Cost Automata, Safe Schemes, and Downward Closures

David Barozzini
Institute of Informatics, University of Warsaw, Poland
dbarozzini@mimuw.edu.pl

Lorenzo Clemente
Institute of Informatics, University of Warsaw, Poland
clementelorenzo@gmail.com

Thomas Colcombet
IRIF-CNRS-Université de Paris, France
thomas.colcombet@irif.fr

Paweł Parys
Institute of Informatics, University of Warsaw, Poland
parys@mimuw.edu.pl

Abstract

Higher-order recursion schemes are an expressive formalism used to define languages of possibly infinite ranked trees. They extend regular and context-free grammars, and are equivalent to simply typed λY-calculus and collapsible pushdown automata. In this work we prove, under a syntactical constraint called safety, decidability of the model-checking problem for recursion schemes against properties defined by alternating B-automata, an extension of alternating parity automata for infinite trees with a boundedness acceptance condition. We then exploit this result to show how to compute downward closures of languages of finite trees recognized by safe recursion schemes.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Rewrite systems

Keywords and phrases Cost logics, cost automata, downward closures, higher-order recursion schemes, safe recursion schemes

Digital Object Identifier 10.4230/LIPIcs.ICALP.2020.109

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

Related Version A full version of the paper is available at [4], `https://arxiv.org/abs/2004.12187`.

Funding *David Barozzini*: Author supported by the National Science Centre, Poland (grant no. 2016/22/E/ST6/00041).
Lorenzo Clemente: Author supported by the National Science Centre, Poland (grant no. 2016/22/E/ST6/00041).
Thomas Colcombet: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No.670624), and the DeLTA ANR project (ANR-16-CE40-0007).
Paweł Parys: Author supported by the National Science Centre, Poland (grant no. 2016/22/E/ST6/00041).

1 Introduction

Higher-order functions are nowadays widely used not only in functional programming languages such as Haskell and the OCAML family, but also in mainstream languages such as Java, JavaScript, Python, and C++. *Recursion schemes* are faithful and algorithmically manageable abstractions of the control flow of higher-order programs [36]. A deterministic recursion scheme normalises into a possibly infinite Böhm tree, and in this respect recursion schemes can equivalently be presented as simply-typed lambda-terms using a higher-order fixpoint combinator Y [50]. There are also nontrivial inter-reductions between recursion schemes and the equi-expressive collapsible higher-order pushdown automata [30] and ordered tree-pushdown automata [13]. In another semantics, also used in this paper, nondeterminstic recursion schemes are recognisers of languages of finite trees, and in this view they are also known as higher-order OI grammars [23, 38], generalising indexed grammars [2] (which are recursion schemes of order two) and ordered multi-pushdown automata [8].

The most celebrated algorithmic result in the analysis of recursion schemes is the decidability of the model-checking problem against properties expressed in monadic second-order logic (MSO): given a recursion scheme $\mathcal{G}$ and an MSO sentence φ, one can decide whether the Böhm tree generated by $\mathcal{G}$ satisfies φ [43]. This fundamental result has been reproved several times, that is, using collapsible higher-order pushdown automata [30], intersection types [37], Krivine machines [48], and it has been extended in diverse directions such as global model checking [11], logical reflection [9], effective selection [12], and a transfer theorem via models of lambda-calculus [49]. When the input property is given as an MSO formula, the model-checking problem is non-elementary already for trees of order 0 (regular trees) [51]; when the input property is presented as a parity tree automaton (which is equi-expressive with MSO on trees, but less succinct), the MSO model-checking problem for recursion schemes of order n is complete for n-fold exponential time [43]. Despite these hardness results, the model-checking problem can be solved efficiently on multiple nontrivial examples, thanks to the development of several recursion-scheme model checkers [36, 35, 10, 47, 42].

Unboundedness problems. Recently, an increasing interest has arose for model checking quantitative properties going beyond the expressive power of MSO. The *diagonal problem* is an example of a quantitative property not expressible in MSO. Over words, the problem asks, for a given set of letters Σ and a language of finite words $\mathcal{L}$, whether for every $n \in \mathbb{N}$ there is a word in $\mathcal{L}$ where every letter from Σ occurs at least n times. Over full trios (classes of languages closed under regular transductions), decidability of the diagonal problem over finite words has interesting algorithmic consequences, such as computability of downward closures [54] and decidability of separability by piecewise testable languages [21]. The diagonal problem for languages of words recognised by recursion schemes is decidable [29, 14, 45].

Over full trios of finite words, the diagonal problem is equivalent to the computability of downward closures [22], which is an important problem in its own right. The *downward closure* of a language $\mathcal{L}$ of finite trees is the set $\mathcal{L}{\downarrow}$ of all trees that can be homeomorphically embedded into some tree in $\mathcal{L}$. By Higman's lemma [31], the embedding relation on finite ranked trees is a well quasi-order. Consequently, the downward closure $\mathcal{L}{\downarrow}$ of an arbitrary set of trees $\mathcal{L}$ is always a regular language. The downward closure of a language offers a nontrivial regular abstraction thereof: even though the actual count of letters is lost, their limit properties are preserved, as well as their order of appearance.

We say that the downward closure is *computable* when a finite automaton for $\mathcal{L}{\downarrow}$ can be effectively constructed (which is not true in general). Downward closures are computable for a wide class of languages of finite words such as those recognised by context-free grammars [20, 41, 3], Petri nets [27], stacked counter automata [55], context-free FIFO rewriting systems and 0L-systems [1], second-order pushdown automata [54], higher-order pushdown automata [29], and (possibly unsafe) recursion schemes over words [14]. Over finite trees, it is known that downward closures are computable for the class of regular tree languages [25]. We are not aware of other such computability results for other classes of languages of finite trees.

In another line of research, B-automata, and among them *alternating B-automata*, have been put forward as a quantitative extension to MSO [15, 18, 52, 39]. They extend alternating automata over infinite trees [26, Chapter 9] by nonnegative integer counters that can be incremented or reset to zero. The extra counters do not constrain the availability of transitions during a run (unlike in other superficially similar models, such as counter machines), but are used in order to define the acceptance condition: an infinite tree is *n-accepted* if n is a bound on the values taken by the counters during an accepting run of the automaton over it.

The *universality problem* consists in deciding whether for every tree there is a bound n for which it is n-accepted. The *boundedness problem* asks whether there exists a bound n for which all trees are n-accepted. These two problems are closely related. Their decidability is an important open problem in the field, and proving the decidability of the boundedness problem would solve the long standing nondeterministic Mostowski index problem [17]. However, though open in general, the boundedness problem is known to be decidable over finite words [15] and trees [18] and infinite words [39], as well as over infinite trees for its weak [53] and the more general quasi-weak [40] version.

Another expressive formalism expressing unboundedness properties beyond MSO is MSO+U, which extends MSO by a new quantifier "$\mathsf{U}X.\varphi$" [7] stating that there exist arbitrarily large finite sets X satisfying φ. This logic is incomparable with B-automata. The model-checking problem of recursion schemes against its weak fragment WMSO+U, where monadic second-order quantifiers are restricted to finite sets, is decidable [46].

Contributions. Our first contribution is the decidability of the model-checking problem of properties expressed by alternating B-automata for an expressive class of recursion schemes called *safe recursion schemes*. As generators of infinite trees, safe recursion schemes are equivalent to higher-order pushdown automata without the collapse operation [34] and are strictly less expressive than general (unsafe) recursion schemes [44, Corollary I.2]. Here, the model-checking problem asks whether a concrete infinite tree (the Böhm tree generated by the safe recursion scheme) is accepted by the B-automaton for some bound. This problem happens to be significantly simpler than the universality/boundedness problem above described. The proof goes by reducing the order of the safe recursion scheme similarly as done in Knapik, Niwiński, and Urzyczyn [34] to show decidability of the MSO model-checking problem, at the expense of making the property automaton two-way. We then rely on the fact that two-way alternating B-automata can be converted to equivalent one-way alternating B-automata [6]. Our result is incomparable with the result of Ong [43], since

(1) alternating B-automata are strictly more expressive than MSO, however

(2) we obtain it under the more restrictive safety assumption.

Whether the safety assumption can be dropped while preserving decidability of the model-checking problem against B-automata properties remains open.

Our second contribution is to define the following generalization of the diagonal problem from words to trees: given a language of finite trees $\mathcal{L}$ and a set of letters Σ, decide whether for every $n \in \mathbb{N}$ there is a tree $T \in \mathcal{L}$ such that every letter from Σ occurs at least n times on every branch of T. This generalization is designed in order to reduce the computation of downward closures to the diagonal problem, in the same fashion as for finite words. Our proof strategy is to represent downward-closed sets of trees $\mathcal{L}\downarrow$ by simple tree regular expressions, which are a subclass of regular expressions for finite trees [24, 25]. By further analysing and simplifying the structure of these expressions, the computation of the downward closure can be reduced to finitely many instances of the diagonal problem. Unlike in the case of finite words, we do not know whether for full trios there exists a converse reduction from the diagonal problem to the problem of computing downward closures.

