Tree Algebras and Bisimulation-Invariant MSO on Finite Graphs

Thomas Colcombet
CNRS, IRIF, Université Paris Cité, France

Amina Doumane
CNRS, LIP, ENS Lyon, France

Denis Kuperberg
CNRS, LIP, ENS Lyon, France

Abstract

We establish that the bisimulation invariant fragment of MSO over finite transition systems is expressively equivalent over finite transition systems to modal μ-calculus, a question that had remained open for several decades.

The proof goes by translating the question to an algebraic framework, and showing that the languages of regular trees that are recognised by finitary tree algebras whose sorts zero and one are finite are the regular ones. This corresponds for trees to a weak form of the key translation of Wilke algebras to omega-semigroup over infinite words, and was also a missing piece in the algebraic theory of regular languages of infinite trees for twenty years.

2012 ACM Subject Classification Theory of computation → Finite Model Theory; Theory of computation → Modal and temporal logics; Theory of computation → Algebraic language theory; Theory of computation → Regular languages

Keywords and phrases MSO, mu-calculus, finite graphs, bisimulation, tree algebra

Digital Object Identifier 10.4230/LIPIcs.ICALP.2025.152

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: https://arxiv.org/abs/2407.12677

Funding *Denis Kuperberg*: ANR ReCiProg.

Acknowledgements We thank Achim Blumensath for helpful discussions, proofreading this work, and finding a major mistake in a previous version of this paper (now solved).

Low CO2 This is a low-CO_2 research paper. The authors did not use plane travel, neither for the development of this work nor its presentation.

1 Introduction

A well-known result of Janin and Walukiewicz [13, Thm 11] states:

▶ **Theorem 1.** *For a property of transition systems, the following statements are equivalent:*
- *being MSO-definable and bisimulation-invariant, and*
- *being μ-calculus-definable*[1]*.*

In which a property of transition systems is bisimulation-invariant if for any two bisimilar transition systems, both satisfy the property, or none.

[1] In this work, *μ-calculus* refers to the standard propositional modal μ-calculus, i.e. the temporal logic constructed from propositions, modalities □ and ◇, boolean connectives, and least and greatest fixpoints.

Keeping the above result in mind, it is enlightening to recall the following well-known finite model property: two μ-calculus sentences that can be separated by some transition system can always be separated by a finite one. In other words, the semantics of a μ-calculus sentence is entirely defined by what it expresses over finite transition systems. For this reason, the question of whether a version of Theorem 1 for finite transition systems would hold is extremely natural. It remained an important open question in the field for almost three decades. One contribution of this paper is to provide a positive answer to it:

▶ **Theorem 2.** *For a property of finite transition systems, the following items are equivalent:*
- *being MSO-definable and bisimulation-invariant, and*
- *being μ-calculus-definable.*

For both Theorem 1 and Theorem 2, the upward direction is the same. It follows from well-known facts: (1) μ-calculus sentences can be effectively translated into equivalent MSO-sentences, and (2) properties definable in μ-calculus are invariant under bisimulation.

The difficult direction is to show that bisimulation-invariant MSO-definable properties can be translated to equivalent μ-calculus sentences. The explanation of why the original approach of Janin and Walukiewicz cannot used in the finite case has been clearly phrased by Blumensath and Wolf [5]:

> The above mentioned result by Janin and Walukiewicz on bisimulation-invariant monadic second-order logic has so far defied all attempts at a similar transfer to the realm of finite structures. The main reason is that the original proof is based on automata-theoretic techniques and an essential ingredient is a reduction to trees, via the unravelling operation. As this operation produces infinite trees, we cannot use it for formulae that are only bisimulation-invariant over finite transition systems.

Our proof of the downward implication of Theorem 2 uses a translation of the problem to an algebraic formalism. Though this approach is similar to other works in the field, it requires to develop several algebraic arguments that are not present in the literature.

Overview of the arguments and key contributions

We present here a high-level overview of the sequence of arguments used in our proof, putting an emphasis on lemmas that involve entirely new arguments. Due to lack of space, only subsets of these arguments will be covered in this extended abstract.

We roughly decompose the arguments in three main parts: from MSO to algebras, from algebras to yield-algebras, and from yield-algebras to μ-calculus.

PART 1: From MSO to algebras. The first part of the proof concerns how to transform monadic second-order logic into an algebraic framework. Technically, it follows the algebraic/compositional approach, which is standard in the field, and does not require to develop new arguments.

1. We abstract transition systems by the notion of systems. Systems are finite structures built by combining symbols from a ranked alphabet. Systems also have an initial vertex and variables $x_1, \ldots, x_n$ (n is the rank of the system). See Section 2.1. Finite transition systems can be seen as systems over a suitably chosen ranked alphabet. See Section 2.5.
2. Systems are equipped with composition operations that allow to build complex systems by combining simpler ones together. In the spirit of category theory, all these operations are uniformly described by a single one, flatten. See Section 2.2.
3. From these composition operations, one derives a notion of algebras, that can be used for recognising sets of systems. See Sections 3.1 and 3.2.

4. At this point, the so-called "composition method" can be used for showing that MSO-definable languages of transition systems are recognisable by rankwise-finite algebras (i.e. an algebra in which all ranks are finite). See Section 3.3.
5. We define morphisms between systems, that correspond to folding the source system to the target system (or unfolding when seen the other way round). Two systems are unfold-equivalent if they share a common unfolding. One can identify the usual notion of "infinite regular tree" as an unfold-equivalence-class of systems. See Section 2.4. We also show that unfold-equivalence is a congruence with respect to flatten (Lemma 7).

PART 2: From algebras to yield-algebras. What we have achieved so far is that we have translated the logical hypothesis of Theorem 2, i.e. an MSO-definable bisimilar-invariant property, into a rankwise-finite algebra that recognises it. The next step toward constructing a μ-calculus formula is to add some form of non-determinism to systems. This is what we review now.

6. We generalise the notion of systems to the richer notion of set-systems. The generalisation goes along two directions: (1) a form of non-determinism, and (2) some special vertices called "root vertices" are added (we will not discuss this point for the moment). See Section 2.1. Again, these set-systems are equipped with composition operations via a flatten operation. See Section 2.2.
7. Using the notion of morphism, we can define what are the systems that a set-system can produce: its yields. Yields are, up to unfold-equivalence, systems that can be produced by "resolving the non-determinism". Two set-systems are yield-equivalent if they share the same yields. Over systems, yield-equivalence coincides with unfold-equivalence. See Section 4.1.
8. As we did for systems, we also have to study how yield-equivalence interacts with the operations of composition of set-systems, and as before, the conclusion is that yield-equivalence is a congruence with respect to flatten See Section 4.3.
9. As a consequence, there is a notion of yield-algebra, which corresponds to set-systems modulo yield-equivalence. Such yield-algebras can be used for recognising sets of systems.
10. A key contribution of this work is Lemma 20: A language of regular-trees which is recognisable by a rankwise-finite algebra is recognisable by a rankwise-finite yield-algebra. See Section 4.4.

