Deciding monadic theories of hyperalgebraic trees

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

[1] **IREMIA**, 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@mimuw.edu.pl
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 [7] 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 lambda terms, in such a way that the MSO theory of a tree can be interpreted in the MSO theory of a lambda term.

Introduction

In 1969, Rabin [16] 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 unwinded structure then relies on the decidability of the MSO theory of the initial structure, and the "regularity" of the unwinding process (see [19]). 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 structures with additional monadic predicates corresponding to the labels, and a tree is *regular* if it has only a finite number of nonisomorphic 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 [7] 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 by higher order grammars, although the expressive power of such grammars was extensively studied by Damm [8] 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 [7].

Our method makes use of the idea of the *infinitary lambda calculus*, already considered by several authors [12, 5]. Here we view infinite lambda 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

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

of the original lambda term, viewed as an appropriate logical structure (Theorem 3.4 below). Let us stress that the reduction is not a mere interpretation (which seems hardly to be 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 lambda term, we use techniques similar to Caucal [6], combined with an idea originated from Geometry of Interaction and the theory of optimal reductions. Namely, we consider deformations of regular paths in lambda terms and push-down store computations along these paths.

The motivation behind the aforementioned reduction of MSO theories is that the MSO theory of a lambda 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 lambda 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 5.4 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., [14] 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 [10] decidability of the monadic second-order theory (S1S) of *paths* of such graphs (not the full MSO theory of graphs).

1 Preliminaries

Throughout the paper, the set of natural numbers is denoted ω and, for $n \in \omega$, the symbol $[n]$ abbreviates $\{1, \ldots, n\}$.

Types. We consider a set of types $\mathcal{T}$ constructed from a unique *basic* type $\mathbf{0}$: $\mathbf{0}$ is a type and, if τ_1, τ_2 are types, so is $(\tau_1 \to \tau_2) \in \mathcal{T}$. 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 $\ell(\mathbf{0}) = 0$, and $\ell(\tau_1 \to \tau_2) = \max(1 + \ell(\tau_1), \ell(\tau_2))$. 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 a type $((\mathbf{0} \to \mathbf{0}) \to \mathbf{0}) \to (\mathbf{0} \to \mathbf{0}) \to (\mathbf{0} \to \mathbf{0} \to \mathbf{0}) \to \mathbf{0} \to \mathbf{0}$ is homogeneous, but a type $\mathbf{0} \to (\mathbf{0} \to \mathbf{0}) \to \mathbf{0}$ is not.

Higher–order terms. A *typed alphabet* is a set Γ of symbols with types in $\mathcal{T}$. Thus Γ can be also presented as a $\mathcal{T}$-indexed family $\{\Gamma_\tau\}_{\tau \in \mathcal{T}}$, 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.

Given a typed alphabet Γ, the set $T(\Gamma) = \{T(\Gamma)_\tau\}_{\tau \in \mathcal{T}}$ 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 $t_0 t_1 \ldots t_n$ instead of $(\cdots((t_0 t_1) t_2) \cdots) t_n$. For applicative terms $t, t_1, \ldots, t_m$, and symbols $z_1, \ldots, z_m$, of appropriate types, the term $t[t_1/z_1, \ldots, t_k/z_m]$ is defined as the result of simultaneous replacement in t of z_i by t_i, for $i = 1, \ldots, m$.

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 typed alphabet of level 1. A symbol f in Σ is of type $\mathbf{0}^n \to \mathbf{0}$, for $n \geq 0$, and can be viewed as a (first–order) function symbol of *arity* n. Let $T \subseteq \omega^*$ 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 $i \in [n]$. 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 also 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 $\{w \in \mathcal{D}\mathrm{om} \mid |w| \leq n\}$. 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 $\mathcal{T}$ can be identified with (nonlabeled) finite binary trees. More specifically, we use the set of directions $\{p, q\}$, and let

- $\mathit{tree}(\mathbf{0}) = \{\varepsilon\}$,
- $\mathit{tree}(\tau_1 \to \tau_2)$ is the unique tree such that $\mathit{tree}(\tau_1).p = \mathit{tree}(\tau_1)$ and $\mathit{tree}(\tau_1).q = \mathit{tree}(\tau_2)$.

In the sequel we will not make notational distinction between τ and $\mathit{tree}(\tau)$.

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.

Clearly, the satisfaction of a formula depends only on the valuation of its free variables. The *monadic second–order theory* of $\mathbf{A}$ is the set of all MSO sentences satisfied in $\mathbf{A}$, in symbols $\mathit{MSO}(\mathbf{A}) = \{\varphi : \mathbf{A} \models \varphi\}$.

Let Σ be a typed alphabet of level 1, 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 [18] for a survey of the results on monadic second-order theory of trees.

2 Infinitary lambda calculus

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

Definition 2.1. 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 *lambda 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 lambda tree of type $\mathbf{0}$.
- Each function symbol $f \in \Sigma^{\perp}$ of type τ is a lambda tree of type τ.
- Otherwise each lambda tree is of type of level at most 1 and is either an *application* (MN) or an *abstraction* $(\lambda x.M)$ (see Figure 1).
- If a lambda tree P of type τ is an application (MN) then M is a lambda tree of type $\mathbf{0} \to \tau$, and N is a lambda tree of type $\mathbf{0}$.
- If a lambda tree P of type τ has the form $(\lambda x.M)$, then $\tau = \mathbf{0} \to \sigma$, and M is a lambda tree of type σ.

Strictly speaking, the above is a co-inductive definition of the two-argument relation "M is a lambda tree of type τ". Formally, a lambda 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 "lambda tree" we actually mean a lambda tree together with its type. We warn the reader about a possible confusion: The types associated with lambda trees by the above definition are *not* the types of these trees viewed as terms over Σ° (but rather as terms over Σ).

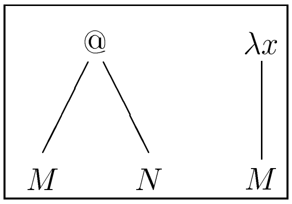


Fig. 1. Application and abstraction

Let M be a lambda 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*, iff it has an ancestor labeled λx. The *binder* of this occurrence of x is the closest of all such ancestors λx (i.e., one occurring at the largest depth). An occurrence of x which is not bound is called *free*. We say that a variable x is *free* in a lambda 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 lambda tree M with $FV(M) = \varnothing$ is called *closed*.

Definition 2.2. We call a lambda tree M *boundedly typed* if the set of types of all subterms of M is finite.

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

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

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

Let $P(z/x)$ denote the result of replacing in P all free occurrences of a variable x by another variable z.

Definition 2.3. A *bisimulation* on lambda trees is a binary relation $\sim$ satisfying the following conditions:

(1) If $M \sim N$ then M and N are of the same type.
(2) If $M \sim N$ then the root labels of M and N are either the same or both M and N are abstractions.
(3) If $\lambda x.M \sim \lambda y.N$, then $M(z/x) \sim N(z/y)$, whenever z is a fresh variable (neither free nor bound in $\lambda x.M$ and $\lambda y.N$).
(4) If $(MN) \sim (M'N')$ then $M \sim M'$ and $N \sim N'$.

Lemma 2.4.