Our third contribution is decidability of the diagonal problem for languages of finite trees recognised by safe recursion schemes (and thus computability of downward closures of those languages). The diagonal problem can directly be expressed in a logic called *weak cost monadic second order logic* (WCMSO) [53], which extends weak MSO with atomic formulae of the form $|X| \leq N$ stating that the cardinality of the monadic variable X is at most N. Since WCMSO can be translated to alternating B-automata [53], the diagonal problem reduces to the model-checking problem of safe recursion schemes against alternating B-automata, which we have shown decidable in the first part. Note that it seems difficult to express the diagonal problem using alternating B-automata directly, and indeed the fact that alternating B-automata can express all WCMSO properties is nontrivial.

Outline. In Section 2, we define recursion schemes and B-automata. In Section 3, we present our first result, namely decidability of model checking of safe recursion schemes against B-automata. In Section 4, we introduce the diagonal problem, and we show how it can be used to compute downward closures. In Section 5, we solve the diagonal problem for schemes. We conclude in Section 6 with some open problems. A full technical report containing full proofs it also available [4].

2 Preliminaries

Recursion schemes. A *ranked alphabet* is a (usually finite) set $\mathbb{A}$ of letters, together with a function *rank*: $\mathbb{A} \to \mathbb{N}$, assigning a *rank* to every letter. When we define trees below, we require that a node labeled by a letter a has exactly $\mathit{rank}(a)$ children. In the sequel, we usually assume some fixed finite ranked alphabet $\mathbb{A}$. The set of *(simple) types* is constructed from a unique ground type o using a binary operation $\to$; namely o is a type, and if α and β are types, so is $\alpha \to \beta$. By convention, $\to$ associates to the right, that is, $\alpha \to \beta \to \gamma$ is understood as $\alpha \to (\beta \to \gamma)$. A type $\mathsf{o} \to \ldots \to \mathsf{o}$ with k occurrences of $\to$ is also written as $\mathsf{o}^k \to o$. The *order* of a type α, denoted $\mathit{ord}(\alpha)$ is defined by induction: $\mathit{ord}(\mathsf{o}) = 0$ and $\mathit{ord}(\alpha_1 \to \ldots \to \alpha_k \to \mathsf{o}) = \max_i(\mathit{ord}(\alpha_i)) + 1$ for $k \geqslant 1$.

We coinductively define both *lambda-terms* and the two-argument relation "M *is a lambda-term of type* α" as follows (cf. [32, 5]):

- a letter $a \in \mathbb{A}$ is a lambda-term of type $\mathsf{o}^{\mathit{rank}(a)} \to \mathsf{o}$;
- for every type α there is a countable set $\{x, y, \ldots\}$ of *variables of type* α which can be used as lambda-terms of type α;
- if M is a lambda-term of type β and x a variable of type α, then $\lambda x.M$ is a lambda-term of type $\alpha \to \beta$; this construction is called a *lambda-binder*;
- if M is a lambda-term of type $\alpha \to \beta$, and N is a lambda-term of type α, then $M\,N$ is a lambda-term of type β, called an *application*.

As usual, we identify lambda-terms up to *alpha-conversion* (i.e., renaming of bound variables). We use here the standard notions of *free variable*, *subterm*, (capture-avoiding) *substitution*, and *beta-reduction* (see for instance [32, 5]). A *closed* lambda-term does not have free variables. For a lambda-term M of type α, the *order* of M, denoted $\mathit{ord}(M)$, is defined as $\mathit{ord}(\alpha)$. It is *first-order* if its order is one. An *applicative term* is a lambda-term not containing lambda-binders (it contains only letters, applications, and variables).

A lambda-term M is *superficially safe* if all its free variables x have order $\mathit{ord}(x) \geqslant \mathit{ord}(M)$. A lambda-term M is *safe* if it is superficially safe, and if for every subterm of the form $K\,L_1\,\ldots\,L_k$, where K is not an application and $k \geqslant 1$, all subterms $K, L_1, \ldots, L_k$ are superficially safe.

A *(higher-order, deterministic) recursion scheme* over the alphabet $\mathbb{A}$ is a tuple $\mathcal{G} = \langle \mathbb{A}, \mathcal{N}, X_0, \mathcal{R} \rangle$, where $\mathcal{N}$ is a finite set of typed *nonterminals*, $X_0 \in \mathcal{N}$ is the *initial nonterminal*, and $\mathcal{R}$ is a function assigning to every nonterminal $X \in \mathcal{N}$ of type $\alpha_1 \to \cdots \to \alpha_k \to \mathsf{o}$ a finite lambda-term of the form $\lambda x_1. \cdots .\lambda x_k.K$, of the same type $\alpha_1 \to \cdots \to \alpha_k \to \mathsf{o}$, in which K is an applicative term with free variables in $\mathcal{N} \uplus \{x_1, \dots, x_k\}$. We refer to $\mathcal{R}(X)$ as the *rule* for X. The *order* of a recursion scheme $ord(\mathcal{G})$ is the maximum order of its nonterminals.

The lambda-term *represented* by a recursion scheme $\mathcal{G}$ as above, denoted $\Lambda(\mathcal{G})$, is the limit of applying recursively the following operation to X_0: take an occurrence of some nonterminal X, and replace it with $\mathcal{R}(X)$ (the nonterminals should be chosen in a fair way, so that every nonterminal is eventually replaced). Thus, $\Lambda(\mathcal{G})$ is a (usually infinite) regular lambda-term obtained by unfolding the nonterminals of $\mathcal{G}$ according to their definition. We remark that when substituting $\mathcal{R}(X)$ for a nonterminal X there is no need for any renaming of variables (capture-avoiding substitution), since $\mathcal{R}(X)$ does not contain free variables other than nonterminals. We only consider recursion schemes for which $\Lambda(\mathcal{G})$ is well-defined (e.g. by requiring that $\mathcal{R}(X)$ is not a single nonterminal). A recursion scheme $\mathcal{G}$ is *safe* if $\Lambda(\mathcal{G})$ is safe.

A *tree* is a closed applicative term of type o. Such lambda-terms are coinductively of the form $a\, M_1 \cdots M_r$, where $a \in \mathbb{A}$ is of rank r, and where $M_1, \dots, M_r$ are again trees. Thus, such a lambda-term can be identified with a tree understood in the traditional sense: a is the label of its root, and $M_1, \dots, M_r$ describe subtrees attached in the r children of the root, from left to right. For trees we also use the notation $a(M_1, \dots, M_r)$ instead of $a\, M_1 \dots M_r$. A tree is *regular* if it has finitely many subtrees (subterms) up to isomorphism.

The *Böhm tree* of a lambda-term M of type o, denoted $BT(M)$, is defined coinductively as follows: if there is a sequence of beta-reductions from M to a lambda-term of the form $a\, M_1 \dots M_r$ (where $a \in \mathbb{A}$ is a letter), then $BT(M) = a(BT(M_1), \dots, BT(M_r))$; otherwise $BT(M) = \perp()$, where $\perp \in \mathbb{A}$ is a distinguished letter of rank 0. It is a classical result that $BT(M)$ exists, and is uniquely defined [32, 5]. Clearly, $BT(M)$ is indeed a tree. The tree *generated* by a recursion scheme $\mathcal{G}$, denoted $BT(\mathcal{G})$, is $BT(\Lambda(\mathcal{G}))$.

A lambda-term N is *normalizing* if $BT(N)$ does not contain the special letter $\perp$; a recursion scheme $\mathcal{G}$ is *normalizing* if $\Lambda(\mathcal{G})$ is normalizing. In other words, in a normalizing recursion scheme/lambda-term beta-reduction always produces a letter. It is possible to transform every recursion scheme $\mathcal{G}$ into a normalizing recursion scheme $\mathcal{G}'$ generating the same tree as $\mathcal{G}$, up to renaming $\perp$ into some non-special letter $\perp'$ (cf. [28, Section 5]). Moreover, the construction preserves safety and the order.

