf{subs}}$, i.e. $\mathbb{Q}[x, y]$ where only y can be substituted. **UCF** is the restriction of $\mathbf{UF}[\circ]^{\mathsf{subs}}$ to its elements with at most one occurrence of $\circ$. This forms a 2-sorted algebra. We consider its natural match in $\mathbb{Q}[x][y]^{\mathsf{subs}}$: Let $\mathbf{A}$ be the 2-sorted algebra:

- The universe is $A := \{p(x) + yq(x) : p, q \in \mathbb{Q}[x]\} \subseteq \mathbb{Q}[x, y]$.
- The types are $A_0 := \mathbb{Q}[x], A_1 := A \setminus A_0$.
- The operations are:
  - multiplication, defined only on pairs of types: $(0, 0), (0, 1), (1, 0)$,
  - $(-)[y := (-)]$.

▶ **Lemma 9.** *UCF is simulated by* **A**.

Proof. We call $\alpha : \mathbf{UF}[\circ]^{\mathsf{subs}} \to \mathbb{Q}[x][y]^{\mathsf{subs}}$ the homomorphism obtained by extending last subsection's ϕ with mapping the substitution variable $\circ$ to y. We restrict α to terms with at most one occurrence of $\circ$. The image of $h \in \mathbf{UCF}$ will then be a term of $\mathbb{Q}[x][y]^{\mathsf{subs}}$ with at most one occurrence of y. If $\circ$ never appears in h, then y never appears in $\alpha(h)$, thus $\alpha(h) \in \mathbf{A}_0$. If $\circ$ appears once in h, then y appears once in $\alpha(h)$, thus $\alpha(h) \in \mathbf{A}_1$. ◀

We now prove that **A** is simulated by $\mathbb{Q}[x]$ *without* substitution.

▶ **Lemma 10.** **A** *is simulated by* $(\mathbb{Q}[x], +, \times)$.

Proof. We will use encoding of **A** in $\mathbb{Q}[x] \times \mathbb{Q}[x]$ given by $p(x) + yq(x) \mapsto (p, q)$. Provided this, we encode operations $+, \times, (-)[y := (-)]$ in a straightforward manner. For example, for the composition operation in **A**, we see that $(p(x) + yq(x))[y := (p'(x) + yq'(x))]$ is equal to $p(x) + p'(x)q(x) + yq'(x)q(x)$. Hence, in pairs of $\mathbb{Q}$[x], $(-)[y := (-)]$ is encoded by $\psi_{(-)[y:=(-)]} : (p, q, p', q') \mapsto (p + p'q, q'q)$. ◀

Since $\preceq_{\mathsf{pol}}$ is a transitive relation, Lemma 10 and Lemma 9 give Theorem 7. Once Theorem 7 is proven, Corollary 5 gives Corollary 8.

Note that this proof extends to contexts with a bounded number of holes. We can add N substitution variables $\circ_1, \ldots, \circ_N$ to **UF**. Lemma 2 gives a homomorphism α_N that ensures $\mathbf{UF}[\circ_1, \ldots, \circ_N]^{\mathsf{subs}} \preceq_{\mathsf{pol}} \mathbb{Q}[x][y_1, \ldots, y_N]^{\mathsf{subs}}$. One could then define contexts with at most M occurrences of variables $\mathbf{UCF}^{\leq \mathrm{M}}$. In a manner similar to Lemma 9, we can find a finitely-sorted algebra that contains $\alpha_N(\mathbf{UCF}^{\leq \mathrm{M}})$, i.e. an algebra of all polynomials of $\mathbb{Q}[x][y_1, \ldots, y_N]$ with a degree $\leqslant M$ regarding the variables $y_1, \ldots, y_N$. Then, in a manner similar to Lemma 10, we can show that finite degree composition can be encoded in $(\mathbb{Q}[x], +, \times)$.

