



**HAL**  
open science

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

Paul D Gallot, Aurélien Lemay, Sylvain Salvati

► **To cite this version:**

Paul D 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.science/hal-02902853v2>**

Submitted on 18 Sep 2020

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# 1 Linear High-Order Deterministic Tree 2 transducers with Regular look-ahead

3 **Paul D. Gallot**

4 INRIA, Université de Lille

5 paul.gallot@inria.fr

6 **Aurélien Lemay**

7 Université de Lille, INRIA, CNRS

8 aurelien.lemay@univ-lille.fr

9 **Sylvain Salvati**

10 Université de Lille, INRIA, CNRS

11 sylvain.salvati@univ-lille.fr

## 12 — Abstract —

---

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

29 **2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Transducers; Theory of computation  $\rightarrow$   
30 Lambda calculus; Theory of computation  $\rightarrow$  Tree languages

31 **Keywords and phrases** Transducers,  $\lambda$ -calculus, Trees

32 **Digital Object Identifier** 10.4230/LIPIcs.MFCS.2020.34

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

35 **Funding** *Paul D. Gallot*: ANR-15-CE25-0001 – Colis

36 *Aurélien Lemay*: ANR-15-CE25-0001 – Colis

37 *Sylvain Salvati*: ANR-15-CE25-0001 – Colis



© Paul Gallot, Aurélien Lemay and Sylvain Salvati;  
licensed under Creative Commons License CC-BY

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

38 **1** Introduction

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

45 Logic offers another, more descriptive, view on tree transformations. In particular,  
 46 Monadic Second Order (MSO) logic defines a class of tree transformations (MSOT) [5, 6] which  
 47 is expressive and is closed under composition. It coincides with the class of transformations  
 48 definable with MTT enhanced with a regular look-ahead and restricted to finite copying  
 49 [9, 10], and also with the class of STT [1].

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

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

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

73 The relationship of HODTR<sub>lin</sub> to MSOT is made via a transformation that *reduces the*  
 74 *order* of transducers. We indeed prove that for any HODTR<sub>lin</sub>, there exists an equivalent  
 75 HODTR<sub>lin</sub> whose order is at most 3. This transformation allows us to prove then that  
 76 HODTR<sub>lin</sub> are equivalent to Attribute Tree Transducers with the single use restriction  
 77 (ATT<sub>sur</sub>). In turn, this shows that HODTR<sub>lin</sub> are equivalent to MSOT [2].

78 One of the main interests of HODTR<sub>lin</sub> is that  $\lambda$ -calculus also offers a simple composition  
 79 algorithm. This approach gives an efficient procedure for composing two HODTR<sub>lin</sub>. In  
 80 general, this procedure raises the order of the produced transducer. In comparison, com-  
 81 position in other equivalent classes are either complex or indirect (through MSOT). In any  
 82 case, our procedure has a better complexity. Indeed, it benefits from higher-order which  
 83 permits a larger number of implementations for a given transduction. The complexity of the  
 84 construction is also lowered by the use of a notion of determinism slightly more liberal than

85 usual that we call *weak determinism*.

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

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

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

## 108 2 Definitions

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

### 111 2.1 $\lambda$ -calculus

112 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  
113 types that are either an atomic type, i.e. an element of  $\mathcal{A}$ , or a functional type  $(A \rightarrow B)$ , with  
114  $A$  and  $B$  being in  $\text{types}(\mathcal{A})$ . The operator  $\rightarrow$  is right-associative and  $A_1 \rightarrow \dots \rightarrow A_n \rightarrow B$   
115 denotes the type  $(A_1 \rightarrow (\dots \rightarrow (A_n \rightarrow B) \dots))$ . The order of a type  $A$  is inductively defined  
116 by  $\text{order}(A) = 0$  when  $A \in \mathcal{A}$ , and  $\text{order}(A \rightarrow B) = \max(\text{order}(A) + 1, \text{order}(B))$ .

