Linear high-order deterministic tree transducers with regular look-ahead

Paul Gallot, Aurélien Lemay, Sylvain Salvati

▶ To cite this version:

Paul Gallot, Aurélien Lemay, Sylvain Salvati. Linear high-order deterministic tree transducers with regular look-ahead. MFCS 2020 : The 45th International Symposium on Mathematical Foundations of Computer Science, Andreas Feldmann; Michal Koucky; Anna Kotesovcova, Aug 2020, Prague, Czech Republic. 10.4230/LIPIcs.MFCS.2020.34 . hal-02902853v2

HAL Id: hal-02902853
https://hal.archives-ouvertes.fr/hal-02902853v2

Submitted on 18 Sep 2020

Linear High-Order Deterministic Tree transducers with Regular look-ahead

Paul D. Gallot
INRIA,Université de Lille
paul.gallot@inria.fr

Aurélien Lemay
Université de Lille, INRIA, CNRS
aurelien.lemay@univ-lille.fr

Sylvain Salvati
Université de Lille, INRIA, CNRS
sylvain.salvati@univ-lille.fr

Abstract

We introduce the notion of high-order deterministic top-down tree transducers (HODT) whose outputs correspond to simply-typed lambda-calculus formulas. These transducers are natural generalizations of known models of top-tree transducers such as: Deterministic Top-Down Tree Transducers, Macro Tree Transducers, Streaming Tree Transducers…We focus on the linear restriction of high order tree transducers with look-ahead ($\mathrm{HODTR}_{\mathrm{lin}}$), and prove this corresponds to tree to tree functional transformations defined by Monadic Second Order (MSO) logic. We give a specialized procedure for the composition of those transducers that uses a flow analysis based on coherence spaces and allows us to preserve the linearity of transducers. This procedure has a better complexity than classical algorithms for composition of other equivalent tree transducers, but raises the order of transducers. However, we also indicate that the order of a $\mathrm{HODTR}_{\mathrm{lin}}$ can always be bounded by 3, and give a procedure that reduces the order of a $\mathrm{HODTR}_{\mathrm{lin}}$ to 3. As those resulting $\mathrm{HODTR}_{\mathrm{lin}}$ can then be transformed into other equivalent models, this gives an important insight on composition algorithm for other classes of transducers. Finally, we prove that those results partially translate to the case of almost linear HODTR: the class corresponds to the class of tree transformations performed by MSO with unfolding (not closed by composition), and provide a mechanism to reduce the order to 3 in this case.

2012 ACM Subject Classification Theory of computation → Transducers; Theory of computation → Lambda calculus; Theory of computation → Tree languages

Keywords and phrases Transducers,λ-calculus,Trees

Digital Object Identifier 10.4230/LIPIcs.MFCS.2020.34

Related Version A full version of the paper is available at `https://hal.archives-ouvertes.fr/hal-02902853v1`.

Funding *Paul D. Gallot*: ANR-15-CE25-0001 – Colis
Aurélien Lemay: ANR-15-CE25-0001 – Colis
Sylvain Salvati: ANR-15-CE25-0001 – Colis

45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020).
Editors: Javier Esparza and Daniel Král'; Article No. 34; pp. 34:1–34:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Tree Transducers formalize transformations of structured data such as Abstract Syntax Trees, XML, JSON, or even file systems. They are based on various mechanisms that traverse tree structures while computing an output: Top-Down and Bottom-Up tree transducers [17, 4] which are direct generalizations of deterministic word transducers [8, 7, 3], but also more complex models such as macro tree transducers [11] (MTT) or streaming tree transducers [1] (STT) to cite a few.

Logic offers another, more descriptive, view on tree transformations. In particular, Monadic Second Order (MSO) logic defines a class of tree transformations (MSOT) [5, 6] which is expressive and is closed under composition. It coincides with the class of transformations definable with MTT enhanced with a regular look-ahead and restricted to finite copying [9, 10], and also with the class of STT [1].

We argue here that simply typed λ-calculus gives a uniform generalisation of all these different models. Indeed, they can all be considered as classes of programs that read input tree structures, and, at each step, compose tree operations which in the end produce the final output. Each of these tree operations can be represented using simply typed λ-terms.

In this paper, we define top-down tree transducers that follow the usual definitions of such machines, except that rules can produce λ-terms of arbitrary types. We call these machines, High-Order Top-down tree transducers, or High-Order Deterministic Tree Transducers (HODT) in the deterministic case. This class of transducers naturally contains top-down tree transducers, as they are HODT of order 0 (the output of rules are trees), but also MTT, which are HODT of order 1 (outputs are tree contexts). They also contain STT, which can be translated directly into HODT of order 3 with some restricted continuations. Also, STT traverse their input tree represented as a string in a leftmost traversal (a stream). This constraint could easily be adapted to our model but would yield technical complications that are not the focus of this paper. Finally, our model generalizes High Level Tree Transducers defined in [12], which also produce λ-term, but restricted to the safe λ-calculus case.

In this paper we focus on the *linear* and *almost linear* restrictions of HODT. In terms of expressiveness, linear HODTR ($\mathrm{HODTR}_{\mathrm{lin}}$) corresponds to the class of MSOT. This links our formalism to other equivalent classes of transducers, such as finite-copying macro-tree transducers [9, 10], with an important difference: the linearity restriction is a simple syntactic restriction, whereas finite-copying or the equivalent single-use-restricted condition are both global conditions that are harder to enforce. For STT, the linearity condition corresponds to the copyless condition described in [1] and where the authors prove that any STT can be made copyless.

The relationship of $\mathrm{HODTR}_{\mathrm{lin}}$ to MSOT is made via a transformation that *reduces the order* of transducers. We indeed prove that for any $\mathrm{HODTR}_{\mathrm{lin}}$, there exists an equivalent $\mathrm{HODTR}_{\mathrm{lin}}$ whose order is at most 3. This transformation allows us to prove then that $\mathrm{HODTR}_{\mathrm{lin}}$ are equivalent to Attribute Tree Transducers with the single use restriction ($\mathrm{ATT}_{\mathrm{sur}}$). In turn, this shows that $\mathrm{HODTR}_{\mathrm{lin}}$ are equivalent to MSOT [2].