▶ **Corollary 11.** $\boldsymbol{UCF}^{\leq \mathrm{M}}$ *is simulated by* $(\mathbb{Q}[x], +, \times)$. *Functionality of* $\boldsymbol{UCF}^{\leq \mathrm{M}}$*-RA and equivalence of functional* $\boldsymbol{UCF}^{\leq \mathrm{M}}$*-RA are decidable.*

3.3 Encompassing of MSO

Corollary 8 gives decidability results on the class of **UCF**-RA. We motivated this class as a relevant extension of **UF**-RA by exhibiting a transformation (see Figure 3) that required contexts to be expressed. However, this class is not immediately relevant in its properties or expressiveness. In this section, we prove that **UCF**-RA can express strictly more than all MSO-definable transformations on unordered trees. Note that **UCF**-RA define functions from *binary ordered* trees to UCF, not from UF to UF. We say that an **UCF**-RA expresses a function $f : \mathsf{UF} \to \mathsf{UF}$ if for a binary tree t that is the "FCNS" encoding of a forest h, its image for the tree t is $f(h)$.

We briefly present a definition of MSO formulae and transformations. More complete definitions exist elsewhere in the literature (e.g. [9]).

The syntax of monadic second order logic (MSO) is:

$$\phi := \phi \wedge \phi \mid \neg\phi \mid \exists x \phi \mid \exists X \phi \mid x \in X$$

where lower cases x are node variables, and upper cases X are set variables. This syntax is enriched by different relations to describe the structure of the objects we consider:

- For binary trees (BT), we add two relations $\mathsf{Ch}_L(x, y)$ and $\mathsf{Ch}_R(x, y)$ that express that y is the left child (resp. right child) of x.
- For unranked ordered forests (OF), we add $\mathsf{FC}(x, y)$, that expresses that y is the first child of x, and $\mathsf{NS}(x, y)$ that express that y is the brother directly to the right of x.
- For unranked unordered forests (UF), we only add the relation $\mathsf{Ch}(x, y)$, that expresses that y is a child of x. The relation "Sibling" would only be syntactic sugar.

An *MSO-definable transformation* with n copies is a transformation that for each input node x, makes n output nodes $x_1, \dots, x_n$. The presence or absence of an edge in the output are dictated by formulae defining the transformation. A MSO-definable transformation is characterized by its formulae $\varphi_{\mathcal{R},i,j}$ for each $1 \leqslant i, j \leqslant n$, and each structure relation $\mathcal{R}$ (e.g. FC and NS if the output is ordered forests).

For example, if one wanted to reverse left and right children in binary trees, this would be a transformation definable in $\mathrm{MSO}_{BT\to BT}$ with one copy, where $\varphi_{\mathsf{Ch}_L,1,1}(x, y) = \mathsf{Ch}_R(x, y)$, i.e. y_1 is x_1's left child in the output iff y was x's right child in the input, and conversely $\varphi_{\mathsf{Ch}_R,1,1}(x, y) = \mathsf{Ch}_L(x, y)$.

We note that this definition can express transformations between any two tree algebras. For example, the "FCNS" decoding of Figure 2 can be encoded in MSO from binary trees to UF. Since we will use different combinations of input-output in this part, we introduce the notation $\mathrm{MSO}_{\bullet\to\bullet}$ to denote MSO from one type of trees to the other. $\mathrm{MSO}_{BT\to OF}$ designs MSO-definable functions from binary trees to ordered forests, and $\mathrm{MSO}_{UF\to UF}$ designs MSO-definable functions from unordered forests to unordered forests.

▶ **Proposition 12.** *Every function of $MSO_{UF\to UF}$ can be described by an **UCF**-RA.*

The proof we provide to show this Proposition has three arguments:

1. $\mathrm{MSO}_{UF\to UF}$ can be represented by functions of $\mathrm{MSO}_{BT\to OF}$.
2. Bottom-UP Streaming Tree Transducers (STT) [3] describe all functions of $\mathrm{MSO}_{BT\to OF}$.
3. Bottom-Up STT can be expressed as register automata.

From $\mathrm{MSO}_{UF\to UF}$ to $\mathrm{MSO}_{BT\to OF}$. We say that a binary tree t *represents* an unordered forest h if the "FCNS" decoding of t as represented in Figure 2 is h. Note that t is not unique for h, but every t represents a unique h. Similarily, we can say that an unranked ordered forest h *represents* an unordered forest h' if by forgetting the siblings' order in h, we get h'. Once again such an h is not unique for h', but every h represents a unique h'. We can extend this notion to MSO transformation.