(1) *The union $\approx$ of all bisimulations is a bisimulation itself.*
(2) *On ordinary lambda-terms, the relation $\approx$ coincides with the α-equivalence $=_\alpha$.*
(3) *$M \approx N$ holds if and only if $M{\restriction}n =_\alpha N{\restriction}n$ holds for all n.*
(4) *If $M \approx N$ then $FV(M) = FV(N)$.*

The above lemma provides evidence that the greatest bisimulation is a correct generalization of the notion of α-equivalence. That is, $M \approx N$ holds iff M and N are identical up to names of bound variables. Following a common practice in lambda calculus, alpha-equivalent lambda trees will be identified. In consequence, all three statements $M \approx N$, $M =_\alpha N$ and $M = N$ will be understood as expressing the same property.

Lemma 2.5 (Principle of co-induction [11]). *In order to prove that $M =_\alpha N$, it is enough to find a bisimulation $\sim$ such that $M \sim N$.*

In order to define beta reduction on lambda graphs, we must define the operation of substitution, denoted by $M[x := N]$. Our definition of substitution is similar to that for ordinary lambda terms.

- $x[x := N] = N$;
- $y[x := N] = y$;
- $f[x := N] = f$;
- $(PQ)[x := N] = (P[x := N]Q[x := N])$;
- $(\lambda x.P)[x := N] = \lambda x.P$;
- $(\lambda y.P)[x := N] = \lambda y.P[x := N]$, provided $y \notin FV(N)$;
- $(\lambda y.P)[x := N] = \lambda z.P(z/y)[x := N]$, with fresh z, otherwise.

Note however, that the above definition is not inductive, but rather co-inductive. That is, the operation of substitution is the greatest (in fact, unique) fixpoint of a monotone operator $\Phi_{x,N}$ on relations. Given a relation r on lambda trees, the relation $\Phi_{x,N}(r)$ associates, for instance, the term PQ with all terms $P'Q'$ such that $P\ r\ P'$ and $Q\ r\ Q'$ hold. An iterated application of $\Phi_{x,N}$ to the relation $\{(P,Q) \mid P \text{ and } Q \text{ are of the same type}\}$ yields an infinite decreasing sequence, the meet of which is a function.

A *redex* in a lambda 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. The symbol $\twoheadrightarrow_\beta$ stands for the reflexive and transitive closure of $\to_\beta$. Note that infinite lambda trees, even simply typed, may have infinite reduction sequences, due to infinitely many redexes.

2.1 Paths in lambda graphs

Once we have agreed to identify α-equivalent lambda trees, a convenient representation of these is to ignore names of bound variables. In fact, we can simply erase these names. Instead, we draw additional arrows going upward from all bound occurrences of variables to their binders. After that, without loss of information, we can replace all labels λx just by "λ" and all bound occurrences of variables by a uniform label "v".

For technical reasons, we also 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.)

In this way, every closed lambda tree M is represented as an (undirected) graph $G(M)$, i.e., a tree with some extra backward edges. 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 and the entry node of an application is the additional c-node (not the @-node). If confusion will not arise, we 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.

Some edges in a graph $G(M)$ will now be assigned labels and orientation. The labels are either p or q, and the orientation is either downward (from an ancestor to a descendant) or upward.

- An edge from the entry node of an application to an @-node is oriented downward and labeled q.
- An edge connecting the argument of an application with the corresponding @-node is labeled p and oriented upward.
- An edge connecting the body of an abstraction with the corresponding λ-node is labeled q and oriented upward.
- An (additional) edge from a variable node to the appropriate λ-node is labeled p and oriented upward.

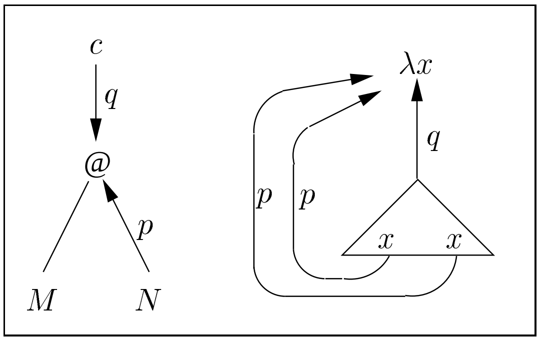


Fig. 2. Labeled application and abstraction

Note that the labeling and orientation are fully determined by the lambda tree M, and are not formally part of $G(M)$. Figure 2 explains the labeling.

Following the ideas of the Geometry of Interaction, see [3, 4], we will now consider paths in lambda graphs.

A sequence of adjacent edges in a lambda 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;
- Two edges connecting two different variable nodes with a lambda 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 [17], we will now consider a certain stateless push down automaton walking on paths in a lambda graph. Informally, the automaton moves along edges of a path Π, and each time it traverses a labeled edge forward or backward, a push-down 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 our pushdown automaton works with the input alphabet $\{p\uparrow, p\downarrow, q\uparrow, q\downarrow, |\}$ ($|$ for a nonlabeled edge). A *configuration* of the automaton 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 the automaton is fully determined by the initial contents of the pds.

An important property of our automaton is that the contents of the push down store 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 2.6. *Assume that the automaton 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.*

Proof. For a single step of the automaton the hypothesis follows by inspection of the possible cases. For many steps proceed by induction. The latter claim follows from the former because a reverse of a computation is a computation on the reversed path. □

In other words, the push down store of our automaton 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 lambda tree is boundedly typed (see Definition 2.2) then the pushdown automaton can essentially be replaced by a finite automaton. Indeed, suppose that our automaton can traverse a path from a node α_M to a node α_N. Then it can also perform a computation along this path in which the pds becomes empty at least once. (If it is not the case originally, we can remove the suffix of the pds which remains unchanged.) Without loss of generality (because passing trough an @-node does not modify the pds), we may assume that it happens in some entry node, i.e., the automaton assumes a configuration (α_P, ε), for some $P : \tau$. Clearly, $\tau.\varepsilon = \tau$ is defined, and hence, by Lemma 2.6, the type pointed to by the pds during the whole computation is τ. 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 lambda term). Then we can convert the pds contents of our pushdown automaton 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 2.7. *Suppose a lambda tree M is boundedly typed. There is a deterministic finite automaton whose set of states is a subset of $\{p, q\}^*$, and whose computations along paths in $G(M)$ coincide with the computations of our pushdown automaton. Whenever the pushdown automaton can traverse a path in $G(M)$, the finite automaton can do it as well, and if the original computation empties pds at least once, the finite automaton can follow the same computation.*

It will now be convenient to define a computation path (of our pushdown automaton) more formally. A *computation path Π* in a lambda tree 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 the automaton'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.

Lemma 2.8. *Let M be a closed lambda 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.*

Proof. Note that the trivial paths consisting of (α_M, ϵ) and of $(\alpha_f, q^{i-1}p)$, are properly ending. We claim that whenever a computation path Π starting from a configuration (α_M, ϵ) or $(\alpha_f, q^{i-1}p)$, and ending in a configuration (α_N, w) is properly ending then either N is a signature symbol, or there is a unique prolongation of Π, by one or two steps, to a computation path Π' of the form $(\Pi, (\alpha_Q, v))$ or $(\Pi, (\alpha', w'), (\alpha_Q, v))$, where α' is an @-node. Moreover, Π' is again properly ending.

The claim follows easily by case analysis. Now the claim of the lemma follows easily by induction argument. □

2.2 Derived trees

An infinite tree in $T^{\infty}(\Sigma^{\perp})$ can be viewed as an infinite lambda 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 lambda 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.