One of the main interests of $\mathrm{HODTR}_{\mathrm{lin}}$ is that λ-calculus also offers a simple composition algorithm. This approach gives an efficient procedure for composing two $\mathrm{HODTR}_{\mathrm{lin}}$. In general, this procedure raises the order of the produced transducer. In comparison, composition in other equivalent classes are either complex or indirect (through MSOT). In any case, our procedure has a better complexity. Indeed, it benefits from higher-order which permits a larger number of implementations for a given transduction. The complexity of the construction is also lowered by the use of a notion of determinism slightly more liberal than usual that we call *weak determinism.*

The last two results allow us to obtain a composition algorithm for other equivalent classes of tree transducer, such as MTT or STT: compile into $\text{HODTR}_{\text{lin}}$, compose, reduce the order, and compile back into the original model. The advantage of this approach over the existing ones is that the complex composition procedure is decomposed into two simpler steps (the back and forth translations between the formalisms are unsurprising technical procedures). We believe in fact that existing approaches [12, 1] combine in one step the two elements, which is what makes them more complex.

The property of order reduction also applies to a wider class of HODT, *almost linear* HODT (HODTR_{al}). Again here, this transformation allows us to prove that this class of tree transformations is equivalent to that of Attribute Tree Transducers which is known to be equivalent to MSO tree transformations with unfolding [2], i.e. MSO tree transduction that produce Directed Acyclic Graphs (i.e. trees with shared sub-trees) that are unfolded to produce a resulting tree. We call these transductions Monadic Second Order Transductions with Sharing (MSOTS). Note however that HODTR_{al} are not closed under composition.

Section 2 presents the technical definitions used throughout the paper. In particular, it gives the definitions of the various notions of transducers studied in the paper and also the notion of weak determinism. Section 3 studies the expressivity of linear and almost linear higher-order transducer by relating them to MSOT and MSOTS. It focuses more specifically on the order reduction procedure that is at the core of the technical work. Section 4 presents the composition algorithm for linear higher-order transducers. This algorithm is based on Girard's coherence spaces and can be interpreted as a form of partial evaluation for linear higher-order programs. Finally we conclude.

2 Definitions

This section presents the main formalisms we are going to use throughout the paper, namely simply typed λ-calculus, finite state automata and high-order transducers.

2.1 λ-calculus

Fix a finite set of atomic types $\mathcal{A}$, we then define the set of types over $\mathcal{A}$, $\text{types}(\mathcal{A})$, as the types that are either an atomic type, i.e. an element of $\mathcal{A}$, or a functional type $(A \to B)$, with A and B being in $\text{types}(\mathcal{A})$. The operator $\to$ is right-associative and $A_1 \to \cdots \to A_n \to B$ denotes the type $(A_1 \to (\cdots \to (A_n \to B)\cdots))$. The order of a type A is inductively defined by $\text{order}(A) = 0$ when $A \in \mathcal{A}$, and $\text{order}(A \to B) = \max(\text{order}(A)+1, \text{order}(B))$.

A signature Σ is a triple $(C, \mathcal{A}, \tau)$ with C being a finite set of *constants*, $\mathcal{A}$ a finite set of *atomic types*, and τ a mapping from C to $\text{types}(\mathcal{A})$, *the typing function.*

We allow ourselves to write $\text{types}(\Sigma)$ to refer to the set $\text{types}(\mathcal{A})$. The order of a signature is the maximal order of a type assigned to a constant (i.e. $\max\{\text{order}(\tau(c)) \mid c \in C\}$). In this work, we mostly deal with tree signatures which are of order 1 and whose set of atomic types is a singleton. In such a signature with atomic type o, the types of constants are of the form $o \to \cdots \to o \to o$. We write $o^n \to o$ for an order-1 type which uses $n+1$ occurrences of o, for example, $o^2 \to o$ denotes $o \to o \to o$. When c is a constant of type A, we may write c^A to make explicit that c has type A. Two signatures $\Sigma_1 = (C_1, \mathcal{A}_1, \tau_1)$ and $\Sigma_2 = (C_2, \mathcal{A}_2, \tau_2)$ so that for every c in $C_1 \cap C_2$ we have $\tau_1(c) = \tau_2(c)$ can be summed, and we write $\Sigma_1 + \Sigma_2$ for the signature $(C_1 \cup C_2, \mathcal{A}_1 \cup \mathcal{A}_2, \tau)$ so that if c is in C_1, $\tau(c) = \tau_1(c)$ and if c is in C_2, $\tau(c) = \tau_2(c)$. The sum operation over signatures being associative and commutative, we write $\Sigma_1 + \cdots + \Sigma_n$ to denote the sum of several signatures.

We assume that for every type A, there is an infinite countable set of variables of type A. When two types are different the set of variables of those types are of course disjoint. As with constants, we may write x^A to make it clear that x is a variable of type A.

When Σ is a signature, we define the family of simply typed λ-terms over Σ, denoted $\Lambda(\Sigma) = (\Lambda^A(\Sigma))_{A \in \text{types}(\Sigma)}$, as the smallest family indexed by types(Σ) so that:

- if c^A is in Σ, then c^A is in $\Lambda^A(\Sigma)$,
- x^A is in $\Lambda^A(\Sigma)$,
- if $A = B \to C$ and M is in $\Lambda^C(\Sigma)$, then $(\lambda x^B.M)$ is in $\Lambda^A(\Sigma)$,
- if M is in $\Lambda^{B \to A}(\Sigma)$ and N is in $\Lambda^B(\Sigma)$, then (MN) is in $\Lambda^A(\Sigma)$.

The term M is a *pure* λ-term if it does not contain any constant c^A from Σ. When the type is irrelevant we write $M \in \Lambda(\Sigma)$ instead of $M \in \Lambda^A(\Sigma)$. We drop parentheses when it does not bring ambiguity. In particular, we write $\lambda x_1 \dots x_n.M$ for $(\lambda x_1(\dots(\lambda x_n.M)\dots))$, and $M_0 M_1 \dots M_n$ for $((\dots(M_0 M_1)\dots)M_n)$.

The set $\text{fv}(M)$ of free variables of a term M is inductively defined on the structure of M:

- $\text{fv}(c) = \emptyset$,
- $\text{fv}(x) = \{x\}$,
- $\text{fv}(MN) = \text{fv}(M) \cup \text{fv}(N)$,
- $\text{fv}(\lambda x.M) = \text{fv}(M) - \{x\}$.

Terms which have no free variables are called *closed*. We write $M[x_1, \dots, x_k]$ to emphasize that $\text{fv}(M)$ is included in $\{x_1, \dots, x_k\}$. When doing so, we write $M[N_1, \dots, N_k]$ for the capture avoiding substitution of variables x_1, …, x_k by the terms N_1, …, N_k. In other contexts, we simply use the usual notation $M[N_1/x_1, \dots, N_k/x_k]$. Moreover given a substitution θ, we write $M.\theta$ for the result of applying this (capture avoiding) substitution and we write $\theta[N_1/x_1, \dots, N_k/x_k]$ for the substitution that maps the variables x_i to the terms N_i but is otherwise equal to θ. Of course, we authorize such substitutions only when the λ-term N_i has the same type as the variable x_i.