▶ **Definition 13.** A function $f \in \mathrm{MSO}_{BT\to OF}$ *represents* a function $f' \in \mathrm{MSO}_{UF\to UF}$ if:

- For every unordered forest h such that f' is defined over h, then there exists at least one binary tree t such that t represents h, and f is defined over t
- For every binary tree t such that f is defined over t, f' is defined over the unordered forests h such that t represents h, and $f(t)$ represents $f'(h)$.

Once again such an f is not unique for f', but every f represents a unique f'. Furthermore, it is always possible to find a representant f for an MSO-definable function f'.

▶ **Lemma 14.** *If $f' \in MSO_{UF\to UF}$, then there exists $f \in MSO_{BT\to OF}$ that represents f'.*

Proof. We start by encoding the input, transforming f' into a function of $\text{MSO}_{BT\to UF}$. To modify f' so that it transforms trees that represent h into $f'(h)$, one has to replace every occurence of $\mathsf{Ch}(x,y)$ into $\varphi_{\mathsf{Ch},i,j}$ by its "FCNS" encoding, i.e. $\exists z \mid \mathsf{Ch}_L(x,z) \wedge \mathsf{Ch}^*_L(z,y)$.

Encoding the output requires to change $\varphi_{\mathsf{Ch},i,j}$ into two relations $\varphi_{\mathsf{FC},i,j}$ and $\varphi_{\mathsf{NS},i,j}$, i.e. to artificially order the siblings of the output forest. To that effect, we note that from the BT-formula $\varphi_{\mathsf{Ch},i,j}$, one can describe in BT-MSO a set $S_{x,i,j} = \{y \mid \varphi_{\mathsf{Ch},i,j}(x,y)\}$. Since $S_{x,i,j}$ is a set of input nodes of a binary tree, it is totally ordered by their occurrence in the infix run. This order can be expressed as a BT-MSO relation. We can then decide to order all the children y_j of the output node x_i.

To find the first child of a node in the output, we say that $\varphi_{\mathsf{FC},i,j}(x,y)$ if j is the first index where $S_{x,i,j} \neq \emptyset$ and y is its first element. Similarily, to find the next sibling of a node in the output, we say that $\varphi_{\mathsf{NS},i,j}(y,z)$ if $\exists x \mid \varphi_{\mathsf{Ch},k,i}(x,y) \wedge \varphi_{\mathsf{Ch},k,j}(x,z)$, and either $i = j$ and y, z are consecutive elements of $S_{x,k,i}$, or y is the last element of $S_{x,k,i}$, z is the first element of $S_{x,k,j}$, and j is the first index bigger than i such that $S_{x,k,j} \neq \emptyset$. ◀

From $\text{MSO}_{BT\to OF}$ to STT to RA. The next step is to use an existing result from the literature [3] that describes a model of transducers that describes all $\text{MSO}_{BT\to OF}$. The formalism in question are *Streaming Tree Transducers* (STT). A STT is an automaton on nested words (words representing trees) that maintains a stack of register configurations. The nesting of the words dictates how this stack behaves: each opening letter $<a$ stores the current variable values in the stack to start with fresh ones, then each closing letter $a>$ uses the current variable values and the top of the stack to generate new values for the registers.

In [3], STT are limited to linear functions for the update of their value. Furthermore, the paper proves that without loss of expression, one can consider *Bottom-Up STT*, where reading an opening symbol $<a$ resets the state as well as the registers. On such STT, the behavior of a STT reading the nested word of a subtree does not depend on what occurs before or after, and its computation behaves like a register automaton reading a tree in a bottom-up manner. We will consider the class of Bottom-Up STT that read nested representations of binary trees (Bottom-Up BT STT).

▶ **Proposition 15.** *Every function of $\text{MSO}_{BT\to UF}$ is described by a Bottom-Up BT STT.*

▶ **Proposition 16.** *Every function of a Bottom-Up BT STT is described by an OCF-RA.*

Proposition 15 comes directly from Theorems 3.7 and 4.6 of [3]: 3.7 explains Bottom-Up BT-STT are as powerful as general BT-STT, and 4.6 states that STT can describe any function of $\text{MSO}_{BT\to UF}$. Proposition 16 is not directly proven in [3] but their definition of Bottom-Up STT is made specifically to that end. We provide more details in the appendix.