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

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

## 34:4 Linear High-Order Deterministic Tree transducers with Regular look-ahead

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

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

- 135 ■ if  $c^A$  is in  $\Sigma$ , then  $c^A$  is in  $\Lambda^A(\Sigma)$ ,
- 136 ■  $x^A$  is in  $\Lambda^A(\Sigma)$ ,
- 137 ■ if  $A = B \rightarrow C$  and  $M$  is in  $\Lambda^C(\Sigma)$ , then  $(\lambda x^B.M)$  is in  $\Lambda^A(\Sigma)$ ,
- 138 ■ if  $M$  is in  $\Lambda^{B \rightarrow A}(\Sigma)$  and  $N$  is in  $\Lambda^B(\Sigma)$ , then  $(MN)$  is in  $\Lambda^A(\Sigma)$ .

139 The term  $M$  is a *pure*  $\lambda$ -term if it does not contain any constant  $c^A$  from  $\Sigma$ . When the type  
 140 is irrelevant we write  $M \in \Lambda(\Sigma)$  instead of  $M \in \Lambda^A(\Sigma)$ . We drop parentheses when it does  
 141 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  
 142  $M_0 M_1 \dots M_n$  for  $((\dots(M_0 M_1)\dots)M_n)$ .

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

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

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

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

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

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

169 For example, given a tree signature  $\Sigma_1$  with one atomic type  $o$  and two constants  $f$  of type  
 170  $o^2 \rightarrow o$  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  
 171  $o$  is linear because each variable ( $y_1$ ,  $y_2$  and  $x$ ) occurs exactly once in  $M$ . The term  $M$  contains  
 172 a  $\beta$ -redex so:  $(\lambda y_1 y_2. f y_1 (f a y_2)) a (f x a) \rightarrow_\beta (\lambda y_2. f a (f a y_2)) (f x a) \rightarrow_\beta f a (f a (f x a))$ .  
 173 The term  $f a (f a (f x a))$  has no  $\beta$ -redex so it is the  $\beta$ -normal form of  $M$ .

174 Another example: the term  $M_2 = (\lambda y. f y y) (x a)$  with free variable  $x$  of type  $o \rightarrow o$  is  
 175 syntactically almost linear because the variable  $y$  which occurs twice in the term is of the  
 176 atomic type  $o$ . It  $\beta$ -reduces to the term  $M'_2 = f (x a) (x a)$  which is not syntactically almost  
 177 linear, so  $\beta$ -reduction does not preserve syntactical almost linearity.

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

## 180 2.2 Tree Automata

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

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

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

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

## 194 2.3 High-Order Deterministic top-down tree Transducers

195 From now on we assume that  $\Sigma_i$  is a tree signature for every number  $i$  and that its atomic  
196 type is  $o_i$ .

197 A Linear High-Order Deterministic top-down Transducer with Regular look-ahead

198 (HODTR<sub>lin</sub>)  $T$  is a tuple  $(\Sigma_Q, \Sigma_1, \Sigma_2, q_0, R, \mathbf{A})$  where:

- 199 ■  $\Sigma_1 = (C_1, \{o_1\}, \tau_1)$  is a first-order tree signature, the *input signature*,
- 200 ■  $\Sigma_2 = (C_2, \{o_2\}, \tau_2)$  is a first-order tree signature, the *output signature*,
- 201 ■  $\Sigma_Q = (Q, \{o_1, o_2\}, \tau_s)$  is the *state signature*, and is such that for every  $q \in Q$ ,  $\tau_s(q)$  is of  
202 the form  $o_1 \rightarrow A_q$  where  $A_q$  is in  $\text{types}(\Sigma_2)$ . Constants of  $Q$  are called *states*,
- 203 ■  $q_0 \in Q$  is the *initial state*,
- 204 ■  $\mathbf{A}$  is a BOT over the tree signature  $\Sigma_1$ , the *look-ahead* automaton, with set of states  $P$ ,
- 205 ■  $R$  is a finite set of rules of the form  
206  $q(a \vec{x}) \langle \vec{p} \rangle \rightarrow M(q_1 x_1) \dots (q_n x_n)$   
207 where:  
208
  - 209 ■  $q, q_1, \dots, q_n \in Q$  are states of  $\Sigma_Q$ ,
  - 210 ■  $a$  is a constant of  $\Sigma_1$  with type  $o_1^n \rightarrow o_1$ ,
  - 211 ■  $\vec{x} = x_1, \dots, x_n$  are variables of type  $o_1$ , they are the child trees of the root labeled  $a$ ,
  - 212 ■  $\vec{p} = p_1, \dots, p_n$  are in  $P$  (the set of states of the look-ahead  $\mathbf{A}$ ),
  - 213 ■  $M$  is a linear term of type  $A_{q_1} \rightarrow \dots \rightarrow A_{q_n} \rightarrow A_q$  built on signature  $\Sigma_2 + \Sigma_Q$ .
  - 214 ■ there is one rule per possible left-hand side (determinism).