Recursion schemes as recognizers of languages of finite trees. The standard semantics of a recursion scheme $\mathcal{G} = \langle \mathbb{A}, \mathcal{N}, X_0, \mathcal{R} \rangle$ is the single infinite tree $BT(\mathcal{G})$ generated by the scheme. An alternative view is to consider a recursion scheme as a recognizer of a language of finite trees $\mathcal{L}(\mathcal{G})$. This alternative view is relevant when discussing downward closures of languages of finite trees. We employ a special letter $\mathsf{nd} \in \mathbb{A}$ of rank 2 in order to represent $\mathcal{L}(\mathcal{G})$ by resolving the nondeterministic choice of nd in the infinite tree $BT(\mathcal{G})$ in all possible ways. Formally, for two trees T, U, we write $T \to_{\mathsf{nd}} U$ if U is obtained from T by choosing an nd-labeled node u of T and a child v thereof, and replacing the subtree rooted at u with the subtree rooted at v. The relation $\to_{\mathsf{nd}}^*$ is the reflexive and transitive closure of $\to_{\mathsf{nd}}$. We define the language of finite trees *recognized* by $\mathcal{G}$ as $\mathcal{L}(\mathcal{G}) = \mathcal{L}(BT(\mathcal{G}))$, where

$$\mathcal{L}(T) = \{U \mid T \to_{\mathsf{nd}}^* U, \text{ with } U \text{ finite and not containing “}\mathsf{nd}\text{” or “}\perp\text{”}\}.$$

For an illustration of this encoding, and simultaneously for an example of a recursion scheme, consider the ranked alphabet $\mathbb{A}$ containing a letter a of rank 2, two letters b_1, b_2 of rank 1, and a letter c of rank 0. We use an initial nonterminal S of order-0 type o, and an additional nonterminal A of order-2 type $(\mathsf{o} \to \mathsf{o}) \to (\mathsf{o} \to \mathsf{o}) \to \mathsf{o} \to \mathsf{o} \to \mathsf{o}$, together with the following two rules:

$$\mathcal{R}(S) = A\, b_1\, b_2\, c\, c,$$
$$\mathcal{R}(A) = \lambda f.\lambda g.\lambda x.\lambda y.\mathsf{nd}\,(a\,x\,y)\,(A\,f\,g\,(f\,x)\,(g\,x)).$$

Then, $BT(\mathcal{G})$ is the infinite non-regular tree $\mathsf{nd}\,(a\,c\,c)\,(\mathsf{nd}\,(a\,(b_1\,c)\,(b_2\,c))\,(\cdots))$, and $\mathcal{L}(\mathcal{G})$ is the non-regular language of all finite trees of the form $a\,(b_1^n\,c)\,(b_2^n\,c)$ with $n \in \mathbb{N}$.

Alternating B-automata. We introduce the model of automata used in this paper, namely *alternating one-way/two-way B-automata* over trees (over a ranked alphabet). We consider counters which can be *incremented* $\mathtt{i}$, *reset* $\mathtt{r}$, or left *unchanged* ε. Let Γ be a finite set of *counters* and let $\mathbb{C} = \{\mathtt{i}, \mathtt{r}, \varepsilon\}$ be the *alphabet of counter actions*. Each counter starts with value zero, and the *value of a sequence* of actions is the supremum of the values achieved during this sequence. For instance $\mathtt{iir}\varepsilon\mathtt{i}\varepsilon$ has value 2, $(\mathtt{ir})^\omega$ has value 1, and $\mathtt{iri}^2\mathtt{ri}^3\mathtt{r}\cdots$ has value ∞. For an infinite sequence of counter actions $w \in \mathbb{C}^\omega$, let $val(w) \in \mathbb{N} \cup \{\infty\}$ be its *value*. In case of several counters, $w = c_1 c_2 \cdots \in (\mathbb{C}^\Gamma)^\omega$, we take the counter with the maximal value: $val(w) = \sup_{c \in \Gamma} val(w(c))$, where $w(c) = c_1(c)c_2(c)\cdots$.

An *(alternating, two-way) B-automaton* over a finite ranked alphabet $\mathbb{A}$ is a tuple $\langle \mathbb{A}, Q, q_0, pr, \Gamma, \delta\rangle$ consisting of a finite set of *states* Q, an *initial state* $q_0 \in Q$, a function $pr\colon Q \to \mathbb{N}$ assigning *priorities* to states, a finite set Γ of counters, and a *transition function*

$$\delta : Q \times \mathbb{A} \to \mathcal{B}^+(\{\uparrow, \circlearrowleft, \downarrow_1, \downarrow_2, \ldots\} \times \mathbb{C}^\Gamma \times Q)$$

mapping a state and a letter a to a (finite) positive Boolean combination of triples of the form (d, c, q); it is assumed that if $d = \downarrow_i$ then $i \leqslant rank(a)$. Such a triple encodes the instruction to send the automaton to state q in direction d while performing action c. The direction $\downarrow_i$ moves to the i-th child, $\uparrow$ moves to the parent, and $\circlearrowleft$ stays in place. We assume that $\delta(q, a)$ is written in disjunctive normal form for all q and a.

The acceptance of an infinite input tree T by an alternating B-automaton $\mathcal{A}$ is defined in terms of a game $(\mathcal{A}, T)$ between two players, called Eve and Adam. Eve is in charge of disjunctive choices and tries to minimize the counter values while satisfying the parity condition. Adam, on the other hand, is in charge of conjunctive choices and tries to either maximize counter values, or to sabotage the parity condition. Since the transition function is given in disjunctive normal form, each turn of the game consists of Eve choosing a disjunct and Adam selecting a single tuple (d, c, q) thereof. In order to guarantee that from every position there is some move, we assume that each disjunction is nonempty and that each disjunct contains a tuple with some direction other than $\uparrow$. A *play* of $\mathcal{A}$ on the tree T is a sequence $q_0, (d_1, c_1, q_1), (d_2, c_2, q_2), \ldots$ compatible with T and δ: q_0 is the initial state, and for all $i \in \mathbb{N}$, $(d_{i+1}, c_{i+1}, q_{i+1})$ appears in $\delta(q_i, T(x_i))$ where x_i is the node of T after following the directions $d_1 d_2 \ldots d_i$ starting from the root. The value $val(\pi)$ of a play π is the value $val(c_1 c_2 \cdots)$ as defined above if the largest number appearing infinitely often among the priorities $pr(q_0), pr(q_1), \ldots$ is even; otherwise, $val(\pi) = \infty$. We say that the play π is *n-winning* (for Eve) if $val(\pi) \leqslant n$.

A *strategy* for one of the players in the game $(\mathcal{A}, T)$ is a function that returns the next choice given the history of the play. Note that choosing a strategy for Eve and a strategy for Adam fixes a play in $(\mathcal{A}, T)$. We say that a play π is *compatible* with a strategy σ if there is some strategy σ' for the other player such that σ and σ' together yield the play π. A strategy for Eve is *n-winning* if every play compatible with it is n-winning. We say that Eve *n-wins* the game if there is some n-winning strategy for Eve. A B-automaton *n-accepts* a tree T if Eve n-wins the game $(\mathcal{A}, T)$; it *accepts* T if it n-accepts T for some $n \in \mathbb{N}$. The language *recognized* by $\mathcal{A}$ is the set of all trees accepted by $\mathcal{A}$.

If no $\delta(q, a)$ uses the direction $\uparrow$, then we call $\mathcal{A}$ *one-way*. The following theorem essentially follows from a result of Blumensath, Colcombet, Kuperberg, Parys, and Vanden Boom [6, Theorem 6] modulo some cosmetic changes (c.f. [4, Appendix A] for more details).

▶ **Theorem 2.1** (c.f. [6, Theorem 6]). *Given an alternating two-way B-automaton, one can compute an alternating one-way B-automaton that recognizes the same language.*

As a special case of a result by Colcombet and Göller [16] we obtain the following fact.

▶ **Fact 2.2.** *One can decide whether a given B-automaton accepts a given regular tree.*

3 Model-checking safe recursion schemes against alternating B-automata

In this section we prove the first main theorem of our paper, the decidability of the *model-checking problem* of safe recursion schemes against properties described by B-automata:

▶ **Theorem 3.1.** *Given an alternating B-automaton $\mathcal{A}$ and a safe recursion scheme $\mathcal{G}$, one can decide whether $\mathcal{A}$ accepts the tree generated by $\mathcal{G}$.*

It is worth noticing that this theorem generalises the result of Knapik et al. [34] on safe recursion schemes from regular (MSO) properties to the more general quantitative realm of properties described by B-automata. On the other hand, our result is incomparable with the celebrated theorem of Ong [43] showing decidability of model checking regular properties of possibly unsafe recursion schemes. Whether model checking of possibly unsafe recursion schemes against properties described by B-automata is decidable remains an open problem.

By Theorem 2.1, every B-automaton can be effectively transformed into an equivalent one-way B-automaton, so it is enough to prove Theorem 3.1 for a one-way B-automaton $\mathcal{A}$. The proof of Theorem 3.1 is based on the following lemma, where we use in an essential way the assumption that the recursion scheme is safe.