End of Proof. To turn that OCF-RA into an **UCF**-RA, we just have to change the ordered concatenation of OCF to the unordered concatenation of **UCF**. By combining Lemma 14, Proposition 15 and Proposition 16, we conclude our proof of Proposition 12.

We note that every MSO-definable function can be described by a **UCF**-RA, however the converse is not true; consider a function that creates output of exponential size (whereas MSO can only describe functions of linear size increase). Consider unary input trees of form $\mathsf{root}_a^n(\bot)$, and a 1-counter **UCF**-RA with rules $\bot \to q(\mathsf{root}())$, and $a(q(h)) \to h + h$. The image doubles in size each time a symbol is read. Unsurprisingly, this counterexample uses the copyful nature of **UCF**-RA, as copyless restrictions tend to limit the expressivity power of register automata to MSO classes [2, 3].

4 On decidability of MTT equivalence. Equivalence of polynomials-RA with composition is undecidable

In this section, we tru to use the "Hilbert Method" to study the equivalence problem on Macro Tree Transducers (MTT) [11]. MTT have numerous definitions. For this paper, we will consider them to be register automata on an algebra of ranked trees with an operation of substitution on the leaves; observe this is exactly $\mathsf{OrderedTrees}[X]^{\mathsf{subs}}$-RA. The algebra $\mathsf{OrderedTrees}$ (ranked trees without substitution on the leaves) can be simulated by words with concatenation (via nested word encoding). Words with concatenation can be encoded by $\mathbb{Q}$ (see, for example, the proof of Corollary 10.11 [6]). Thus, $\mathsf{OrderedTrees} \preceq_{\mathsf{pol}} \mathbb{Q}$. Finally, by Lemma 2, we have that $\mathsf{OrderedTrees}[X]^{\mathsf{subs}} \preceq_{\mathsf{pol}} \mathbb{Q}[X]^{\mathsf{subs}}$. This means that if equivalence is decidable for $\mathbb{Q}[X]^{\mathsf{subs}}$-RA, then MTT equivalence is decidable. Unfortunately, we will show that even with one variable x, the register automata of $\mathbb{Q}[x]^{\mathsf{subs}}$-RA have undecidable functionality and equivalence:

▶ **Theorem 17.** *The functionality problem for $\mathbb{Q}[x]^{\mathsf{subs}}$-RA and equivalence problem for functional $\mathbb{Q}[x]^{\mathsf{subs}}$-RA are undecidable.*

We prove this undecidability result by reducing the reachability problem for 2-counter machines to the equivalence problem on deterministic $\mathbb{Q}[x]^{\mathsf{subs}}$-RA with a monadic input (i.e. that reads words rather than trees). This means that the actual theorem we prove is slightly more powerful than Theorem 17. Its full extent is described in Theorem 20. We recall the definition of a 2-counter machine.

▶ **Definition 18.** A 2-counter machine (2CM) is a pair $M = (Q, \delta)$, where:
- Q is a finite set of states,
- $\delta : Q \times \{0,1\} \times \{0,1\} \to Q \times \{-1,0,1\} \times \{-1,0,1\}$ is a *total* transition function.

A configuration of M is a triplet of one state and two nonnegative integer values (or counters) $(q, c_1, c_2) \in \mathbb{Q} \times \mathbb{N} \times \mathbb{N}$. We describe how to use transitions between configurations: $(q, c_1, c_2) \to (q', c_1', c_2')$ if there exists $(q, b_1, b_2) \to (q', d_1, d_2)$ in δ such that for $i \in \{1, 2\}$: $c_i = 0 \iff b_i = 0$ and $c_i' = c_i + d_i$. Note that to ensure that no register wrongfully goes into the negative, we assume wlog that if there exists $(q, b_1, b_2) \to (q', d_1, d_2)$ in δ, then $d_i = -1 \implies b_i = 1$ (i.e. we can only decrease a non-zero counter).

The *2CM reachability problem* can be expressed as such: starting from an initial configuration $(q_0, 0, 0)$, can we access the state $q_f \in Q$, i.e. is there a configuration (q_f, c_1, c_2) such that $(q_0, 0, 0) \to^* (q_f, c_1, c_2)$. It is well known that 2CM reachability is undecidable.

