MSO is decidable on trees generated by safe order-2 grammars

Deciding Monadic Theories of Hyperalgebraic Trees

Teodor Knapik[1], Damian Niwiński[2⋆], and Paweł Urzyczyn[2⋆⋆]

[1] Dept. de Mathématiques et Informatique, Université de la Réunion, BP 7151, 97715 Saint Denis Messageries Cedex 9, Réunion
knapik@univ-reunion.fr
[2] Institute of Informatics, Warsaw University
ul. Banacha 2, 02–097 Warszawa, Poland
{niwinski,urzy}@mimuw.edu.pl

Abstract. We show that the monadic second–order theory of any infinite tree generated by a higher–order grammar of level 2 subject to a certain syntactic restriction is decidable. By this we extend the result of Courcelle [6] that the MSO theory of a tree generated by a grammar of level 1 (algebraic) is decidable. To this end, we develop a technique of representing infinite trees by infinite λ–terms, in such a way that the MSO theory of a tree can be interpreted in the MSO theory of a λ–term.

Introduction

In 1969, Rabin [13] proved decidability of the monadic second–order (MSO) theory of the full n–ary tree, which is perhaps one of the most widely applied decidability results. There are several ways in which Rabin's Tree Theorem can be extended. One possibility is to consider a more general class of structures obtained in tree–like manner, i.e., by unwinding some initial structure. The decidability of the MSO theory of the unwound structure then relies on the decidability of the MSO theory of the initial structure, and the "regularity" of the unwinding process (see [16]). Another direction, which we will pursue here, is to remain with trees but consider more sophisticated modes of generation than unwinding.

To this end, it is convenient to rephrase Rabin's Tree Theorem for *labeled* trees, as follows: The MSO theory of any regular tree is decidable. Here, a labeled tree is seen as a logical structure with additional monadic predicates corresponding to the labels, and a tree is *regular* if it has only a finite number of non–isomorphic subtrees. An equivalent definition of regularity says that a tree is generated by a (deterministic) regular tree grammar, which gives rise to further generalizations. Indeed, Courcelle [6] proved that the MSO theory of any tree generated by an *algebraic* (or, context–free) tree grammar is also decidable. However nothing general is known about the MSO theories of trees generated

⋆ Partly supported by KBN Grant 8 T11C 027 16.
⋆⋆ Partly supported by KBN Grant 8 T11C 035 14.

S. Abramsky (Ed.): TLCA 2001, LNCS 2044, pp. 253–267, 2001.

by higher order grammars, although the expressive power of such grammars was extensively studied by Damm [7] in the early eighties. This is the question we address in the present paper.

It is plausible to think that any tree generated by a higher–order grammar (see Section 4 below) has decidable MSO theory, this however is only a conjecture. At present, we are able to show decidability of the MSO theory of trees generated by grammars of level 2 satisfying some additional condition restricting occurrences of individual parameters in scope of functional ones. This however properly extends the aforementioned result of Courcelle [6].

Our method makes use of the idea of the *infinitary* λ*–calculus*, already considered by several authors [9,4]. Here we view infinite λ–terms as infinite trees additionally equipped with edges from bound variables to their binders. In course of a possibly infinite sequence of β–reductions, these additional edges may disappear, and the result is a tree consisting of constant symbols only. We show that the MSO theory of the resulting tree can be reduced to the MSO theory of the original λ–term, viewed as an appropriate logical structure (Theorem 2 below). Let us stress that the reduction is not a mere interpretation (which seems to be hardly possible). Instead, we use the μ–calculus as an intermediate logic, and an intermediate structure obtained by folding the tree. In order to interpret the MSO theory of the folded tree in the MSO theory of the λ–term, we use techniques similar to Caucal [5], combined with an idea originated from Geometry of Interaction and the theory of optimal reductions. Namely, we consider deformations of regular paths in λ–terms and push–down store computations along these paths.

The motivation behind the of MSO theories is that the MSO theory of a λ–term should be easier to establish than that of the tree after reduction. This is indeed the case of grammars satisfying our restriction. More specifically, for each such grammar, we are able to construct an infinite λ–term which is essentially an algebraic tree, and whose result of β–reduction is precisely the tree generated by the grammar. Hence, by the aforementioned Courcelle's theorem, we get our decidability result (Theorem 4 below).

Let us mention that the interest in deciding formal theories of finitely presentable infinite structures has grown among the verification community during last decade (see, e.g., [11] and references therein). In particular, a problem related to ours was addressed by H. Hungar, who studied graphs generated by some specific higher–order graph grammars. He showed [8] decidability of the monadic second–order theory (S1S) of *paths* of such graphs (not the full MSO theory of graphs).

1 Preliminaries

Types. We consider a set of types T constructed from a unique *basic* type $\mathbf{0}$. That is $\mathbf{0}$ is a type and, if τ_1, τ_2 are types, so is $(\tau_1 \to \tau_2) \in \mathrm{T}$. The operator $\to$ is assumed to associate to the right. Note that each type is of the form $\tau_1 \to \cdots \to \tau_n \to \mathbf{0}$, for some $n \geq 0$. A type $\mathbf{0} \to \cdots \to \mathbf{0}$ with $n+1$ occurrences of $\mathbf{0}$ is also written $\mathbf{0}^n \to \mathbf{0}$. The level $\ell(\tau)$ of a type τ is defined by $\ell(\tau_1 \to \tau_2) =$

$\max(1+\ell(\tau_1),\ell(\tau_2))$, and $\ell(\mathbf{0})=0$. Thus $\mathbf{0}$ is the only type of level 0 and each type of level 1 is of the form $\mathbf{0}^n\to\mathbf{0}$ for some $n>0$. A type $\tau_1\to\cdots\to\tau_n\to\mathbf{0}$ is *homogeneous* (where $n\geq 0$) if each τ_i is homogeneous and $\ell(\tau_1)\geq\ell(\tau_2)\geq\ldots\geq\ell(\tau_n)$. For example $((\mathbf{0}\to\mathbf{0})\to\mathbf{0})\to(\mathbf{0}\to\mathbf{0})\to(\mathbf{0}\to\mathbf{0}\to\mathbf{0})\to\mathbf{0}\to\mathbf{0}$ is homogeneous, but $\mathbf{0}\to(\mathbf{0}\to\mathbf{0})\to\mathbf{0}$ is not.

Higher–order terms. A *typed alphabet* is a set Γ of symbols with types in T. Thus Γ can be also presented as a T–indexed family $\{\Gamma_\tau\}_{\tau\in\mathrm{T}}$, where Γ_τ is the set of all symbols of Γ of type τ. We let the *type level* $\ell(\Gamma)$ of Γ be the supremum of $\ell(\tau)$, such that Γ_τ is nonempty. A *signature* is a typed alphabet of level 1.

Given a typed alphabet Γ, the set $T(\Gamma)=\{T(\Gamma)_\tau\}_{\tau\in\mathrm{T}}$ of *applicative terms* is defined inductively, by

(1) $\Gamma_\tau\subseteq T(\Gamma)_\tau$; (2) if $t\in T(\Gamma)_{\tau_1\to\tau_2}$ and $s\in T(\Gamma)_{\tau_1}$ then $(ts)\in(\Gamma)_{\tau_2}$.