215 Notice that we have given states a type of the form  $o_1 \rightarrow A$  where  $A \in \text{types}(o_2)$ . The  
216 reason why we do this is to have a uniform notation. Indeed, a state  $q$  is meant to transform,  
217 thanks to the rules in  $R$ , a tree built in  $\Sigma_1$  into a  $\lambda$ -term built on  $\Sigma_2$  with type  $A_q$ . So  
218 we simply write  $q M N_1 \dots N_n$  when we want to transform  $M$  with the state  $q$  and pass  
219  $N_1, \dots, N_n$  as arguments to the result of the transformation. We write  $\Sigma_T$  for the signature  
220  $\Sigma_1 + \Sigma_2 + \Sigma_Q$ . Notice also that the right-hand part of a rule is a term that is built only  
221 with constants of  $\Sigma_2$ , states from  $\Sigma_Q$  and variables of type  $o_1$ . Thus, in order for this  
222 term to have a type in  $\text{types}(\Sigma_2)$ , it is necessary that the variables of type  $o_1$  only occur as

223 the first argument of a state in  $\Sigma_Q$ . Finally, remark that we did not put any requirement  
 224 on the type of the initial state. So as to restrict our attention to transducers as they are  
 225 usually understood, it suffices to add the requirement that the initial state is of type  $o_1 \rightarrow o_2$ .  
 226 However, we consider as well that transducers may produce *programs* instead of first order  
 227 terms.

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

235 Almost linear high-order deterministic top-down transducer with regular look-ahead  
 236 (HODTR<sub>al</sub>) are defined similarly, with the distinction that a term  $M$  appearing as a right-  
 237 hand side of a rule should be almost linear.

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

252 Given a HODTR<sub>lin</sub>, a HODTR<sub>al</sub> or a HOWDTR<sub>lin</sub>  $T$ , we write  $T :: \Sigma_1 \longrightarrow \Sigma_2$  to mean  
 253 that the input signature of  $T$  is  $\Sigma_1$  and its output signature is  $\Sigma_2$ .

254 A transducer  $T$  induces a notion of reduction on terms. A  $T$ -redex is a term of the form  
 255  $q(a M_1 \dots M_n)$  if and only if  $q(a x_1 \dots x_n) \langle p_1, \dots, p_n \rangle \rightarrow M[x_1, \dots, x_n]$  is a rule of  $T$  and  
 256 (the  $\beta$ -normal forms of)  $M_1, \dots, M_n$  are respectively accepted by  $\mathbf{A}$  with the states  $p_1, \dots, p_n$ .  
 257 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  
 258 are typed terms and that they have the same type as their corresponding  $T$ -redices. The  
 259 relation of  $T$ -contraction relates a term  $M$  and a term  $M'$  when  $M'$  is obtained from  $M$   
 260 by replacing one of its  $T$ -redex with a corresponding  $T$ -contractum. We write  $M \rightarrow_T M'$   
 261 when  $M$   $T$ -contracts to  $M'$ . The relation of  $\beta$ -reduction is confluent, and so is the relation  
 262 of  $T$ -reduction as transducers are deterministic, moreover, the union of the two relations is  
 263 terminating. It is not hard to prove that it is also locally confluent and thus confluent. It  
 264 follows that  $\rightarrow_{\beta, T}$  (which is the union of  $\rightarrow_{\beta}$  and  $\rightarrow_T$ ) is confluent and strongly normalizing.  
 265 Given a term  $M$  built on  $\Sigma_T$ , we write  $|M|_T$  to denote its normal form modulo  $=_{\beta, T}$ .