Reduction of 2CM Reachability to $\mathbb{Q}[x]^{\mathsf{subs}}$-RA Equivalence. Let $M = (Q, \delta)$ be a n-states 2CM. We rename its states $Q = \{1, \dots, n\}$. We consider the 2CM reachability problem in M from state 1 to state n.

We simulate this machine with a $\mathbb{Q}[x]^{\mathsf{subs}}$-RA M'. It will have only one state $q_{M'}$: the configurations (q, c_1, c_2) of M will be encoded in 3 registers of M'. It will work on a signature $\bot \cup \delta$, where $\bot$ is of rank 0 and every transition $(q, b_1, b_2) \to (q', d_1, d_2)$ of δ is a symbol of rank 1. Intuitively, reading a symbol $(q, b_1, b_2) \to (q', d_1, d_2)$ in M' models executing this transition in M. The automaton will have 6 registers: 3 to encode the configurations of M' and 3 containing auxiliary polynomials useful to test if the input sequence of transitions describes a valid computation in M.

We encode the configurations (q, n_1, n_2) in 3 registers as follows:
- register r_q holds the (number of the) current state.
- registers r_{c_1}, r_{c_2} hold n_1, n_2 – the current values of the counters.

We now encode transitions in M as register operations in M'. When reading a transition $(i, b_1, b_2) \to (j, d_1, d_2)$, the update of the configuration is natural. However, we must ensure that we are allowed to use this transition in the current configuration.

To this end, we keep in M' a witness register. Its value will be 0 if and only if the sequence of transitions read as an input does not constitute a valid path in M. To update such a register, when a transition $(i, b_1, b_2) \to (j, d_1, d_2)$ is read, we need to check that $r_q = i$ and that $r_{c_i} = 0 \iff b_i = 0$.

For the state $q \in \{1, \dots, n\}$, we design $P_{q=i}$, a polynomial such that $P_{q=i}(i) \neq 0$ and for every other value $1 \leqslant j \leqslant n$, $P_{q=i}(j) = 0$: $P_{q=i}(x) := \prod_{1 \leqslant j \leqslant n}^{i \neq j} (x - j)$. This approach cannot work for counters, as there is no absolute bound to their value. To remedy that problem, we will design *for each* m a polynomial $P_{c=0}^{\leqslant m}$ such that $P_{c=0}^{\leqslant m}(0) \neq 0$ and for every other value, $1 \leqslant j \leqslant m$, $P_{c=0}^{\leqslant m}(j) = 0$. $P_{c=0}^{\leqslant m}(x) := \prod_{1 \leqslant j \leqslant m} (x - j)$. Intuitively, $P_{c=0}^{\leqslant m}$ works as a test for counters in the m-th step of M, since counters c_1, c_2 cannot exceed the value m at that point. This means that $P_{c=0}^{\leqslant m}$ will have to be stored and updated in a register of its own. To this end, we introduce the last three registers of M':

- the register r_+. After m steps, $r_+ = m$.
- the register r_{zt}. After m steps, $r_{zt} = P_{c=0}^{\leqslant m}$.
- the witness register r_w. After m steps, $r_w \neq 0 \iff$ we read a valid path in M.

We describe how to update the registers of M' when reading an input symbol $(i, b_1, b_2) \to (j, d_1, d_2)$. Note that according to our definition of $\mathbb{Q}[x]^{\mathsf{subs}}$-RA, the new values $\overline{r}'$ are computed as a function of the old value of $\overline{r}$. This means that any value on the right of the assignation symbol $\leftarrow$ is the value before reading $(i, b_1, b_2) \to (j, d_1, d_2)$.

- $r_q \leftarrow j$, $r_{c_1} \leftarrow r_{c_1} + d_1$, $r_{c_2} \leftarrow r_{c_2} + d_2$,
- $r_+ \leftarrow r_+ + 1$, $r_{zt} \leftarrow r_{zt} \times (x - r_+ - 1)$,
- $r_w \leftarrow r_w \times T_q \times T_1 \times T_2$, where:
  - $T_q = (P_{q=i})[x := (r_q)]$,
  - for $i \in \{1, 2\}$, $T_i = \begin{cases} r_{c_i} \text{ if } b_i = 1, \\ (r_{zt})[x := (r_{c_i})] \text{ if } b_i = 0. \end{cases}$

This update strategy ensures that each counter does what we established its role to be. The only register for which this is not trivial is r_w. We show that $r_w = 0$ if and only if we failed to read a proper path in M.