PART 3: From yield-algebras to μ-calculus. At this moment of the proof, the original MSO-definable bisimulation-invariant property has been transformed into a rankwise-finite yield-algebra. The final arguments involve a careful study of such algebraic objects that we overview in Section 5:

11. We first identify inside yield-algebras some elements that we call deterministic. Deterministic elements form a sub-yield-algebra.
12. We then introduce the key automaton property for a yield-algebra that recognises a language L: it signifies informally that "all elements can be under-approximated by deterministic ones in a faithful way".
13. The reason of this definition is that if a language of regular-trees L is recognised by a rankwise-finite yield-algebra that has the automaton property, there exists an automaton of infinite trees (in the classical sense) recognising L. If the language is furthermore bisimilar-invariant, by adapting the techniques of [13], one can produce a μ-calculus formula along these lines.
14. The last missing piece, and the second technical contribution of this work is Lemma 25: Syntactic rankwise-finite yield-algebras have the automaton property.

Related work

Bisimulation-invariant logics. The seminal Hennessy-Milner theorem [12] gives a first characterisation of modal logics via bisimulation-equivalence. Van Benthem generalised it to First-Order logic (FO) in [20]. The result was then transferred to finite structures [18], and specialised to particular classes of finite structures [9, 11]. In [14], the equivalence between the bisimulation-invariant fragment of monadic path logic with CSL* is shown. An intermediate logic strictly between FO and MSO, obtained by extending FO with fixed-point operators, is considered in [16], where it is shown that the two-way bisimulation-invariant fragment of this logic is equivalent to two-way μ-calculus on finite structures. Concerning the logic MSO, as already explained, Janin and Walukiewicz prove the equivalence with μ-calculus on general structures [13]. The case of weak Monadic logic was shown to correspond to continuous μ-calculus in [8].

On finite structures, this equivalence was shown for structures of bounded Cantor-Bendixson rank in [5].

Composition method and tree algebra. The composition method that we use here was pioneered in [10] for FO and used intensively for MSO in [19, 17]. There is a long line of research pursuing a well-behaved notion of algebra for regular languages of infinite words and trees [21, 15, 1, 6, 2, 3]. The theory has inherent difficulties. For instance [7] gives an example of rankwise-finite algebras that define non-regular languages of infinite trees. In [4], the specific notion of powerset is investigated, it shows that distributivity laws do not exist if there are non-linear identities in the theory (and this is relevant to the present work).

2 Set-systems and Systems

In this section, we introduce systems and set-systems. It may be meaningful to read this part first while concentrating on systems.

We define set-systems and systems in Section 2.1, how to compose them in Section 2.2, and the notion of [set]-context in Section 2.3. In Section 2.4 we introduce morphisms of set-systems, and the notions of folding, unfold-equivalence and regular-trees. We finally explain what are transition systems, how these can be encoded as systems, and how to phrase bisimulation in this framework (Section 2.5).

2.1 Set-Systems and Systems

A *ranked set* Σ (or *alphabet*) is a family of sets Σ_n indexed by natural numbers. We write $a \in \Sigma$ for $a \in \Sigma_n$ for some n. For $a \in \Sigma$, the *rank* of Σ, noted $\mathrm{rk}(a)$, is the integer n such that $a \in \Sigma_n$. A *map of ranked sets* f from Σ to Γ is a map that preserves the ranks, i.e. a family of maps $f_n \colon \Sigma_n \to \Gamma_n$ for all $n \in \mathbb{N}$. A ranked set Σ is *rankwise-finite* if Σ_n is finite for all n. Note that ranked sets equipped with their maps form a category, that we call the *category of ranked sets*. We fix an infinite set $\mathsf{Vars} = \{x_1, x_2, \dots\}$ of *variables*. Variables can be seen as having rank 0. For $n \in \mathbb{N}$, we denote with $[n]$ the set $\{1, \dots, n\}$, and $\mathsf{Vars}[n]$ the variables $\{x_1, \dots, x_n\}$.

Let us define now the notions of set-systems and systems, illustrated on Figure 1.

▶ **Definition 3** (set-systems). *Given a ranked set Σ, a Σ-set-system of rank n is a tuple $S = (V, Vi, Vr, label, Edges)$, composed of*

- *a finite set of vertices V, a set of initial vertices $Vi \subseteq V$, a set of root vertices $Vr \subseteq V$,*
- *a labelling function $label : V \to \Sigma$; in practice, we simply write $S(s)$ for $label(s)$,*

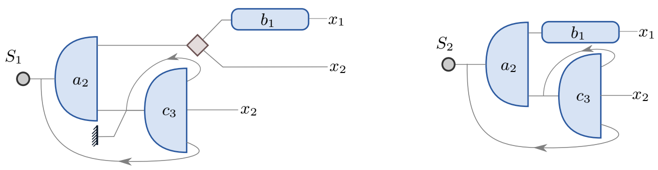


Figure 1 A Σ-set-system S_1 and a Σ-system S_2, both over variables $\{x_1, x_2\}$ (i.e. S_1 and S_2 are of rank 2), for Σ containing symbols b_1 of rank 1, a_2 of rank 2, and c_3 of rank 3. The topmost outgoing edge of a symbol has direction 1, the next one 2, and so on. The circle ◯ is placed before an initial vertex, the symbol ⫳ is placed before a root vertex, and ◇ emphasizes the presence of multiple successors in the same direction (i.e. non-determinism). Implicitly, edges are directed from left to right unless explicitly using an arrow notation.

- *an edge relation* $Edges \subseteq V \times \mathbb{N} \times (V \uplus \{x_1, \ldots, x_n\})$ *consisting of edges of the form* (v, d, f) *with* $d \in [\mathrm{rk}(S(v))]$*;* d *is called the direction of the edge; if* f *is a vertex, then* (v, d, f) *is called a transition edge; otherwise* f *is a variable and* (v, d, f) *is a variable edge. We also denote* $Edges(v)$ *the set* $\{(d, f) \mid (v, d, f) \in Edges\}$*, and* $Edges(v, d) = \{f \mid (v, d, f) \in Edges\}$*.*

We may add subscripts to these elements to identify the set-system they belong to, e.g. V_S *for the set of vertices of* S*. We will note* $S + S'$ *the disjoint union of* S *and* S'*. A Σ-set-system is closed if it has rank* 0*. Two Σ-set-systems* S, S' *are of the same shape if they differ only on their labelling, i.e.* $V_S = V_{S'}$*,* $Vi_S = Vi_{S'}$*,* $Vr_S = Vr_{S'}$*, and* $Edges_S = Edges_{S'}$*.*

Given a map of ranked sets η *from* Σ *to* Γ*, we denote* $\overline{\eta}$ *the map from Σ-set-systems to Γ-set-systems that sends* S *to* $\overline{\eta}(S) = (V_S, Vi_S, Vr_S, \eta \circ label_S, Edges_S)$*.*

We may drop the mention of the alphabet Σ, and simply talk about set-systems. The variables used in a [set-]system of rank n are $x_1, \ldots, x_n$. We sometimes use other variable names such as x, y, z for convenience if there is no ambiguity about the meaning.