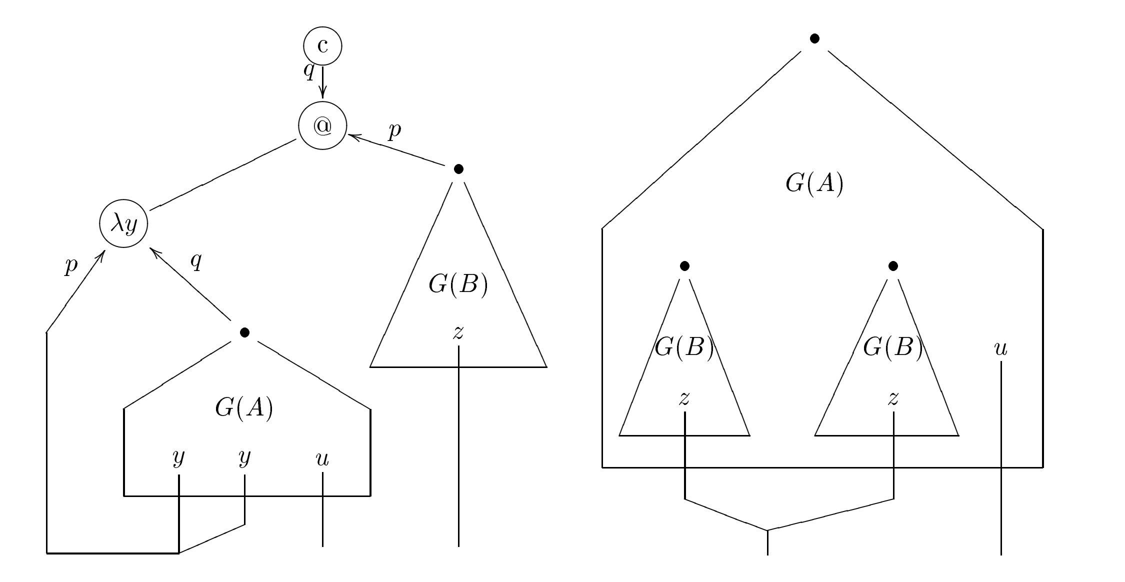
A tree $t_M : \mathcal{D}\mathrm{om}\, t_M \to \Sigma^{\perp}$ will be defined along with a partial mapping I_M from $\mathcal{D}\mathrm{om}\, t_M$ to $G(M)$ in such a way that the label of $I_M(w)$ in $G(M)$ coincides with $t_M(w)$. More specifically, the domain of I_M will be $\mathcal{D}\mathrm{om}_{+} t_M = \{w \in \mathcal{D}\mathrm{om}\, t_M : t_M(w) \neq \perp\}$. At first, consider the maximal computation path Π in $G(M)$ starting from (α_M, ε) (cf. Lemma 2.8). If it is infinite, we let $t_M = \perp$ and $I_M = \varnothing$. Otherwise, again by Lemma 2.8, 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 $I_M : \varepsilon \mapsto s_M$, 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 2.8, 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 $I_M(wi) = \alpha_i$, whenever Π_i is finite, and $t_M(wi) = \perp$, otherwise.

2.3 Beta reduction

We will now examine how a beta reduction step applied to a lambda tree M may affect a path Π in $G(M)$. Suppose M_1 is obtained from M by a beta-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 is free in A, the other case is easier and left to the reader. Figure 2.3 presents a redex, and Figure 2.3 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.



(a) Before reduction

(b) After reduction

Fig. 3.

- 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 Figure 2.3) should be considered an offset of the entry node of the body of the abstraction (the entry to $G(A)$ at Figure 2.3) 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.

Lemma 2.9. *Let $M \to_\beta N$ and let Π_1 be a computation path in $G(N)$ starting in (β_1, w) and properly ending in (β_2, v). Let β_1 and β_2 be respectively offsets of α_1 and α_2. Then there is a computation path Π in $G(M)$ starting from (α_1, w), and properly ending in (α_2, v), which is not shorter than Π_1.*

Proof. Of course, Π is such that Π_1 is a deformation of Π. $\square$

Proposition 2.10. *Let M and N be closed lambda 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. By Lemma 2.9, each computation path in $G(N)$ used in this construction is a deformation of a computation path $G(M)$. 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. $\square$

3 Moving between MSO theories

Let M be a closed boundedly typed lambda 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. We consider a signature $\Sigma^\bullet = \Sigma \cup \{@, \lambda, c, v\}$, 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.

Now let t_M and $I_M : \mathcal{D}\mathrm{om}_+ t \to W$ be defined as in Section 2.2. Note that the mapping I_M cannot be used directly for interpretation of t_M in $G(M)$ since it is not, in general, injective. Instead, we will use the fact that MSO over trees on finite signature is equivalent to a weaker logic, which does not distinguish between t_M and its image under I. It will be the propositional modal μ-calculus, which we mention here only very briefly, referring the reader to [13] or [9] for detailed presentations.

We consider the vocabulary $\{p_f : f \in \Sigma^\perp\} \cup \{d_i : 1 \le i \le m_\Sigma\}$, derived from the signature $\Sigma^\perp$ as in section 1 (page 3). Fix an infinite list of variables $X_0, X_1, \ldots$. The formulas of the μ-calculus are defined by the following grammar.

$$p ::= X_i \mid p_f \mid p \vee p \mid p \wedge p \mid \langle d_i \rangle p \mid [d_i]p \mid \mu X_i.p \mid \nu X_i.p$$

The formulas of the μ calculus (of our vocabulary) are interpreted in structures (labeled transitions systems) of the form $\mathcal{S} = \langle S, \{p_f^{\mathcal{S}} : f \in \Sigma\} \cup \{d_i^{\mathcal{S}} : 1 \le i \le m_\Sigma\}\rangle$, where $p_f^{\mathcal{S}} \subseteq S$, and $d_i^{\mathcal{S}} \subseteq S \times S$. Note that, in particular, $\mathbf{t}$ is such a structure, for any tree $t \in T^\infty(\Sigma^\perp)$. The satisfaction of a μ-calculus sentence ψ in a point s of a model $\mathcal{S}$ is denoted $\mathcal{S}, s \models \psi$.

The following fact is known (see [15], section 6).

Proposition 3.1. *For every sentence φ of monadic second order logic (over the vocabulary R_Σ), one can construct a sentence p_φ of the μ-calculus, such that, for any tree $t \in T^\infty(\Sigma)$,*

$$\mathbf{t} \models \varphi \quad \textit{iff} \quad \mathbf{t}, \varepsilon \models p_\varphi.$$

Applying the above proposition to $\Sigma := \Sigma^\perp$ and $t := t_M$, we see that the MSO theory of t_M can be translated to the μ-calculus. Now we will define a structure induced by the mapping I_M which will have the same μ-calculus theory as $\mathbf{t_M}$, and will be interpretable in the MSO theory of $\mathbf{G}_M$. To this end, we first need to extend I_M to a mapping I_M^* defined on the whole $\mathcal{D}\mathrm{om}\ t_M$. Without loss of generality, we can assume that the root of $G(M)$ (ε) is not labeled by a signature symbol[1]. We let

$$\begin{array}{ll} I_M^*(w) = I(w) & \text{if } I(w) \neq \perp \\ I_M^*(w) = \varepsilon & \text{otherwise} \end{array}$$

[1] Otherwise $G(M)$ consists of a single node labeled by a constant, and all the results of this section 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.