▶ **Lemma 3.2.** *For every safe recursion scheme $\mathcal{G}$ of order m and for every alternating one-way B-automaton $\mathcal{A}$, one can effectively construct a safe recursion scheme $\mathcal{G}'$ of order $m-1$ and an alternating two-way B-automaton $\mathcal{A}'$ such that*

$$\mathcal{A} \text{ accepts } BT(\mathcal{G}) \quad \text{if and only if} \quad \mathcal{A}' \text{ accepts } BT(\mathcal{G}').$$

Theorem 3.1 follows easily: Using Lemma 3.2 we can reduce the order of the considered safe recursion scheme by one. We obtain a two-way B-automaton, which we convert back to a one-way B-automaton using Theorem 2.1. It is then sufficient to repeat this process, until we end up with a recursion scheme of order 0. A recursion scheme of order 0 generates a regular tree and, by Fact 2.2, we can decide whether the resulting B-automaton accepts this tree, answering our original question.

Lambda-trees. We now come to the proof of Lemma 3.2. The construction of $\mathcal{G}'$ from of $\mathcal{G}$ follows an analogous result for MSO [33, 34], which we generalise to B-automata. We represent some lambda-terms as trees. For a *finite* set $\mathcal{X}$ of variables of type o, we define a new ranked alphabet $\mathbb{A}_{\mathcal{X}}$ that contains

1) a letter $\overline{a}$ of rank 0 for every letter $a \in \mathbb{A}$;
2) a letter $\overline{x}$ of rank 0 for every variable $x \in \mathcal{X}$;
3) a letter $\overline{\lambda x}$ of rank 1 for every variable $x \in \mathcal{X}$;
4) a letter @ of rank 2.

We remark that $\mathbb{A}_{\mathcal{X}}$ is a usual finite ranked alphabet. A *lambda-tree* is a tree over the alphabet $\mathbb{A}_{\mathcal{X}}$, where $\mathcal{X}$ is clear from the context. Intuitively, a lambda-tree is a tree representation of a first-order lambda-term.

The semantics $[\![T]\!]_{\mathcal{X},s}$ of a lambda-tree T is defined in such a way that if T "corresponds" to the lambda-term M, then $[\![T]\!]_{\mathcal{X},s} = BT(M)$. Since T uses only variables of type o we can read the resulting Böhm tree directly, without performing any reduction. Essentially, we walk down through T, skipping all lambda-binders and choosing the left branch in all applications. Whenever we reach some variable x, we go up to the corresponding lambda-binder, then up to the corresponding application, and then we again start going down in the argument of this application. Formally, let $\mathcal{X}$ be a finite set of variables of type o, and let $s \in \mathbb{N}$. The intended meaning is that $\mathcal{X}$ contains variables that may potentially appear in the considered lambda-tree T, and that s is a bound for the arity of types in the lambda-term represented by T (types of all its subterms should be of the form $\mathsf{o}^k \to \mathsf{o}$ for $k \leq s$). We take $\mathit{Dirs}_{\mathcal{X},s} = \{\downarrow\} \cup \{\uparrow_x \mid x \in \mathcal{X}\} \cup \{\uparrow_i \mid 1 \leq i \leq s\}$. Intuitively, $\downarrow$ means to go down to the left child, $\uparrow_x$ means that we are looking for the value of (lambda)variable x, and $\uparrow_i$ means that we are looking for the i-th argument of an application. For a node v of T denote its parent by $\mathit{par}(v)$, and its i-th child by $\mathit{ch}_i(v)$. For $d \in \mathit{Dirs}_{\mathcal{X},s}$, and for a node v of T labeled by $a \in \mathbb{A}$, we define the $(\mathcal{X}, s)$*-successor* of (d, v) as

1. $(\downarrow, \mathit{ch}_1(v))$ if $d = \downarrow$ and $a = \overline{\lambda x}$ (for some x) or @,
2. $(\uparrow_x, v)$ if $d = \downarrow$ and $a = \overline{x}$ (for some x),
3. $(\uparrow_x, \mathit{par}(v))$ if $d = \uparrow_x$ and $a \neq \overline{\lambda x}$ (including the case when $a = \overline{\lambda y}$ for $y \neq x$),
4. $(\uparrow_1, \mathit{par}(v))$ if $d = \uparrow_x$ and $a = \overline{\lambda x}$,
5. $(\uparrow_{i+1}, \mathit{par}(v))$ if $d = \uparrow_i$ for $i < s$ and $a = \overline{\lambda y}$ (for some y),
6. $(\uparrow_{i-1}, \mathit{par}(v))$ if $d = \uparrow_i$ for $i > 1$ and $a =$ @,
7. $(\downarrow, \mathit{ch}_2(v))$ if $d = \uparrow_1$ and $a =$ @.

Rule 1 allows us go to down to the first child in the case of lambda-binders and applications. Rule 2 records that we have seen x, and thus we need to find its value by going up. Rule 3 climbs the tree upwards as long as we do not see the corresponding binder $\overline{\lambda x}$. Rule 4 records that we have seen $\overline{\lambda x}$ and initialises its level to 1. We now need to find the corresponding application. Rule 5 increments the level and goes up when we encounter a binder $\overline{\lambda y}$, and Rule 6 decrements it for applications @. Finally, when we see an application at level 1 we apply Rule 7 which searches for the value of x in the right child. An $(\mathcal{X}, s)$*-maximal path* from (d_1, v_1) is a sequence of pairs $(d_1, v_1), (d_2, v_2), \ldots$ in which every (d_{i+1}, v_{i+1}) is the $(\mathcal{X}, s)$-successor of (d_i, v_i), and which is either infinite or ends in a pair that has no $(\mathcal{X}, s)$-successor. For $d \in \mathit{Dirs}_{\mathcal{X},s}$, and for a node v of T, we define the $(\mathcal{X}, s)$*-derived tree* from (T, d, v), denoted by $[\![T, d, v]\!]_{\mathcal{X},s}$, by coinduction:

- if the $(\mathcal{X}, s)$-maximal path from (d, v) is finite and ends in $(\downarrow, w)$ for a node w labeled by $\overline{a}$, then $[\![T, d, v]\!]_{\mathcal{X},s} = a([\![T, \uparrow_1, w]\!]_{\mathcal{X},s}, \ldots, [\![T, \uparrow_{\mathit{rank}(a)}, w]\!]_{\mathcal{X},s})$;
- otherwise, $[\![T, d, v]\!]_{\mathcal{X},s} = \perp$.

The $(\mathcal{X}, s)$*-derived tree* from T is $[\![T]\!]_{\mathcal{X},s} = [\![T, \downarrow, v_0]\!]_{\mathcal{X},s}$, where v_0 is the root of T. We say that T is *normalizing* if $[\![T]\!]_{\mathcal{X},s}$ does not contain $\perp$.

The following lemma performs the order reduction. It crucially relies on the safety assumption. It is a variant of results proved in Knapik et al. [33, 34] (c.f. [4, Appendix C] for more details). Intuitively, it says that a lambda-tree representation T of a safe recursion scheme $\mathcal{G}$ of order m can be computed by a safe recursion scheme of order $m-1$ in a semantic-preserving way.

▶ **Lemma 3.3** ([33, 34]). *For every safe recursion scheme $\mathcal{G}$ of order $m \geq 1$ one can construct a safe recursion scheme $\mathcal{G}'$ of order $m-1$, a finite set of variables $\mathcal{X}$, and a number $s \in \mathbb{N}$ such that*

$$[\![BT(\mathcal{G}')]\!]_{\mathcal{X},s} = BT(\mathcal{G}).$$

▶ Remark 3.4. In Knapik et al. [33, 34] the lambda-tree T is denoted $\daleth_{\mathcal{G}}(X_0)$ (where X_0 is the starting nonterminal of $\mathcal{G}$), and the recursion scheme $\mathcal{G}'$ is denoted $\mathcal{G}^\alpha$. The set $\mathcal{X}$ is just the set of variables appearing in the letters used in $\mathcal{G}^\alpha$; the number s can be also read out of $\mathcal{G}^\alpha$. They prove that the $(\mathcal{X}, s)$-derived tree of $\daleth_{\mathcal{G}}(X_0)$ equals the tree generated by $\mathcal{G}$ [33, Proposition 4], that $\mathcal{G}^\alpha$ is safe [34, Lemma 3.5], and that $\mathcal{G}^\alpha$ generates $\daleth_{\mathcal{G}}(X_0)$.