We take for granted the notions of β-contraction, noted $\to_\beta$, β-reduction, noted $\xrightarrow{*}_\beta$, β-conversion, noted $=_\beta$, and β-normal form for terms.

Consider closed terms of type o that are in β-normal form and that are built on a tree signature, they can only be of the form $a\, t_1 \dots t_n$ where a is a constant of type $o^n \to o$ and t_1, …, t_n are closed terms of type o in β-normal form. This is just another notation for ranked trees. So when the type o is meant to represent trees, types of order 1 which have the form $o \to \cdots \to o \to o$ represent functions from trees to trees, or more precisely tree contexts. Types of order 2 are types of trees parametrized by contexts. The notion of order captures the complexity of the operations that terms of a certain type describe.

A term M is said *linear* if each variable (either bound or free) in M occurs exactly once in M. A term M is said *syntactically almost linear* when each variable in M of non-atomic type occurs exactly once in M. Note that, through β-reduction, linearity is preserved but not syntactic almost linearity.

For example, given a tree signature Σ_1 with one atomic type o and two constants f of type $o^2 \to o$ and a of type o, the term $M = (\lambda y_1 y_2.f\, y_1\, (f\, a\, y_2))\, a\, (f\, x\, a)$ with free variable x of type o is linear because each variable (y_1, y_2 and x) occurs exactly once in M. The term M contains a β-redex so: $(\lambda y_1 y_2.f\, y_1\, (f\, a\, y_2))\, a\, (f\, x\, a) \to_\beta (\lambda y_2.f\, a\, (f\, a\, y_2))\, (f\, x\, a) \to_\beta f\, a\, (f\, a\, (f\, x\, a))$. The term $f\, a\, (f\, a\, (f\, x\, a))$ has no β-redex so it is the β-normal form of M.

Another example: the term $M_2 = (\lambda y.f\, y\, y)\, (x\, a)$ with free variable x of type $o \to o$ is syntactically almost linear because the variable y which occurs twice in the term is of the atomic type o. It β-reduces to the term $M_2' = f\, (x\, a)\, (x\, a)$ which is not syntactically almost linear, so β-reduction does not preserve syntactical almost linearity.

We call a term *almost linear* when it is β-convertible to a syntactically almost linear term. Almost linear terms are characterized also by typing properties (see [15]).

2.2 Tree Automata

We present here the classical definition of deterministic bottom-up tree automaton (BOT) adapted to our formalism. A BOT $\mathtt{A}$ is a tuple (Σ_P, Σ, R) where:

- $\Sigma = (C, \{o\}, \tau)$ is a first-order tree signature, the *input signature*,
- $\Sigma_P = (P, \{o\}, \tau_P)$ is the *state signature*, and is such that for every $p \in P$, $\tau_P(p) = o$. Constants of P are called *states*,
- R is a finite set of rules of the form $a\, p_1 \dots p_n \to p$ where:
  - p,$p_1, \dots, p_n$ are states of P,
  - a is a constant of Σ with type $o^n \to o$.

  An automaton is said *deterministic* when there is at most one rule in R for each possible left hand side. It is *non-deterministic* otherwise.

Apart from the notation, our definition differs from the classical one by the fact there are no final states, and hence, the automaton does not describe a language. This is due to the fact that BOT will be used here purely for look-ahead purposes.

2.3 High-Order Deterministic top-down tree Transducers

From now on we assume that Σ_i is a tree signature for every number i and that its atomic type is o_i.

A Linear High-Order Deterministic top-down Transducer with Regular look-ahead ($\mathrm{HODTR_{lin}}$) T is a tuple $(\Sigma_Q, \Sigma_1, \Sigma_2, q_0, R, \mathtt{A})$ where:

- $\Sigma_1 = (C_1, \{o_1\}, \tau_1)$ is a first-order tree signature, the *input signature*,
- $\Sigma_2 = (C_2, \{o_2\}, \tau_2)$ is a first-order tree signature, the *output signature*,
- $\Sigma_Q = (Q, \{o_1, o_2\}, \tau_s)$ is the *state signature*, and is such that for every $q \in Q$, $\tau_s(q)$ is of the form $o_1 \to A_q$ where A_q is in types(Σ_2). Constants of Q are called *states*,
- $q_0 \in Q$ is the *initial state*,
- $\mathtt{A}$ is a BOT over the tree signature Σ_1, the *look-ahead* automaton, with set of states P,
- R is a finite set of rules of the form

  $$q(a\overrightarrow{x})\langle\overrightarrow{p}\rangle \to M(q_1x_1)\dots(q_nx_n)$$

  where:
  - $q, q_1, \dots, q_n \in Q$ are states of Σ_Q,
  - a is a constant of Σ_1 with type $o_1^n \to o_1$,
  - $\overrightarrow{x} = x_1, \dots, x_n$ are variables of type o_1, they are the child trees of the root labeled a,
  - $\overrightarrow{p} = p_1, \dots, p_n$ are in P (the set of states of the look-ahead $\mathtt{A}$),
  - M is a linear term of type $A_{q_1} \to \dots \to A_{q_n} \to A_q$ built on signature $\Sigma_2 + \Sigma_Q$.
  - there is one rule per possible left-hand side (determinism).

Notice that we have given states a type of the form $o_1 \to A$ where $A \in$ types(o_2). The reason why we do this is to have a uniform notation. Indeed, a state q is meant to transform, thanks to the rules in R, a tree built in Σ_1 into a λ-term built on Σ_2 with type A_q. So we simply write $q\,M\,N_1 \dots N_n$ when we want to transform M with the state q and pass N_1,..., N_n as arguments to the result of the transformation. We write Σ_T for the signature $\Sigma_1 + \Sigma_2 + \Sigma_Q$. Notice also that the right-hand part of a rule is a term that is built only with constants of Σ_2, states from Σ_Q and variables of type o_1. Thus, in order for this term to have a type in types(Σ_2), it is necessary that the variables of type o_1 only occur as the first argument of a state in Σ_Q. Finally, remark that we did not put any requirement on the type of the initial state. So as to restrict our attention to transducers as they are usually understood, it suffices to add the requirement that the initial state is of type $o_1 \to o_2$. However, we consider as well that transducers may produce *programs* instead of first order terms.

The linearity constraint on M affects both bound variables and the free variables $x_1, \dots, x_n$, meaning that all of the subtrees $x_1, \dots, x_n$ are used in computing the output. That will be important for the composition of two transducers because if the first transducer fails in a branch of its input tree then the second transducer, applied to that tree, must fail too. This restriction forcing the use of input subtrees does not reduce the model's expressivity because we can always add a state q which visits the subtree but only produces the identity function on type o_2 (this state then has type $A_q = o_1 \to o_2 \to o_2$).

