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

Paul Gallot, Aurélien Lemay, Sylvain Salvati

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 single-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 ($\text{HODTR}_{\text{lin}}$), 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 $\text{HODTR}_{\text{lin}}$ can always be bounded by 3, and give a procedure that reduces the order of a $\text{HODTR}_{\text{lin}}$ to 3. As those resulting $\text{HODTR}_{\text{lin}}$ 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

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

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

One of the main interests of $\text{HODTR}_{\text{lin}}$ is that λ-calculus also offers a simple composition algorithm. This approach gives an efficient procedure for composing two $\text{HODTR}_{\text{lin}}$. 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 $\mathrm{HODTR}_{\mathrm{al}}$ 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 $\mathcal{A}$, $\mathrm{types}(\mathcal{A})$, 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 $\mathrm{types}(\mathcal{A})$. 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 $\mathrm{order}(A) = 0$ when $A \in \mathcal{A}$, and $\mathrm{order}(A \to B) = \max(\mathrm{order}(A) + 1, \mathrm{order}(B))$.

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 $\mathrm{types}(\mathcal{A})$, *the typing function*.

We allow ourselves to write $\mathrm{types}(\Sigma)$ to refer to the set $\mathrm{types}(\mathcal{A})$. The order of a signature is the maximal order of a type assigned to a constant (i.e. $\max\{\mathrm{order}(\tau(c)) \mid c \in C\}$). 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 $\Lambda(\Sigma) = (\Lambda^A(\Sigma))_{A \in \mathrm{types}(\Sigma)}$, as the smallest family indexed by $\mathrm{types}(\Sigma)$ 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 $\mathrm{fv}(M)$ of free variables of a term M is inductively defined on the structure of M:

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

Terms which have no free variables are called *closed*. We write $M[x_1, \dots, x_k]$ to emphasize that $\mathrm{fv}(M)$ 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 $x_1, \dots, x_k$ by the terms $N_1, \dots, N_k$. 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 $\overset{*}{\to}_\beta$, 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 $t_1, \dots, t_n$ 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 $o \to \dots \to o \to o$ 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 [16]).

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 ($\mathrm{HODTR}_{\mathrm{lin}}$) 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
  $$q(a\,\overrightarrow{x})\langle\overrightarrow{p}\rangle \to M(q_1 x_1)\dots(q_n x_n)$$
  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 $A \in \mathrm{types}(o_2)$. 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 $\mathrm{add}(S(S\,Z))(S(S(S\,Z)))$:

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

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 [15]. 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 proof that $\mathrm{HODTR}_{\mathrm{lin}}$ 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 $\mathrm{HODTR}_{\mathrm{al}}$ are equivalent to MSOTS.

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

As we are interested in limiting the size of the transducer that is computed, and even though our primary goal is to compose $\mathrm{HODTR}_{\mathrm{lin}}$, this section is devoted to the composition of $\mathrm{HOWDTR}_{\mathrm{lin}}$. 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: $q(a\,\overrightarrow{x})\langle\overrightarrow{\ell}\rangle \to M\,(q_1\ x_1)\dots(q_n\ x_n)$ where $q, q_1, \dots, q_n \in Q$ are states of T_1, $\overrightarrow{\ell} = \ell_1, \dots, \ell_n$ 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 $\mathrm{HOWDTR}_{\mathrm{lin}}$ $T :: \Sigma_1 \to \Sigma_3$ 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:*

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

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:

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

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

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

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 [14], the proof is detailed in the appendix.

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:

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

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 $\mathrm{HOWDTR}_{\mathrm{lin}}$, $Q = \{q_1, \dots, q_m\}$ 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: $L = L_1 \times [\![A_{q_1}]\!] \times \cdots \times [\![A_{q_m}]\!]$
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 $\vdash M : f_{1,i_1} \multimap \cdots \multimap f_{n,i_n} \multimap f_i$. 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 $\vdash M : f_{1,i_1} \multimap \cdots \multimap f_{n,i_n} \multimap f$.