In order to prove Lemma 3.3, one needs to replace in $\mathcal{G}$ every variable x of type o by $\overline{x}$, every lambda-binder concerning such a variable by $\overline{\lambda x}$, and every application with an argument of type o by a construct creating a @-labeled node. Types of subterms change and the order of the recursion scheme decreases by one. Notice, however, that while computing $BT(\Lambda(\mathcal{G}))$ we may need to rename variables during capture-avoiding substitutions, while in the tree generated by the modified recursion scheme we leave original variable names. In general (i.e., when the transformation is applied to an arbitrary recursion scheme) this causes a problem of overlapping variable names. The assumption that $\mathcal{G}$ is safe is crucial here and there is no need to rename variables when applying the transformation to a safe recursion scheme.

Having Lemma 3.3, it remains to transform a one-way B-automaton $\mathcal{A}$ operating on the tree generated by $\mathcal{G}$ into a two-way B-automaton $\mathcal{A}'$ operating on the lambda-tree generated by $\mathcal{G}'$, as described by the following lemma (as mentioned on page 5, we can assume that $\mathcal{G}$ is normalizing, which implies that $BT(\mathcal{G}')$ is normalizing: the tree $[\![BT(\mathcal{G}')]\!]_{\mathcal{X},s} = BT(\mathcal{G})$ does not contain $\perp$).

▶ **Lemma 3.5.** *Let $\mathcal{A}$ be an alternating one-way B-automaton over a finite alphabet $\mathbb{A}$, let $\mathcal{X}$ be a finite set of variables, and let $s \in \mathbb{N}$. One can construct an alternating two-way B-automaton $\mathcal{A}'$ such that for every normalizing lambda-tree T over $\mathbb{A}_{\mathcal{X}}$,*

$$\mathcal{A} \text{ accepts } [\![T]\!]_{\mathcal{X},s} \quad \text{if and only if} \quad \mathcal{A}' \text{ accepts } T.$$

Proof. The B-automaton $\mathcal{A}'$ simulates $\mathcal{A}$ on the lambda-tree. Whenever $\mathcal{A}$ wants to go down to the i-th child, $\mathcal{A}'$ has to follow the $(\mathcal{X}, s)$-maximal path from $(\uparrow_i, v)$ (where v is the current node). To this end, it has to remember the current pair (d, v), and repeatedly find its $(\mathcal{X}, s)$-successor. Here v is always just the current node visited by the B-automaton; the d component comes from the (finite) set $Dirs_{\mathcal{X},s}$, and thus it can be remembered in the state. It is straightforward to encode the definition of an $(\mathcal{X}, s)$-successor in transitions of an automaton. We do not have to worry about infinite $(\mathcal{X}, s)$-maximal paths, because by assumption the $(\mathcal{X}, s)$-derived tree does not contain $\perp$-labeled nodes. ◀

4 Downward closures of tree languages

In this section we lay down a method for the computation of the downward closure for classes of languages of finite trees closed under linear FTT transductions. This method is analogous to the one of Zetzsche [54] for the case of finite words. In Section 4.1 we define the downward closure of languages of finite ranked trees with respect to the embedding well-quasi order and in Section 4.2 we define the simultaneous unboundedness problem for trees and show how computing the downward closure reduces to it. In Section 4.3 we define the diagonal problem for finite trees and show how the previous problem reduces to it. We will then solve the diagonal problem for languages of finite trees recognized by safe recursion schemes in Section 5.

Let us emphasize that this section can be applied to any class of languages of finite trees closed under linear FTT transductions, not just those recognized by safe recursion schemes.

4.1 Preliminaries

Given two finite trees $S = a(S_1, \ldots, S_k)$ and $T = b(T_1, \ldots, T_r)$, we say that S *homeomorphically embeds into* T, written $S \sqsubseteq T$, if, either

1) there exists $i \in \{1, \ldots, r\}$ such that $S \sqsubseteq T_i$, or

2) $a = b$, $k = r$, and $S_i \sqsubseteq T_i$ for all $i \in \{1, \ldots, r\}$.

For a language of finite trees $\mathcal{L}$, its *downward closure*, denoted by $\mathcal{L}{\downarrow}$, is the set of trees S such that $S \sqsubseteq T$ for some tree $T \in \mathcal{L}$.

Pure products. Goubault-Larrecq and Schmitz [25] describe downward-closed sets of trees using so-called *simple tree regular expressions*. Among those expressions they distinguish *products*, which describe *ideals* of trees. Because every downward-closed set of trees is a finite union of ideals, such a set can be described by a finite list of products. Since their definition of a product is rather indirect, we consider the stronger notion of *pure products*.

A *context* is a tree possibly containing one or more occurrences of a special leaf $\Box$, called a *hole*. Given a context C and a set of trees $\mathcal{L}$, we write $C[\mathcal{L}]$ for the set of trees obtained from C by replacing every occurrence of the hole $\Box$ by some tree from $\mathcal{L}$. Different occurrences of $\Box$ are replaced by possibly different trees from $\mathcal{L}$. The definition readily extends to a set of contexts $\mathcal{C}$, by writing $\mathcal{C}[\mathcal{L}]$ for $\bigcup_{C\in\mathcal{C}} C[\mathcal{L}]$. If C does not have any $\Box$, then $C[\mathcal{L}]$ is just $\{C\}$. A *pure product* is defined according to the following abstract syntax:

$$P ::= a^?(P, \ldots, P) \mid I^*.P, \qquad C ::= a(P_\Box, \ldots, P_\Box),$$
$$I ::= C + \cdots + C, \qquad P_\Box ::= \Box \mid P,$$

where the sum of contexts is nonempty, and where in a context $C = a(P_{\Box,1}, \ldots, P_{\Box,r})$ it is required that at least one $P_{\Box,i}$ is a hole $\Box$. A pure product P denotes a set of trees $[\![P]\!]$ downward-closed for $\sqsubseteq$, which is defined recursively as follows:

$$\begin{aligned}
[\![a^?(P_1, \ldots, P_r)]\!] &= \{a(T_1, \ldots, T_r) \mid \forall i\,.\, T_i \in [\![P_i]\!]\} \cup [\![P_1]\!] \cup \cdots \cup [\![P_r]\!],\\
[\![I^*.P]\!] &= \bigcup_{n\in\mathbb{N}} \underbrace{[\![I]\!][\ldots[[\![I]\!]}_{n}[\![P]\!]]]\ldots],\\
[\![C_1 + \cdots + C_k]\!] &= [\![C_1]\!] \cup \cdots \cup [\![C_k]\!],\\
[\![a(P_{\Box,1}, \ldots, P_{\Box,r})]\!] &= \{a(T_1, \ldots, T_r) \mid \forall i\,.\, T_i \in [\![P_{\Box,i}]\!]\} \cup [\![P_{\Box,1}]\!] \cup \cdots \cup [\![P_{\Box,r}]\!],\\
[\![\Box]\!] &= \{\Box\}\,.
\end{aligned}$$

For example, $[\![(a(b(), \Box))^*.c^?()]\!]$ is the set of trees of the form either $b()$, or $c()$, or $a(b(), a(b(), \ldots a(b(), x) \ldots))$ with x either $b()$ or $c()$. Based on the results of Goubault-Larrecq and Schmitz [25] it is not difficult to deduce the following lemma (see [4, Appendix D] for a proof).

▶ **Lemma 4.1.** *Every set of trees $\mathcal{L}$ downward-closed for $\sqsubseteq$ can be represented as $\mathcal{L} = [\![P_1]\!] \cup \cdots \cup [\![P_k]\!]$, in which $P_1, \ldots, P_k$ are pure products.*

This decomposition result strengthens the results of Goubault-Larrecq and Schmitz [25] by showing that pure products (instead of just products) suffice in order to represent downward-closed sets of trees.