Almost linear high-order deterministic top-down transducer with regular look-ahead ($\mathrm{HODTR}_{\mathrm{al}}$) are defined similarly, with the distinction that a term M appearing as a right-hand side of a rule should be almost linear.

As we are concerned with the size of the composition of transducers, we wish to relax a bit the notion of $\mathrm{HODTR}_{\mathrm{lin}}$. Indeed, when composing $\mathrm{HODTR}_{\mathrm{lin}}$ we may have to determinize the look-ahead so as to obtain a $\mathrm{HODTR}_{\mathrm{lin}}$, which may cause an exponential blow-up of the look-ahead. However if we keep the look-ahead non-deterministic, the transducer stays deterministic in the weaker sense that only one rule of the transducer can apply when it is actually run. For this we adopt a slightly relaxed notion of deterministic transducer that we call high-order weakly deterministic top-down transducer with regular look-ahead ($\mathrm{HOWDTR}_{\mathrm{lin}}$). They are similar to $\mathrm{HODTR}_{\mathrm{lin}}$ but they can have *non-deterministic automata* as look-ahead with the proviso that when $q(a\, x_1 \dots x_n)\langle p_1, \dots, p_n\rangle \to M[x_1, \dots, x_n]$ and $q(a\, x_1 \dots x_n)\langle p'_1, \dots, p'_n\rangle \to M'[x_1, \dots, x_n]$ are two distinct rules of the transducer then it must be the case that for some i there is no tree that is recognized by both p_i and p'_i. This property guarantees that when transforming a term at most one rule can apply for every possible state. Notice that it suffices to determinize the look-ahead so as to obtain a $\mathrm{HODTR}_{\mathrm{lin}}$ from a $\mathrm{HOWDTR}_{\mathrm{lin}}$, and therefore the two models are equivalent.

Given a $\mathrm{HODTR}_{\mathrm{lin}}$, a $\mathrm{HODTR}_{\mathrm{al}}$ or a $\mathrm{HOWDTR}_{\mathrm{lin}}$ T, we write $T :: \Sigma_1 \longrightarrow \Sigma_2$ to mean that the input signature of T is Σ_1 and its output signature is Σ_2.

A transducer T induces a notion of reduction on terms. A T-redex is a term of the form $q(a\, M_1 \dots M_n)$ if and only if $q(a\, x_1 \dots x_n)\langle p_1, \dots, p_n\rangle \to M[x_1, \dots, x_n]$ is a rule of T and (the β-normal forms of) $M_1, \dots, M_n$ are respectively accepted by $\mathtt{A}$ with the states $p_1, \dots, p_n$. In that case, a T-contractum of $q(a\, M_1 \dots M_n)$ is $M[M_1, \dots, M_n]$. Notice that T-contracta are typed terms and that they have the same type as their corresponding T-redices. The relation of T-contraction relates a term M and a term M' when M' is obtained from M by replacing one of its T-redex with a corresponding T-contractum. We write $M \to_T M'$ when M T-contracts to M'. The relation of β-reduction is confluent, and so is the relation of T-reduction as transducers are deterministic, moreover, the union of the two relations is terminating. It is not hard to prove that it is also locally confluent and thus confluent. It follows that $\to_{\beta,T}$ (which is the union of $\to_\beta$ and $\to_T$) is confluent and strongly normalizing. Given a term M built on Σ_T, we write $|M|_T$ to denote its normal form modulo $=_{\beta,T}$.

Then we write $\mathrm{rel}(T)$ for the relation:
$\{(M, |q_0 M|_T) \mid \text{M is a closed term of type } o_1 \text{ and } |q_0 M|_T \in \Lambda(\Sigma_2)\}$.

Notice that when $|q_0 M|_T$ contains some states of T, as it is usual, the pair $(M, |q_0 M|_T)$ is not in the relation.

Given a finite set of trees L_1 on Σ_1 and L_2 included in $\Lambda^{A_{q_0}}$, we respectively write $T(L_1)$ and $T^{-1}(L_2)$ for the image of L_1 by T and the inverse image of L_2 by T.

We give an example of a $\mathrm{HODTR}_{\mathrm{lin}}$ T that computes the result of additions of numeric expressions (numbers being represented in unary notation). For this we use an input tree signature with type o_1, and constants Z^{o_1}, S^{o_1} and $\mathrm{add}^{o_1 \to o_1 \to o_1}$ which respectively denote zero, the successor function and addition. The output signature is similar but different to avoid confusion: it uses the type o_2 and constants O^{o_2}, $N^{o_2 \to o_2}$ which respectively denote zero and successor.

We do not really need the look-ahead automaton for this computation, so we omit it for this example. We could have a blank look-ahead automaton $\mathtt{A}$ with one state l and rules: $\mathtt{A}(Z) = l$, $\mathtt{A}(S\,l) = l$, $\mathtt{A}(\mathrm{add}\,l\,l) = l$; which would not change the result of the transducer.

The transducer has two states: q_0 of type $o_1 \to o_2$ (the initial state), and q_i of type $o_1 \to o_2 \to o_2$. The rules of the transducer are the following:

- $q_0(Z) \to O$, $q_0(S\,x) \to N(q_i\,x\,O)$,
- $q_0(\mathrm{add}\,x\,y) \to q_i\,x\,(q_i\,y\,O)$,
- $q_i(Z) \to \lambda x.x$,
- $q_i(S\,x) \to \lambda y.N(q_i\,x\,y)$,
- $q_i(\mathrm{add}\,x\,y) \to \lambda z.q_i\,x\,(q_i\,y\,z)$,

As an example, we perform the transduction of the following term $\mathrm{add}(S(S\,Z))(S(S(S\,Z)))$:

$$\begin{aligned} & q_0(\mathrm{add}(S(S\,Z))(S(S(S\,Z)))) \quad \to_T \quad (q_i(S(S\,Z)))(q_i(S(S(S\,Z)))O) \\ & \quad \overset{*}{\to}_T \quad (\lambda y_1.N((\lambda y_2.N((\lambda x.x)y_2))y_1))((\lambda y_3.N((\lambda y_4.N((\lambda y_5.N((\lambda x.x)y_5))y_4))y_3))O) \\ & \quad \overset{*}{\to}_\beta \quad N(N(N(N(N\,O)))) \end{aligned}$$

The state q_i transforms a sequence of n symbols S into a λ-term of the form $\lambda x.N^n(x)$, and the *add* maps both its children into such terms and composes them. The state q_0 simply applies O to the resulting term.

Note that our reduction strategy here has consisted in first computing the T-redices and then reducing the β-redices. This makes the computation simpler to present. As we mentioned above a head-reduction strategy would lead to the same result.