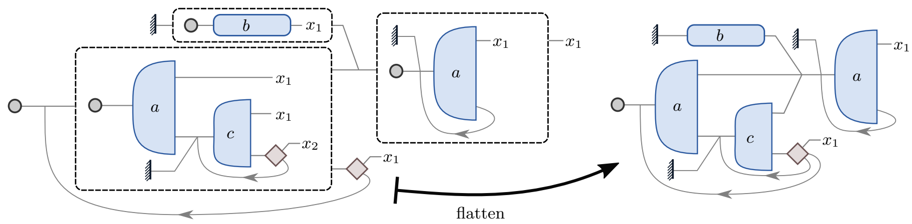
▶ **Definition 4** (systems)**.** *A Σ-system* T *is a Σ-set-system such that* Vi_S *is a singleton,* Vr_S *is empty, and for all vertices* $v \in V_S$ *and all directions* $d \in [\mathrm{rk}(label(v))]$*,* $Edges_S(v, d)$ *is a singleton.*

2.2 Composing systems and set-systems: the flatten operation

We describe in this section the operations used for composing set-systems (or systems), allowing to build complex ones out of simpler elements. We do it using the categorical approach of monads: the idea is to have a single complex operation of composition, called flatten, that can then be specialized into simpler ones.

The operation flatten, takes a set-system of set-systems as input, and produces a set-system as output. We explain it here through an example: the following picture shows a set-system of set-systems and its *flattening*. The subsystems are drawn inside the dashed boxes, and boxes are organised themselves as a set-system structure that we call the pattern[2]. Direction names for edges going out of a node v will always be implicit, the topmost one being 1, then 2, and so on, ending with $\mathrm{rk}(label(v))$ for the bottommost.

[2] Note that the definition enforces that the rank of subsystems (i.e. the number of variables they use) should be consistent with the number of outgoing directions of the vertex in the pattern.



As is shown, the flatten operation glues the subsystems together, keeping their internal structures. If an edge of a subsystem ends in a variable x_d, then it is connected instead to all the initial vertices of the subsystems reachable following direction d in the pattern, as well as to the variables reachable following direction d. After flattening, the initial vertices are the initial vertices of the subsystems that were themselves initial vertices in the pattern. Root vertices are treated differently: the root vertices from subsystems remain root vertices during the process, while initial vertices of subsystems that are root get promoted to roots.

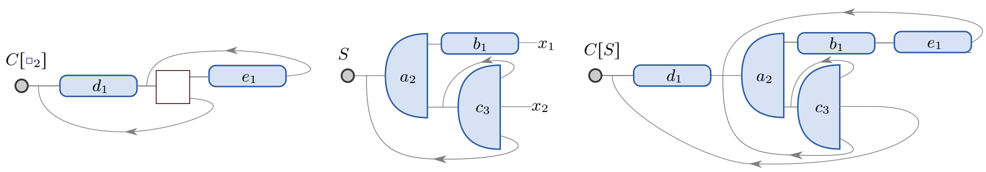
Note that the flatten operation, when given a system of systems as input, produces a system. Hence, the same operation shall be used for composing systems or set-systems depending on the context. In a categorical description, flatten, together with the corresponding operation unit (that turns a single symbol into a one-vertex system), equips the set-system definition (resp. system definition) with a monad structure.

2.3 Set-contexts, and contexts

A special form of composition, which is derived from flatten is the notion of set-context, which are set-systems with a hole that can be filled.

A *Σ-set-context* with a *k-hole* is a $(\Sigma \uplus \{\square_k\})$-set-system, in which $\square_k$ is a new symbol of rank k, called the *hole symbol*, and which labels exactly one vertex, called the *hole vertex*. Set-contexts are denoted $C[\square_k]$, $D[\square_k]$, ... A set-context is *closed* if it is of rank 0. Given a Σ-set-context $C[\square_k]$ and a Σ-set-system S of rank k, $C[S]$ denotes the Σ-set-system obtained by substituting S for the hole. A *context* is a set-context which is a system.

An example of a closed context $C[\square_2]$, a system S of rank 2, and the result $C[S]$ are pictured below. The hole vertex is depicted as an empty box.



Formally, this amounts to construct the (set-system)-set-system S' of the same shape as C such that $S'(h) = S$ for h the hole vertex of C, and $S'(s) = \text{unit}(C(s))$ for all other vertices $s \in V_C$. We then define $C[S]$ to be $\text{flatten}(S')$.

Note in particular that this substitution follows the "rules of flattening", which means that (1) if the hole vertex is initial in S, then all initial vertices of S remain initial in $C[S]$, (2) all the root vertices of S remain root vertices in $C[S]$, and (3) if the hole vertex is root in S, then all initial vertices of S get to be promoted into root vertices in $C[S]$.

Depending on the situations, we shall use flatten or [set-]contexts. In many situations, this is just a choice of presentation.

2.4 Morphisms, unfoldings, and regular-trees

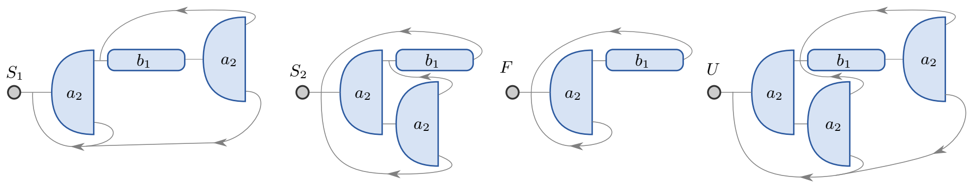
So far, we have seen systems and set-systems as “rigid structures”. The notions of morphisms allows us to compare them in more subtle ways.

Set-systems come with a natural definition of morphisms: a *morphism of set-systems* from S to S', set-systems of same rank, is a map from vertices of S to vertices of S' that preserves initial vertices, root vertices and edges.

If we specialise morphisms to Σ-systems we obtain the notions of unfolding, unfold-equivalence and regular-trees:

▶ **Definition 5** (unfolding and regular-trees)**.** *If there is a morphism of set-systems from a Σ-system T to another one T', then T is called an* unfolding *of T', and T' a* folding *of T. Two Σ-systems that have a common unfolding are called* unfold-equivalent[3]*. A* Σ-regular-tree *is an unfold-equivalence class of Σ-systems.*

For instance, in the pictures below, the systems S_1 and S_2 have both U as unfolding, and thus are unfold-equivalent. The system F is a folding of both S_1 and S_2:



▶ Remark 6. Note that if one would “*fully unfold*” a system (as is classically done), one would obtain an infinite tree which is “regular in the classical sense” (meaning that it has a finite number of subtrees), and that all regular trees (in the classical sense) can be obtained as the full unfolding of some system. It is also easy to check that two systems are unfold-equivalent if and only if they have isomorphic full unfoldings. Hence, the map which to a regular tree (in the classical sense) associates the set of systems that fully unfold to it is a bijection between regular trees (in the classical sense) and regular-trees as defined in this work.