We proceed by induction on the number of steps. The induction hypothesis is that a mistake happened before the m-th step if and only if $r_w = 0$ before reading the m-th symbol. If such is the case, r_w will stay at zero for every subsequent step, as the new value of r_w is always a multiple of the previous ones. If the error occurs exactly at the m-th step, it means that the m-th letter of the input was a transition $(i, b_1, b_2) \to (j, d_1, d_2)$, but r_q was not i (and hence $T_q = (P_{q=i})[x := (r_q)] = 0$), or that for this transition to apply we need the counter c_i to be 0 when it was not (or conversely assumed it > 0 when it was 0). This last case is caught by T_i. By using $T_i = r_{c_i}$, we have $T_i = 0$ exactly when we were wrong. If $b_i = 0$ then we assume $c_i = 0$. We know that $c_i \leq m$, where m is the number of step taken. By using $T_i = (r_{zt})[x := (r_{c_i})] = P_{c=0}^{\leqslant m}(c_i)$, we have that $T_i = 0$ exactly when $0 < c_i \leq m$.

The final step of the reduction comes by picking the output function for the only state of M'. We pick $f(\overline{r}) := (P_{q=i})[x := (r_q)] \times r_w$. The only way for the output to not be 0 is if r_q ends in n (i.e. we reached state n) and if $r_w \neq 0$ (i.e. we used a valid path). In other words, the following Lemma holds.

▶ **Lemma 19.** *$[\![M']\!]$ is the constant 0 function if and only if n is not reachable from 1 in M*

By comparing M' to a $\mathbb{Q}[x]^{\mathsf{subs}}$-RA M_0 performing the constant 0 function, we get that deciding equivalence on functional $\mathbb{Q}[x]^{\mathsf{subs}}$-RA would allow to decide 2CM Reachability. Similarly, running nondeterministically M' and M_0, we get that deciding functionality on $\mathbb{Q}[x]^{\mathsf{subs}}$-RA would allow to decide 2CM Reachability. This leads to the proof of Theorem 17. More than that, we show a more thorough result:

▶ **Theorem 20.** *The equivalence problem for deterministic $\mathbb{Q}[x]^{\mathsf{subs}}$-RA is undecidable, even with a monadic input alphabet. The functionality problem for $\mathbb{Q}[x]^{\mathsf{subs}}$-RA is undecidable, even with a one-letter monadic alphabet.*

The first point is given directly by the point above: M' is a deterministic $\mathbb{Q}[x]^{\mathsf{subs}}$-RA on a monadic alphabet $\bot \cup \delta$, that is to say, the input of M' is a word where each letter is an element of δ.

For the second point, we imagine a slight alteration of this proof where the input alphabet is $\{a, \bot\}$ where $\bot$ is of rank 0 and a of rank 1, that is to say, the input of the $\mathbb{Q}[x]^{\mathsf{subs}}$-RA would be a word of form a^k. In this new version, M' is no longer deterministic, but guesses each time what transition of M to emulate. When M' reads a^k, it either guesses a correct path of length k, or makes a mistake and returns 0. M' is functional iff it cannot guess a run that produces something else than 0, i.e. iff n is not reachable from 1 in M.

5 Conclusion

We use "Hilbert Methods" to study equivalence problems on register automata. To apply these methods to register automata on contexts, we consider algebras with a substitution operation. To show the decidability of equivalence on **UCF**-RA, a class that subsumes MSO-definable transformations in unordered forests, we use the fact that bounded degree substitution can be encoded into $+, \times$ in $\mathbb{Q}[X]$. However, when applying the same method to Macro Tree Transducers, we are led to consider register automata on $\mathbb{Q}[X]^{\mathsf{subs}}$, whose equivalence we prove to be undecidable. In essence, for the "Hilbert Methods" we consider to provide positive results, it seems necessary to limit the use of composition.