Because of Theorem 5 proved in the appendix, 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 HOWDTR$_{\text{lin}}$ instead of HODTR$_{\text{lin}}$ 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 HOWDTR$_{\text{lin}}$ and HODTR$_{\text{lin}}$ 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, 19, 17], 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. 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. Z. Fulop. On attributed tree transducers. *Acta Cybernet.*, 5:261–279, 1981.
14. J. Y. Girard. Linear logic. *TCS*, 50:1–102, 1987.
15. M Kanazawa and R Yoshinaka. Distributional learning and context/substructure enumerability in nonlinear tree grammars. In *Formal Grammar*, pages 94–111. Springer, 2016.
16. Makoto Kanazawa. *Almost affine lambda terms*. National Institute of Informatics, 2012.
17. 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`.
18. 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.
19. 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.

A Order reduction

A.1 Templates

A.1.1 Linear templates

Proof of Theorem 1

In order to show that the set of linear templates of a given type A is finite, we use notions and properties defined in [16]: the definitions of positive and negative subtype occurrences and subpremises in A and what it entails in the structure of terms of type A.

For any type A, we can label occurrences of subtypes in A as *positive* or *negative* using the following rules:

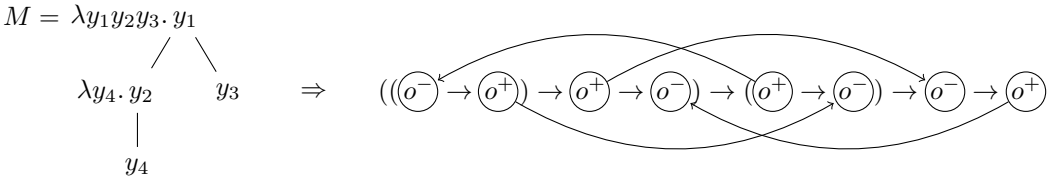
- A is positive, we note it A^+,
- if $B \to C$ is a positive subtype of A then B is negative and C is positive, we note it $(B^- \to C^+)^+$,
- if $B \to C$ is a negative subtype of A then B is positive and C is negative, we note it $(B^+ \to C^-)^-$.

For example, if $A = ((o \to o) \to (o \to o)) \to ((o \to o) \to (o \to o))$ is a type built on the atomic tree type o, then we can label occurrences of subtypes of A as follows: $((o^- \to o^+)^+ \to (o^+ \to o^-)^-)^- \to ((o^+ \to o^-)^- \to (o^- \to o^+)^+)^+$.

So, for all subtype occurrence $A' = A_1 \to \dots A_n \to o$, if A' is positive then $A_1^- \to \dots A_n^- \to o^+$, if A' is negative then $A_1^+ \to \dots A_n^+ \to o^-$.

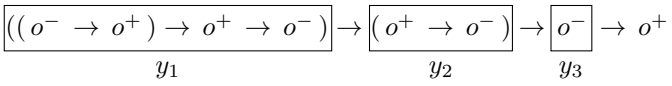
With any closed linear term M in β-normal form of type A we associate a bijection from the set of positive occurrences of the atomic type o in A to the set of negative occurrences of the atomic type o in A, we call it the *trace* of M and note it $\Theta(M)$.