Note that each applicative term can be presented in a form $Zt_1\ldots t_n$, where $n\geq 0$, $Z\in\Gamma$, and $t_1,\ldots,t_n$ are applicative terms. We say that a term $t\in T(\Gamma)_\tau$ is of type τ, which we also write $t:\tau$. We adopt the usual notational convention that application associates to the left, i.e. we write $t_0t_1\ldots t_n$ instead of $(\cdots((t_0t_1)t_2)\cdots)t_n$.

Trees. The free monoid generated by a set X is written X^* and the empty word is written ε. The length of word $w\in X^*$ is denoted by $|w|$. A *tree* is any nonempty prefix–closed subset T of X^* (with ε considered as the *root*). If $u\in T$, $x\in X$, and $ux\in T$ then ux is an *immediate successor* of u in T. For $w\in T$, the set $T.w=\{v\in X^*: wv\in T\}$ is the *subtree* of T induced by w. Note that $T.w$ is also a tree, and $T.\varepsilon=T$.

Now let Σ be a signature and let $T\subseteq\omega^*$, where ω is the set of natural numbers, be a tree. A mapping $t{:}\,T\to\Sigma$ is called a *Σ–tree* provided that if $t(w):\mathbf{0}^k\to\mathbf{0}$ then w has exactly k immediate successors which are $w1,\ldots,wk$ (hence w is a leaf whenever $t(w):\mathbf{0}$). The set of Σ–trees is written $T^\infty(\Sigma)$.

If $t{:}\,T\to\Sigma$ is a Σ–tree, then T is called the *domain* of t and denoted by $T=\mathcal{D}\mathrm{om}\ t$. For $v\in\mathcal{D}\mathrm{om}\ t$, the *subtree* of t induced by v is a Σ–tree $t.v$ such that $\mathcal{D}\mathrm{om}\ t.v=(\mathcal{D}\mathrm{om}\ t).v$, and $t.v(w)=t(vw)$, for $w\in\mathcal{D}\mathrm{om}\ t.v$. It is convenient to organize the set $T^\infty(\Sigma)$ into an algebra over the signature Σ, where for each $f\in\Sigma_{\mathbf{0}^n\to\mathbf{0}}$, the operation associated with f sends an n–tuple of trees $t_1,\ldots,t_n$ onto the unique tree t such that $t(\varepsilon)=f$ and $t.i=t_i$, for $i\in[n]$. (The notation $[n]$ abbreviates $\{1,\ldots,n\}$). Finite trees in $T^\infty(\Sigma)$ can be also identified with applicative terms of type $\mathbf{0}$ over the alphabet Σ in the usual manner.

We introduce a concept of limit. For a Σ–tree t, let $t\restriction n$ be its truncation to the level n, i.e., the restriction of the function t to the set $\{w\in\mathcal{D}\mathrm{om}\ t:|w|\leq n\}$. Suppose $t_0,t_1,\ldots$ is a sequence of Σ–trees such that, for all k, there is an m, say $m(k)$, such that, for all $n,n'\geq m(k)$, $t_n\restriction k=t_{n'}\restriction k$. (This is a Cauchy condition in a suitable metric space of trees.) Then the *limit* of the sequence t_n, in symbols $\lim t_n$, is a Σ–tree t which is the set–theoretical union of the functions $t_n\restriction m(n)$ (understanding a function as a set of pairs).

Types as trees. Types in T can be identified with finite (unlabeled) binary trees. More specifically, we use the set of directions $\{p,q\}$, and let $\mathit{tree}(\tau_1\to\tau_2)$ be the

unique tree such that $tree(\tau_1 \to \tau_2).p = tree(\tau_1)$, $tree(\tau_1 \to \tau_2).q = tree(\tau_2)$ and $tree(\mathbf{0}) = \{\varepsilon\}$. In the sequel we will not make notational distinction between τ and $tree(\tau)$.

Monadic second–order logic. Let R be a *relational vocabulary*, i.e., a set of relational symbols, each r in R given with an arity $\rho(r) > 0$. The formulas of *monadic second order (MSO) logic* over vocabulary R use two kinds of variables : *individual variables* $x_0, x_1, \ldots$, and *set variables* $X_0, X_1, \ldots$. Atomic formulas are $x_i = x_j$, $r(x_{i_1}, \ldots, x_{i_{\rho(r)}})$, and $X_i(x_j)$. The other formulas are built using propositional connectives $\vee, \neg$, and the quantifier $\exists$ ranging over both kinds of variables. (The connectives $\wedge, \Rightarrow$, etc., as well as the quantifier $\forall$ are introduced in the usual way as abbreviations.) A formula without free variables is called a *sentence*. Formulas are interpreted in relational structures over the vocabulary R, which we usually present by $\mathbf{A} = \langle A, \{r^{\mathbf{A}} : r \in R\}\rangle$, where A is the *universe* of $\mathbf{A}$, and $r^{\mathbf{A}} \subseteq A^{\rho(r)}$ is a $\rho(r)$–ary relation on A. A *valuation* is a mapping v from the set of variables (of both kinds), such that $v(x_i) \in A$, and $v(X_i) \subseteq A$. The *satisfaction* of a formula φ in $\mathbf{A}$ under the valuation v, in symbols $\mathbf{A}, v \models \varphi$ is defined by induction on φ in the usual manner. The *monadic second–order theory* of $\mathbf{A}$ is the set of all MSO sentences satisfied in $\mathbf{A}$.

Let Σ be a signature and suppose that the maximum of the arities of symbols in Σ exists and equals m_Σ. A tree $t \in T^\infty(\Sigma)$ can be viewed as a logical structure $\mathbf{t}$, over the vocabulary $R_\Sigma = \{p_f : f \in \Sigma\} \cup \{d_i : 1 \leq i \leq m_\Sigma\}$, with $\rho(p_f) = 1$, and $\rho(d_i) = 2$:

$$\mathbf{t} = \langle \mathcal{D}\mathrm{om}\, t, \{p_f^{\mathbf{t}} : f \in \Sigma\} \cup \{d_i^{\mathbf{t}} : 1 \leq i \leq m_\Sigma\}\rangle.$$

The universe of $\mathbf{t}$ is the domain of t, and the predicate symbols are interpreted by $p_f^{\mathbf{t}} = \{w \in \mathcal{D}\mathrm{om}\, t : t(w) = f\}$, for $f \in \Sigma$, and $d_i^{\mathbf{t}} = \{(w, wi) : wi \in \mathcal{D}\mathrm{om}\, t\}$, for $1 \leq i \leq m_\Sigma$. We refer the reader to [15] for a survey of the results on monadic second–order theory of trees.

2 Infinitary λ–Calculus

We will identify infinite λ–terms with certain infinite trees. More specifically, we fix a finite signature Σ and let $\Sigma^\perp = \Sigma \cup \{\perp\}$, where $\perp$ is a fresh symbol of type $\mathbf{0}$. All our finite and infinite terms, called *λ–trees* are simply typed and may involve constants from $\Sigma^\perp$, and variables from a fixed countably infinite set. In fact, we only consider λ–trees of types of level at most 1.