Transductions. A (nondeterministic) *finite tree transducer* (*FTT*) is a tuple $\mathcal{A} = (\mathbb{A}_{in}, \mathbb{A}_{out}, S, p^I, \delta)$, where $\mathbb{A}_{in}, \mathbb{A}_{out}$ are the input and output alphabets (finite, ranked), S is a finite set of *control states*, $p^I \in S$ is an *initial state*, and δ is a finite set of *transition rules* of the form either $(p, a(\mathsf{x}_1, \dots, \mathsf{x}_r)) \to T$ or $(p, \mathsf{x}) \to T$, where $p \in S$ is a control state, $a \in \mathbb{A}_{in}$ is a letter of rank r, and T is a finite tree over the alphabet $\mathbb{A}_{out} \cup (S \times \{\mathsf{x}_1, \dots, \mathsf{x}_r\})$ or $\mathbb{A}_{out} \cup (S \times \{\mathsf{x}\})$, respectively. The rank of all the pairs from $S \times \{\mathsf{x}_1, \dots, \mathsf{x}_r\}$ or $S \times \{\mathsf{x}\}$ is 0. An FTT is *linear* if for each rule of the form $(p, a(\mathsf{x}_1, \dots, \mathsf{x}_r)) \to T$ and for each $i \in \{1, \dots, r\}$, in T there is at most one letter from $S \times \{\mathsf{x}_i\}$, and moreover for each rule of the form $(p, \mathsf{x}) \to T$, in T there is at most one letter from $S \times \{\mathsf{x}\}$. An FTT $\mathcal{A}$ defines in a natural way a relation between finite trees, also denoted $\mathcal{A}$ (c.f. Comon et al. [19]). For a language $\mathcal{L}$ we write $\mathcal{A}(\mathcal{L})$ for the set of trees U such that $(T, U) \in \mathcal{A}$ for some $T \in \mathcal{L}$. A function that maps $\mathcal{L}$ to $\mathcal{A}(\mathcal{L})$ for some linear FTT $\mathcal{A}$ is called a *linear FTT transduction.*

▶ **Fact 4.2.** *The downward closure operation $\mathcal{L} \mapsto \mathcal{L}{\downarrow}$ and the regular restriction operation $\mathcal{L} \mapsto \mathcal{L} \cap \mathcal{R}$ (for every regular language $\mathcal{R}$) are effectively linear FTT transductions.*

▶ **Lemma 4.3** (c.f. [4, Appendix E])**.** *The class of languages of finite trees recognized by safe recursion schemes is effectively closed under linear FTT transductions.*

4.2 The simultaneous unboundedness problem for trees

We say that a pure product P is *diversified*, if no letter appears in P more than once. The *simultaneous unboundedness problem* (SUP) for a class $\mathfrak{C}$ of finite trees asks, given a diversified pure product P and a language $\mathcal{L} \in \mathfrak{C}$ such that $\mathcal{L} \subseteq [\![P]\!]$, whether $[\![P]\!] \subseteq \mathcal{L}{\downarrow}$.

▶ Remark 4.4. This is a generalization of SUP over finite words. In the latter problem, one is given a language of finite words $\mathcal{L}$ such that $\mathcal{L} \subseteq a_1^* \dots a_k^*$, and must check whether $a_1^* \dots a_k^* \subseteq \mathcal{L}{\downarrow}$. A word in $a_1^* \dots a_k^*$ can be represented as a linear tree by interpreting $a_1, \dots, a_k$ as unary letters and by appending a new leaf e at the end. Thus $a_1^* \dots a_k^*$ can be represented as the language of the diversified pure product $(a_1(\Box))^*.(a_2(\Box))^*. \cdots .(a_k(\Box))^*.e^?()$.

Following Zetzsche [54], we can reduce the computation of the downward closure to SUP.

▶ **Theorem 4.5** (c.f. [4, Appendix F])**.** *Let $\mathfrak{C}$ be a class of languages of finite trees closed under linear FTT transductions. One can compute a finite tree automaton recognizing the downward closure of a given language from $\mathfrak{C}$ if and only if SUP is decidable for $\mathfrak{C}$.*

▶ Remark 4.6. Pure products for trees correspond to expressions of the form $a_0^? A_1^* a_1^? \dots A_k^* a_k^?$ for words (where A_i are sets of letters). In SUP for words simpler expressions of the form $b_1^* \dots b_k^*$ suffice. This is not possible for trees:

1) expressions of the form $a^?(\cdot, \cdot)$ cannot be removed since they are responsible for branching, and
2) reducing the two contexts in $(a(P_1, \Box) + b(P_2, \Box))^*.P_3$ to a single one would require changing trees of the form $a(T_1, b(T_2, T_3))$ into trees of the form $c(T_1, T_2, T_3)$, which is not a linear FTT transduction.

4.3 The diagonal problem for trees

In SUP for words, instead of checking whether $a_1^* \dots a_k^* \subseteq \mathcal{L}{\downarrow}$, one can equivalently check whether, for each $n \in \mathbb{N}$, there is a word $a_1^{x_1} \dots a_k^{x_k} \in \mathcal{L}$ such that $x_1, \dots, x_k \geq n$. The latter problem is known as the *diagonal problem* for words. In this section, we define an analogous diagonal problem for trees, and we show how to reduce SUP to it.

Given a set of letters Σ, we say that a language of finite trees $\mathcal{L}$ is Σ-*diagonal* if, for every $n \in \mathbb{N}$, there is a tree $T \in \mathcal{L}$ such that for every letter $a \in \Sigma$ and every branch B of T there are at least n occurrences a in B. The *diagonal problem* for a class $\mathfrak{C}$ of finite trees asks, given a language $\mathcal{L} \in \mathfrak{C}$ and a set of letters Σ, whether $\mathcal{L}$ is Σ-diagonal.

Versatile trees. Contrary to the case of words, the presence of sums in our expressions creates some complications in reducing from SUP to the diagonal problem. We deal with these sums by introducing the notion of *versatile trees*. Intuitively, in order to obtain a versatile tree of a pure product P, for every sum $I = C_1 + \dots + C_k$ in P we fix some order of the contexts $C_1, \dots, C_k$, and we allow the contexts to be appended in this order. Formally, the set $(\!|P|\!)$ of versatile trees of a pure product P is defined by structural induction on P:

$$\begin{aligned}
(\!|I^*.P|\!) &= \bigcup_{n\in\mathbb{N}} (\!|I|\!)\underbrace{[((\!|I|\!) \cup \{\Box\})[\dots[((\!|I|\!) \cup \{\Box\})[}_{n}(\!|P|\!)]]\dots]],\\
(\!|a^?(P_1, \dots, P_r)|\!) &= (\!|a(P_1, \dots, P_r)|\!),\\
(\!|C_1 + \dots + C_k|\!) &= (\!|C_1|\!)[\dots[(\!|C_k|\!)]\dots],\\
(\!|a(P_{\Box,1}, \dots, P_{\Box,r})|\!) &= \{a(T_1, \dots, T_r) \mid \forall i\,.\, T_i \in (\!|P_{\Box,i}|\!)\}\,,\\
(\!|\Box|\!) &= \{\Box\}\,.
\end{aligned}$$

For example, if $I = a(S_1, \Box, \Box) + b(\Box, S_2)$, then $(\!|I|\!) = \{a(S_1, b(\Box, S_2), b(\Box, S_2))\}$. Notice that all trees in $(\!|P|\!)$ have the same root's label; denote this label by $\mathit{root}(P)$.

From SUP to the diagonal problem. Assuming that P is diversified, for a number $n \in \mathbb{N}$ we say that a tree T is *n-large with respect to P* if, for every subexpression of P of the form $I^*.P'$, above every occurrence of $\mathit{root}(P')$ in T there are at least n ancestors labeled by $\mathit{root}(I^*.P')$. In other words, for $T \in (\!|P|\!)$ this means that in T every context appearing in P was appended at least n times, on all branches where it was possible to append it. Clearly $(\!|P|\!) \subseteq [\![P]\!]$. On the other hand, every tree from $[\![P]\!]$ can be embedded into every large enough versatile tree. We thus obtain the following lemma.

▶ **Lemma 4.7.** *For every diversified pure product P, and for every sequence of trees $T_1, T_2, \dots \in (\!|P|\!)$ such that every T_n is n-large, $\{T_n \mid n \in \mathbb{N}\}{\downarrow} = [\![P]\!]$.*

Using versatile trees we can reduce from SUP to the diagonal problem.

▶ **Lemma 4.8** (c.f. [4, Appendix G]). *Let $\mathfrak{C}$ be a class of languages of finite trees closed under linear FTT transductions. SUP for $\mathfrak{C}$ reduces to the diagonal problem for $\mathfrak{C}$.*

▶ Remark 4.9. Another formulation of the diagonal problem for languages of finite trees [29, 14, 45] requires that, for every $n \in \mathbb{N}$, there is a tree $T \in \mathcal{L}$ containing at least n occurrences of every letter $a \in \Sigma$ (not necessarily on the same branch, unlike in our case). Such a formulation of the diagonal problem seems too weak to compute downward closures for languages of finite trees.

5 Languages of safe recursion schemes

In the previous section, we have developed a general machinery allowing one to compute downward closures for classes of languages of finite trees closed under linear FTT transductions. In this section, we apply this machinery to the particular case of languages recognized by safe recursion schemes. The following is the main theorem of this section.

▶ **Theorem 5.1.** *Finite tree automata recognizing downward closures of languages of finite trees recognised by safe recursion schemes are computable.*

In order to prove the theorem we need to recall a formalism necessary to express the diagonal problem in logic.