We show how to compute $\Theta(M)$ on an example. To a term $M = \lambda y_1 y_2 y_3 . y_1\ (\lambda y_4 . y_2\ y_4)\ y_3$ of type $A = ((o^- \to o^+) \to o^+ \to o^-) \to (o^+ \to o^-) \to o^- \to o^+$ we have:

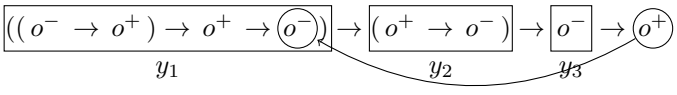


The trace is computed by induction on M:

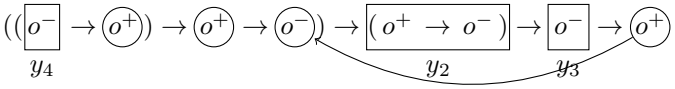
First M introduces y_1,y_2 and y_3:



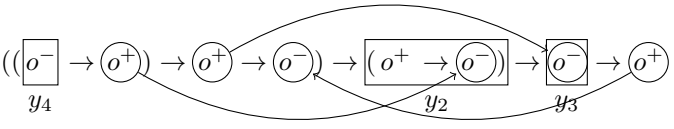
Then, because y_1 is the head variable of M, the output type of M corresponds to the output type of y_1:



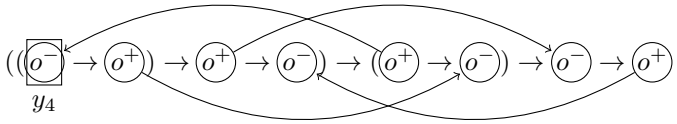
Then in the arguments of y_1 we introduce y_4 and we have two terms of type o^+ to match with output types o^- of variables:



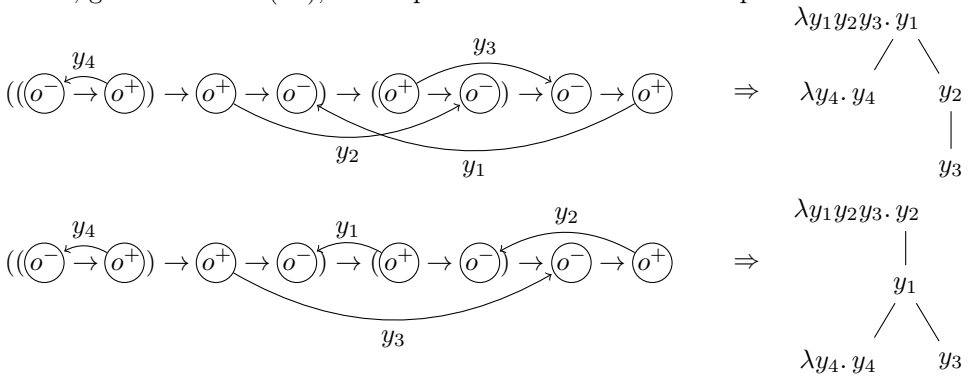
Those are mapped to y_2 and y_3:



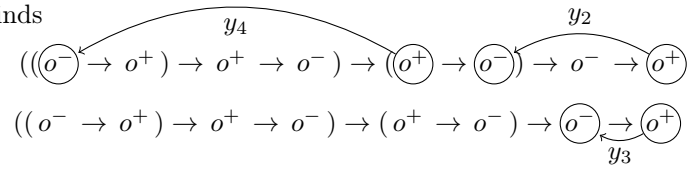
Finally the argument of y_2 is y_4:



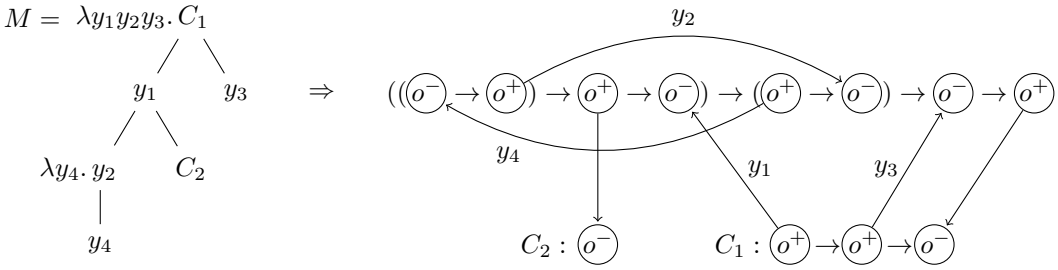
This is how we compute the trace of a linear term in linear normal form. The function which associates a trace with any linear term in linear normal form is injective, and it is possible, given a trace $\Theta(M)$, to compute the term M. For example:



However injective, the Θ function is not surjective in general, meaning there are bijections from positive to negative atomic subtype occurrences that do not correspond to any term. For example, for type $A = ((o \to o) \to o \to o) \to (o \to o) \to o \to o$, there are only 3 terms in linear normal form of type A, and only 3 corresponding traces (the three examples we have shown so far). Any other bijection between positive and negative atomic subtype occurrences is not a trace either because it binds variable y_4 outside of its scope: or because some variable would not appear in the term:



The consequence of this is that the number of closed linear terms in linear normal form of a given type A is bounded by the number of bijections between A's sets of positive and negative atomic subtype occurrences. In order to have a bound on the number of linear templates of a type, we extend the trace function from closed linear terms to linear terms with free variables which represent tree contexts, i.e. with type of the form $o^n \to o$. Again we show how it works on an example: the template $M = \lambda y_1 y_2 y_3 . C_1\,(y_1\,(\lambda y_4 . y_2\; y_4)\; C_2)\; y_3$ with tree contexts C_1 and C_2 of respective types $o \to o \to o$ and o,



Naturally, the free variables provide new atomic subtype occurrences and the positivity and negativity of those are computed as if C_1 and C_2 were variables like y_2 and y_3. If a

tree context is of the form $o^n \to o$ then it has 1 negative and n positive atomic subtype occurrences.

In order to show that the set of linear templates of a type is finite, we use the fact that templates are *minimal* decompositions: it means that there can not be a tree context that is directly applied to another tree context. This implies that, in the trace of a template, a positive atomic subtype occurrence of a tree context can not be mapped to a negative atomic subtype occurrence in a tree context. Since there is exactly one negative atomic subtype occurrence per tree context, the number of tree contexts in a template of type A is bounded by the number of positive atomic subtype occurrences in A. On the other hand, the number of positive atomic subtype occurrences in the tree contexts is bounded by the number of negative atomic subtype occurrences in A. So, for any given type A, the number of tree contexts of a linear template is bounded, the arity n of these tree contexts is bounded and, for each tree contexts setting, the number of traces (and therefore the number of templates) is bounded. Consequently, for all type A the number of linear templates of type A is bounded (by n^n where n is the size of type A).

A.1.2 Almost linear templates

Before we get to almost linear templates, we need to introduce η-contraction and η-long form for terms. An η-redex is a term of the form $(\lambda x.(M\,x))$ when $x \notin \mathrm{fv}(M)$ and its *η-contractum* is the term M. The relation of η-contraction, $\to_\eta$, η-reduction, $\overset{*}{\to}_\eta$, and η-conversion, $=_\eta$, are defined similarly to β-contraction. So as to compare λ-terms, we use the union of β-contraction and η-contration, $\to_{\beta\eta}$. But this can be done by putting terms in a particular form: the *η-long form*. A term M is said to be in η-long form whenever if N is a subterm of M that has type $A \to B$ then either N is of the form $\lambda x.N'$, or its occurrence in M is applied to some argument. For every term M there is a term M' in η-long form such that $M =_\eta M'$ and moreover $M =_{\beta\eta} N$ iff given M' and N' that are η-long forms of M and N, $M' =_\beta N'$.

In the case of almost linear templates, we first define an almost linear normal form for terms that are equivalent to almost linear terms. For this we use results by M. Kanazawa [16] (2012) on almost affine lambda terms. Note that these results are applicable to both almost affine and almost linear terms. This report characterizes almost linear terms as terms that have the *negatively non-duplicated* property, consequently almost linear terms are terms that are both non-erasing (each bound variable is used at least once) and have the *negatively non-duplicated* property.