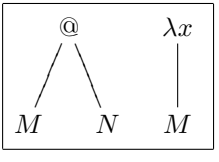


Fig. 1. Application and abstraction

Let Σ° be an infinite alphabet of level 1, consisting of a binary function symbol @, all symbols from $\Sigma^\perp$ as individual constants, regardless of their actual types, infinitely many individual variables as individual constants, unary function symbols λx for all variables x. The set of all *λ–trees* (over a signature Σ) is the greatest set of Σ°–trees, given together with their *types*, such that the following conditions hold.

- Each variable x is a λ–tree of type $\mathbf{0}$.
- Each function symbol $f \in \Sigma^{\perp}$ of type τ is a λ–tree of type τ.
- Otherwise each λ–tree is of type of level at most 1 and is either an *application* (MN) or an *abstraction* $(\lambda x.M)$ (see Fig. 1).
- If a λ–tree P of type τ is an application (MN) then M is a λ–tree of type $\mathbf{0} \to \tau$, and N is a λ–tree of type $\mathbf{0}$.
- If a λ–tree P of type τ has the form $(\lambda x.M)$, then $\tau = \mathbf{0} \to \sigma$, and M is a λ–tree of type σ.

Strictly speaking, the above is a co–inductive definition of the two–argument relation "M is a λ–tree of type τ". Formally, a λ–tree can be presented as a pair (M, τ), where M is a Σ°–tree, and τ is its type satisfying the conditions above. Whenever we talk about a "λ–tree" we actually mean a λ–tree together with its type.

Let M be a λ–tree and let x be a variable. Each node of M labeled x is called an *occurrence* of x in M. An occurrence of x is *bound* (resp. *free*), iff it has an (resp. no) ancestor labeled λx. The *binder* of this occurrence of x is the closest of all such ancestors λx. A variable x is *free* in a λ–tree M iff it has a free occurrence in M. The (possibly infinite) set of all free variables of M will be denoted by $FV(M)$. A λ–tree M with $FV(M) = \varnothing$ is called *closed*.

Definition 1. We call a λ–tree M *boundedly typed* if the set of types of all subterms of M is finite.

Clearly, ordinary λ–terms can be seen as a special case of λ–trees, and the notion of a free variable in a λ–tree generalizes the notion of a free variable in a λ–term. The *n–th approximant* of a λ–tree M, denoted $M{\restriction}n$ is defined by induction as follows:

- $M{\restriction}0 = \perp$, for all M;
- $(MN){\restriction}(n{+}1) = (M{\restriction}n)(N{\restriction}n)$
- $(\lambda x.M){\restriction}(n{+}1) = \lambda x(M{\restriction}n)$

That is, the n–th approximant is obtained by replacing all subtrees rooted at depth n by the constant $\perp$.

We denote by $M[x := N]$ the result of substitution of all free occurrences of x in M by N. The definition of the substitution of λ–trees is similar to that for ordinary λ–terms. (An α–conversion of some subterms of M may be necessary in order to avoid the capture of free variables of N.)

A *redex* in a λ–tree M is a subtree of the form $(\lambda x.P)Q$. The *contractum* of such a redex is of course $P[x := Q]$. We write $M \to_{\beta} N$ iff N is obtained from M by replacing a redex by its contractum. Note that infinite λ–trees, even simply typed, may have infinite reduction sequences, due to infinitely many redexes.

2.1 Paths in λ–Graphs

To each λ–tree M, we associate a *λ–graph* $G(M)$. Some edges of $G(M)$ are oriented and labeled either p or q. Other edges are non–oriented and unlabeled. To construct $G(M)$ we start with M where (for technical reasons) we add an additional node labeled "c" above each application, i.e., above any @–node.

(We refer to a node labeled by a symbol σ as to a σ–node.) We add an edge, oriented downward and labeled q from each c–node to the corresponding @–node. We also add an edge oriented upward and labeled p, form each bound occurrence of a variable to its binder. Since the α–equivalent λ–trees may be identified, without loss of information, we can replace all labels λx just by "λ" and all bound occurrences of variables by a uniform label "v". In addition we assign labels and orientation to the following existing edges. Each edge connecting the argument of an application with the corresponding @–node is labeled p and oriented upward. Each edge connecting the body of an abstraction with the corresponding λ–node is labeled q and oriented upward. Different cases of nodes and edges of $G(M)$ are depicted on Fig. 2.

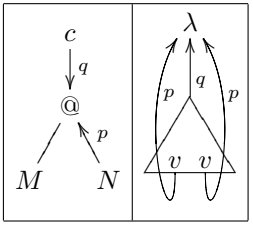


Fig. 2. Labeled application and abstraction

Each subterm N of M corresponds in an obvious way to a subgraph of $G(M)$, which will be denoted $G(N)$. Observe that nodes corresponding to free variables of N are connected to λ–nodes outside of $G(N)$, and these connecting edges are not part of $G(N)$. Each of these graphs $G(N)$ has a distinguished *entry* node (drawn at the top). The entry node of an abstraction is the appropriate λ–node, the entry node of an application is the additional c–node (not the @–node) and the entry node of a variable is the v–node itself. If confusion does not arise, we will write $\alpha := \alpha_N$ to mean that α is an entry node of a subterm N in M. Note that each node in $G(M)$, except for @–nodes, is an entry node of $G(N)$, for some subterm N of M.

Following the ideas of the Geometry of Interaction, see [2,3], we will now consider paths in λ–graphs. A sequence of adjacent edges in a λ–graph (possibly including the extra arcs from variable nodes to their binders), is called a *straight path* provided there is no backtracking, i.e., no edge is taken twice in a row and two edges connecting two different variable nodes with a λ–node which is their common binder may not directly follow one another. From now on by a *path* we always mean a straight path. Note that a path can pass an oriented edge forward (obeying the orientation) or backward (against the orientation).

Following [14], we will now consider a certain stateless pushdown automaton $\mathcal{P}$ walking on paths in a λ–graph. Informally, $\mathcal{P}$ moves along edges of a path Π, and each time it traverses a labeled edge forward or backward, a pushdown store (pds, for short) operation is performed (no pds operation if there is no label). Whenever we follow an arrow labeled p, we push p on the pds, and in order to traverse such an arrow backward, we must pop p from the pds. We proceed in an analogous way when we use edges labeled by q, forward or backward. In particular, in order to take a p–arrow backward, the top of the pds must be p, and similarly for q. This can be described more formally, considering that $\mathcal{P}$ works with the input alphabet $\{p\uparrow, p\downarrow, q\uparrow, q\downarrow, |\}$ ($|$ for a nonlabeled edge). A *configuration* of $\mathcal{P}$ is defined as a pair of the form (α, w) consisting of a node α and a word w representing the pds contents (top at the left). Note that once the

path Π is fixed, the behaviour of $\mathcal{P}$ is fully determined by the initial contents of the pds.