We now define the structure $\mathbf{I}_M$ over the same vocabulary as $\mathbf{t}_M$, of universe $I_M^*(\mathcal{D}\text{om } t)$ by letting, for $f \in \Sigma^\perp$, we let $p_f^{\mathbf{I}_M} = \{I_M^*(w) : w \in \mathcal{D}\text{om } t_M\}$, and, for $1 \le i \le m_\Sigma$, $d_i^{\mathbf{I}_M} = \{(I_M^*(w), I_M^*(wi)) : wi \in \mathcal{D}\text{om } t_M\}$.

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 and allows to conclude the following (see, e.g., [1]).

Lemma 3.2. *For any μ-calculus sentence p, $\mathbf{t}_M, \varepsilon \models p$ if and only if $\mathbf{I}_M, s_M \models p$.*

The next lemma allows to accomplish the interpretation of the MSO theory of t_M in the MSO of $G(M)$.

Lemma 3.3. *For any μ-calculus sentence p, one can construct an MSO sentence φ_p such that $\mathbf{I}_M, s_M \models p$ if and only if $\mathbf{G}_M \models \varphi_p$.*

Proof. It is easy to translate directly the formulas of the μ-calculus into the MSO of a transition system (see, e.g., [2]). So, 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 $1 \le i \le m_\Sigma$, defining the relations $I_M^*(\mathcal{D}\text{om } t_M)$, $p_f^{\mathbf{I}_M}$, 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 2.8, there is at most one such path. Moreover, by Proposition 2.7, this computation can be carried by a finite automaton. (It follows easily from Lemma 2.8 that the computation empties pds at least once.) Therefore, it is routine to express the desired property by the known techniques, see Caucal [6]. 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 $Uni(x)$. □

We are ready to conclude this section.

Theorem 3.4. *Let M be a closed boundedly typed lambda 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 $\mathbf{G}_M \models \varphi'$.*

Proof. It is the composition of mappings of Proposition 3.1 and Lemma 3.3. □

4 Grammars

We now fix two disjoint typed alphabets, $N = \{N_\tau\}_{\tau \in \mathcal{T}}$ and $\mathcal{X} = \{\mathcal{X}_\tau\}_{\tau \in \mathcal{T}}$ of *nonterminals* and *variables* (or *parameters*), respectively. A *grammar* is a tuple $\mathcal{G} = (\Sigma, V, S, E)$, where Σ is a *signature* (i.e., a finite alphabet of level 1), $V \subseteq N$ 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

$$F z_1 \ldots z_m = w$$

where $F : \tau_1 \to \tau_2 \cdots \to \tau_m \to \mathbf{0}$ is a nonterminal in V, z_i is a variable of type τ_i, and w is an applicative term in $T(\Sigma \cup V \cup \{z_1 \ldots z_m\})$.

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 $m \ge 1$ 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. Let, as before $\Sigma^\perp = \Sigma \cup \{\perp\}$, with $\perp : \mathbf{0}$. First, with any applicative term t over $\Sigma \cup V$, we associate an expression $t^\perp$ over signature $\Sigma^\perp$ inductively as follows.

- If $t = f$, $f \in \Sigma$, then $t^\perp = f$.
- If $t = X$, $X \in V$, then $t^\perp = \perp$.
- If $t = (sr)$ then if $s^\perp \neq \perp$ then $t^\perp = (s^\perp r^\perp)$, otherwise $t^\perp = \perp$.

Informally speaking, the operation $t \mapsto t^\perp$ replaces in t each nonterminal, together with its arguments, by $\perp$. 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 of terms in $T(\Sigma \cup V)$, $t_0 \to_{\mathcal{G}} t_1 \to_{\mathcal{G}} \dots$. 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 $[\![\mathcal{G}]\!]$.

4.1 From grammar terms to lambda trees

Given a grammar $\mathcal{G}$, we define a relation $\gimel_{\mathcal{G}}$ co-inductively as the greatest relation between terms in $T(\Sigma \cup V \cup \mathcal{X})$ and lambda trees (over Σ) such that, for all $(t, M) \in \gimel_{\mathcal{G}}$,

(1) if t is a function symbol f then $M = f$,
(2) if t is a variable $x \in \mathcal{X}_{\mathbf{0}}$ then $M = x$,
(3) if $t = Ft_1, \dots t_m$, where $t_1 \dots t_m$ are terms such that $\mathrm{type}(\phi_i) = \mathrm{type}(t_i)$, for $i \in [m]$, and $F\phi_1 \dots \phi_m x_1 \dots x_n = r$ is an equation of $\mathcal{G}$ with variables $\phi_1, \dots, \phi_m$ of level 1 and variables $x_1, \dots, x_n$ of level 0, then

$$M \in \{\lambda x'_1 \dots x'_n.N \mid (r[\phi_1{:=}t_1, \dots, \phi_m{:=}t_m, x_1{:=}x'_1, \dots, x_n{:=}x'_n], N) \in \gimel_{\mathcal{G}}\},$$

where the variables $x'_1, \dots, x'_n$ are chosen so that no x'_i occurs free in any of t_j,
(4) if $t = t_1 t_2$ where $t_1 : \mathbf{0} \to \tau$ and $t_2 : \mathbf{0}$ then $M \in \{M_1 M_2 \mid (t_1, M_1), (t_2, M_2) \in \gimel_{\mathcal{G}}\}$.

Lemma 4.1. *$\daleth_{\mathcal{G}}$ is a function, i.e., for any $t \in T(\Sigma \cup V \cup \mathcal{X})$, $n \geq 0$, there exists a unique M (up to α-equivalence) such that $(t, M) \in \daleth_{\mathcal{G}}$.*

Proof. See Appendix A.

From now on $\daleth_{\mathcal{G}}$ is considered as a function from $T(\Sigma \cup V \cup \mathcal{X})$ into the set of lambda trees. The following properties follow from definition.

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

Lemma 4.2. *If $t, t' \in T(\Sigma \cup V \cup \mathcal{X})$ and a variable x has the same type as t', namely $\mathbf{0}$, then*

$$\daleth_{\mathcal{G}}(t[x{:=}t']) =_{\alpha} \daleth_{\mathcal{G}}(t)[x{:=}\daleth_{\mathcal{G}}(t')] \ .$$

Proof. See Appendix B.