The order of the $\mathrm{HODTR}_{\mathrm{lin}}$ T is $\max\{\mathrm{order}(A_q) \mid q \in Q\}$. Before going further, we want to discuss how our framework relates to other transduction models. More specifically how the notion of order of transformations generalizes the DTOP and MTT transduction models: if we relax the constraint of linearity of our transducers, then DTOP and MTT can be seen as non-linear transducers of order 0 and 1 respectively. In contrast of these, we chose to study the constraint of linearity instead of the constraint of order and, in this paper, we will explore the benefits of this approach. Firstly we will explain why increasing the order beyond order 3 does not increase the expressivity of neither $\mathrm{HODTR}_{\mathrm{lin}}$ nor $\mathrm{HODTR}_{\mathrm{al}}$. Next we will show how $\mathrm{HODTR}_{\mathrm{lin}}$ and $\mathrm{HOWDTR}_{\mathrm{lin}}$ both capture the expressivity of tree transformations defined by monadic second order logic. Lastly, we will prove that, contrary to MTT, the class of $\mathrm{HODTR}_{\mathrm{lin}}$ transformations is closed under composition, we will give an algorithm for computing the composition of $\mathrm{HODTR}_{\mathrm{lin}}$ and $\mathrm{HOWDTR}_{\mathrm{lin}}$, and explain why using $\mathrm{HOWDTR}_{\mathrm{lin}}$ avoids an exponential blow-up in the size of the composition transducer.

3 Order reduction and expressiveness

In this section we outline a construction that transforms a transducer of $\mathrm{HODTR}_{\mathrm{lin}}$ or $\mathrm{HODTR}_{\mathrm{al}}$ into an equivalent linear or almost linear transducer of order ≤ 3. These two constructions are similar and central to proving that $\mathrm{HODTR}_{\mathrm{lin}}$ and $\mathrm{HODTR}_{\mathrm{al}}$ are respectively equivalent to Monadic Second Order Transductions from trees to trees (MSOT) and to Monadic Second Order Transductions from trees to terms (i.e. trees with sharing) (MSOTS). We will later show that there are translations between $\mathrm{HODTR}_{\mathrm{lin}}$ of order 3 and attribute tree transducers with the *single use restriction* and between $\mathrm{HODTR}_{\mathrm{al}}$ of order 3 and attribute tree transducers. These two models are known to be respectively equivalent to MSOT and MSOTS [2].

The central idea in the construction consists in decomposing λ-terms M into pairs $\langle M', \sigma\rangle$ where M' is a pure λ-term and σ is a substitution of variables with the following properties:

- $M =_\beta M'.\sigma$,
- the free variables of M' have at most order 1,
- for every variable x, $\sigma(x)$ is a closed λ-term,
- the number of free variables in M' is minimal.

In such a decomposition, we call the term M' a *template*. In case M is of type A, linear or almost linear, it can be proven that M' can be taken from a finite set [14]. The linear case is rather simple, but the almost linear case requires some precaution as one needs first to put M in syntactically almost linear form and then make the decomposition. Though the almost linear case is more technical the finiteness argument is the same in both cases and is based on proof theoretical arguments in multiplicative linear logic which involves polarities in a straightforward way.

The linear case conveys the intuition of decompositions in a clear manner. One takes the normal form of M and then delineates the largest contexts of M, i.e. first order terms that are made only with constants and that are as large as possible. These contexts are then replaced by variables and the substitution σ is built accordingly. The fact that the contexts are chosen as large as possible makes it so that no introduced variable can have as argument a term of the form $x\, M_1 \dots M_n$ where x is another variable introduced in the process. Therefore, the new variables introduced in the process bring one negative atom and several (possibly 0) positive ones and all of them need to be matched with positive and negative atoms in the type of M as, under these conditions, they cannot be matched together. This explains why there are only finitely many possible templates for a fixed type.

▶ **Theorem 1.** *For all type A built on tree signature Σ, the set of templates of closed linear (or almost linear) terms of type A is finite.*

Moreover, the templates associated with a λ-term can be computed compositionally (i.e. from the templates of its parts). As a result, templates can be computed by the look-ahead of $\mathrm{HODTR}_{\mathrm{lin}}$ or of $\mathrm{HODTR}_{\mathrm{al}}$. When reducing the order, we enrich the look-ahead with template information while the substitution that is needed to reconstruct the produced term is outputted by the new transducer. The substitution is then performed by the initial state used at the root of the input tree which then outputs the same result as the former transducer. The substitution can be seen as a tuple of order 1 terms. It is represented as a tuple using Church encoding, i.e. a continuation. This makes the transducer we construct be of order 3.

▶ **Theorem 2.** *Any $HODTR_{lin}$ (resp. $HODTR_{al}$) has an equivalent $HODTR_{lin}$ (resp. $HODTR_{al}$) of order* 3.

The proof of this result shows that every $\mathrm{HODTR}_{\mathrm{lin}}$ (or $\mathrm{HODTR}_{\mathrm{al}}$) can be seen as mapping trees to tuples of contexts and combining these contexts in a linear (resp. almost linear) way. This understanding of $\mathrm{HODTR}_{\mathrm{lin}}$ and of $\mathrm{HODTR}_{\mathrm{al}}$ allows us to prove that they are respectively equivalent to Attribute Tree Transducers with Single Use Restriction ($\mathrm{ATT}_{\mathrm{sur}}$); and to Attribute Tree Transducers (ATT). Then, using [2], we can conclude with the following expressivity result:

▶ **Theorem 3.** *$HODTR_{lin}$ are equivalent to MSOT and $HODTR_{al}$ are equivalent to MSOTS.*

The full proofs of these are rather technical and are not detailed here, but they appear in the full version of the article. The proof that HODTR$_{\mathrm{lin}}$ are equivalent to MSOT could have been simpler by using the equivalence with MTT with the *single-use restricted* property instead of ATT, but we would still need to use ATT to show that HODTR$_{\mathrm{al}}$ are equivalent to MSOTS.

4 Composition of HODTR$_{\mathrm{lin}}$

As we are interested in limiting the size of the transducer that is computed, and even though our primary goal is to compose HODTR$_{\mathrm{lin}}$, this section is devoted to the composition of HOWDTR$_{\mathrm{lin}}$. Indeed, working with non-deterministic look-aheads allows us to save the possibly exponential cost of determinizing an automaton.