266 Then we write  $\text{rel}(T)$  for the relation:

$$267 \{(M, |q_0 M|_T) \mid M \text{ is a closed term of type } o_1 \text{ and } |q_0 M|_T \in \Lambda(\Sigma_2)\}.$$

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

270 Given a finite set of trees  $L_1$  on  $\Sigma_1$  and  $L_2$  included in  $\Lambda^{A_{q_0}}$ , we respectively write  $T(L_1)$   
 271 and  $T^{-1}(L_2)$  for the image of  $L_1$  by  $T$  and the inverse image of  $L_2$  by  $T$ .

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

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

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

283 ■  $q_0(Z) \rightarrow O$ ,  $q_0(Sx) \rightarrow N(q_i x O)$ ,

284 ■  $q_0(\text{add } xy) \rightarrow q_i x (q_i y O)$ ,

285 ■  $q_i(Z) \rightarrow \lambda x.x$ ,

286 ■  $q_i(Sx) \rightarrow \lambda y.N(q_i x y)$ ,

287 ■  $q_i(\text{add } xy) \rightarrow \lambda z.q_i x (q_i y z)$ ,

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

$$\begin{aligned} & q_0(\text{add}(S(SZ))(S(S(SZ)))) \rightarrow_T (q_i(S(SZ)))(q_i(S(S(SZ)))O) \\ 289 & \xrightarrow{*}_T (\lambda y_1.N((\lambda y_2.N((\lambda x.x)y_2))y_1))((\lambda y_3.N((\lambda y_4.N((\lambda y_5.N((\lambda x.x)y_5))y_4))y_3))O) \\ & \xrightarrow{*}_{\beta} N(N(N(N(O)))) \end{aligned}$$

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

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

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

### 309 **3 Order reduction and expressiveness**

310 In this section we outline a construction that transforms a transducer of  $\text{HODTR}_{\text{lin}}$  or  
 311  $\text{HODTR}_{\text{al}}$  into an equivalent linear or almost linear transducer of order  $\leq 3$ . These two  
 312 constructions are similar and central to proving that  $\text{HODTR}_{\text{lin}}$  and  $\text{HODTR}_{\text{al}}$  are respect-  
 313 ively equivalent to Monadic Second Order Transductions from trees to trees (MSOT) and to  
 314 Monadic Second Order Transductions from trees to terms (i.e. trees with sharing) (MSOTS).  
 315 We will later show that there are translations between  $\text{HODTR}_{\text{lin}}$  of order 3 and attribute tree

316 transducers with the *single use restriction* and between  $\text{HODTR}_{\text{al}}$  of order 3 and attribute  
 317 tree transducers. These two models are known to be respectively equivalent to MSOT and  
 318 MSOTS [2].

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

- 321 ■  $M =_{\beta} M'.\sigma$ ,
- 322 ■ the free variables of  $M'$  have at most order 1,
- 323 ■ for every variable  $x$ ,  $\sigma(x)$  is a closed  $\lambda$ -term,
- 324 ■ the number of free variables in  $M'$  is minimal.

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

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

342 ► **Theorem 1.** *For all type  $A$  built on tree signature  $\Sigma$ , the set of templates of closed linear*  
 343 *(or almost linear) terms of type  $A$  is finite.*

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

352 ► **Theorem 2.** *Any  $\text{HODTR}_{\text{lin}}$  (resp.  $\text{HODTR}_{\text{al}}$ ) has an equivalent  $\text{HODTR}_{\text{lin}}$  (resp.*  
 353  *$\text{HODTR}_{\text{al}}$ ) of order 3.*

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

360 ► **Theorem 3.**  *$\text{HODTR}_{\text{lin}}$  are equivalent to MSOT and  $\text{HODTR}_{\text{al}}$  are equivalent to MSOTS.*