At this point, one can prove that unfold-equivalence interacts nicely with flatten:

▶ **Lemma 7** (unfold-equivalence is a congruence)**.** *Let T, T' be (Σ-system)-systems of the same shape such that $T(t)$ is unfold-equivalent to $T'(t)$ for all vertices t, then* flatten(T) *and* flatten(T') *are unfold-equivalent.*

Equivalently, for all Σ-contexts $C[\square_k]$ and unfold-equivalent systems S, S' of rank k, then $C[S]$ and $C[S']$ are unfold-equivalent.

From a categorical point of view, this means that systems modulo unfold-equivalence, i.e. regular-trees, equipped with the flatten and unit operations form a monad.

2.5 Transition systems as systems

In this section, we recall what is the standard definition of a transition system, and we explain how these can be seen as systems as defined above. We also explain how bisimilarity can be phrased in this setting.

[3] Showing that it is an equivalence relation requires a proof. Note that this would be equivalent to require to have a common folding, but this direction is of no use in the present work.

A *transition system* (V, i, T, γ) consists of a finite[4] set of *vertices* V, an *initial vertex* i, a binary relation T called the *transition relation*, and a labelling function $\gamma : V \to 2^{\mathsf{P}}$. We now explain how to encode transition systems as systems over a specific ranked alphabet $\mathbf{Tr}$.

Let $\mathbf{Tr}$ be the ranked alphabet which has an element ν_n of rank n for all $\mathtt{P}$-*valuation* $\nu \in 2^{\mathsf{P}}$, and all $n \in \mathbb{N}$. Given a closed $\mathbf{Tr}$-system S, call its *decoding* the transition system $\mathrm{decode}(S)$ that has the same set of vertices with same labelling, the same initial vertex, and $T(u, v)$ holds if (u, d, v) is an edge for some d. Conversely, S is called an *encoding* of $\mathrm{decode}(S)$. Note that the decoding is unique, while several non-isomorphic $\mathbf{Tr}$-systems may encode the same transition system. Note also that all transition systems admit at least one encoding (we use here the assumption that transition systems are finite).

We define the *bisimilarity* relation over closed $\mathbf{Tr}$-systems as the least equivalence relation that contains unfold-equivalence and the relation "encoding the same transition system".

The following lemma states why this is consistent with the standard terminology.

▶ **Lemma 8.** *Two $\mathbf{Tr}$-systems are bisimilar if and only if their decodings are "bisimilar in the standard sense".*

From now on, we shall only consider transition systems through this encoding as $\mathbf{Tr}$-systems.

3 Algebras

We have seen in the previous section all the necessary material for introducing a natural notion of algebras and using them for recognising languages of systems.
In this section, we define these algebras (Section 3.1), and describe how they can be used to recognise languages of systems (Section 3.2). We then explain why MSO-definable languages of transition systems are recognised by rankwise-finite algebras (Section 3.3).

3.1 Algebras

The following definitions follow the standard approach via monads.

▶ **Definition 9** (algebras). *An algebra $\mathcal{A}$ is a ranked set A together with a map of ranked sets eval (called the evaluation) from A-systems to A, such that*

- $\mathrm{eval}(\mathrm{unit}(a)) = a$, *for all* $a \in A$,
- $\mathrm{eval}(\overline{\mathrm{eval}}(S)) = \mathrm{eval}(\mathrm{flatten}(S))$ *for all (A-system)-systems* S.

A morphism from an algebra $\mathcal{A} = (A, \mathrm{eval})$ to an algebra $\mathcal{A}' = (A', \mathrm{eval}')$ is a map ρ from A to A' such that $\mathrm{eval}'(\overline{\rho}(S)) = \rho(\mathrm{eval}(S))$ for all A-systems S.

An algebra is unfold-invariant *if $\mathrm{eval}(S) = \mathrm{eval}(S')$ for all unfold-equivalent A-systems S, S'.*
Unfold-invariant algebras are called regular-tree algebras. *The ranked set of Σ-systems equipped with* flatten *as evaluation is an algebra called the* free algebra generated by Σ, *or the* Σ-free algebra.

▶ **Example 10** (regular-tree algebra). Consider $\mathcal{A} = (A, \mathrm{eval})$ defined by $A_n = \mathcal{P}([n]) \uplus \{\bot\}$, and eval to be mapping an A-system T of rank n to $\mathrm{eval}(T) = \bot$ if there is a "$\bot$" element reachable from the initial vertex. Otherwise $\mathrm{eval}(T)$ is the set of indices $i \in [n]$ such that the variable x_i is reachable in T from the initial vertex. We leave to the reader to check that $\mathcal{A}$ satisfies the identities of algebras. This algebra is unfold-invariant, since the symbol $\bot$ (resp. the variable x_i) is reachable in T from the initial vertex if and only if this is also the case in any unfolding of T.

[4] In this work, as for systems, transition systems are finite.

3.2 Languages and their recognition by algebras

Call *language of Σ-systems* a set of closed Σ-systems. A *language of Σ-regular-trees* is a language of Σ-systems L which is invariant under unfold-equivalence, i.e. such that if T, T' are unfold-equivalent and $T \in L$, then $T' \in L$, for all Σ-systems T, T'.

Let ρ be a morphism (called the *recognising morphism*) from the Σ-free algebra to an algebra $\mathcal{A}$, and $P \subseteq A_0$ (called the *accepting set*), the *language recognised* by $(\mathcal{A}, \rho, P)$ is the set of closed Σ-systems defined as

$$\mathrm{Rec}(\mathcal{A}, \rho, P) := \{T \mid T \text{ closed } \Sigma\text{-system such that } \rho(T) \in P\}.$$

We say that L is recognised by $\mathcal{A}$ if it is recognised by $(\mathcal{A}, \rho, P)$ for some ρ and P.

This is the standard definition of recognition, as for word languages. The only subtlety is that we focus on closed systems, which is reflected in the fact that $P \subseteq A_0$. Quite naturally, we shall be interested in languages that are recognisable by rankwise-finite algebras, i.e. algebras $\mathcal{A} = (A, \mathrm{eval})$ such that A_n is finite for all $n \in \mathbb{N}$.

Note, as expected, that the language of systems recognised by regular-tree algebras are languages of regular-trees.

▶ **Example 11** (reachable symbol)**.** Consider some ranked alphabet Σ, and a set of letters $R \subseteq \Sigma$. Let also $\mathcal{A}$ be the algebra from Example 10, and define ρ as the unique algebra morphism from the Σ-free algebra to $\mathcal{A}$ such that for all symbols $a \in \Sigma$ of rank k:

$$\rho(a(x_1, \dots, x_k)) = \begin{cases} \bot & \text{if } a \in R \\ \{1, \dots, k\} & \text{otherwise.} \end{cases}$$

This morphism sends all Σ-systems S of rank n to $\bot$ if there is a symbol from R reachable from the initial vertex, and otherwise the set of numbers of the variables reachable from the initial vertex otherwise. The language recognised by $(\mathcal{A}, \rho, \{\bot\})$ is the set of closed Σ-systems that contain a symbol of R reachable from the initial vertex. Since this language is preserved under unfold-equivalence, it is a language of regular-trees. Similarly, the language recognised by $(\mathcal{A}, \rho, \{\varnothing\})$ is the set of closed Σ-systems that contain no symbol of R reachable from the initial vertex.