An important property of $\mathcal{P}$ is that the contents of the pds in a configuration (α, w) may contain some information about the type of the subterm N whose entry node is α (i.e., $\alpha = \alpha_N$). Note that, according to our representation of types as trees, the subtree $\tau.w$ of a type τ is again a type.

Lemma 1. *Assume that $\mathcal{P}$ can move from (α_N, w) to (α_P, v), and that $N : \tau$ and $P : \sigma$. Suppose that $\tau.w$ is defined. Then $\sigma.v$ is defined and $\tau.w = \sigma.v$. Similarly, if $\sigma.v$ is defined then so is $\tau.w$ and the equality holds.*

In other words, the pds of $\mathcal{P}$ can be seen as a pointer to a certain subtype of the type of currently visited node. A crucial consequence of this fact is that if our λ–tree is boundedly typed (see Definition 1) then $\mathcal{P}$ can essentially be replaced by a finite automaton, if we only consider computations beginning with configurations (α_N, w), such that $\tau.w$ is defined, where τ is the type of N. Indeed, by Lemma 1, the type pointed to by the pds during the whole computation is $\tau' = \tau.w$. Now, if there are altogether only finitely many types in use, the type τ' can occur in all these types in a finite number of positions only. This means that there is only a finite number of possible values of the pds (uniformly bounded for a boundedly typed λ–term). Then we can convert the pds contents of $\mathcal{P}$ into states of a *finite* automaton. Thus we can compare computations of both automata in an obvious way.

We summarize above considerations in the following.

Proposition 1. *Suppose a λ–tree M is boundedly typed. There is a deterministic finite automaton $\mathcal{A}$ whose set of states is a subset of $\{p, q\}^*$, and whose computations along paths in $G(M)$ coincide with the computations of $\mathcal{P}$. Whenever $\mathcal{P}$ can traverse a path in $G(M)$, $\mathcal{A}$ can do it as well.*

It will now be convenient to define a computation path (of $\mathcal{P}$) more formally. A *computation path* Π in a λ–graph M is a finite or infinite sequence of configurations $(\alpha_0, w_0), (\alpha_1, w_1), \ldots$, such that the corresponding sequence of edges $(\alpha_0, \alpha_1), (\alpha_1, \alpha_2), \ldots$ forms a straight path (if Π is infinite, we mean that each initial segment is straight), and $w_0, w_1, \ldots$ are consecutive contents of the pds in $\mathcal{P}$'s computation along this path. Note that we allow a *trivial* computation path consisting of a single configuration (α_0, w_0) (no edges). A computation path is *maximal* if it is infinite or finite but cannot be extended to a longer computation path.

Now suppose a computation path Π ends in a configuration (α_N, w). If Π is nontrivial, there are two ways in which Π may reach the last configuration: It either comes from outside of $G(N)$ (i.e., the last but one node is not in $G(N)$) or from inside of $G(N)$. We will call Π *South–oriented* in the first case, and *North–oriented* in the second (as Π comes "from above" or "from below", respectively). If Π is trivial, and hence consists only of (α_N, w), we will qualify it as South–oriented if N is an application or a signature symbol of arity 0, and North–oriented if N is a signature symbol of arity > 0 (we do not care for other

cases). Now, let $N:\tau$. We will say that Π is *properly ending* if the type $\tau.w$ is defined and equals $\mathbf{0}$, and moreover

- if Π is South–oriented then $w = q^n$, for some $n \geq 0$,
- if Π is North–oriented then $w = q^n p$, for some $n \geq 0$.

We are ready to state a lemma that will be crucial for further applications (see [10] for a proof).

Lemma 2. *Let M be a closed λ–tree of type $\mathbf{0}$.*

(1) *There is exactly one maximal computation path Π starting from configuration (α_M, ε). If Π is finite then it must end in a configuration (α_g, q^n), for some signature symbol g of arity n.*
(2) *Let $\alpha = \alpha_f$, be a node of M, where f is a signature symbol of arity $k > 0$, and let $i \in [k]$. There is exactly one maximal computation path Π starting from configuration $(\alpha_f, q^{i-1}p)$. If Π is finite then it must end in a configuration (α_g, q^n), for some signature symbol g of arity n.*

2.2 Derived Trees

An infinite tree in $T^\infty(\Sigma^\perp)$ can be viewed as an infinite λ–term in a natural way if we read $f(t_1, \ldots, t_k)$ as the nested application $(\ldots((ft_1)t_2)\ldots t_k)$.

Conversely, we will show a method to derive a tree in $T^\infty(\Sigma^\perp)$ from a closed boundedly typed λ–tree M of type $\mathbf{0}$. Intuitively, this will be a tree to which M eventually evaluates, after performing all β–reductions. However, not all branches of reduction need converge, and therefore we will sometimes put $\perp$ instead of a signature symbol.

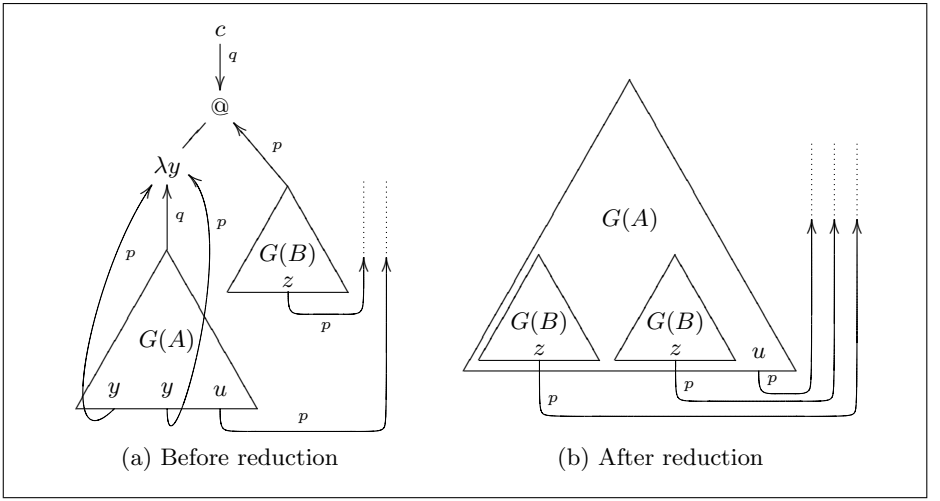
A tree $t_M\colon \mathcal{D}\mathrm{om}\, t_M \to \Sigma^\perp$ will be defined along with a partial mapping I_M from $\mathcal{D}\mathrm{om}\, t_M$ to $G(M)$ in such a way that the label of $I_M(w)$ in $G(M)$ coincides with $t_M(w)$. More specifically, the domain of I_M will be $\mathcal{D}\mathrm{om}_+ t_M = \{w \in \mathcal{D}\mathrm{om}\, t_M : t_M(w) \neq \perp\}$. At first, consider the maximal computation path Π in $G(M)$ starting from (α_M, ε) (cf. Lemma 2). If it is infinite, we let $t_M = \perp$ and $I_M = \varnothing$. Otherwise, again by Lemma 2, Π ends in a node labeled by a signature symbol. We call this node the *source of* M and denote it by s_M. We let $I_M\colon \varepsilon \mapsto s_M$, and $t_M(\varepsilon) = f$, where f is the label of s_M. Now suppose I_M and t_M are defined for a node w, and $t_M(w) = g$, $I_M(w) = \alpha = \alpha_g$, where g is a signature symbol of arity k. For each $i = 1, \ldots, k$, we consider the maximal computation path Π_i in $G(M)$ starting from $(\alpha, q^{i-1}p)$. By Lemma 2, the path Π_i is well defined, and if it is finite, the last node, say α_i, is labeled by a signature symbol, say g_i. Then we define t_M and (possibly) I_M for the k successors of w, by $t_M(wi) = g_i$ and $I_M(wi) = \alpha_i$, if Π_i is finite, and $t_M(wi) = \perp$, otherwise.