The other result of that paper we are using is a lemma (Lemma 8 page 13), which, for every negatively non-duplicated term M in η-long β-normal form, builds, through a deterministic procedure, an almost affine term M' that β-reduces to M. The way M' is computed from M is by successively factorizing variables y that are not of atomic type but occur at several places in M. For any such variable y, the *negatively non-duplicated* property implies that there are terms $N_1, \ldots, N_m$ such that y always occurs in a term $y\,N_1 \ldots N_m$ of atomic type in M; then there is a subterm M_y of M containing all occurrences of $y\,N_1 \ldots N_m$, that term M_y is β-equivalent to the term $(\lambda y'.M'_y)\,(y\,N_1 \ldots N_m)$ where $M'_y = M_y[y\,N_1 \ldots N_m/y']$. By replacing M_y with $(\lambda y'.M'_y)\,(y\,N_1 \ldots N_m)$ in M we remove the copying of the non atomic variable y and instead have the copying of variable y' which is of atomic type. By applying this process to every copied variable of non-atomic type in M we get the almost linear term M' β-equivalent to M.

With any term M equivalent to an almost linear term, we associate the almost linear term M' obtained by applying that process to the η-long β-normal form of M. Since two

equivalent terms M_1 and M_2 have the same η-long β-normal form, they are associated with the same almost linear term M'. Therefore we have a normal form for all term that is equivalent to an almost linear term, we call it the almost linear normal form.

Once we have the almost linear normal form, we can apply the same reasoning as the one for linear templates. Because of the process of factorizing copied non-atomic variables, almost linear templates can be more complex than linear ones. But since the number of distinct non-atomic variables in a term M is bounded by the size of the type of M, the number of almost linear templates of a type A is bounded by $n_{templates} * (n_{fact})^{n_{var}}$ where $n_{templates}$ is the number of linear templates of type A, n_{fact} is a bound on the number of templatewise distinct possible factorizations of a non-atomic variable (i.e. two factorizations are templatewise distinct only if the templates of the factorized terms are distinct) and n_{var} is a bound on the number of non-atomic variables. We saw before that $n_{templates} \leq n^n$ where n is the size of the type A. The number of non-atomic variables is bounded by the size n of the type A. The template of a factorized term only depends on at which subterm M_y of M the factorization happens, and the number of templatewise distinct such M_y is bounded by the size of the template, so $n_{fact} \leq 2n$. Therefore the number of almost linear templates of a given type A of size n is bounded by n^{2n}.

A.2 Effective order reduction

We will use the following notation: if a λ-term M is associated to the decomposition $\langle M', \sigma\rangle$ where M' is a template and σ a substitution of the free variables in M', then we note $\mathfrak{T}(M) = (M', (\sigma(y_1), \dots, \sigma(y_n)))$ where $y_1, \dots, y_n$ are the free variables in M. In this case we allow $=$ to mean *equal up to renaming of free variables.* For all type A we note $t\langle A\rangle$ the set of templates of terms of type A.

A.2.1 Linear case

Before proving theorem 2 we first prove a useful lemma:

▶ **Lemma 8.** *Let $M[x_1, \dots, x_n]$ be a linear term built on signature Σ_1 with typed free variables $x_1^{A_1}, \dots, x_n^{A_n}$, let $t_1, \dots, t_n$ be linear templates of $x_1, \dots, x_n$. Then there is a linear template t and tree contexts $C_1, \dots, C_\ell$ with free variables $y_{1,1}, \dots, y_{1,\ell_1}, \dots, y_{n,1}, \dots, y_{n,\ell_n}$ such that, for all linear terms $N_1, \dots, N_n$ with $\mathfrak{T}(N_i) = (t_i, (C_{i,1}, \dots, C_{i,\ell_i}))$ for all i :*

$$\mathfrak{T}(M[x_1/N_1, \dots, x_n/N_n]) = (t, (C_1, \dots, C_\ell)[y_{i,j}/C_{i,j}]_{i\leq n, j\leq \ell_i})$$