Future developments of this work could then consist of finding other acceptable restrictions on the use of composition in $\mathbb{Q}[X]$ that still allows for decidability results in register automata. Another possible avenue is to use the properties of $\preceq_{\mathsf{pol}}$ to prove negative results: if $\mathbf{A} \preceq_{\mathsf{pol}} \mathbf{B}$, and register automata have undecidable problems in $\mathbf{A}$, then this negative results propagates to $\mathbf{B}$. Finally, "Hilbert Methods" can apply to a huge variety of algebras (e.g. **UCF** in this paper or $\mathbb{Q}^n$ in [5]). They provide decidability results on register automata on algebras with nontrivial structure properties like commutativity of operations (e.g. children in **UCF**) that make the usual methods to decide equivalence difficult to apply.

References

1 Michael H. Albert and J. Lawrence. A Proof of Ehrenfeucht's Conjecture. *Theor. Comput. Sci.*, 41:121–123, 1985.

2 Rajeev Alur and Pavol Cerný. Expressiveness of streaming string transducers. In *FSTTCS*, volume 8 of *LIPIcs*, pages 1–12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

3 Rajeev Alur and Loris D'Antoni. Streaming Tree Transducers. *J. ACM*, 64(5):31:1–31:55, 2017. doi:10.1145/3092842.

4 Rajeev Alur, Loris D'Antoni, Jyotirmoy V. Deshmukh, Mukund Raghothaman, and Yifei Yuan. Regular Functions and Cost Register Automata. In *LICS*, pages 13–22. IEEE Computer Society, 2013.

5 Michael Benedikt, Timothy Duff, Aditya Sharad, and James Worrell. Polynomial automata: Zeroness and applications. In *LICS*, pages 1–12. IEEE Computer Society, 2017.

6 Mikołaj Bojańczyk. Automata toolbox, May 2018. URL: `https://www.mimuw.edu.pl/~bojan/paper/automata-toolbox-book`.

7 Mikołaj Bojańczyk and Igor Walukiewicz. Forest algebras. In *Logic and Automata*, volume 2 of *Texts in Logic and Games*, pages 107–132. Amsterdam University Press, 2008.

8 Christian Choffrut. Minimizing subsequential transducers: a survey. *Theor. Comput. Sci.*, 292(1):131–143, 2003.

9 Bruno Courcelle. Monadic Second-Order Definable Graph Transductions: A Survey. *Theor. Comput. Sci.*, 126(1):53–75, 1994.

10 Joost Engelfriet and Sebastian Maneth. Macro Tree Translations of Linear Size Increase are MSO Definable. *SIAM J. Comput.*, 32(4):950–1006, 2003.

11 Joost Engelfriet and Heiko Vogler. Macro Tree Transducers. *J. Comput. Syst. Sci.*, 31(1):71–146, 1985.

12 Juha Honkala. A short solution for the HDT0L sequence equivalence problem. *Theor. Comput. Sci.*, 244(1-2):267–270, 2000.

13 Karel Culik II and Juhani Karhumäki. The Equivalence of Finite Valued Transducers (On HDT0L Languages) is Decidable. *Theor. Comput. Sci.*, 47(3):71–84, 1986.

14 Sebastian Maneth. A Survey on Decidable Equivalence Problems for Tree Transducers. *Int. J. Found. Comput. Sci.*, 26(8):1069–1100, 2015. `doi:10.1142/S0129054115400134`.

15 William C. Rounds. Mappings and Grammars on Trees. *Mathematical Systems Theory*, 4(3):257–287, 1970.

16 Helmut Seidl, Sebastian Maneth, and Gregor Kemper. Equivalence of Deterministic Top-Down Tree-to-String Transducers is Decidable. In *FOCS*, pages 943–962. IEEE Computer Society, 2015.

A Appendix

Bottom-Up Streaming Tree Transducers. We go more into detail into *Streaming Tree Transducers* (STT), that read and output nested words. This formalism is central to the proof of Proposition 15, and of Proposition 16.