Cost logics. *Cost monadic logic* (*CMSO*) was introduced in Colcombet [15] as a quantitative extension of monadic second-order logic. As usual, the logic can be defined over any relational structure, but we restrict our attention to CMSO over trees. In addition to *first-order variables* ranging over nodes of the tree and *monadic second-order variables* (also called *set variables*) ranging over sets of nodes, CMSO uses a single additional variable N, called the *numeric variable*, which ranges over $\mathbb{N}$. The atomic formulas in CMSO are those from *MSO* (the membership relation $x \in X$ and relations $a(x, x_1, \dots, x_r)$ asserting that $a \in \mathbb{A}$ of rank r is the label at node x with children $x_1, \dots, x_r$ from left to right), as well as a new predicate $|X| \leqslant N$, where X is any set variable and N is the numeric variable. Arbitrary CMSO formulas are built inductively by applying Boolean connectives and by quantifying (existentially or universally) over first-order or set variables. We require that any predicates of the form $|X| \leqslant N$ appear positively in the formula (i.e., within the scope of an even number of negations). We regard N as a parameter. As usual, a *sentence* is a formula without first-order or monadic *free variables*; however, the parameter N is allowed to occur in a sentence. If we fix a value $n \in \mathbb{N}$ for N, the semantics of $|X| \leqslant N$ is what one would expect: the predicate holds when X has cardinality at most n. We say that a sentence φ *n-accepts* a tree T if it holds in T when n is used as value of N; it *accepts* T if it n-accepts T for some $n \in \mathbb{N}$. The language *defined* by φ is the set of all trees (over a fixed alphabet $\mathbb{A}$) accepted by φ.

Weak cost monadic logic (*WCMSO* for short) is the variant of CMSO where the second-order quantification is restricted to finite sets. Vanden Boom [53, Theorem 2] proves that WCMSO is effectively equivalent to a subclass of alternating B-automata, called *weak B-automata*. Thanks to Theorem 3.1, we obtain the following corollary.

▶ **Corollary 5.2.** *The model-checking problem of safe recursion schemes against WCMSO properties is decidable.*

▶ Remark 5.3. The same holds for a more expressive logic called *quasi-weak cost monadic logic* (*QWCMSO*) [6], whose expressive power lies between WCMSO and the CMSO. Indeed, Blumensath et al. [6, Theorem 2] prove that QWCMSO is effectively equivalent to a subclass of alternating B-automata called *quasi-weak B-automata*, and thus by Theorem 3.1 even model checking of safe recursion schemes against QWCMSO properties is decidable.

Solving the diagonal problem. By Theorem 4.5 and Lemma 4.8, all we need to do is to show that the diagonal problem is decidable for languages recognized by safe recursion schemes, that is, that given a safe recursion scheme $\mathcal{G}$ and a set of letters Σ, one can check whether for every $n \in \mathbb{N}$ there is a tree $T \in \mathcal{L}(\mathcal{G})$ such that there are at least n occurrences of every letter $a \in \Sigma$ on every branch of T (we say that such a tree T is *n-large with respect* *to* Σ). In order to do this, given a set of letters Σ, we write a WCMSO sentence φ_Σ that n-accepts an (infinite) tree T if and only if no tree in $\mathcal{L}(T)$ is n-large with respect to Σ. Consequently, φ_Σ accepts T if for some n no tree in $\mathcal{L}(T)$ is n-large with respect to Σ, that is, if $\mathcal{L}(T)$ is not Σ-diagonal. Thus, in order to solve the diagonal problem, it is enough to check whether φ_Σ accepts $BT(\mathcal{G})$ (recall that $\mathcal{L}(\mathcal{G})$ is defined as $\mathcal{L}(BT(\mathcal{G}))$), which is decidable by Corollary 5.2. It remains to construct the aforementioned sentence φ_Σ.

First, observe that the process of producing a finite tree recognized by $\mathcal{G}$ from the infinite tree $BT(\mathcal{G})$ generated by $\mathcal{G}$ is expressible by a formula of WCMSO (actually, by a first-order formula). More precisely we can write a WCMSO formula $\mathsf{tree}(X)$ that holds in a tree T if and only if X is instantiated to a set of nodes of a tree $T' \in \mathcal{L}(T)$, together with their nd-labeled ancestors. See [4, Appendix H] for more details. Using $\mathsf{tree}(X)$ we now construct the desired formula φ_Σ, and thus we finish the proof of Theorem 5.1.

▶ **Lemma 5.4.** *Given a set of letters Σ, one can compute a WCMSO sentence φ_Σ that, for every $n \in \mathbb{N}$, n-accepts a tree T if and only if no tree in $\mathcal{L}(T)$ is n-large with respect to Σ.*

Proof. We can reformulate the property as follows: for every tree $T' \in \mathcal{L}(T)$ there is a letter $a \in \Sigma$, and a leaf x that has less than n a-labeled ancestors. This is expressed by the following formula of WCMSO (where $\mathsf{leaf}(x)$ states that the node x is a leaf, $a(x)$ that x has label a, and $z \le x$ that z is an ancestor of x, all being easily expressible):

$$\forall X.\Big(\mathsf{tree}(X) \to \bigvee_{a\in\Sigma} \exists x \exists Z.\big(x \in X \wedge \mathsf{leaf}(x) \wedge \forall z.(z \le x \wedge a(z) \to z \in Z) \wedge |Z| < N\big)\Big). \quad ◀$$

6 Conclusions

A tantalising direction for further work is to drop the safety assumption from Theorem 3.1, that is, to establish whether the model-checking problem against B-automata is decidable for trees generated by (not necessarily safe) recursion schemes. We also leave open whether downward closures are computable for this more expressive class. Another direction for further work is to analyse the complexity of the considered model-checking problem. The related problem described in Remark 4.9 is k-EXP-complete for languages of finite trees recognised by recursion schemes of order k [45], and thus not harder than the nonemptiness problem [43]. Does the same upper bound hold for the more general diagonal problem that we consider in this paper? Zetzsche [56] has shown that the downward closure inclusion problem is co-k-NEXP-hard for languages of finite trees recognised by safe recursion schemes of order k. Is it possible to obtain a matching upper bound?

References

1. Parosh Aziz Abdulla, Luc Boasson, and Ahmed Bouajjani. Effective lossy queue languages. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 639–651. Springer, 2001. doi:10.1007/3-540-48224-5_53.
2. Alfred V. Aho. Indexed grammars - an extension of context-free grammars. *J. ACM*, 15(4):647–671, 1968. doi:10.1145/321479.321488.
3. Georg Bachmeier, Michael Luttenberger, and Maximilian Schlund. Finite automata for the sub- and superword closure of CFLs: Descriptional and computational complexity. In Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings*, volume 8977 of *Lecture Notes in Computer Science*, pages 473–485. Springer, 2015. doi:10.1007/978-3-319-15579-1_37.

4 David Barozzini, Lorenzo Clemente, Thomas Colcombet, and Paweł Parys. Cost Automata, Safe Schemes, and Downward Closures. *arXiv e-prints*, page arXiv:2004.12187, April 2020. arXiv:2004.12187.

5 Alessandro Berarducci and Mariangiola Dezani-Ciancaglini. Infinite lambda-calculus and types. *Theor. Comput. Sci.*, 212(1-2):29–75, 1999. doi:10.1016/S0304-3975(98)00135-2.

6 Achim Blumensath, Thomas Colcombet, Denis Kuperberg, Paweł Parys, and Michael Vanden Boom. Two-way cost automata and cost logics over infinite trees. 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 16:1–16:9. ACM, 2014. doi:10.1145/2603088.2603104.

7 Mikołaj Bojańczyk. A bounding quantifier. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic, 18th International Workshop, CSL 2004, 13th Annual Conference of the EACSL, Karpacz, Poland, September 20-24, 2004, Proceedings*, volume 3210 of *Lecture Notes in Computer Science*, pages 41–55. Springer, 2004. doi:10.1007/978-3-540-30124-0_7.

8 Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996. doi:10.1142/S0129054196000191.

9 Christopher H. Broadbent, Arnaud Carayol, C.-H. Luke Ong, and Olivier Serre. Recursion schemes and logical reflection. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 120–129. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.40.

10 Christopher H. Broadbent and Naoki Kobayashi. Saturation-based model checking of higher-order recursion schemes. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPIcs*, pages 129–148. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2013. doi:10.4230/LIPIcs.CSL.2013.129.

11 Christopher H. Broadbent and C.-H. Luke Ong. On global model checking trees generated by higher-order recursion schemes. In Luca de Alfaro, editor, *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5504 of *Lecture Notes in Computer Science*, pages 107–121. Springer, 2009. doi:10.1007/978-3-642-00596-1_9.

12 Arnaud Carayol and Olivier Serre. Collapsible pushdown automata and labeled recursion schemes: Equivalence, safety and effective selection. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 165–174. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.73.

13 Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. Ordered tree-pushdown systems. In Prahladh Harsha and G. Ramalingam, editors, *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16-18, 2015, Bangalore, India*, volume 45 of *LIPIcs*, pages 163–177. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.163.