4.1 Semantic analysis

Let $T_1 = (\Sigma_Q, \Sigma_1, \Sigma_2, q_0, R_1, \mathtt{A}_1)$ and $T_2 = (\Sigma_P, \Sigma_2, \Sigma_3, p_0, R_2, \mathtt{A}_2)$ be two Linear High-Order Weakly Deterministic tree Transducers with Regular look-ahead. The rules of T_1 can be written: $q(a\,\vec{x})\langle\vec{\ell}\,\rangle \to M\,(q_1\ x_1)\dots(q_n\ x_n)$ where $q, q_1, \dots, q_n \in Q$ are states of T_1, $\vec{\ell} = \ell_1, \dots, \ell_n$ are states of $\mathtt{A}_1$ and the λ-term M is of type $A_{q_1} \to \dots \to A_{q_n} \to A_q$. Our goal is to build a HOWDTR$_{\mathrm{lin}}$ $T :: \Sigma_1 \to \Sigma_3$ that does the composition of T_1 and T_2, so we want to replace a rule such as that one with a new rule which corresponds to applying T_2 to the term M.

In order to do so, we need, for each o_2 tree in M, to know the associated state $\ell \in L_2$ of T_2's look-ahead, and the state $p \in P$ of T_2 which is going to process that node. So with any such tree we associate the pair (p, ℓ). In this case we call (p, ℓ) the *token* which represents the behavior of the tree. In general, we want to associate *tokens* not only with trees, but also with λ-terms of higher order. For example, we map an occurrence of a symbol $a \in \Sigma_2$ of type $o_2 \to o_2 \to o_2$, whose arguments x_1 and x_2 (of type o_2) respectively have look-ahead states ℓ_1 and ℓ_2 and are processed by states p_1 and $p_2 \in P$ of T_2, to the *token* $(p_1, \ell_1) \multimap (p_2, \ell_2) \multimap (p, \ell)$ where (p, ℓ) is the token of the tree $a\ x_1 x_2$ (of type o_2). We formally define *tokens* as follows:

▶ **Definition 4.** *The set of* semantic tokens $[\![A]\!]$ *over a type A built on atomic type o_2 is defined by induction:*

$$[\![o_2]\!] = \{(p, \ell) \mid p \in P, \ell \in L_2\} \qquad\qquad [\![A \to B]\!] = \{f \multimap g \mid f \in [\![A]\!], g \in [\![B]\!]\}$$

Naturally, the semantic token associated with a λ-term M of type A built on atomic type o_2 will depend on the context where the term M appears. For example a tree of atomic type o_2 can be processed by any state $p \in P$ of T_2, and a term of type $A \to B$ can be applied to any argument of type A. But for any such M taken out of context, there exists a finite set of possible tokens for it. For example, a given tree of type o_2 can be processed by any state $p \in P$ depending on the context, but it has always the same look-ahead $\ell \in L_2$.

In order to define the set of possible semantic tokens for a term, we use a system of derivation rules. The following derivation rules are used to derive judgments that associate a term with a semantic token. So a judgment $\Gamma \vdash M : f$ associates term M with token f, where Γ is a substitution which maps free variables in M to tokens. The rules are:

$$\frac{p(a\,\vec{x})\langle\ell_1, \dots, \ell_n\rangle \xrightarrow{T_2} M(p_1\,x_1)\dots(p_n\,x_n) \qquad\qquad \mathtt{A}_2(a\,(\ell_1, \dots, \ell_n)) = \ell}{\vdash a : (p_1, \ell_1) \multimap \dots \multimap (p_n, \ell_n) \multimap (p, \ell)}$$

$$\frac{\Gamma_1 \vdash M : f \multimap g \qquad \Gamma_2 \vdash N : f}{\Gamma_1, \Gamma_2 \vdash M\,N : g}$$

$$\frac{\Gamma, x^A : f \vdash M : g}{\Gamma \vdash \lambda x^A.M : f \multimap g} \qquad \frac{f \in [\![A]\!]}{x^A : f \vdash x^A : f}$$

Using this system we can derive, for any term M^A, all the semantic tokens that correspond to possible behaviours of M^A when it is processed by T_2.

4.2 Unicity of derivation for semantic token judgements

We will later show that we can compute the image of M from the derivation of the judgement $\vdash M : f$, assuming that f is the token that represents the behaviour of T_2 on M. But before that we need to prove that for a given term M and token f the derivation of the judgement $\vdash M : f$ is unique:

▶ **Theorem 5.** *For every type A, for every term M of type A and every token $f \in [\![A]\!]$, there is at most one derivation $\mathcal{D} :: \vdash M : f$.*

This theorem relies in part on the fact that tokens form a *coherent space*, as introduced by Girard in [13]. The full proof of this theorem is not detailed here but can be found in the full version of the article on Hal.

Now that we have shown that there is only one derivation per judgement $\vdash M : f$, we are going to see how to use that derivation in order to compute the term N that is the image of M by transducer T_2.

4.3 Collapsing of token derivations

We define a function (we call it collapsing function) which maps every derivation $\mathcal{D} :: \vdash M : f$ to a term $\overline{\mathcal{D}}$ which corresponds to the output of transducer T_2 on term M assuming that M has behaviour f.

▶ **Definition 6.** *Let $\mathcal{D}$ be a derivation. We define $\overline{\mathcal{D}}$ by induction on $\mathcal{D}$, there are different cases depending on the first rule of $\mathcal{D}$:*

If $\mathcal{D}$ is of the form:

$$\frac{p(a\,\overrightarrow{x})\langle \ell_1, \dots, \ell_n\rangle \xrightarrow{T_2} N(p_1\,x_1)\dots(p_n\,x_n) \qquad \mathtt{A}_2(a\,(\ell_1,\dots,\ell_n)) = \ell}{\vdash a : (p_1,\ell_1) \multimap \cdots \multimap (p_n,\ell_n) \multimap (p,\ell)}$$

then $\overline{\mathcal{D}} = N$,

if $\mathcal{D}$ is of the form:

$$\frac{\mathcal{D}_1 :: \Gamma_1 \vdash N_1 : f \multimap g \qquad \mathcal{D}_2 :: \Gamma_2 \vdash N_2 : f}{\Gamma_1, \Gamma_2 \vdash N_1\,N_2 : g}$$

then $\overline{\mathcal{D}} = \overline{\mathcal{D}_1}\,\overline{\mathcal{D}_2}$,

if $\mathcal{D}$ is of the form:

$$\frac{\mathcal{D}_1 :: \Gamma, x^A : f \vdash N : g}{\Gamma \vdash \lambda x^A.N : f \multimap g}$$

then $\overline{\mathcal{D}} = \lambda x.\overline{\mathcal{D}_1}$, *if $\mathcal{D}$ is of the form:*

$$\frac{f \in [\![A]\!]}{x^A : f \vdash x^A : f}$$

then $\overline{\mathcal{D}} = x^{\overline{f}}$.

We can check that, for all derivation $\mathcal{D} :: \vdash M : f$, the term $\overline{\mathcal{D}}$ is of type $\overline{f}$ given by: $\overline{(p, \ell)} = A_p$ and $\overline{f \multimap g} = \overline{f} \to \overline{g}$.