For Sect. 3, we also need an extension I^*_M of I_M defined on the whole $\mathcal{D}\mathrm{om}\, t$. Without loss of generality, we can assume that the root ε of $G(M)$ is not labeled by a signature symbol.[1] We let $I^*_M(w) = I_M(w)$ if $I_M(w)$ is defined and $I^*_M(w) = \varepsilon$ otherwise.

[1] Otherwise $G(M)$ consists of a single node labeled by a constant, and all the results in the sequel become trivial. We choose ε for concreteness, but any other node not in $I_M(\mathcal{D}\mathrm{om}_+ t_M)$ could be used instead.

2.3 Beta Reduction

We will now examine how a β–reduction step applied to a λ–tree M may affect a path Π in $G(M)$. Suppose M_1 is obtained from M by a β–reduction replacing a redex $(\lambda y.A)B$ by $A[y := B]$. In order to transform $G(M)$ into $G(M_1)$ one finds out all the variable nodes bound by the λ–node at the top of $G(\lambda y.A)$. Then the subgraph $G((\lambda y.A)B)$ is replaced by $G(A)$ where all such variable nodes are replaced by copies of the whole graph $G(B)$. We consider only the case when y



(a) Before reduction (b) After reduction

Fig. 3.

is free in A, the other case is easier and left to the reader. Fig. 3(a) presents a redex, and Fig. 3(b) shows its contractum. Consider a weakly regular path Π_1 in M_1. We define a path Π in M as follows:

- Outside of the redex the path is unchanged.
- The same for portions of the path within $G(A[y := B])$ but outside of $G(B)$.
- Each copy of $G(B)$ in $G(A[y := B])$ is represented in the redex by the single argument $G(B)$. Every portion of the path Π_1 that goes through any of the multiple $G(B)$'s in $G(A[y := B])$ is replaced by an identical portion going through $G(B)$ in the redex.
- Whenever Π_1 enters $G(A[y := B])$ through its top node, the path Π enters $G((\lambda y.A)B)$ through its top node (a q–arrow), then goes to $G(\lambda y.A)$ and takes the q–arrow backward to enter $G(A)$. (Note that there is no other choice.)
- Whenever Π_1 enters $G(B)$ through its top node the path Π reaches a v–node within $G(A)$. Then it must go to the argument $G(B)$, traversing p and then p backward.
- Whenever Π_1 enters or leaves $G(A[y := B])$ through a variable (i.e., via an edge between a variable node to its binder), the path Π enters or leaves the redex through the corresponding variable. (Note that no variable free in B can be bound in $A[y := B]$.)

In this case we say that Π_1 is a *deformation* of Π.

Suppose now that $M \to_\beta N$. Then every node β of $G(N)$ may be seen as obtained from a node α of $G(M)$. We say that β is an *offset* of α. This notion should be obvious up to one specific point: the entry node to a contractum of a redex (the entry to $G(A)$ at Fig. 3(b)) should be considered an offset of the entry node of the body of the abstraction (the entry to $G(A)$ at Fig. 3(a)) and *not* of the entry node of the redex or of the abstraction.

In this way we can say that a node of $G(M)$ may have one or more offsets or no offset at all, but each node of $G(N)$ is an offset of exactly one node of $G(M)$. In addition, a variable or constant may only be an offset of an identical variable or constant. It should be obvious that the type associated to a node and to its offset must be the same.

Let a path Π_1 in $G(N)$ from node β_1 to node β_2 be a deformation of a path Π in $G(M)$. Let α_1 and α_2 be respectively the initial and final nodes of Π. Then of course β_1 and β_2 are respectively offsets of α_1 and α_2.

Proposition 2. *Let M and N be closed λ–trees of level $\mathbf{0}$, and let t_M and t_N be the respective derived trees. If $M \to_\beta N$ then $t_M = t_N$.*

Proof. Consider the inductive construction of the tree t_N, as described in Section 2.2. It may be readily established that each computation path in $G(N)$ used in this construction is a deformation of a computation path in $G(M)$. If the former is infinite, the latter must be infinite too. If the former reaches a signature symbol f, so must the latter, because α_f must be an offset of α_f. Hence the result follows by induction on the levels of the tree t_N. □

3 Moving between MSO Theories

Let M be a closed boundedly typed λ–tree of level $\mathbf{0}$. We are going to show that the MSO theory of the derived tree t_M can be interpreted in the MSO theory of $G(M)$, viewed as a specific logical structure $\mathbf{G}_M$ defined below. We shall see that both structures are bisimilar in the usual process–theoretic sense (see e.g. Definition 6.3.10 of [1]). By composing several well–known facts about the propositional modal μ–calculus (see e.g. [1,12]), we can establish the following.

Proposition 3. *There is a recursive mapping ρ of MSO sentences such that for every MSO sentence φ, every tree $t \in T^\infty(\Sigma)$ and every countable structure $\mathbf{A}$ which is bisimilar to $\mathbf{t}$, the following holds:* $\mathbf{t} \models \varphi \iff \mathbf{A} \models \rho(\varphi)$.

We define $\Sigma^\bullet := \Sigma \cup \{@, \lambda, c, v\}$, where the symbol @ is binary, λ and c are unary, and v, as well as all symbols from Σ are 0–ary. Recall that $G(M)$ is a tree over $\Sigma^\bullet$ additionally equipped with the edges from v–nodes to their binders (λ–nodes). Let us denote the domain of $G(M)$ by W. We consider the structure $\mathbf{G}_M = \langle W, \{p_f^{\mathbf{G}_M} \mid f \in \Sigma^\bullet\} \cup \{d_i^{\mathbf{G}_M} \mid 1 \le i \le m_{\Sigma^\bullet}\} \cup \{E^{\mathbf{G}_M}\}\rangle$ where $p_f^{\mathbf{G}_M}$ and $d_i^{\mathbf{t}}$ are defined as in Section 1, and $(u, w) \in E^{\mathbf{G}_M}$, whenever $G(M)(u) = v$, $G(M)(w) = \lambda$, and there is an edge in $G(M)$ from u to w.