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

## 366 4 Composition of $\text{HODTR}_{\text{lin}}$

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

### 371 4.1 Semantic analysis

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

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

388 ► **Definition 4.** *The set of semantic tokens  $\llbracket A \rrbracket$  over a type  $A$  built on atomic type  $o_2$  is*  
 389 *defined by induction:*

$$390 \quad \llbracket o_2 \rrbracket = \{(p, \ell) \mid p \in P, \ell \in L_2\} \quad \llbracket A \rightarrow B \rrbracket = \{f \multimap g \mid f \in \llbracket A \rrbracket, g \in \llbracket B \rrbracket\}$$

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

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

$$401 \quad \frac{p(a \vec{x}) \langle \ell_1, \dots, \ell_n \rangle \xrightarrow{T_2} M(p_1 x_1) \dots (p_n x_n) \quad \mathbf{A}_2(a(\ell_1, \dots, \ell_n)) = \ell}{\vdash a : (p_1, \ell_1) \multimap \dots \multimap (p_n, \ell_n) \multimap (p, \ell)}$$

402

$$403 \quad \frac{\Gamma_1 \vdash M : f \multimap g \quad \Gamma_2 \vdash N : f}{\Gamma_1, \Gamma_2 \vdash MN : g}$$

$$404 \quad \frac{\Gamma, x^A : f \vdash M : g}{\Gamma \vdash \lambda x^A. M : f \multimap g} \quad 405 \quad \frac{f \in \llbracket A \rrbracket}{x^A : f \vdash x^A : f}$$

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

## 408 4.2 Unicity of derivation for semantic token judgements

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

413 ► **Theorem 5.** *For every type  $A$ , for every term  $M$  of type  $A$  and every token  $f \in \llbracket A \rrbracket$ , there  
414 is at most one derivation  $\mathcal{D} :: \vdash M : f$ .*

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

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

## 421 4.3 Collapsing of token derivations

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

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

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

$$428 \quad \frac{p(a \vec{x}) \langle \ell_1, \dots, \ell_n \rangle \xrightarrow{T_2} N(p_1 x_1) \dots (p_n x_n) \quad \mathbf{A}_2(a(\ell_1, \dots, \ell_n)) = \ell}{\vdash a : (p_1, \ell_1) \multimap \dots \multimap (p_n, \ell_n) \multimap (p, \ell)}$$

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

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

$$431 \quad \frac{\mathcal{D}_1 :: \Gamma_1 \vdash N_1 : f \multimap g \quad \mathcal{D}_2 :: \Gamma_2 \vdash N_2 : f}{\Gamma_1, \Gamma_2 \vdash N_1 N_2 : g}$$

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

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

$$434 \quad \frac{\mathcal{D}_1 :: \Gamma, x^A : f \vdash N : g}{\Gamma \vdash \lambda x^A. N : f \multimap g}$$

435 *then  $\overline{\mathcal{D}} = \lambda x. \overline{\mathcal{D}_1}$ ,*

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

$$437 \frac{f \in \llbracket A \rrbracket}{x^A : f \vdash x^A : f}$$

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

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

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

#### 444 4.4 Construction of the transducer which realizes the composition

445 We recall some notations:  $T_1 = (\Sigma_Q, \Sigma_1, \Sigma_2, q_0, R_1, A_1)$  and  $T_2 = (\Sigma_P, \Sigma_2, \Sigma_3, p_0, R_2, A_2)$  are  
446 two HOWDTR<sub>lin</sub>,  $Q = \{q_1, \dots, q_m\}$  is the set of states of  $T_1$  and, for every state  $q_i \in Q$ , we  
447 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  
448 tokens of terms of type  $A$  is noted  $\llbracket A \rrbracket$  and is finite.