3.3 Monadic second-order logic, and the composition method

We have seen in Section 2.5 how to see transition systems as **Tr**-systems. Let us explain now how logic interacts with this view. We assume the reader familiar with *monadic second-order logic* (*MSO* for short). The signature here is the one of transition systems (one binary relation, unary relations for each predicate, and an initial constant). We shall say that an MSO-sentence ψ is *bisimulation-invariant* if for all bisimilar transition systems S, S', we have $S \models \psi$ if and only if $S' \models \psi$.

Using the "composition method" approach, we get the translation from logic to algebra:

▶ **Lemma 12.** *Given an MSO-sentence φ, the set $L := \{S$ **Tr**-system $\mid \mathrm{decode}(S) \models \varphi\}$ is recognisable by a rankwise-finite algebra.*

4 Yield Algebras

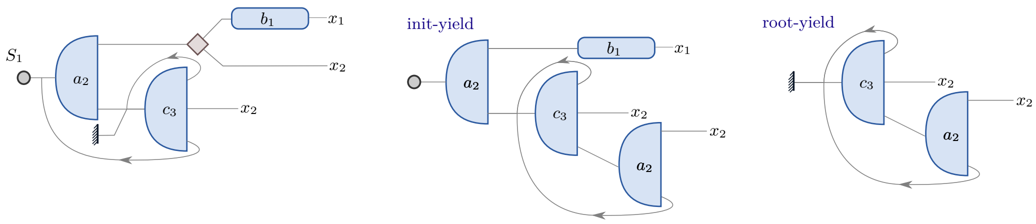
We aim to introduce a richer notion of algebras, called yield-algebras. For this, we need to first give some form of semantics to set-systems. This is done in Section 4.1 with the notion of yields and yield-equivalence. The notions of yield-algebras and recognition by them follows

(Section 4.3). In Section 4.2, we explain a difficulty in working with this definition, and hint at a tool for circumventing it. The section culminates with Lemma 20 explaining why we can restrict our attention to rankwise-finite yield-algebras (Section 4.4).

4.1 Yields of set-systems and yield-equivalence

We shall now see how set-systems can be understood as "finite-non-deterministic machines that would non-deterministically produce systems". This is the notion of yields, that come in two variants: init and root-yields.

We introduce the notion through an example. A set-system is pictured below, as well as one of its init-yields and one of its root-yields.



Informally, an *init-yield* of a set-system S is a system T that unfolds to a system that can be obtained starting from some initial vertex of S by resolving all the non-deterministic choices, potentially partially unravelling the set-system at the same time. A *root-yield* is similar, starting from a root vertex instead of an initial vertex. Note that the definition is such that the set of init-yields (resp. root-yields) of a set-system are closed under unfold-equivalence. The real definition makes use of morphisms of set-systems. We set:

$$\mathrm{InitYields}(S) = \{T \mid T \text{ init-yield of } S\} \quad \text{and} \quad \mathrm{RootYields}(S) = \{T \mid T \text{ root-yield of } S\}.$$

Two set-systems S, S' of same rank are *yield-equivalent* if $\mathrm{InitYields}(S) = \mathrm{InitYields}(S')$ and $\mathrm{RootYields}(S) = \mathrm{RootYields}(S')$.

Note that if S is a system, then an init-yield of S is nothing but an unfold-equivalent system, and there is no root-yield. As a consequence yield-equivalence coincides with unfold-equivalence over systems.

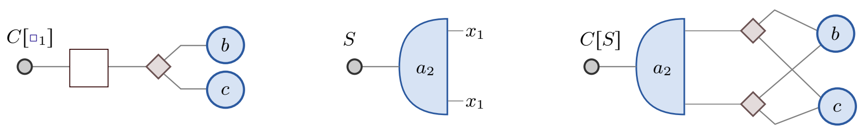
From now, for simplification of the presentation, we shall do as if only init-yields existed.

4.2 Resolutions

This section illustrates a difficulty arising from working with yields, and briefly describes the tool of resolution that we use to circumvent it in the complete version of this work.

At a high level, we would like, given a set-context $C[\square_n]$, to write the yields of $C[S]$ for some set-system S of rank n as a "composition" of the yields of S. The following example shows that it does not work.

▶ **Example 13** (yields are not compositional). Let us work with a ranked alphabet that has symbols b and c of rank 0, and a_2 of rank 2. Consider the set-context $C[\square_1] = \square_1(b + c)$ and the system $S = a_2(x_1, x_1)$:



Then:
- The set-context $C[\square_1]$ has two init-yields: $C_1[\square_1] = \square_1(b)$ and $C_2[\square_1] = \square_1(c)$.
- The system S has exactly one init-yield: S itself.
- The set-system $C[S]$ has four init-yields up to unfold-equivalence: $a_2(b,b)$, $a_2(b,c)$, $a_2(c,b)$, and $a_2(c,c)$.

If we combine the init-yields of $C[\square_1]$ and S in all possible ways, we only obtain $C_1[S] = a_2(b,b)$ and $C_2[S] = a_2(c,c)$. These are indeed init-yields of $C[S]$, but we still miss two of them, namely $a_2(b,c)$ and $a_2(c,b)$. The issue is that the double occurrence of x_1 in $a_2(x_1,x_1)$ forbids to use it in a combination of systems for producing, eg, $a_2(a,b)$.

This situation is in fact a well-identified mathematical difficulty. For instance, this is what prevents the existence of a distributive law with the powerset monad over the monads of non-linear trees [4].

To circumvent this problem, we introduce the notion of *resolution*, which combines the yield with some variable renaming.

▶ **Definition 14.** *Given a Σ-set-system S of rank n, a* resolution (T,σ) *of S consists of:*
- *a map σ from $[m]$ to $[n]$ for some m, and*
- *a Σ-system T of rank m, such that*
- *$\widehat{\sigma}(T)$ is an init-yield of S, where $\widehat{\sigma}(T)$ is T with all variables x_i renamed into $x_{\sigma(i)}$.*

In Example 13, we see that $(a_2(x_1,x_2),\sigma)$ with the constant map $\sigma\colon [2] \to [1]$ is a resolution of S, and that it conveys the information that could not be caught by yields.