Lemma 4.3. *For all $t, t' \in T(\Sigma \cup V)$, if $t \to_{\mathcal{G}} t'$, then $\daleth_{\mathcal{G}}(t) \xrightarrow{*}_{\beta} \daleth_{\mathcal{G}}(t')$.*

Proof. We first consider the case when t is of the form $Ft_1 \ldots t_m t'_1 \ldots t'_n$, and there is an equation $F\phi_1 \ldots \phi_m x_1 \ldots x_n = r$ of $\mathcal{G}$. Thus $t' = r\Big[\phi_1{:=}t_1, \ldots, \phi_m{:=}t_m, x_1{:=}t'_1, \ldots x_n{:=}t'_n\Big]$. We have to show

$$\daleth_{\mathcal{G}}(Ft_1 \ldots t_m t'_1 \ldots t'_n) \xrightarrow{*}_{\beta} \daleth_{\mathcal{G}}(r[\phi_1{:=}t_1, \ldots, \phi_m{:=}t_m, x_1{:=}t'_1, \ldots, x_n{:=}t'_n]) \ .$$

We have $\daleth_{\mathcal{G}}(Ft_1 \ldots t_m t'_1 \ldots t'_n) =$ by definition of $\daleth_{\mathcal{G}}$

$$\begin{aligned}
&= \daleth_{\mathcal{G}}(Ft_1 \ldots t_m)\daleth_{\mathcal{G}}(t'_1) \ldots \daleth_{\mathcal{G}}(t'_n) \qquad \text{by definition of } \daleth_{\mathcal{G}} \\
&= \lambda x'_1 \ldots x'_n . \daleth_{\mathcal{G}}(r[\phi_1{:=}t_1, \ldots, \phi_m{:=}t_m, x_1{:=}x'_1, \ldots, x_n{:=}x'_n])\daleth_{\mathcal{G}}(t'_1) \ldots \daleth_{\mathcal{G}}(t'_n) \\
&\xrightarrow{*}_{\beta} \daleth_{\mathcal{G}}(r[\phi_1{:=}t_1, \ldots, \phi_m{:=}t_m, x_1{:=}x'_1, \ldots, x_n{:=}x'_n])[x'_1{:=}\daleth_{\mathcal{G}}(t'_1), \ldots, x'_n{:=}\daleth_{\mathcal{G}}(t'_n)] \\
&\qquad \text{by Lemma 4.2} \\
&= \daleth_{\mathcal{G}}(r[\phi_1{:=}t_1, \ldots, \phi_m{:=}t_m, x_1{:=}x'_1, \ldots, x_n{:=}x'_n][x'_1{:=}t'_1, \ldots, x'_n{:=}t'_n]) \\
&\qquad \text{because } x'_1, \ldots, x'_n \text{ do not occur free in } t_1, \ldots, t_m \\
&= \daleth_{\mathcal{G}}(r[\phi_1{:=}t_1, \ldots, \phi_m{:=}t_m, x_1{:=}t'_1, \ldots, x_n{:=}t'_n])
\end{aligned}$$

The case of $t = rs$ follows by induction. □

Given a grammar $\mathcal{G}$ with an axiom S, we call $\daleth_{\mathcal{G}}(S)$ the *lambda tree of $\mathcal{G}$*.

We are now going to establish a connection between the lambda tree of $\mathcal{G}$ and the tree $[\![\mathcal{G}]\!]$ generated by $\mathcal{G}$. (For definition of $t^{\perp}$, see page 12.)

Lemma 4.4. *Let $t \in T(\Sigma \cup V)$ be of type $\mathbf{0}$ and let $M = \daleth_{\mathcal{G}}(t)$. Then $t^{\perp} \sqsubseteq t_M$.*

Proof. See Appendix C.

Lemma 4.5. *Suppose $[\![\mathcal{G}]\!] = \perp$, and let $M = \daleth_{\mathcal{G}}(S)$. Then $t_M = \perp$, too.*

Proof. We show that the maximal computation path in $G(M)$ starting from (α_M, ε) induces a reduction in $\mathcal{G}$ which permanently rewrites the head symbol of a term. This follows that the path is infinite, hence the result. □

Thus we obtain a characterization of $[\![\mathcal{G}]\!]$ in terms of operation $\daleth_{\mathcal{G}}$ and derivation of trees from lambda graphs.

Proposition 4.6. *Let $M = \daleth_{\mathcal{G}}(S)$. Then $t_M = [\![\mathcal{G}]\!]$.*

Proof. Let $S = t_0 \to_{\mathcal{G}} t_1 \to_{\mathcal{G}} \ldots$ be a reduction in $\mathcal{G}$. By Lemma 4.3, we have $\daleth_{\mathcal{G}}(t_0) \xrightarrow{*}_{\beta} \daleth_{\mathcal{G}}(t_1) \xrightarrow{*}_{\beta} \ldots$, and, by Proposition 2.10, the trees derived from $\daleth_{\mathcal{G}}(t_0), \daleth_{\mathcal{G}}(t_1), \ldots$ are all equal to t_M. By Lemma 4.4, it follows that $t_n^{\perp} \sqsubseteq t_M$, and consequently $\lim t_n^{\perp} \sqsubseteq t_M$. This implies $[\![\mathcal{G}]\!] \sqsubseteq t_M$. If $\perp$ does not occur in $[\![\mathcal{G}]\!]$ then $[\![\mathcal{G}]\!] = t_M$ and we are done.

For the proof of inclusion $t_M \sqsubseteq [\![\mathcal{G}]\!]$ in the case when $\perp$ does occur in $[\![\mathcal{G}]\!]$, see Appendix C.

□

5 Decidability result

By Proposition 4.6 and Theorem 3.4, the decision problem of the MSO theory of a tree generated by a grammar reduces to that of the graph $G(\daleth_{\mathcal{G}}(S))$. 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 lambda 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 (see Section 5.1 below). A good situation occurs if in part (3) of the definition of $\daleth_{\mathcal{G}}$ we need not to rename the bound variables (i.e., we can take $x'_i := x_i$, for $i = 1, \ldots, m$).

Definition 5.1. Let $\mathcal{G}$ be a grammar of level 2. We call the grammar *unsafe* if there are two equations (not necessarily distinct) $F\phi_1 \ldots \phi_m x_1 \ldots x_n = r$ and $F'\phi'_1 \ldots \phi'_{m'} x'_1 \ldots x'_{n'} = r'$ (where the ϕ's are of level 1 and the x's of level 0) such that r has a subterm $F't_1, \ldots t_{m'}$, such that some variable x_i occurs free in some term t_j. Otherwise the grammar is *safe*. (Note that in 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 $\daleth_{\mathcal{G}}(S)$ 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 $\mathcal{X}^{\mathcal{G}_\mathbf{0}} = \{x_1, \ldots, x_L\}$ 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.

It is convenient first to define a transformation α of (homogeneous) types of level 2 to types of level 1, that maps $\mathbf{0}^\ell \to \mathbf{0}$ to $\mathbf{0}$, and $(\mathbf{0}^{k_1} \to \mathbf{0}) \to \ldots \to (\mathbf{0}^{k_m} \to \mathbf{0}) \to \mathbf{0}^\ell \to \mathbf{0}$ to $\mathbf{0}^m \to \mathbf{0}$. We will denote $\alpha(\tau)$ by τ^α. Let $\Sigma^\alpha = \Sigma \cup \{@, c, \lambda x_1, \ldots, \lambda x_L, x_1, \ldots, x_L\}$, 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. The set $V^\alpha = \{F^\alpha : F \in V\}$ is a copy of V, with $F^\alpha : \tau^\alpha$, whenever $F : \tau$. Finally, whenever there is an equation $F\phi_1 \ldots \phi_m x_1 \ldots x_n = r$ in E (where the ϕ's are of level 1 and the x's of level 0), there is an equation $F^\alpha \phi_1 \ldots \phi_m = \lambda x_1 \ldots \lambda x_n . r^\alpha$ in E^α, where the transformation of typed terms $r : \tau \mapsto r^\alpha : \tau^\alpha$ is defined inductively by

- $\alpha : F \mapsto F^\alpha$,
- $\alpha : x_i \mapsto x_i$, for $x_i \in \mathcal{X}^{\mathcal{G}_\mathbf{0}}$,
- $\alpha : y \mapsto y$, for parameters of level 1,
- if $r = (ts)$ with $s : \mathbf{0}$ then $r^\alpha = c(@t^\alpha s^\alpha)$,
- if $r = Ft_1 \ldots t_m$, where $F : (\mathbf{0}^{k_1} \to \mathbf{0}) \to \ldots \to (\mathbf{0}^{k_m} \to \mathbf{0}) \to \mathbf{0}^\ell \to \mathbf{0}$ is a nonterminal then $r^\alpha = F^\alpha t_1^\alpha \ldots t_m^\alpha$

Note that the above definition is correct since whenever a nonterminal F of type as above occurs in an equation of the grammar, all its arguments of level 1 must be provided (otherwise the grammar would not be of level 2).