We now define the structure $\mathbf{I}_M$ over the same vocabulary as $\mathbf{t}_M$, of universe $I_M^*(\mathcal{D}\mathrm{om}\ t_M)$ by letting, for $f \in \Sigma^\perp$, we let $p_f^{\mathbf{I}_M} = \{I_M^*(w) : t_M(w) = f\}$, and, for $1 \leq i \leq m_\Sigma$, $d_i^{\mathbf{I}_M} = \{(I_M^*(w), I_M^*(wi)) : wi \in \mathcal{D}\mathrm{om}\ t_M\}$. Clearly I_M^* is an epimorphism from $\mathbf{t}_M$ onto $\mathbf{I}_M$, and moreover $(I_M^*(w), I_M^*(v)) \in d_i^{\mathbf{I}_M}$ implies $(w, v) \in d_i^{\mathbf{t}_M}$, and $I_M^*(w) \in p_f^{\mathbf{I}_M}$ implies $w \in p_f^{\mathbf{t}_M}$. This follows that $\mathbf{t}_M$ and $\mathbf{I}_M$ are bisimilar as transition systems.

The next lemma allows to accomplish the interpretation of the MSO theory of $\mathbf{t}_M$ in that of $\mathbf{G}_M$.

Lemma 3. *For any MSO sentence φ, one can construct effectively an MSO sentence ψ such that $\mathbf{I}_M \models \varphi$ if and only if $\mathbf{G}_M \models \psi$.*

Proof. It is enough to interpret the structure $\mathbf{I}_M$ in the MSO theory of $\mathbf{G}_M$. Since the universe of $\mathbf{I}_M$ is already a subset of the universe of $G(M)$, it is enough to write MSO formulas, say $Uni(x)$, $P_f(x)$, for $f \in \Sigma^\perp$, and $D_i(x, y)$, for $1 \leq i \leq m_\Sigma$, defining the relations $I_M^*(\mathcal{D}\mathrm{om}\ t_M)$, $p_f^{\mathbf{I}_M}$, and $d_i^{\mathbf{I}_M}$, respectively.

The formulas $P_f(x)$ are obvious. To write formulas $D_i(x, y)$, the key point is to express a property "there is a finite computation path Π starting from configuration $(\alpha_f, q^{i-1}p)$ (where f is a signature symbol of arity k), and ending in a node labeled by a signature symbol". Note that such a path must be maximal, and hence, by Lemma 2, there is at most one such path. Moreover, by Proposition 1, this computation can be carried by a finite automaton. (It follows easily from Lemma 2 that the computation empties pds at least once.) Therefore, it is routine to express the desired property by the known techniques, see Caucal [5]. The existence of an infinite computation path can be expressed by negation of the existence of finite maximal paths. The argument for expressing a computation path starting from (α_M, ε) is similar. This allows to write formulas $D_i(x, y)$. Using formulas $P_f(x)$ and $D_i(x, y)$, it is routine to write the desired formula $Uni(x)$. □

By combining Proposition 3 and Lemma 3, we get the following result.

Theorem 2. *Let M be a closed boundedly typed λ–tree of type $\mathbf{0}$, and let $t_M \in T^\infty(\Sigma^\perp)$ be the tree derived from M. Then the MSO theory of $\mathbf{t}_M$ is reducible to the MSO theory of $\mathbf{G}_M$, that is, there exists a recursive mapping of sentences $\varphi \mapsto \varphi'$ such that $\mathbf{t}_M \models \varphi$ iff $\mathbf{G}_M \models \varphi'$.*

4 Grammars

We now fix two disjoint typed alphabets, $N = \{N_\tau\}_{\tau \in \mathrm{T}}$ and $\mathrm{X} = \{\mathrm{X}_\tau\}_{\tau \in \mathrm{T}}$ of *nonterminals* and *variables* (or *parameters*), respectively. A *grammar* is a tuple $\mathcal{G} = (\Sigma, V, S, E)$, where Σ is a signature, $V \subseteq N$ is a finite set of nonterminals, $S \in V$ is a *start symbol* of type $\mathbf{0}$, and E is a set of equations of the form $Fz_1 \dots z_m = w$, where $F : \tau_1 \to \dots \to \tau_m \to \mathbf{0}$ is a nonterminal in V, z_i is a variable of type τ_i, and w is an applicative term in $T(\Sigma \cup V \cup \{z_1 \dots z_m\})$.

We assume that for each $F \in V$, there is exactly one equation in E with F occurring on the left hand side. Furthermore, we make a *proviso* that each nonterminal in a grammar has a homogeneous type, and that if $m \geq 1$ then $\tau_m = \mathbf{0}$. This implies that each nonterminal of level > 0 has at least one parameter of level 0 (which need not, of course, occur at the right–hand side). The *level* of a grammar is the highest level of its nonterminals.

In this paper, we are interested in grammars as generators of Σ–trees. First, for any applicative term t over $\Sigma \cup V$, let $t^\perp$ be the result of replacing in t each nonterminal, together with its arguments, by $\perp$. (Formally, $t^\perp$ is defined by induction: $f^\perp = f$, for $f \in \Sigma$, $X^\perp = \perp$, for $X \in V$, and $(sr)^\perp = (s^\perp r^\perp)$ if $s^\perp \neq \perp$, otherwise $(sr)^\perp = \perp$.) It is easy to see that if t is an applicative term (over $\Sigma \cup V$) of type $\mathbf{0}$ then $t^\perp$ is an applicative term over $\Sigma^\perp$ of type $\mathbf{0}$. Recall that applicative terms over $\Sigma^\perp$ of type $\mathbf{0}$ can be identified with finite trees.

We will now define the single–step rewriting relation $\to_{\mathcal{G}}$ among the terms over $\Sigma \cup V$. Informally speaking, $t \to_{\mathcal{G}} t'$ whenever t' is obtained from t by replacing some occurrence of a nonterminal F by the right–hand side of the appropriate equation in which all parameters are in turn replaced by the actual arguments of F. Such a replacement is allowed only if F occurs as a head of a subterm of type $\mathbf{0}$. More precisely, the relation $\to_{\mathcal{G}} \subseteq T(\Sigma \cup V) \times T(\Sigma \cup V)$ is defined inductively by the following clauses.

- $Ft_1 \dots t_k \to_{\mathcal{G}} t[z_1{:=}t_1, \dots, z_k{:=}t_k]$ if there is an equation $Fz_1 \dots z_k = t$ (with $z_i : \rho_i$, $i = 1, \dots, k$), and $t_i \in T(\Sigma \cup V)_{\rho_i}$, for $i = 1, \dots, k$.
- If $t \to_{\mathcal{G}} t'$ then $(st) \to_{\mathcal{G}} (st')$ and $(tq) \to_{\mathcal{G}} (t'q)$, whenever the expressions in question are applicative terms.

A *reduction* is a finite or infinite sequence $t_0 \to_{\mathcal{G}} t_1 \to_{\mathcal{G}} \dots$ of terms in $T(\Sigma \cup V)$. We define the relation $t \twoheadrightarrow^\infty_{\mathcal{G}} t'$, where t is an applicative term in $T(\Sigma \cup V)$ and t' is a tree in $T^\infty(\Sigma^\perp)$, by

- t' is a finite tree, and there is a finite reduction sequence $t = t_0 \to_{\mathcal{G}} \dots \to_{\mathcal{G}} t_n = t'$, or
- t' is infinite, and there is an infinite reduction sequence $t = t_0 \to_{\mathcal{G}} t_1 \to_{\mathcal{G}} \dots$ such that $t' = \lim t_n^\perp$.