Now that we have associated, with any pair (M, f) such that f is a semantic token of term M, a term $N = \overline{\mathcal{D}}$ which represents the image of M by T_2, we need to show that replacing M with N in the computation of transducers leads to the same results.

4.4 Construction of the transducer which realizes the composition

We recall some notations: $T_1 = (\Sigma_Q, \Sigma_1, \Sigma_2, q_0, R_1, \mathtt{A}_1)$ and $T_2 = (\Sigma_P, \Sigma_2, \Sigma_3, p_0, R_2, \mathtt{A}_2)$ are two HOWDTR$_{\text{lin}}$, $Q = \{q_1, \dots, q_m\}$ is the set of states of T_1 and, for every state $q_i \in Q$, we note A_{q_i} the type of $q_i(t)$ when t is a tree of type o_1. For all type A built on o_2, the set of tokens of terms of type A is noted $[\![A]\!]$ and is finite.

Previously, we saw how to apply transducer T_2 to terms M of type A built on the atomic type o_2, so we can apply T_2 to terms which appear on the left side of rules of T_1: $q(a\,\overrightarrow{x})\langle \overrightarrow{\ell}\,\rangle \to M\,(q_{i_1}\,x_1)\dots(q_{i_n}\,x_n)$. In a rule such as this one, in order to replace term M with term $N = \overline{\mathcal{D}}$ where $\mathcal{D}$ is the unique derivation of the judgement $\vdash M : f$, we need to know which token f properly describes the behaviour of T_2 on M. The computation of that token is done in the look-ahead automaton $\mathtt{A}$ of T.

We define the set of states of $\mathtt{A}$ as: $L = L_1 \times [\![A_{q_1}]\!] \times \dots \times [\![A_{q_m}]\!]$
With any tree t (of type o_1) we want to associate the look-ahead of T_1 on t and, for each state $q_i \in Q$ of T_1, a token of $q_i(t)$. The transition function of the look-ahead automaton $\mathtt{A}$ is defined by, for all $(\ell_1, f_{1,1}, \dots, f_{1,n}), \dots, (\ell_n, f_{m,1}, \dots, f_{m,n}) \in L$:

$a\,(\ell_1, f_{1,1}, \dots, f_{1,m}) \dots (\ell_n, f_{n,1}, \dots, f_{n,m}) \xrightarrow{\mathtt{A}} (\ell, f_1, \dots, f_m)$

where $a\,\ell_1 \dots \ell_n \xrightarrow{\mathtt{A}_1} \ell$ and, for all state $q_i \in Q$, f_i is such that in T_1 there exists a rule $q_i(a\,\overrightarrow{x})\langle \ell_1, \dots, \ell_n\rangle \xrightarrow{T_1} M\,(q_{i_1}\,x_1)\dots(q_{i_n}\,x_n)$ and a derivation of the judgement $\vdash M : f_{1,i_1} \multimap \dots \multimap f_{n,i_n} \multimap f_i$. Note that this look-ahead automaton is non-deterministic in general, but the transducer is weakly deterministic in the sense that, at each step, even if several look-ahead states are possible, only one rule of the transducer can be applied.

We define the set of states Q' of transducer T by:

$Q' = \{(q, f) \mid q \in Q, f \in [\![A_q]\!]\} \cup \{q'_0\}$

Then we define the set R of rules of transducer T as the set of rules of the form:

$(q, f)(a\,\overrightarrow{x})\langle(\ell_1, f_{1,1}, \dots, f_{1,m}), \dots\rangle \xrightarrow{T} \overline{\mathcal{D}}\,((q_{i_1}, f_1)\,x_1)\dots((q_{i_n}, f_n)\,x_n)$

such that there exists in T_1 a rule: $q(a\,\overrightarrow{x})\langle \ell_1, \dots\rangle \xrightarrow{T_1} M\,(q_{i_1}\,x_1)\dots(q_{i_n}\,x_n)$ and $\mathcal{D}$ is a derivation of the judgement $\vdash M : f_{1,i_1} \multimap \dots \multimap f_{n,i_n} \multimap f$.

Because of Theorem 5 that set of rules is weakly deterministic.

To that set R we then add rules for the initial state q'_0, which simply replicate the rules of states of the form $(q_0, (p_0, \ell))$: for all $a \in \Sigma_1$, all $(\ell_1, f_{1,1}, \dots, f_{1,m}), \dots, (\ell_n, f_{n,1}, \dots, f_{n,m}) \in L$ and all rule in R of the form:

$(q_0, (p_0, l))(a\,\overrightarrow{x})\langle(\ell_1, f_{1,1}, \dots, f_{1,m}), \dots\rangle \xrightarrow{T} M\,((q_1, f_1)\,x_1)\dots((q_n, f_n)\,x_n)$

where p_0 is the initial state of T_2 and $l \in L_2$ is a state of the look-ahead automaton of T_2, we add the rule :

$q'_0(a\,\overrightarrow{x})\langle(\ell_1, f_{1,1}, \dots, f_{1,m}), \dots\rangle \xrightarrow{T} M\,((q_1, f_1)\,x_1)\dots((q_n, f_n)\,x_n)$

This set R of rules is still weakly deterministic according to Theorem 5.

We have thus defined the HOWDTR$_{\text{lin}}$ $T = (\Sigma_{Q'}, \Sigma_1, \Sigma_3, q'_0, R, \mathtt{A})$.

▶ **Theorem 7.** $T = T_2 \circ T_1$

Finally, we will analyze the complexity of this algorithm and show that using the algorithm on $\mathrm{HOWDTR}_{\mathrm{lin}}$ instead of $\mathrm{HODTR}_{\mathrm{lin}}$ avoids an exponential blow-up of the size of the produced transducer.

First the set of states Q' of T is of size $|Q'| = 1 + \Sigma_{q\in Q}|[\![A_q]\!]|$ where $|[\![A_q]\!]|$ is the number of tokens of type A_q. $|[\![A_q]\!]| = (|P|\,|L_2|)^{|A_q|}$ where $|P|$ is the number of states of transducer T_2, $|L_2|$ is the number of states of the look-ahead automaton of transducer T_2 and $|A_q|$ is the size of the type A_q. So the size of Q' is $O(\Sigma_{q\in Q}(|P|\,|L_2|)^{|A_q|})$, that is a polynomial in the size of T_2 to the power of the size of types of states of T_1.