Proof. For all $i \leq n$: $N_i =_{\beta\eta} t_i[y_{i,1}/C_{i,1}, \dots, y_{i,\ell_i}/C_{i,\ell_i}]$, where $y_{i,1}, \dots, y_{i,\ell_i}$ are the free variables of t_i, because $\mathfrak{T}(N_i) = (t_i, (C_{i,1}, \dots, C_{i,\ell_i}))$. Then we define t and $(C_1, \dots, C_\ell)$ as the template and tree-contexts of the λ-term $M[x_1/t_1, \dots, x_n/t_n]$ on the signature $\Sigma \cup \{y_{i,j}\}_{i\leq n, j\leq \ell_i}$ (it is a tree signature because variables $y_{i,j}$ are tree-contexts and therefore of order at most 1). Consequently :

$$\begin{aligned} M[x_1/N_1, \dots, x_n/N_n] &= M[x_1/t_1, \dots, x_n/t_n][y_{i,1}/C_{i,1}, \dots, y_{i,\ell_i}/C_{i,\ell_i}] \\ &= t[z_1/C_1, \dots, z_\ell/C_\ell][y_{1,1}/C_{1,1}, \dots, y_{n,\ell_n}/C_{n,\ell_n}] \end{aligned}$$

and so :

$$\mathfrak{T}(M[x_1/N_1, \dots, x_n/N_n]) = (t, (C_1, \dots, C_\ell)[y_{i,j}/C_{i,j}]_{i\leq n, j\leq \ell_i})$$

◀

Now we can prove theorem 2 in the linear case:

Proof. Let $T = (\Sigma_Q, \Sigma_1, \Sigma_2, q_0, R, \mathtt{A})$ be a $\text{HODTR}_{\text{lin}}$. We note L the set of states of $\mathtt{A}$. We want to define a $\text{HODTR}_{\text{lin}}$ $T' = (\Sigma_{Q'}, \Sigma_1, \Sigma_2, q'_0, R', \mathtt{A}')$ of order 3 equivalent to T.

We start by defining the look-ahead automaton $\mathtt{A}'$ and its set of states $L' = L \times t\langle A_{q_0}\rangle \dots t\langle A_{q_m}\rangle$ where $A_{q_0}, \dots, A_{q_m}$ are the output type of the states in Q and $t\langle A\rangle$ is the set of templates of type A. So this look-ahead associates, with every input tree N, the look-ahead $\mathtt{A}$ on tree N and, for each state q_i, the template of $q_i(N)$. Lemma 8 shows how to compute the template of a term $M[x_1, \dots, x_n]$ using the templates of $x_1, \dots, x_n$, then we define the rules of $\mathtt{A}'$ accordingly so that, for all input tree N, the state of the look-ahead $\mathtt{A}'$ on tree N is $(l, t_0, t_1, \dots, t_m)$ where l is the look-ahead of $\mathtt{A}$ on N and, for all $i \leq m$, t_i is the template of $q_i(N)$. We prove this by induction on the input tree, the induction step is a direct application of lemma 8.

Then we define the set of states Q' of T' : $Q' = \{(q_i, t) \mid q_i \in Q, t \in t\langle A_{q_i}\rangle\} \cup \{q'_0\}$. We will now define the rules in R' so that, for all $q_i \in Q, t \in t\langle A_{q_i}\rangle$ and for all input tree N : $(q_i, t)(N) = (C_1, \dots, C_\ell)$ (using continuations to represent the tuple) such that $\mathfrak{T}(q_i(N)) = (t, (C_1, \dots, C_\ell))$. For all state $(q_i, t) \in Q'$, input tree constant f of arity n, input tree variables $x_1, \dots, x_n$ and their look-ahead states $l_1, \dots, l_n$ in L and $l'_1, \dots, l'_n$ in L', and for all rule in R of the form : $q_i(f\, x_1 \dots x_n)\langle l_1, \dots, l_n\rangle \to M[x_1, \dots, x_n]$ where variable x_1 is processed by state q_{i_1}, x_2 by q_{i_2} and so on, we add the following rule in R' :