To define a unique tree produced by the grammar, we recall a standard *approximation ordering* on $T^\infty(\Sigma^\perp)$: $t' \sqsubseteq t$ if $\mathcal{D}\mathrm{om}\ t' \subseteq \mathcal{D}\mathrm{om}\ t$ and, for each $w \in \mathcal{D}\mathrm{om}\ t'$, $t'(w) = t(w)$ or $t'(w) = \perp$. (In other words, t' is obtained from t by replacing some of its subtrees by $\perp$.) Then we let $[\![\mathcal{G}]\!] = \sup\{t \in T^\infty(\Sigma^\perp) : S \twoheadrightarrow^\infty_{\mathcal{G}} t\}$. It is easy to see that, by the Church–Rosser property of our grammar, the above set is directed, and hence $[\![\mathcal{G}]\!]$ is well defined since $T^\infty(\Sigma^\perp)$ with the approximation ordering is a cpo. Furthermore, it is routine to show that if an infinite reduction $S = t_0 \to_{\mathcal{G}} t_1 \to_{\mathcal{G}} \dots$ is *fair*, i.e., any occurrence of a nonterminal symbol is eventually rewritten, then its result $t' = \lim t_n^\perp$ is $[\![\mathcal{G}]\!]$.

From grammar terms to λ–trees. Given a grammar $\mathcal{G}$, we define a map $\rrbracket_{\mathcal{G}}$ of $T(\Sigma \cup V \cup \mathrm{X})$ into the set of λ–trees (over Σ) such that

(1) $\mathfrak{I}_{\mathcal{G}}(t) = f$, if t is a function symbol $f \in \Sigma$,
(2) $\mathfrak{I}_{\mathcal{G}}(t) = x$, if t is a variable $x \in \mathrm{X}_{\mathbf{0}}$,
(3) $\mathfrak{I}_{\mathcal{G}}(t) = \lambda x'_1 \dots x'_n.\mathfrak{I}_{\mathcal{G}}(r[\phi_1{:=}t_1, \dots, \phi_m{:=}t_m, x_1{:=}x'_1, \dots, x_n{:=}x'_n])$, where the variables $x'_1, \dots, x'_n$ are chosen so that no x'_i occurs free in any of t_j, if $t = Ft_1 \dots t_m$, $F\phi_1 \dots \phi_m x_1 \dots x_n = r$ is an equation of $\mathcal{G}$ and $\mathrm{type}(\phi_i) = \mathrm{type}(t_i)$ for $i \in [m]$.
(4) $\mathfrak{I}_{\mathcal{G}}(t) = \mathfrak{I}_{\mathcal{G}}(t_1)\mathfrak{I}_{\mathcal{G}}(t_2)$, if $t = t_1 t_2$ where $t_1 : \mathbf{0} \to \tau$ and $t_2 : \mathbf{0}$.

It is a routine exercise to prove that $\mathfrak{I}_{\mathcal{G}}$ is well defined. To this end one may first define (co–inductively) an appropriate relation, establish its functionality and show that it corresponds to the above definition of $\mathfrak{I}_{\mathcal{G}}$.

We have the following characterization of $[\![\mathcal{G}]\!]$ in terms of operation $\mathfrak{I}_{\mathcal{G}}$ and derivation of trees from λ–graphs (see [10] for a proof).

Proposition 4. *Let $M = \mathfrak{I}_{\mathcal{G}}(S)$. Then $t_M = [\![\mathcal{G}]\!]$.*

5 Decidability Result

By Proposition 4 and Theorem 2, the decision problem of the MSO theory of a tree generated by a grammar reduces to that of the graph $G(\mathfrak{I}_{\mathcal{G}}(S))$. We are now interested in generating, in a sense, the last graph by a grammar of level 1. Note, however, that the underlying tree structure of $G(M)$ does not keep the complete information about the tree M. Indeed, while converting a λ–tree M into a graph $G(M)$ we have replaced (possibly) infinite number of labels λx_i and x_i, by only two labels, λ and v, at the expense of introducing "back edges". One might expect that these back edges are MSO definable in the underlying tree structure of $G(M)$, but it is not always the case. A good situation occurs if in part (3) of the definition of $\mathfrak{I}_{\mathcal{G}}$ we need not to rename the bound variables (i.e., we can take $x'_i := x_i$, for $i = 1, \dots, m$).

Definition 3. Let $\mathcal{G}$ be a grammar of level 2. We call the grammar *unsafe* if there are two equations (not necessarily distinct) $F\phi_1 \dots \phi_m x_1 \dots x_n = r$ and $F'\phi'_1 \dots \phi'_{m'} x'_1 \dots x'_{n'} = r'$ (where the ϕ's are of level 1 and the x's of level 0) such that r has a subterm $F't_1, \dots t_{m'}$, such that some variable x_i occurs free in some term t_j. Otherwise the grammar is *safe*. (Note that in the above, x_i may occur in arguments of F' of type $\mathbf{0}$, but not in those of level 1.)

It is easy to see that if a grammar is safe then in the definition on $\mathfrak{I}_{\mathcal{G}}(S)$ we are not obliged to introduce any new variables.

Let $\mathcal{G} = (\Sigma, V, S, E)$ be a safe grammar of level 2. We may assume that the parameters of type $\mathbf{0}$ occurring in distinct equations are different. Let $\mathrm{X}^{\mathcal{G}_{\mathbf{0}}} = \{x_1, \dots, x_L\}$ be the set of all parameters of type $\mathbf{0}$ occurring in grammar $\mathcal{G}$. We define an *algebraic* (i.e., level 1) grammar $\mathcal{G}^\alpha = (\Sigma^\alpha, V^\alpha, S^\alpha, E^\alpha)$ as follows.

First we define a translation α of (homogeneous) types of level 2 to types of level 1 that maps $(\mathbf{0}^k \to \mathbf{0})$ to $\mathbf{0}$, and $(\mathbf{0}^{k_1} \to \mathbf{0}) \to \dots \to (\mathbf{0}^{k_m} \to \mathbf{0}) \to \mathbf{0}^\ell \to \mathbf{0}$ to $\mathbf{0}^m \to \mathbf{0}$. We will denote $\alpha(\tau)$ by τ^α. Let $\Sigma^\alpha = \Sigma \cup \{@, c, \lambda x_1, \dots, \lambda x_L, x_1, \dots, x_L\}$, where all symbols from Σ as well as (former)

parameters x_i are considered constant, the symbol @ is binary, and the symbol c as well as all symbols λx_i are unary. Now, for a typed term $r : \tau$ over signature Σ, we define a term $r^\alpha : \tau^\alpha$ over Σ^α, as follows:

- F^α, for a variable $F : \tau$, is a fresh variable of type τ^α,
- $s^\alpha = s$ for each parameter of G (thus parameters of level 0 become constants, and parameters of level 1 change their types to $\mathbf{0}$),
- if $r = Ft_1 \dots t_m$ then $r^\alpha = F^\alpha t_1^\alpha \dots t_m^\alpha$, whenever F is a nonterminal of type $(\mathbf{0}^{k_1} \to \mathbf{0}) \to \dots \to (\mathbf{0}^{k_m} \to \mathbf{0}) \to \mathbf{0}^\ell \to \mathbf{0}$,
- if $r = (ts)$ with $s : \mathbf{0}$ then $r^\alpha = c(@t^\alpha s^\alpha)$.