It is important to note that the set $[\![A_q]\!]$ of tokens of type A_q is where $\mathrm{HOWDTR}_{\mathrm{lin}}$ and $\mathrm{HODTR}_{\mathrm{lin}}$ differ in their complexity: the deterministic alternative to the weakly deterministic T would require to store with the state not a single token, but a set of two-by-two coherent tokens, that would bring the size of Q' to $1 + \Sigma_{q\in Q}2^{|[\![A_q]\!]|}$ which would be exponential in the size of T_2 and doubly exponential in the size of types of T_1.

Then there is the look-ahead automaton: its set of states is $L = L_1 \times [\![A_{q_1}]\!] \times \cdots \times [\![A_{q_m}]\!]$. So the number of states is in $O(|L_1|\,(|P|\,|L_2|)^{\Sigma_{q\in Q}|A_q|})$. The size of the set of rules of the look-ahead automaton is in $O(\Sigma_{a^{(n)}\in\Sigma_1}|L|^{n+1})$ where n is the arity of the constant $a^{(n)}$.

Finally there is the set R of rules of T. For every judgement $\vdash M : f_{1,i_1} \multimap \cdots \multimap f_{n,i_n} \multimap f$, finding a derivation $\mathcal{D}$ of that judgement and computing the corresponding $\overline{\mathcal{D}}$ is in $O(|M|^2)$ time where $|M|$ is the size of M. The number of possible rules is in $O(\Sigma_{a^{(n)}\in\Sigma_1}(|Q'|)^{n+1})$. So computing R is done in time $O(|R|^2\,\Sigma_{a^{(n)}\in\Sigma_1}(|Q'|)^{n+1})$ where R is the set of rules of T_1. With a fixed input signature Σ_1, the time complexity of the algorithm computing T is a polynomial in the sizes of T_1 and T_2, with only the sizes of types of states of T_1 as exponents.

Note that, as our model generalizes other classes of transducers, it is possible to perform their composition in our setting. Thanks to results of Theorem 2, it is then possible to reduce the order of the result of the composition, and obtain a $\mathrm{HODTR}_{\mathrm{lin}}$ that can be converted back in those other models. This methods gives an important insight on the composition procedure for those other formalisms.

In comparison, the composition algorithms for equivalent classes of transductions are either not direct or very complex as they essentially perform composition and order reduction at once. For instance, composition of single used restricted MTT is obtained through MSO ([11]). High-level tree transducers [12] go through a reduction to iterated pushdown tree transducers and back. The composition algorithm for Streaming Tree Transducers described in [1] is direct, but made complex by the fact that the algorithm hides this reduction of order.

The double-exponential complexity of composition of $\mathrm{HODTR}_{\mathrm{lin}}$ compares well to the non-elementary complexity of composition in equivalent non-MSOT classes of transducers. Although the simple exponential complexity of composition in MSOT is better, we should account for the fact that the MSOT model does not attempt to represent the behavior of programs.

5 Conclusion and future work

In this paper we have presented a new mechanical characterization of Monadic Second Order Transductions. This characterization is based on simply typed λ-calculus which allows us to generalize with very few primitives most of the mechanisms used to compute the output in the transducer literature. The use of higher-order allows us to propose an arguably simple algorithm for computing the composition of linear higher-order transducers which coincide with MSOT. The correctness of this algorithm is based on denotation semantics (coherence spaces) of λ-calculus and the heart of the proof uses logical relations. Thus, the use of λ-calculus allows us to base our work on standard tools and techniques rather than developing our own tools as is often the case when dealing with transducers. Moreover, this work sheds some light on how composition is computed in other formalisms. Indeed, we argue that for MTT_{sur}, STT, or ARR_{sur}, the composition must be the application of our composition algorithm followed by the order reduction procedure that we use to prove the equivalence with logical transductions.

The notion of higher-order transducer has already been studied [12, 18, 16], however, there is still some work to be done to obtain direct composition algorithms. We plan to generalize our approach of the linear case to the general one and devise a semantic based partial evaluation for the composition of higher-order transducers.

References

1 R. Alur and L. D'Antoni. Streaming tree transducers. *J. ACM*, 64(5):31:1–31:55, 2017.

2 Roderick Bloem and Joost Engelfriet. A comparison of tree transductions defined by monadic second order logic and by attribute grammars. *J. Comput. Syst. Sci.*, 61(1):1–50, 2000. URL: `https://doi.org/10.1006/jcss.1999.1684`, `doi:10.1006/jcss.1999.1684`.

3 C. Choffrut. A generalization of Ginsburg and Rose's characterisation of g-s-m mappings. In *ICALP 79*, number 71 in LNCS, pages 88–103. SV, 1979.

4 H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. release October, 12th, 2007. URL: `http://tata.gforge.inria.fr/`.

5 B. Courcelle. Monadic second-order definable graph transductions: a survey. *Theoretical Computer Science*, 126(1):53–75, 1994.

6 B. Courcelle. Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations. In Rozenberg, editor, *Handbook of Graph Grammars*, 1997.

7 S. Eilenberg. *Automata, Languages and Machines.* Acad. Press, 1974.

8 C. C. Elgot and G. Mezei. On relations defined by generalized finite automata. *IBM J. of Res. and Dev.*, 9:88–101, 1965.

9 J. Engelfriet and S. Maneth. Macro tree transducers, attribute grammars, and mso definable tree translations. *Information and Computation*, 154(1):34 – 91, 1999.

10 J. Engelfriet and S. Maneth. The equivalence problem for deterministic MSO tree transducers is decidable. *Inf. Process. Lett.*, 100(5):206–212, 2006.

11 J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. Syst. Sci.*, 31(1):71–146, 1985.

12 Joost Engelfriet and Heiko Vogler. High level tree transducers and iterated pushdown tree transducers. *Acta Informatica*, 26(1):131–192, Oct 1988. URL: `https://doi.org/10.1007/BF02915449`, `doi:10.1007/BF02915449`.

13 J. Y. Girard. Linear logic. *TCS*, 50:1–102, 1987.

14 M Kanazawa and R Yoshinaka. Distributional learning and context/substructure enumerability in nonlinear tree grammars. In *Formal Grammar*, pages 94–111. Springer, 2016.

15 Makoto Kanazawa. *Almost affine lambda terms.* National Institute of Informatics, 2012.

16 Naoki Kobayashi, Naoshi Tabuchi, and Hiroshi Unno. Higher-order multi-parameter tree transducers and recursion schemes for program verification. *SIGPLAN Not.*, 45(1):495–508, January 2010. URL: `http://doi.acm.org/10.1145/1707801.1706355`, `doi:10.1145/1707801.1706355`.

17 J. Thatcher and J. Wright. Generalized Finite Automata Theory With an Application to a Decision Problem of Second-Order Logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

18 Akihiko Tozawa. Xml type checking using high-level tree transducer. In Masami Hagiya and Philip Wadler, editors, *Functional and Logic Programming*, pages 81–96, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.