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

455 We define the set of states of  $A$  as:  $L = L_1 \times \llbracket A_{q_1} \rrbracket \times \dots \times \llbracket A_{q_m} \rrbracket$

456 With any tree  $t$  (of type  $o_1$ ) we want to associate the look-ahead of  $T_1$  on  $t$  and, for each  
457 state  $q_i \in Q$  of  $T_1$ , a token of  $q_i(t)$ . The transition function of the look-ahead automaton  $A$   
458 is defined by, for all  $(\ell_1, f_{1,1}, \dots, f_{1,m}), \dots, (\ell_n, f_{n,1}, \dots, f_{n,m}) \in L$ :

$$459 a(\ell_1, f_{1,1}, \dots, f_{1,m}) \dots (\ell_n, f_{n,1}, \dots, f_{n,m}) \xrightarrow{A} (\ell, f_1, \dots, f_m)$$

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

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

$$466 Q' = \{(q, f) \mid q \in Q, f \in \llbracket A_q \rrbracket\} \cup \{q'_0\}$$

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

$$468 (q, f)(a \overrightarrow{x}) \langle (\ell_1, f_{1,1}, \dots, f_{1,m}), \dots \rangle \xrightarrow{T} \overline{\mathcal{D}}((q_{i_1}, f_1) x_1) \dots ((q_{i_n}, f_n) x_n)$$

469 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  
470 derivation of the judgement  $\vdash M : f_{1,i_1} \multimap \dots \multimap f_{n,i_n} \multimap f$ .

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

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

$$475 (q_0, (p_0, \ell))(a \overrightarrow{x}) \langle (\ell_1, f_{1,1}, \dots, f_{1,m}), \dots \rangle \xrightarrow{T} M((q_1, f_1) x_1) \dots ((q_n, f_n) x_n)$$

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

$$478 q'_0(a \overrightarrow{x}) \langle (\ell_1, f_{1,1}, \dots, f_{1,m}), \dots \rangle \xrightarrow{T} M((q_1, f_1) x_1) \dots ((q_n, f_n) x_n)$$

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

480 We have thus defined the HOWDTR<sub>lin</sub>  $T = (\Sigma_{Q'}, \Sigma_1, \Sigma_3, q'_0, R, A)$ .

481 ► **Theorem 7.**  $T = T_2 \circ T_1$

482 Finally, we will analyze the complexity of this algorithm and show that using the  
483 algorithm on HOWDTR<sub>lin</sub> instead of HODTR<sub>lin</sub> avoids an exponential blow-up of the size  
484 of the produced transducer.

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

490 It is important to note that the set  $\llbracket A_q \rrbracket$  of tokens of type  $A_q$  is where HOWDTR<sub>lin</sub> and  
491 HODTR<sub>lin</sub> differ in their complexity: the deterministic alternative to the weakly deterministic  
492  $T$  would require to store with the state not a single token, but a set of two-by-two coherent  
493 tokens, that would bring the size of  $Q'$  to  $1 + \sum_{q \in Q} 2^{\llbracket A_q \rrbracket}$  which would be exponential in the  
494 size of  $T_2$  and doubly exponential in the size of types of  $T_1$ .

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

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

504 Note that, as our model generalizes other classes of transducers, it is possible to perform  
505 their composition in our setting. Thanks to results of Theorem 2, it is then possible to reduce  
506 the order of the result of the composition, and obtain a HODTR<sub>lin</sub> that can be converted  
507 back in those other models. This methods gives an important insight on the composition  
508 procedure for those other formalisms.

509 In comparison, the composition algorithms for equivalent classes of transductions are  
510 either not direct or very complex as they essentially perform composition and order reduction  
511 at once. For instance, composition of single used restricted MTT is obtained through MSO  
512 ([11]). High-level tree transducers [12] go through a reduction to iterated pushdown tree  
513 transducers and back. The composition algorithm for Streaming Tree Transducers described  
514 in [1] is direct, but made complex by the fact that the algorithm hides this reduction of order.

515 The double-exponential complexity of composition of HODTR<sub>lin</sub> compares well to the  
516 non-elementary complexity of composition in equivalent non-MSOT classes of transducers.  
517 Although the simple exponential complexity of composition in MSOT is better, we should  
518 account for the fact that the MSOT model does not attempt to represent the behavior of  
519 programs.