Now $E^\alpha := \{F^\alpha \phi_1 \dots \phi_m = \lambda x_1 \dots \lambda x_n . r^\alpha \mid F\phi_1 \dots \phi_m x_1 \dots x_n = r \in E\}$ (where the ϕ's are of level 1 and the x's of level 0) and $V^\alpha = \{F^\alpha : F \in V\}$ which completes the definition of $\mathcal{G}^\alpha$.

Now let $t^\alpha = [\![\mathcal{G}^\alpha]\!]$ be the tree over $\Sigma^{\alpha\,\perp}$ generated by $\mathcal{G}^\alpha$, and let $\mathbf{t}^\alpha$ be the logical structure associated with it. We transform $\mathbf{t}^\alpha$ into a structure $\mathbf{t}_0^\alpha$ over the vocabulary $\{p_f : f \in \Sigma^\bullet\} \cup \{d_i \mid 1 \le i \le m_{\Sigma^\alpha}\} \cup \{E\}$ as follows. The universe remains the same as in $\mathbf{t}^\alpha$ as well as the interpretation of symbols d_i and p_f, for f different from λx_i and x_i. Furthermore, $w \in p_\lambda^{\mathbf{t}_0^\alpha}$ whenever $w \in p_{\lambda x_i}^{\mathbf{t}^\alpha}$, and $w \in p_v^{\mathbf{t}_0^\alpha}$ whenever $w \in p_{x_i}^{\mathbf{t}^\alpha}$, for some $x_i \in \mathrm{X}^{\mathcal{G}_0}$. Finally, we let $(u, w) \in E^{\mathbf{t}_0^\alpha}$ whenever w is binder of u, i.e., $t^\alpha(u) = x_i$, $t^\alpha(w) = \lambda x_i$, and w is the closest ancestor of u labeled by λx_i.

Lemma 4. *The structure $\mathbf{t}_0^\alpha$ is MSO definable in the structure $\mathbf{t}^\alpha$.*

Furthermore we claim the following.

Lemma 5. *Let grammars $\mathcal{G}$ and $\mathcal{G}^\alpha$ be as above, and let $M = \mathfrak{I}_{\mathcal{G}}(S)$. Then the structure $\mathbf{t}_0^\alpha$ coincides with the structure $\mathbf{G}_M$ defined for M as in Section 3.*

We conclude by the main result of the paper.

Theorem 4. *Let $\mathcal{G}$ be a safe grammar of level 2. Then the monadic theory of $[\![\mathcal{G}]\!]$ is decidable.*

Proof. Since the tree $\mathbf{t}^\alpha$ is algebraic, its MSO theory is decidable, by the result of Courcelle [6]. Let $M = \mathfrak{I}_{\mathcal{G}}(S)$. By Lemmas 5 and 4, the MSO theory of $\mathbf{G}_M$ is decidable. By Proposition 4, $[\![\mathcal{G}]\!] = t_M$. It is easy to see that, by construction, M is boundedly typed. Hence the result follows from Theorem 2. □

Example 1. Let f, g, c be signature symbols of arity 2,1,0, respectively. Consider a grammar of level 2 with nonterminals $S : \mathbf{0}$, and $F, G : (\mathbf{0} \to \mathbf{0}) \to \mathbf{0} \to \mathbf{0}$, and equations

$$S = Fgc \qquad F\varphi x = f\left(F(G\varphi)\,(\varphi x)\right) x \qquad G\psi y = \psi\,(\psi y)$$

It is easy to see that this grammar generates a tree t with $\mathrm{Dom}\, t = \{1^n 2^m : m \le 2^n\}$, such that $t(1^n) = f$, $t(1^n 2^{2^n}) = c$, and $t(w) = g$ otherwise. Since $\mathrm{Dom}\, t$ considered as a language is not context-free, the tree t is not algebraic (see [6]). Since the grammar is safe, the decidability of the MSO of $\mathbf{t}$ follows from Theorem 4.

References

1. A. Arnold and D. Niwiński. *Rudiments of μ–calculus.* Elsevier, 2001.
2. A. Asperti, V. Danos, C. Laneve, and L. Regnier. Paths in the lambda-calculus. In *Proc. 9th IEEE Symp. on Logic in Comput. Sci.*, pages 426–436, 1994.
3. A. Asperti and S. Guerrini. The optimal implementation of functional programming languages. In *Cambridge Tracts in Theoretical Computer Science*, volume 45. Cambridge University Press, 1998.
4. A. Berarducci and M. Dezani-Ciancaglini. Infinite lambda-calculus and types. *Theoret. Comput. Sci.*, 212:29–75, 1999.
5. D. Caucal. On infinite transition graphs having a decidable monadic second–order theory. In F. M. auf der Heide and B. Monien, editors, *23th International Colloquium on Automata Languages and Programming*, LNCS 1099, pages 194–205, 1996. A long version will appear in TCS.
6. B. Courcelle. The monadic second–order theory of graphs IX: Machines and their behaviours. *Theoretical Comput. Sci.*, 151:125–162, 1995.
7. W. Damm. The IO– and OI–hierarchies. *Theoretical Comput. Sci.*, 20(2):95–208, 1982.
8. H. Hungar. Model checking and higher-order recursion. In L. P. M. Kutyłowski and T. Wierzbicki, editors, *Mathematical Foundations of Computer Science 1999*, LNCS 1672, pages 149–159, 1999.
9. J. R. Kennaway, J. W. Klop, M. R. Sleep, and F. J. de Vries. Infinitary lambda calculus. *Theoret. Comput. Sci.*, 175:93–125, 1997.
10. T. Knapik, D. Niwiński, and P. Urzyczyn. Deciding monadic theories of hyperalgebraic trees, 2001. `http://www.univ-reunion.fr/~knapik/publications/`
11. O. Kupferman and M. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In *Computer Aided Verification, Proc. 12th Int. Conference*, Lecture Notes in Computer Science. Springer-Verlag, 2000.
12. D. Niwiński. Fixed points characterization of infinite behaviour of finite state systems. *Theoretical Comput. Sci.*, 189:1–69, 1997.
13. M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Soc*, 141:1–35, 1969.
14. Z. Spławski and P. Urzyczyn. Type fixpoints: iteration vs. recursion. In *Proc. 4th ICPF*, pages 102–113. ACM, 1999.
15. W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 389–455. Springer-Verlag, 1997.
16. I. Walukiewicz. Monadic second-order logic on tree-like structures. In C. Puech and R. Reischuk, editors, *Proc. STACS '96*, pages 401–414. Lect. Notes Comput. Sci. 1046, 1996.