Now, it makes sense to understand a yield of $C[S]$ as a combination of resolutions of C and S. This notion of resolution is one of the arguments used for showing that yield-equivalence is a congruence:

▶ **Lemma 15** (yield-equivalence is a congruence)**.** *Let S, S' be (Σ-set-system)-set-systems of the same shape such that $T(t)$ is yield-equivalent to $T'(t)$ for all vertices t, then* flatten(S) *and* flatten(S') *are yield-equivalent.*

4.3 Yield-algebras and recognisability

The definition of a yield-algebra corresponds to the notion of algebras naturally arising from set-systems, quotiented by yield-equivalence. Since set-systems are equipped with a form of non-determinism, yield-algebras are naturally endowed with an order $\sqsubseteq$ that provides an inf-semi-lattice structure (note that the non-deterministic sum will be noted $\sqcap$, i.e. as an infimum rather than a supremum; this is a choice of presentation that corresponds to the fact that we see in this work this non-determinism as "adversarial", i.e. controlled by an opponent).

▶ **Definition 16.** *A* yield-algebra $\mathcal{Y}$ *is a ranked set Y together with a map of ranked sets* eval *from Y-set-systems to Y, such that*
- eval(unit(a)) $= a$*, for all $a \in Y$,*
- eval($\overline{\text{eval}}(S)$) $=$ eval(flatten(S)) *for all (Y-set-system)-set-systems S, and*
- *for all unfold-equivalent*[5] *Y-set-systems S, S' of same rank n,* eval(S) $=$ eval(S').

[5] This is a generalised version of unfold-equivalence for set-systems, which relies on the natural notion of locally surjective morphisms.

- *for all Y-set-systems S of rank n and all $f \in Y_n$,*

$$f \sqsubseteq \mathrm{eval}(T) \text{ for all yields } T \text{ of } S \text{ implies } f \sqsubseteq \mathrm{eval}(S).$$

where for $a, b \in Y_n$, we note $a \sqcap b := \mathrm{eval}(a + b)$, and $a \sqsubseteq b$ holds if $a \sqcap b = a$.

For $a \in Y_n$, let $a\Uparrow$ be $\{b \in Y_n \mid a \sqsubseteq b\}$.

A morphism from the yield-algebra $\mathcal{Y} = (Y, \mathrm{eval})$ to the yield-algebra $\mathcal{Y}' = (Y', \mathrm{eval}')$ is a map of ranked sets ρ from Y to Y' such that $\mathrm{eval}'(\overline{\rho}(S)) = \rho(\mathrm{eval}(S))$ for all Y-set-systems S.

▶ **Lemma 17.** *The operation $\sqcap$ is associative, commutative, and idempotent. The relation $\sqsubseteq$ is an order, and $\sqcap$ computes the infimum with respect to $\sqsubseteq$.*

Note that since systems are particular cases of set-systems, yield-algebras are in particular algebras. Furthermore, since yield-equivalence coincides with unfold-equivalence over systems, yield-algebras seen as algebras are unfold-invariant, and hence regular-tree algebras. This downgrading of yield-algebras to regular-tree algebras is made implicitly in the rest of this work. This means that we can use yield-algebras for recognising languages of regular-trees:

▶ **Definition 18.** *A language of regular-trees is recognisable by a yield-algebra $\mathcal{Y}$ if it is recognisable by $\mathcal{Y}$ seen as an algebra using an accepting set of the form $f\Uparrow$ for some $f \in Y_0$.*

The following lemma shows the intention behind the definition.

▶ **Lemma 19.** *Let $\mathcal{Y}$ be a yield-algebra recognising a language L of Σ-systems, then, for all closed Σ-set-systems S, $f \sqsubseteq \mathrm{eval}(S)$ if and only if $\mathrm{InitYields}(S) \subseteq L$.*

4.4 rankwise-finite yield-algebras

One can now state the key lemma of this section (which involves new arguments compared to the literature).

▶ **Lemma 20.** *If a language of regular-trees L is recognised by a rankwise-finite algebra, then it is recognised by a rankwise-finite yield-algebra.*

For other kind of algebras, such as monoids, deterministic automata over words or trees, forest algebras, ..., a similar result is classically obtained by applying a form of powerset construction. Let us explain by an example why this standard approach fails here.

▶ **Example 21** (standard approach fails)**.** Given a rankwise-finite algebra $\mathcal{A}$, the "standard approach" would be to add new elements to $\mathcal{A}$ in order to build a yield-algebra $\mathcal{Y}$ for the same language. With this approach $\mathcal{A}$ is a sub-algebra of $\mathcal{Y}$[6]. This is not possible here.

Indeed, consider the language of regular-trees L over the alphabet Σ with $b, c \in \Sigma_0$, $a_1 \in \Sigma_1$ and $a_2 \in \Sigma_2$ that contains the closed Σ-systems that have all their leaves carrying the same letter (only b's or only c's). Let $\mathcal{A}$ be its syntactic-algebra[7], and P be the accepting set. For simplicity, we identify the letters with their image in the algebra and use $\mathrm{eval}_{\mathcal{A}}$ as recognising morphism. We have $\mathrm{eval}_{\mathcal{A}}(a_1(x_1)) = \mathrm{eval}_{\mathcal{A}}(a_2(x_1, x_1))$ since for all closed Σ-contexts $C[\square_1]$, $C[a_1(x_1)] \in L$ if and only if $C[a_2(x_1, x_1)] \in L$. We also have $\mathrm{eval}_{\mathcal{A}}(a_1(b)) \in P$, $\mathrm{eval}_{\mathcal{A}}(a_1(c)) \in P$, but $\mathrm{eval}_{\mathcal{A}}(a_2(b, c)) \notin P$.

---

[6] For instance, if we apply the powerset operation to a monoid $\mathcal{M}$, then by distributive law, we obtain a new monoid $\mathcal{P}(\mathcal{M})$, and $\mathcal{M}$ is isomorphic to the submonoid of $\mathcal{P}(\mathcal{M})$ restricted to its singletons.

[7] It always exists by generic algebraic arguments.

Assume now that we have built a yield-algebra $\mathcal{Y}$ that has $\mathcal{A}$ as a sub-algebra, and $f\Uparrow$ is the accepting part. This implies that $\mathrm{eval}_{\mathcal{Y}}(a_1(x_1)) = \mathrm{eval}_{\mathcal{Y}}(a_2(x_1, x_1))$. However, when composed with $C[\square_1] = \square_1(b + c)$ (as in Example 13), we would have by Lemma 19 $f \sqsubseteq \mathrm{eval}_{\mathcal{Y}}(C[a_1(x_1)])$ since $\mathrm{InitYields}(C[a_1(x_1)]) \subseteq L$ but $f \not\sqsubseteq \mathrm{eval}_{\mathcal{Y}}(C[a_2(x_1, x_1)])$ since $\mathrm{InitYields}(C[a_2(x_1, x_1)]) \ni a_2(b, c) \notin L$. A contradiction.

Our argument for avoiding the problem and proving Lemma 20 involves two ideas. Let $\mathcal{A}$ be a rankwise-finite regular-tree algebra recognising a language of regular-trees L.