$$(q_i, t)\,(f\, x_1 \dots x_n)\langle l'_1, \dots, l'_n\rangle \to$$
$$\lambda k.(q_{i_1}, t_1)\, x_1\, (\lambda y_{1,1}, \dots y_{1,\ell_1}.\ \dots (q_{i_n}, t_n)\, x_n\, (\lambda y_{n,1} \dots y_{n,\ell_n}.k\, C_1 \dots C_\ell) \dots)$$

This is a way of setting variables $y_{1,1}, \dots, y_{1,\ell_1}$ to the tree contexts $(C_{1,1}, \dots, C_{1,\ell_1}) = (q_{i_1}, t_1)\,(x_1)$, it is necessary because using a projection on the tuple every time a tree context $C_{1,i}$ is used would break linearity.

The output type of such a state (q_i, t) is $(A_1 \to \dots A_\ell \to o) \to o$ where o is the atomic output tree type and A_i is the type of the i-th free variable of t, then, since the order of one of the A_i is at most 1, the order of the output type of (q_i, t) is at most 3. So the order of T' is at most 3.

Note that if state q_0 has output type o, the only template for that type is the term x where x is a free variable of type o. Then for the initial state q'_0 of output type o, we add special rules in R'. For all rule already in R' of the form : $(q_0, t)(f\, x_1 \dots x_n)\langle \overrightarrow{\ell}\rangle \to (C_1)$ where (C_1) is the unary tuple of type $(o \to o) \to o$ containing the tree C_1 of type o, we add the rule : $q'_0(f\, x_1 \dots x_n)\langle \overrightarrow{\ell}\rangle \to C_1$.

For all $q_i \in Q, t \in t\langle A_{q_i}\rangle$ and for all input tree N such that $\mathfrak{T}(q_i(N)) = (t, (C_1, \dots, C_\ell))$: $(q_i, t)(N) \to^*_{R'} (C_1, \dots, C_\ell)$; we prove this by induction on the input tree N. Again the induction is a direct application of Lemma 8.

Finally we conclude by applying this property to state $q_0 \in Q$ and template $x \in t\langle o\rangle$, and replacing the first rule applied to (q_0, x) by the corresponding rule on q'_0.

◀

A.2.2 Almost linear case

We first prove the equivalent of lemma 8 for the almost linear case :

▶ **Lemma 9.** *Let $M[x_1, \dots, x_n]$ be an almost linear term on signature Σ_1 with typed free variables $x_1^{A_1}, \dots, x_n^{A_n}$, let $t_1, \dots, t_n$ be almost linear templates of $x_1, \dots, x_n$. Then there is*

an almost linear template t and tree contexts $C_1, \dots, C_\ell$ with free variables $y_{1,1}, \dots, y_{n,\ell_n}$ such that, for all almost linear terms $N_1, \dots, N_n$ with $\mathfrak{T}(N_i) = (t_i, (C_{i,1}, \dots, C_{i,\ell_i}))$ for all i :

$$\mathfrak{T}(M[x_1/N_1, \dots, x_n/N_n]) = (t, (C_1, \dots, C_\ell)[y_{i,j}/C_{i,j}]_{i \le n, j \le \ell_i})$$

Proof. The key to this proof is to notice that the property of being an almost linear λ-term is preserved by substitution of variables with almost linear λ-terms and by $\beta\eta$-equivalence. It ensures that the term $M[x_1/N_1, \dots, x_n/N_n]$ is $\beta\eta$-equivalent to an almost linear λ-term.