In intuition, an STT works with a configuration composed of a state, a finite number of typed variables (or registers) that contains nested words with at most one occurrence of a context symbol (this corresponds to the Ordered Forest Algebra (OCF) in the sense of [7]), and a stack containing pairs of stack symbols and variable valuations. The nesting of the words dictates how this stack behaves: each opening letter $<a$ stores the current variable values in the stack to start with fresh ones, then each closing letter $a>$ uses the current variable values and the top of the stack to generate new values for the registers. The operations on nested words that can be performed in such cases correspond to polynomial operations on OCF: one can use concatenation, context application (which translates directly into OCF), or use a constant nested word, that can be simulated by the roots and concatenation: $(<a \circ a>)[\circ := r \cdot r']<b\ b>$ can be seen in forests as $\mathsf{root}_a(r + r') + \mathsf{root}_b()$.

The general definitions are available on [3]. We use specifically *bottom-up* STT, where reading an opening symbol $<a$ resets the state as well as the registers. On such STT, the behavior of a STT reading the nested word of a subtree does not depend on what occurs before or after. The original paper also imposes a single-use restriction, to ensure each operation can use each register only once. We can keep this restriction, but will not need it. We add a few restrictions to this model:

- We do not allow letters beyond nesting letters. In the language of [3] this means we ignore internal transitions.
- The input domain is a language of nested words of binary trees.

We will call this subclass Bottom-Up BT STT. The first result we need comes directly from the results of [3] that states that single-use STT (even limited to bottom-up) describe exactly MSO functions on nested words:

▶ **Proposition 21.** *Every function of $MSO_{BT\to UF}$ is described by a Bottom-Up BT STT.*

From STT to UCF-RA. To complete the proof, we show that if a Bottom-Up BT STT describes f, then we can find a **UCF**-RA that describes the function f' that f represents, by forgetting the order in the output.

▶ **Proposition 22.** *Every function of a Bottom-Up BT STT is described by an OCF-RA.*

Proof. We propose in a figure below the run of a bottom-up STT in a tree $c(t, t')$. The subtrees t and t' are of root a and b. The second line corresponds to its configuration (state q, register values $\overline{r}$), while the third line keeps track of the top symbol of the STT's stack. The state q_0 and register valuation $\overline{r_0}$ are respectively the initial state and register values. The symbol that was at the top of the stack when reaching $<c$ is denoted as λ.

$<c$	$<a$	h	$a>$	$<b$	h'	$b>$	$c>$
$(q_0, \overline{r_0})$	$(q_0, \overline{r_0})$	$\dots (q, \overline{r})$	$(q_1, \overline{r_1})$	$(q_0, \overline{r_0})$	$\dots (q', \overline{r'})$	$(q_2, \overline{r_2})$	
λ	$(p_a, \overline{r_0})$	$\dots (p_a, \overline{r_0})$	λ	$(p_b, \overline{r_1})$	$\dots (p_b, \overline{r_1})$	λ	
	t			t'			

Figure 4 The run of a bottom-up STT on a binary tree $c(t, t')$.

From $(q, \overline{r})$, when we read $a>$, we use a transition depending only on q, p_a to get q_1 and apply to $\overline{r}, \overline{r_0}$ a polynomial function depending only on q, p_a. We note that p_a depends only of a. Similarly, from $(q', \overline{r'})$, when we read $b>$, we use a transition depending only on q', p_b to get q_2 and apply to $\overline{r}, \overline{r_0}$ a polynomial function depending only on q', p_b. We note that p_b depends of b and q_1.

This means that, if we have prior knowledge of a, b – the roots of the left and right child of c – and q, q' – the states reached by our STT after reading the left and right child of c – we have enough information to find the state reached by our STT after reading $c(t, t')$. We can call this state $q_{a,b,q,q'}$. From a, b, q, q', we can also deduce the polynomial function that links $\overline{r}, \overline{r'}$ to $\overline{r_2}$, the values of the registers after reading $c(t, t')$. We call such a function $\phi_{a,b,q,q'}$, and it can be seen as a polynomial function on nested words or on OCF.

To find an OCF-RA that computes this function, we say that an input subtree leads to a state (q, a), where a is the label of its root, and q is the state reached by the STT right before reading the final $a>$. The registers have the same values as the STT's right before reading the final $a>$. The transitions of our OCF-RA will then be of form:

$$c((q, a), (q', b)) \to ((q_{a,b,q,q'}, c), \phi_{a,b,q,q'})$$ ◀