Idea 1. Define the map profile which maps each A-set-systems S to the set $\{(\mathrm{eval}(T), \sigma) \mid (T, \sigma)$ is a resolution of $S\}$. The arguments of Section 4.2 can be used to show that *profile-equivalence* (i.e. the equivalence relation over A-set-systems "having the same profile") is a congruence for flatten over A-set-systems, i.e. for all A-set-contexts $C[\square_n]$ and A-set-systems S_1, S_2 of rank n that are profile-equivalent, then $C[S_1]$ and $C[S_2]$ are profile-equivalent. It follows that A-set-systems quotiented by profile-equivalence is a yield-algebra that recognises L.

The problem is that it is not rankwise-finite: profile-equivalence is too fine.

Idea 2. This is where the second argument is put in action: we consider small resolutions. A resolution (T, σ) is *small* if the map σ satisfies the condition that for each i in its codomain, $|\sigma^{-1}(i)| \leqslant |A_1|$. We define small−profile(S) as profile but considering only small resolutions. Two set-systems are *small–profile-equivalent* if they have the same small profile. Note that over each rank, small-profile-equivalence is of finite index, is coarser than profile-equivalence, and over rank 0 profile-equivalence and small-profile-equivalence coincide.

At this point, a key technical result is used: a "*context smallification lemma*", that has as consequence that for closed A-set-contexts $C[\square_n]$ and A-set-systems S_1, S_2 of rank n that are small-profile-equivalent, we obtain that $C[S_1]$ and $C[S_2]$ are profile-equivalent. Note the only difference in this statement compared to the analogue one for profile-equivalence, which is that here C is assumed to be closed.

The consequence of this is that the *syntactic yield-algebra* for L (defined in a natural way) is a yield-algebra that recognises L, and is coarser than small-profile-equivalence, and hence rankwise-finite. Lemma 20 has been established.

5 The Automaton Property

In this section, we identify a combinatorial property that allows us to translate languages recognisable by yield-algebras to automata or μ-calculus sentences.

Consider some rankwise-finite yield-algebra $\mathcal{Y}$, its *deterministic elements* are the ones obtainable by composing only elements of the two lowest ranks, i.e. from $Y_{\leqslant 1} := Y_0 \uplus Y_1$:

$$\mathrm{Det}(\mathcal{Y})_k := \{\, \mathrm{eval}(S) \mid S \text{ is a } Y_{\leqslant 1}\text{-set-system of rank } k \,\}$$

Note that $\mathrm{Det}(\mathcal{Y})$ is in fact a sub-yield-algebra of $\mathcal{Y}$. The name of deterministic element comes from the following application (we assume knowledge of automata over infinite trees: our automata have finite states, use a parity condition with finitely many priorities, but there is no limit on the cardinality of the size of the input alphabet):

▶ **Lemma 22.** *Let L be a language of regular-trees recognised by $(\mathcal{Y}, \rho, f\Uparrow)$ where ρ maps symbols to deterministic elements. There exists a deterministic top-down parity automaton over infinite trees that accepts the full unfolding of a system T if and only if $T \in L$.*

Consider now that $\mathcal{Y}$ is used to recognise some language of regular-trees L, using the accepting set $P = f{\Uparrow}$ with $f \in Y_0$. Let ρ be the recognising morphism. The automaton property intuitively states that deterministic elements are sufficient for describing the behaviour of all elements when put in a closed context.

▶ **Definition 23.** *$(\mathcal{Y}, P)$ have the automaton property if for all closed Y-set–contexts $C[\square_n]$ and every $a \in Y_n$ with $\mathrm{eval}(C[a]) \in P$, there is a deterministic element $\delta \in \mathrm{Det}(\mathcal{Y})_n$ such that $\delta \sqsubseteq a$ and $\mathrm{eval}(C[\delta]) \in P$.*

It is easy to verify that, if $(\mathcal{Y}, P)$ has the automaton property, setting $\Delta_a := \{\delta \in \mathrm{Det}(\mathcal{Y})_n \mid \delta \sqsubseteq a\}$ for all $a \in Y_n$, we get that for all Y-set-contexts $C[\square_n]$:

$$\mathrm{eval}(C[a]) \in P \quad \text{if and only if} \quad \mathrm{eval}(C[\delta]) \in P \text{ for some } \delta \in \Delta_a\ . \tag{1}$$

Let us very informally attempt to explain how this property allows us to build a finite state automaton over infinite trees that coincides with L over regular trees. This automaton is defined as follows: it guesses a labelling of the tree by deterministic elements such that for all nodes v labelled by some a it associates some $\delta \in \Delta_a$, and then checks that the resulting infinite tree is accepted by the automaton from Lemma 22.

Let us explain now why this automaton accepts the full unfolding of a system T if and only if $\mathrm{eval}(T) \in P$. For this, we first establish, using inductively Equation (1), that:

- $\mathrm{eval}(T) \in P$, if and only if
- there exists a $\mathrm{Det}(\mathcal{Y})$-system R of the same shape as T such that $R(v) \in \Delta_{T(v)}$ for all vertices v, and moreover $\mathrm{eval}(R) \in P$. Let us call such R a *direct run* over T.

The downward implication indicates that if $\mathrm{eval}(T) \in P$, then the automaton accepts the full unfolding of T using the full unfolding of the direct run R as a witness of acceptance. Conversely, if the automaton accepts the full unfolding of T, this means, by Rabin's lemma[8], that there is a regular witness of acceptance over the full unfolding of T. This can be rephrased as the existence of a $\mathrm{Det}(\mathcal{Y})$-system R such that $\mathrm{eval}(R) \in P$ and R is a direct run over some unfolding T' of T. Hence $\mathrm{eval}(T) = \mathrm{eval}(T') \in P$.

If we combine these arguments with the ones of Janin and Walukiewicz, we get:

▶ **Lemma 24.** *If $\mathcal{Y}$ is a rankwise-finite yield-algebra such that $(\mathcal{Y}, f{\Uparrow})$ has the automaton property and recognises a bisimulation-invariant language of $\mathbf{Tr}$-regular trees L, then L is definable by a μ-calculus sentence.*

We obtain this by showing that if we assume that the original language L is bisimulation-invariant, then the automaton built via our procedure going through deterministic elements can be turned into an automaton of the shape requested by [13, Theorem 7]. Thus, we obtain a μ-calculus sentence that defines L, using directly [13].

The last missing argument, which is also a key contribution of this work is to show:

▶ **Lemma 25.** *Let $\mathcal{Y}$ be the syntactic yield-algebra for a language of regular-trees L, and $P \subseteq \mathcal{Y}$ be the accepting set, if $\mathcal{Y}$ is rankwise-finite, then $(\mathcal{Y}, P)$ has the automaton property.*

We recall the general intuition behind syntactic algebra. In general, the syntactic algebra for a language is the minimal one that recognises it, which is reflected by the fact that any two distinct elements in the algebra can be distinguished by the language in some context.

[8] Rabin's lemma states that a non-empty regular language of infinite trees contains a regular tree.