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 vocabulary $\{p_f : f \in \Sigma^\bullet\} \cup \{d_i \mid 1 \leq i \leq m_{\Sigma^\alpha}\} \cup \{E\}$ 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 $x_i \in \mathcal{X}^{\mathcal{G}_\mathbf{0}}$. 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 (cf. Definition 2.1).

Lemma 5.2. *The structure $\mathbf{t}_0^\alpha$ is MSO definable in the structure $\mathbf{t}^\alpha$.*

Furthermore we claim the following.

Lemma 5.3. *Let grammars $\mathcal{G}$ and $\mathcal{G}^\alpha$ be as above, and let $M = \daleth_{\mathcal{G}}(S)$. 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 5.4. *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 is decidable, by the result of Courcelle [7]. Let $M = \daleth_{\mathcal{G}}(S)$. By Lemmas 5.3 and 5.2, the MSO theory of $\mathbf{G}_M$ is decidable. By Proposition 4.6, $[\![\mathcal{G}]\!] = t_M$. It is easy to see that, by construction, M is boundedly typed. Hence the result follows from Theorem 3.4. □

5.1 Examples

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

$$\begin{aligned} S &= Fgc \\ F\varphi x &= f\Big(F(G\varphi)\,(\varphi x)\Big)\,x \\ G\psi y &= \psi\,(\psi y) \end{aligned}$$

It is easy to see that this grammar generates a tree t with $\mathcal{D}\mathrm{om}\, t = \{1^n 2^m : m \leq 2^n\}$, such that $t(1^n) = f$, $t(1^n 2^{2^n}) = c$, and $t(w) = g$ otherwise. Since $\mathcal{D}\mathrm{om}\, t$ considered as a language is not context-free, the tree t is not algebraic (see [7]). Since the grammar is safe, the decidability of the MSO of $\mathbf{t}$ follows from out method.

Now consider a grammar with equations

$$\begin{aligned} S &= Fgc \\ F\varphi x &= f\Big(F(f(\varphi x))x\Big)\,(\varphi x) \end{aligned}$$

This grammar is not safe, and our method does not apply here. Indeed, it turns out that Lemma 5.2 does not hold in this case. By analysing the structure of $\mathbf{t}_0^\alpha$, one can see that it is possible to define (a half of) an infinite grid in it, which implies undecidability of the MSO theory of $\mathbf{t}_0^\alpha$, while the MSO theory of $\mathbf{t}^\alpha$ is decidable by virtue of its algebraicity. On the other hand, it is easy to see that the tree generated by our grammar is also algebraic, and hence has decidable MSO theory. Hence, non–applicability of our method is not an evidence for undecidability.

References

[1] A. Arnold. *Finite transition systems.* Prentice Hall Int., 1994.

[2] A. Arnold and D. Niwiński. Fixed point characterization of weak monadic logic definable sets of trees. In M. Nivat and A. Podelski, editors, *Tree automata and Languages*, pages 159–188. Elsevier, 1992.

[3] 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.

[4] 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.

[5] A. Berarducci and M. Dezani-Ciancaglini. Infinite lambda-calculus and types. *Theoret. Comput. Sci.*, 212:29–75, 1999.

[6] 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.

[7] B. Courcelle. The monadic second–order theory of graphs IX: Machines and their behaviours. *Theoretical Comput. Sci.*, 151:125–162, 1995.

[8] W. Damm. The IO– and OI–hierarchies. *Theoretical Comput. Sci.*, 20(2):95–208, 1982.

[9] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic.* MIT Press, Cambridge, MA, 2000.

[10] 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.

[11] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of EATCS*, 1997(62):222–259.

[12] J. R. Kennaway, J. W. Klop, M. R. Sleep, and F. J. de Vries. Infinitary lambda calculus. *Theoret. Comput. Sci.*, 175:93–125, 1997.

[13] D. Kozen. Results on the proposotional μ-calculus. *Theoretical Comput. Sci.*, 27:333–354, 1983.

[14] 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.

[15] D. Niwiński. Fixed points characterization of infinite behaviour of finite state systems. *Theoretical Comput. Sci.*, 189:1–69, 1997.

[16] M. O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Soc*, 141:1–35, 1969.

[17] Z. Spławski and P. Urzyczyn. Type fixpoints: iteration vs. recursion. In *Proc. 4th ICPF*, pages 102–113. ACM, 1999.

[18] 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.

[19] 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.

Appendix

A Proof of Lemma 4.1

Lemma 4.1. *$\daleth_{\mathcal{G}}$ is a function, i.e., for any $t \in T(\Sigma \cup V \cup \mathcal{X})$, $n \geq 0$, there exists a unique M (up to α-equivalence) such that $(t, M) \in \daleth_{\mathcal{G}}$.*

Proof. We show by induction that the following holds for each $k \in \mathbb{N}$:

$$\forall t\, \forall M\, \forall M' \Big(((t, M) \in \daleth_{\mathcal{G}} \wedge (t, M') \in \daleth_{\mathcal{G}}) \;\Rightarrow\; M{\restriction}k = M'{\restriction}k\Big) \enspace .$$

The base step, $k = 0$, is trivial. Assume therefore that $k > 0$.

- The case of t being a variable or a function symbol is trivial.
- If $t = Ft_1, \ldots t_m$, where $t_1 \ldots . t_m$ are terms such that $\mathrm{type}(\phi_i) = \mathrm{type}(t_i)$, for $i \in [m]$, and $F\phi_1 \ldots \phi_m x_1 \ldots x_n = r$ is an equation of $\mathcal{G}$, then

$$\begin{aligned} & M = \lambda x'_1 \ldots x'_n.N, \;\text{ for some } N \text{ such that} \\ & \qquad\qquad (r[\phi_1{:=}t_1, \ldots, \phi_m{:=}t_m x_1{:=}x'_1, \ldots, x_n{:=}x'_n], N) \in \daleth_{\mathcal{G}} \\ & \Big(\text{resp. } M' = \lambda x'_1 \ldots x'_n.N', \;\text{ for some } N' \text{ such that} \\ & \qquad\qquad (r[\phi_1{:=}t_1, \ldots, \phi_m{:=}t_m, x_1{:=}x'_1, \ldots, x_n{:=}x'_n], N') \in \daleth_{\mathcal{G}}\Big), \end{aligned}$$

  where the variables $x'_1, \ldots, x'_n$ are chosen so that no x'_i occurs free in any of t_j. If $k \leq n$ then $M{\restriction}k = \lambda x'_1 \ldots x'_k = M'{\restriction}k$. If $k > n$, then, by induction hypothesis,

$$N{\restriction}(k-n) = N'{\restriction}(k-n)$$

  because $n > 0$ according to the proviso of Page 11. Hence

$$M{\restriction}k = (\lambda x'_1 \ldots x'_n.N){\restriction}k = \lambda x'_1 \ldots x'_n.(N{\restriction}k-n) = \lambda x'_1 \ldots x'_n.(N'{\restriction}k-n) = (\lambda x'_1 \ldots x'_n.N'){\restriction}k = M'{\restriction}k$$