14 Lorenzo Clemente, Paweł Parys, Sylvain Salvati, and Igor Walukiewicz. The diagonal problem for higher-order recursion schemes is decidable. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 96–105. ACM, 2016. doi:10.1145/2933575.2934527.

15 Thomas Colcombet. Regular cost functions, part I: Logic and algebra over words. *Logical Methods in Computer Science*, 9(3), 2013. doi:10.2168/LMCS-9(3:3)2013.

16 Thomas Colcombet and Stefan Göller. Games with bound guess actions. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 257–266. ACM, 2016. doi:10.1145/2933575.2934502.

17 Thomas Colcombet and Christof Löding. The non-deterministic Mostowski hierarchy and distance-parity automata. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 398–409. Springer, 2008. doi:10.1007/978-3-540-70583-3_33.

18 Thomas Colcombet and Christof Löding. Regular cost functions over finite trees. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 70–79. IEEE Computer Society, 2010. doi:10.1109/LICS.2010.36.

19 Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications, 2007. URL: http://tata.gforge.inria.fr/.

20 Bruno Courcelle. On constructing obstruction sets of words. *Bulletin of EATCS*, 1991.

21 Wojciech Czerwiński, Wim Martens, Lorijn van Rooijen, and Marc Zeitoun. A note on decidable separability by piecewise testable languages. In Adrian Kosowski and Igor Walukiewicz, editors, *Fundamentals of Computation Theory - 20th International Symposium, FCT 2015, Gdańsk, Poland, August 17-19, 2015, Proceedings*, volume 9210 of *Lecture Notes in Computer Science*, pages 173–185. Springer, 2015. doi:10.1007/978-3-319-22177-9_14.

22 Wojciech Czerwiński, Wim Martens, Lorijn van Rooijen, Marc Zeitoun, and Georg Zetzsche. A Characterization for Decidable Separability by Piecewise Testable Languages. *Discrete Mathematics & Theoretical Computer Science*, Vol. 19 no. 4, FCT '15, December 2017. doi:10.23638/DMTCS-19-4-1.

23 Werner Damm. The IO- and OI-hierarchies. *Theor. Comput. Sci.*, 20:95–207, 1982. doi:10.1016/0304-3975(82)90009-3.

24 Alain Finkel and Jean Goubault-Larrecq. Forward analysis for WSTS, part I: Completions. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, volume 3 of *LIPIcs*, pages 433–444. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, Germany, 2009. doi:10.4230/LIPIcs.STACS.2009.1844.

25 Jean Goubault-Larrecq and Sylvain Schmitz. Deciding piecewise testable separability for regular tree languages. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 97:1–97:15. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.97.

26 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.

27 Peter Habermehl, Roland Meyer, and Harro Wimmel. The downward-closure of Petri net languages. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 466–477. Springer, 2010. doi:10.1007/978-3-642-14162-1_39.

28 Axel Haddad. IO vs OI in higher-order recursion schemes. In Dale Miller and Zoltán Ésik, editors, *Proceedings 8th Workshop on Fixed Points in Computer Science, FICS 2012, Tallinn, Estonia, 24th March 2012.*, volume 77 of *EPTCS*, pages 23–30, 2012. doi:10.4204/EPTCS.77.4.

29 Matthew Hague, Jonathan Kochems, and C.-H. Luke Ong. Unboundedness and downward closures of higher-order pushdown automata. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 151–163. ACM, 2016. doi:10.1145/2837614.2837627.

30 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 452–461. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.34.

31 Graham Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc.*, s3-2(1):326–336, January 1952. doi:10.1112/plms/s3-2.1.326.

32 Richard Kennaway, Jan Willem Klop, M. Ronan Sleep, and Fer-Jan de Vries. Infinitary lambda calculus. *Theor. Comput. Sci.*, 175(1):93–125, 1997. doi:10.1016/S0304-3975(96)00171-5.

33 Teodor Knapik, Damian Niwiński, and Paweł Urzyczyn. Deciding monadic theories of hyperalgebraic trees. In Samson Abramsky, editor, *Typed Lambda Calculi and Applications, 5th International Conference, TLCA 2001, Kraków, Poland, May 2-5, 2001, Proceedings*, volume 2044 of *Lecture Notes in Computer Science*, pages 253–267. Springer, 2001. doi:10.1007/3-540-45413-6_21.

34 Teodor Knapik, Damian Niwiński, and Paweł Urzyczyn. Higher-order pushdown trees are easy. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002. doi:10.1007/3-540-45931-6_15.

35 Naoki Kobayashi. A practical linear time algorithm for trivial automata model checking of higher-order recursion schemes. In Martin Hofmann, editor, *Foundations of Software Science and Computational Structures - 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6604 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2011. doi:10.1007/978-3-642-19805-2_18.

36 Naoki Kobayashi. Model checking higher-order programs. *J. ACM*, 60(3):20:1–20:62, 2013. doi:10.1145/2487241.2487246.

37 Naoki Kobayashi and C.-H. Luke Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science, LICS 2009, 11-14 August 2009, Los Angeles, CA, USA*, pages 179–188. IEEE Computer Society, 2009. doi:10.1109/LICS.2009.29.

38 Gregory M. Kobele and Sylvain Salvati. The IO and OI hierarchies revisited. *Inf. Comput.*, 243:205–221, 2015. doi:10.1016/j.ic.2014.12.015.

39 Denis Kuperberg and Michael Vanden Boom. Quasi-weak cost automata: A new variant of weakness. In Supratik Chakraborty and Amit Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011, December 12-14, 2011, Mumbai, India*, volume 13 of *LIPIcs*, pages 66–77. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011.

40 Denis Kuperberg and Michael Vanden Boom. Quasi-weak cost automata: A new variant of weakness. In Supratik Chakraborty and Amit Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011, December 12-14, 2011, Mumbai, India*, volume 13 of *LIPIcs*, pages 66–77. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.66.

41 Jan van Leeuwen. Effective constructions in well-partially- ordered free monoids. *Discrete Mathematics*, 21(3):237–252, 1978. doi:10.1016/0012-365X(78)90156-5.

42 Robin P. Neatherway and C.-H. Luke Ong. TravMC2: Higher-order model checking for alternating parity tree automata. In Neha Rungta and Oksana Tkachuk, editors, *2014 International Symposium on Model Checking of Software, SPIN 2014, Proceedings, San Jose, CA, USA, July 21-23, 2014*, pages 129–132. ACM, 2014. doi:10.1145/2632362.2632381.

43 C.-H. Luke Ong. On model-checking trees generated by higher-order recursion schemes. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 81–90. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.38.

44 Paweł Parys. On the significance of the collapse operation. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 521–530. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.62.

45 Paweł Parys. The complexity of the diagonal problem for recursion schemes. In Satya Lokam and R. Ramanujam, editors, *Proc. of FSTTCS'17*, volume 93 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 45:1–45:14, Dagstuhl, Germany, 2017. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2017.45.

46 Paweł Parys. Recursion schemes and the WMSO+U logic. In Rolf Niedermeier and Brigitte Vallée, editors, *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, volume 96 of *LIPIcs*, pages 53:1–53:16. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.STACS.2018.53.

47 Steven J. Ramsay, Robin P. Neatherway, and C.-H. Luke Ong. A type-directed abstraction refinement approach to higher-order model checking. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 61–72. ACM, 2014. doi:10.1145/2535838.2535873.

48 Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. *Inf. Comput.*, 239:340–355, 2014. doi:10.1016/j.ic.2014.07.012.

49 Sylvain Salvati and Igor Walukiewicz. A model for behavioural properties of higher-order programs. In Stephan Kreutzer, editor, *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 229–243. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.CSL.2015.229.

50 Sylvain Salvati and Igor Walukiewicz. Simply typed fixpoint calculus and collapsible pushdown automata. *Mathematical Structures in Computer Science*, 26(7):1304–1350, 2016. doi:10.1017/S0960129514000590.

51 Larry J. Stockmeyer. *The complexity of decision problems in automata theory and logic.* PhD thesis, MIT, 1974.

52 Michael Vanden Boom. Weak cost monadic logic over infinite trees. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 2011.

53 Michael Vanden Boom. Weak cost monadic logic over infinite trees. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*, volume 6907 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 2011. doi:10.1007/978-3-642-22993-0_52.

54 Georg Zetzsche. An approach to computing downward closures. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Proc. of ICALP'15*, volume 9135 of *LNCS*, pages 440–451. Springer, 2015. doi:10.1007/978-3-662-47666-6_35.

55 Georg Zetzsche. Computing downward closures for stacked counter automata. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 743–756. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.743.

56 Georg Zetzsche. The complexity of downward closure comparisons. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 123:1–123:14. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.123.