The proof of Lemma 25 is the second place where new arguments are used. Let $C[\square_n]$ be a Y-set-context, and $a \in Y_n$, such that $C[a] \in P$. Our goal is to find a deterministic element δ such that $\delta \sqsubseteq a$ and $C[\delta] \in P$. Notice that here we will informally write $C[\delta] \in P$ instead of $\text{eval}(C[\delta]) \in P$ to lighten notations.

We proceed in two major steps. First, we transform $C[\square_n]$ into a "*maximally difficult context*" $M[\square_n]$ such that $M[a] \in P$, and $M[b] \sqsubseteq C[b]$ for all $b \in Y_n$. This is possible since $\mathcal{Y}$ is rankwise-finite and ordered by $\sqsubseteq$. This also requires decomposition arguments for set-contexts. In particular we show that they can always be refactored into a normal form that is easy to manipulate, as a context of rank n in normal form is completely given by a tuple of $n + 1$ elements of rank 1.

Second, we combine the element a with the structure of the context $M[\square_n]$ in normal form, and crucially make use of roots, to define a $Y_{\leqslant 1}$-set-system Δ of rank n, i.e. whose evaluation $\delta := \text{eval}(\Delta)$ is deterministic, and that satisfies $M[\delta] \in P$. The maximality assumption in the construction of $M[\square_n]$ then results in that for all Y-set-context $D[\square_n]$, $D[\delta] \in P$ implies that D is "less or as difficult" as M, which in turns implies $D[a] \in P$. By definition of the order $\sqsubseteq$, this shows that $\delta \sqsubseteq a$ as desired. Since we also have $M[\delta] \in P$ by construction, and $M[\delta] \sqsubseteq C[\delta]$, we obtain $C[\delta] \in P$ since P is upward-closed. The automaton property is proved.

6 Conclusion

Our notion of systems is equivalent to notions of tree algebras developed elsewhere in the literature. The notion of set-systems is a kind of extended powerset construction on it, expanded with root vertices, something that is new to the best of our knowledge, though close to more classical powerset constructions. One originality here is mainly that all operations are performed while keeping the structures folded, and only finitely unfolding it when necessary. More crucial is the argument that transforms algebras to yield-algebras, which circumvents the impossibility to have the powerset distribute over the monad of an algebraic theory that has non-linear identities. We are studying further this categorical aspect. Another interesting direction is to see to which extent an algebraic approach can be used to bypass automata for deciding the MSO-theory of infinite trees, i.e. a purely algebraic proof or Rabin's theorem.

References

1 Peter Aczel, Jiří Adámek, and Jiří Velebil. A coalgebraic view of infinite trees and iteration. *Electronic Notes in Theoretical Computer Science*, 44(1):1–26, 2001. CMCS 2001, Coalgebraic Methods in Computer Science (a Satellite Event of ETAPS 2001). doi:10.1016/S1571-0661(04)80900-9.

2 Achim Blumensath. Recognisability for algebras of infinite trees. *Theor. Comput. Sci.*, 412(29):3463–3486, 2011. doi:10.1016/J.TCS.2011.02.037.

3 Achim Blumensath. An algebraic proof of rabin's tree theorem. *Theor. Comput. Sci.*, 478:1–21, 2013. doi:10.1016/J.TCS.2013.01.026.

4 Achim Blumensath. The Power-Set Construction for Tree Algebras. *Logical Methods in Computer Science*, Volume 19, Issue 4, November 2023. doi:10.46298/lmcs-19(4:9)2023.

5 Achim Blumensath and Felix Wolf. Bisimulation invariant monadic-second order logic in the finite. *Theoretical Computer Science*, 823:26–43, 2020. doi:10.1016/j.tcs.2020.03.001.

6 Mikołaj Bojańczyk and Tomasz Idziaszek. Algebra for infinite forests with an application to the temporal logic EF. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR 2009 - Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy,*

September 1-4, 2009. Proceedings, volume 5710 of *Lecture Notes in Computer Science*, pages 131–145. Springer, 2009. doi:10.1007/978-3-642-04081-8_10.

7 Mikołaj Bojańczyk and Bartek Klin. A non-regular language of infinite trees that is recognizable by a sort-wise finite algebra. *Logical Methods in Computer Science*, Volume 15, Issue 4, November 2019. doi:10.23638/LMCS-15(4:11)2019.

8 Facundo Carreiro, Alessandro Facchini, Yde Venema, and Fabio Zanasi. Weak MSO: automata and expressiveness modulo bisimilarity. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 27:1–27:27. ACM, 2014. doi:10.1145/2603088.2603101.

9 Anuj Dawar and Martin Otto. Modal characterisation theorems over special classes of frames. *Annals of Pure and Applied Logic*, 161(1):1–42, 2009. doi:10.1016/J.APAL.2009.04.002.

10 S. Feferman and R. Vaught. The first order properties of products of algebraic systems. *Fundamenta Mathematicae*, 47(1):57–103, 1959. URL: http://eudml.org/doc/213526.

11 Helle Hvid Hansen, Clemens Kupke, and Eric Pacuit. Neighbourhood structures: Bisimilarity and basic model theory. *Logical Methods in Computer Science*, Volume 5, Issue 2, April 2009. doi:10.2168/LMCS-5(2:2)2009.

12 Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309. Springer, 1980. doi:10.1007/3-540-10003-2_79.

13 David Janin and Igor Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96, Concurrency Theory, 7th International Conference, Pisa, Italy, August 26-29, 1996, Proceedings*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 1996. doi:10.1007/3-540-61604-7_60.

14 Faron Moller and Alexander Moshe Rabinovich. On the expressive power of CTL*. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 360–368. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782631.

15 Damian Niwiński. Fixed point characterization of infinite behavior of finite-state systems. *Theoretical Computer Science*, 189(1):1–69, 1997. doi:10.1016/S0304-3975(97)00039-X.

16 Maximilian Pflueger, Johannes Marti, and Egor V. Kostylev. A characterisation theorem for two-way bisimulation-invariant monadic least fixpoint logic over finite structures. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, pages 63:1–63:14. ACM, 2024. doi:10.1145/3661814.3662107.

17 Alexander Rabinovich. On compositionality and its limitations. *ACM Trans. Comput. Log.*, 8(1):4, 2007. doi:10.1145/1182613.1182617.

18 Eric Rosen. Modal logic over finite structures. *J. Log. Lang. Inf.*, 6(4):427–439, 1997. doi:10.1023/A:1008275906015.

19 Saharon Shelah. The monadic theory of order. *Annals of Mathematics*, 102(3):379–419, 1975.

20 Johan Van Benthem. Correspondence theory. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic: Volume II: Extensions of Classical Logic*, pages 167–247. Springer Netherlands, Dordrecht, 1984. doi:10.1007/978-94-009-6259-0_4.

21 Thomas Wilke. Algebras for classifying regular tree languages and an application to frontier testability. In Andrzej Lingas, Rolf Karlsson, and Svante Carlsson, editors, *Automata, Languages and Programming*, pages 347–358, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg. doi:10.1007/3-540-56939-1_85.