- If $t = t_1 t_2$ where $t_1 : \mathbf{0} \to \tau$ and $t_2 : \mathbf{0}$ then

$$\begin{aligned} M &= M_1 M_2, \;\text{ for some } M_1 \text{ and } M_2 \text{ such that } (t_1, M_1), (t_2, M_2) \in \daleth_{\mathcal{G}} \\ (\text{resp. } M' &= M'_1 M'_2, \;\text{ for some } M'_1 \text{ and } M'_2 \text{ such that } (t_1, M'_1), (t_2, M'_2) \in \daleth_{\mathcal{G}}). \end{aligned}$$

  Now, by induction hypothesis, $M_1{\restriction}(k-1) = M'_1{\restriction}(k-1)$ and $M_2{\restriction}(k-1) = M'_2{\restriction}(k-1)$. Hence

$$M{\restriction}k = (M_1 M_2){\restriction}k = M_1{\restriction}(k-1) M_2{\restriction}(k-1) = M'_1{\restriction}(k-1) M'_2{\restriction}(k-1) = (M'_1 M'_2){\restriction}k = M'{\restriction}k.$$

□

B Proof of Lemma 4.2

Lemma 4.2. *If $t, t' \in T(\Sigma \cup V \cup \mathcal{X})$ and a variable x has the same type as t', namely $\mathbf{0}$, then*

$$\daleth_{\mathcal{G}}(t[x{:=}t']) =_\alpha \daleth_{\mathcal{G}}(t)[x{:=}\daleth_{\mathcal{G}}(t')] \enspace .$$

Proof. According to the principle of co–induction, it is sufficient to show that the relation

$$\rho = \{(\daleth_{\mathcal{G}}(t[x{:=}t']), \daleth_{\mathcal{G}}(t)[x{:=}\daleth_{\mathcal{G}}(t')]) \mid t, t' \in T(\Sigma \cup V \cup \mathcal{X}),\; x \in \mathcal{X},\; \mathrm{type}(t') = \mathrm{type}(x)\}$$

is a bisimulation. Let $t, t' \in T(\Sigma \cup V \cup \mathcal{X})$ be two terms and $x \in \mathcal{X}$ be a variable such that $\mathrm{type}(t') = \mathrm{type}(x)$. Since ρ contains the identity of $\mathcal{R}\mathrm{an}(\daleth_{\mathcal{G}})$, we check the four conditions of Definition 2.3 assuming that $t \neq x$. (The case of $t = x$ is obvious.)

(1) Obviously $\daleth_{\mathcal{G}}(t[x{:=}t'])$ and $\daleth_{\mathcal{G}}(t)[x{:=}\daleth_{\mathcal{G}}(t')]$ have the same type.

(2) The trees $\daleth_{\mathcal{G}}(t[x{:=}t'])$ and $\daleth_{\mathcal{G}}(t)[x{:=}\daleth_{\mathcal{G}}(t')]$ have the same root. This is the root of $\daleth_{\mathcal{G}}(t)$.

(3) Let M and N be lambda trees such that $\daleth_{\mathcal{G}}(t[x{:=}t']) = \lambda y.M$ and $\daleth_{\mathcal{G}}(t)[x{:=}\daleth_{\mathcal{G}}(t')] = \lambda z.N$. It may only happen when $t = Ft_1 \ldots t_m$ for some equation $F\phi_1 \ldots \phi_m x_1 \ldots x_n = r$ of $\mathcal{G}$ and some terms $t_1, \ldots t_n$ such that $\mathrm{type}(\phi_i) = \mathrm{type}(t_i)$, for $i \in [m]$. Then

$$\lambda y.M = \lambda x_1' \ldots x_n'.\daleth_{\mathcal{G}}(r[\phi_1{:=}t_1, \ldots, \phi_m{:=}t_m, x_1{:=}x_1', \ldots, x_n{:=}x_n'][x{:=}t'])$$
$$(\text{resp. } \lambda z.N = \lambda x_1' \ldots x_n'.\daleth_{\mathcal{G}}(r[\phi_1{:=}t_1, \ldots \phi_m{:=}t_m, x_1{:=}x_1', \ldots, x_n{:=}x_n'])[x{:=}\daleth_{\mathcal{G}}(t')]),$$

where the variables $x_1', \ldots, x_n'$ are chosen so that no x_i' occurs free in any of t_j. Consequently $y = x_1'$ (resp. $z = x_1'$). Now, from

$$\Big(\daleth_{\mathcal{G}}(r[\phi_1{:=}t_1, \ldots, \phi_m{:=}t_m, x_1{:=}x_1', \ldots, x_n{:=}x_n'][x{:=}t']),$$
$$\daleth_{\mathcal{G}}(r[\phi_1{:=}t_1, \ldots, \phi_m{:=}t_m, x_1{:=}x_1', \ldots, x_n{:=}x_n'])[x{:=}\daleth_{\mathcal{G}}(t')]\Big) \in \rho,$$

we have

$$\Big(\daleth_{\mathcal{G}}(\lambda x_2' \ldots x_n'.r[\phi_1{:=}t_1, \ldots \phi_m{:=}t_m, x_1{:=}x_1', \ldots, x_n{:=}x_n'][x{:=}t']),$$
$$\daleth_{\mathcal{G}}(\lambda x_2' \ldots x_n'.r[\phi_1{:=}t_1, \ldots \phi_m{:=}t_m, x_1{:=}x_1', \ldots, x_n{:=}x_n'])[x{:=}\daleth_{\mathcal{G}}(t')]\Big) \in \rho,$$

viz., $(M, N) \in \rho$.

(4) Let M, M', N, N' be lambda trees such that

$$\daleth_{\mathcal{G}}(t[x{:=}t']) = MN \quad \text{and} \quad \daleth_{\mathcal{G}}(t)[x{:=}\daleth_{\mathcal{G}}(t')] = M'N' \ .$$

It may only happen when $t = t_1 t_2$ for some terms $t_1 : \mathbf{0} \to \tau$ and $t_2 : \mathbf{0}$. Consequently

$$\daleth_{\mathcal{G}}(t[x{:=}t']) = \daleth_{\mathcal{G}}((t_1t_2)[x{:=}t']) = \daleth_{\mathcal{G}}(t_1[x{:=}t']t_2[x{:=}t']) = \daleth_{\mathcal{G}}(t_1[x{:=}t'])\daleth_{\mathcal{G}}(t_2[x{:=}t']),$$
$$\daleth_{\mathcal{G}}(t)[x{:=}\daleth_{\mathcal{G}}(t')] = (\daleth_{\mathcal{G}}(t_1)\daleth_{\mathcal{G}}(t_2))[x{:=}\daleth_{\mathcal{G}}(t')] = \daleth_{\mathcal{G}}(t_1)[x{:=}\daleth_{\mathcal{G}}(t')]\daleth_{\mathcal{G}}(t_2)[x{:=}\daleth_{\mathcal{G}}(t')].$$