## 520 **5 Conclusion and future work**

521 In this paper we have presented a new mechanical characterization of Monadic Second Order  
522 Transductions. This characterization is based on simply typed  $\lambda$ -calculus which allows us to  
523 generalize with very few primitives most of the mechanisms used to compute the output in  
524 the transducer literature. The use of higher-order allows us to propose an arguably simple  
525 algorithm for computing the composition of linear higher-order transducers which coincide  
526 with MSOT. The correctness of this algorithm is based on denotation semantics (coherence

spaces) of  $\lambda$ -calculus and the heart of the proof uses logical relations. Thus, the use of  $\lambda$ -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.

## 538 — References —

- 539 1 R. Alur and L. D'Antoni. Streaming tree transducers. *J. ACM*, 64(5):31:1–31:55, 2017.
- 540 2 Roderick Bloem and Joost Engelfriet. A comparison of tree transductions defined by monadic  
541 second order logic and by attribute grammars. *J. Comput. Syst. Sci.*, 61(1):1–50, 2000. URL:  
542 <https://doi.org/10.1006/jcss.1999.1684>, doi:10.1006/jcss.1999.1684.
- 543 3 C. Choffrut. A generalization of Ginsburg and Rose's characterisation of g-s-m mappings. In  
544 *ICALP 79*, number 71 in LNCS, pages 88–103. SV, 1979.
- 545 4 H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and  
546 M. Tommasi. Tree Automata Techniques and Applications. release October, 12th, 2007. URL:  
547 <http://tata.gforge.inria.fr/>.
- 548 5 B. Courcelle. Monadic second-order definable graph transductions: a survey. *Theoretical*  
549 *Computer Science*, 126(1):53–75, 1994.
- 550 6 B. Courcelle. Handbook of Graph Grammars and Computing by Graph Transformations,  
551 Volume 1: Foundations. In Rozenberg, editor, *Handbook of Graph Grammars*, 1997.
- 552 7 S. Eilenberg. *Automata, Languages and Machines*. Acad. Press, 1974.
- 553 8 C. C. Elgot and G. Mezei. On relations defined by generalized finite automata. *IBM J. of*  
554 *Res. and Dev.*, 9:88–101, 1965.
- 555 9 J. Engelfriet and S. Maneth. Macro tree transducers, attribute grammars, and mso definable  
556 tree translations. *Information and Computation*, 154(1):34 – 91, 1999.
- 557 10 J. Engelfriet and S. Maneth. The equivalence problem for deterministic MSO tree transducers  
558 is decidable. *Inf. Process. Lett.*, 100(5):206–212, 2006.
- 559 11 J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. Syst. Sci.*, 31(1):71–146, 1985.
- 560 12 Joost Engelfriet and Heiko Vogler. High level tree transducers and iterated pushdown tree  
561 transducers. *Acta Informatica*, 26(1):131–192, Oct 1988. URL: [https://doi.org/10.1007/](https://doi.org/10.1007/BF02915449)  
562 [BF02915449](https://doi.org/10.1007/BF02915449), doi:10.1007/BF02915449.
- 563 13 J. Y. Girard. Linear logic. *TCS*, 50:1–102, 1987.
- 564 14 M Kanazawa and R Yoshinaka. Distributional learning and context/substructure enumerability  
565 in nonlinear tree grammars. In *Formal Grammar*, pages 94–111. Springer, 2016.
- 566 15 Makoto Kanazawa. *Almost affine lambda terms*. National Institute of Informatics, 2012.
- 567 16 Naoki Kobayashi, Naoshi Tabuchi, and Hiroshi Unno. Higher-order multi-parameter tree  
568 transducers and recursion schemes for program verification. *SIGPLAN Not.*, 45(1):495–  
569 508, January 2010. URL: <http://doi.acm.org/10.1145/1707801.1706355>, doi:10.1145/  
570 1707801.1706355.
- 571 17 J. Thatcher and J. Wright. Generalized Finite Automata Theory With an Application to a  
572 Decision Problem of Second-Order Logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.
- 573 18 Akihiko Tozawa. Xml type checking using high-level tree transducer. In Masami Hagiya and  
574 Philip Wadler, editors, *Functional and Logic Programming*, pages 81–96, Berlin, Heidelberg,  
575 2006. Springer Berlin Heidelberg.