Thus

$$M = \daleth_{\mathcal{G}}(t_1[x{:=}t']), \qquad N = \daleth_{\mathcal{G}}(t_2[x{:=}t']),$$
$$M' = \daleth_{\mathcal{G}}(t_1)[x{:=}\daleth_{\mathcal{G}}(t')], \qquad N' = \daleth_{\mathcal{G}}(t_2)[x{:=}\daleth_{\mathcal{G}}(t')].$$

Hence, $(M, M') \in \rho$ and $(N, N') \in \rho$. □

C Proof of Lemma 4.4 and Proposition 4.6

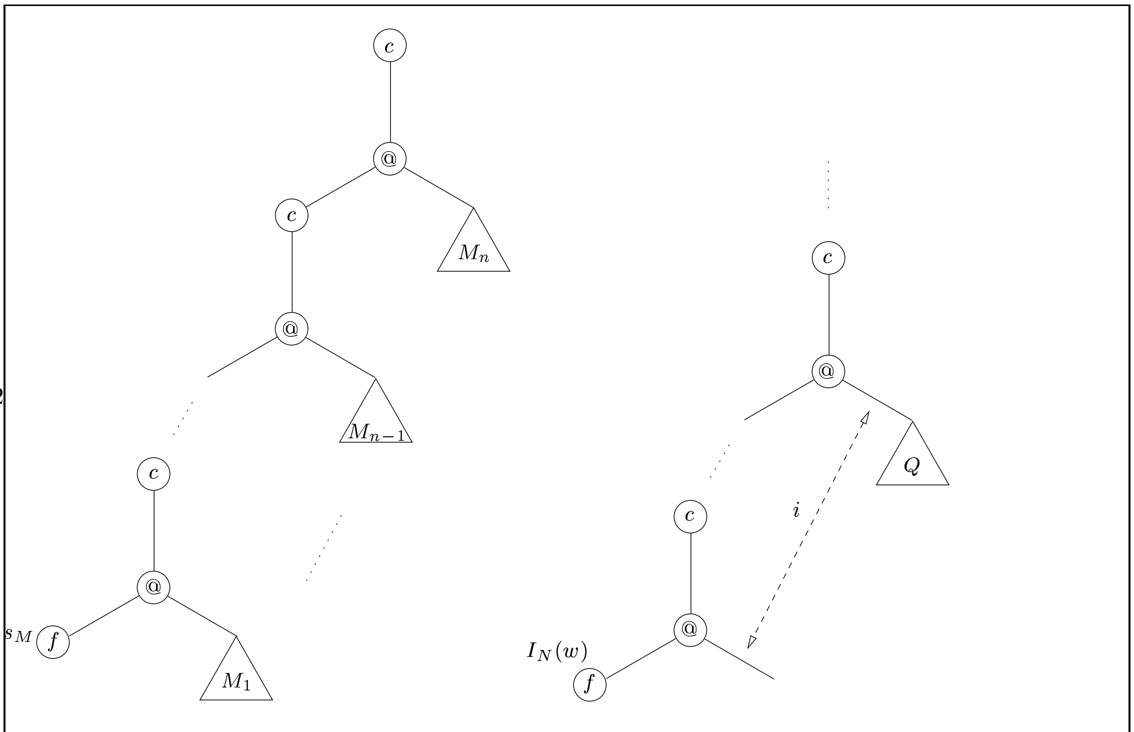
Lemma 4.4. *Let $t \in T(\Sigma \cup V)$ be of type $\mathbf{0}$ and let $M = \daleth_{\mathcal{G}}(t)$. Then $t^{\perp} \sqsubseteq t_M$.*

Proof. By induction on t, which must be of the form $Zt_1 \ldots t_n$. If $Z \in V$ then $t^{\perp} = \perp$ and the result holds trivially. Otherwise $t = ft_1 \ldots t_n$ with $t_i : \mathbf{0}$, for $i = 1, \ldots, n$, and $t^{\perp} = ft_1^{\perp} \ldots t_n^{\perp}$. Let $M_i = \daleth_{\mathcal{G}}(t_i)$. By induction hypothesis, $t_i^{\perp} \sqsubseteq t_{M_i}$. It is now enough to show that (in algebraic notation) $t_M = f(t_{M_1}, \ldots, t_{M_n})$. By definition of $\daleth_{\mathcal{G}}$, we easily see that $G(M)$ is of the form represented in Figure C, and the source of M, denoted s_M, is labeled f. Now, the unique computation path starting from a configuration $(s_M, q^{i-1}p)$ necessarily enters $G(M_i)$ with the empty pds. Hence it must subsequently reach the source of M_i, provided it exists. The claim follows from the definition of derived trees. □

Proposition 4.6. *Let $M = \daleth_{\mathcal{G}}(S)$. Then $t_M = [\![\mathcal{G}]\!]$.*

Proof. (Continuation) We proof the inclusion $t_M \sqsubseteq [\![\mathcal{G}]\!]$ in the remaining case, i.e., when $\perp$ does occur in $[\![\mathcal{G}]\!]$.

Let $t = [\![\mathcal{G}]\!]$. The case when $t = \perp$ has been considered in Lemma 4.5. Otherwise, there is some $w \in \mathcal{D}\mathrm{om}\, t$, such that $t(w) = f$, for some k-ary signature symbol f, and $t(wi) = \perp$, for



(a) A path in $G(M)$ to the source of M.   (b) The image of w under I_N.

Fig. 4.

some $i \in [k]$. Clearly we can find a finite reduction from S to, say, r, such that $r^{\perp}(w) = f$. It is convenient to momentarily represent r as a finite tree which coincides with $r^{\perp}$ except for that its leaves are labeled by the corresponding subterms of r instead of $\perp$. By hypothesis, the term, r' say, standing at the position wi is an application (remember our assumption that each nonterminal has at least one individual argument) whose head symbol is a nonterminal. Moreover, no reduction of r' in $\mathcal{G}$ leads to a term whose head symbol would be a signature symbol.

Let $N = \daleth_{\mathcal{G}}(r)$. We know from Lemma 4.4 that $t_N(w) = f$. It is not difficult to see that the image of w in $G(N)$ under the mapping I_N (cf. Section 2.2) is situated as represented by Figure C, where $Q = \daleth_{\mathcal{G}}(r')$. Moreover, the unique computation path starting from the configuration $(\alpha_f = I_N(w), q^{i-1}p)$ must enter the graph $G(Q)$ with pds ε. In order to apply Lemma 4.5, consider a new grammar $\mathcal{G}'$ which differs from $\mathcal{G}$ only in that it has a new starting symbol S' and equation $S' = r'$. We clearly have $G(\daleth_{\mathcal{G}'}(S')) = G(\daleth_{\mathcal{G}}(r'))$, and $[\![\mathcal{G}']\!] = \perp$, by the choice of r'. On the other hand, the suffix of Π from α_Q coincides with the unique computation path $\daleth_{\mathcal{G}'}(S')$ starting from the configuration (α_Q, ε) (where α_Q is the entry node of $\daleth_{\mathcal{G}'}(S')$). By Lemma 4.5, this path is infinite. Therefore, $t_N(wi) = \perp$.

But, as we have noticed in the first part of the proof, $t_N = t_M$, hence $t_M(w) = \perp$, and so we are done. $\square$