The rest of the proof works like that of lemma 8. ◀

Then the order reduction theorem for almost linear transducers (theorem 2) is proven similarly to the linear case, but using lemma 9 as the almost linear extension of lemma 8.

B Equivalence with MSOT and MSOTS

B.1 Definition of ATT

Attribute grammars [13] are ways to formalize a class of syntax directed translation based on context free grammar. They amount to equip a context-free grammar with semantics attributes that propagate along the abstract syntax tree. These semantics attributes are *synthesized* when their value is propagated bottom-up and *inherited* when they are propagated top-down.

Attributed tree transducers, as defined by [2, 13], correspond to the combination of a relabeling attribute grammar and an attribute grammar whose attributes are trees. The relabeling simulates both the finite state control and the look-ahead automaton of usual transducers. In our setting, they can be seen as HODT with look-ahead whose rules are of the form $q(a\, x_1 \dots x_n) \to b\, q_1(x_1) \dots q_n(x_n)$, where $a \in \Sigma$, $b \in \Delta$ and a and b have the same arity. We call REL the class of transductions defined this way.

Formally, an attributed tree transducer from the input alphabet Σ to the output alphabet Δ is a tuple $(\Sigma, \Delta, S, I, out, R, root)$ where:

- Σ is the input alphabet,
- Δ is the output alphabet,
- S and I are the finite set of respectively *synthesized* and *inherited* attributes,
- $out \in S$, the *meaning attribute*,
- R, the *rules*, is a function that maps elements a of Σ of arity n to equations of the form $(\alpha, i) = M(\alpha_1, i_1) \dots (\alpha_k, i_k)$ for every (α, i) in $(S \times \{0\} \cup I \times [1, n])$ where M is a linear λ-term of type $o^k \to o$ built on the signature Δ and where (α_j, i_j) are pairwise distinct constants that have atomic type and where α_j is in $S \cup I$ and i_j is in $[0, n]$.
- *root*, the *initialization of inherited attributes* which maps elements a of Σ to equations of the form $(\alpha, 0) = M(\alpha_1, 0) \dots (\alpha_k, 0)$ for every α in I, where M is a linear λ-term of type $o^k \to o$ built on the signature Δ and, for all $j \le k$, $(\alpha_j, 0)$ is a constant of atomic type and α_j is in $S \cup I$.

Now given an input tree N built on signature Σ, we let V_N be the set of paths of N that is inductively defined by, for $N = a\, N_1 \dots N_n$: $V_N = \{\epsilon\} \cup \bigcup_{i=1}^{n} \{i.u \mid u \in V_{N_i}\}$. For u in V_N, we write $N \restriction_u$ for the subterm of N that is at path u and which is defined as $N \restriction_\epsilon = N$, $(a\, N_1 \dots N_n) \restriction_{iu} = N_i \restriction_u$. For u in V_N, we let $lab_N(u)$ be the constant a in Σ such that $N \restriction_u = a\, N_1 \dots N_n$. Consider v in $V_{N \restriction_u}$, we have that $(N \restriction_u) \restriction_v = N \restriction_{uv}$. Therefore the operation that appends u in front of an element of $V_{N \restriction_u}$ defines an injection from $V_{N \restriction_u}$ into V_N that preserves the designated term.

The attribute transducer associates with each element of V_N a set of attributes. Formally, it builds a set of equations whose left-hand side belong to $A(N) = (S \cup I) \times V_N$. We call the elements of $A(N)$ *attribute instances* or simply *attributes* of N when the context is clear. For $u \in V_N$, the subset $A_u(N) = \{(\alpha, u) \mid \alpha \in S \cup I\}$ is the set of attributes associated with N at path u. For each attribute $(\alpha, v) \in A(N\restriction_u)$ we define $u.(\alpha, v)$ as the attribute $(\alpha, uv) \in A(N)$. Given a set of attribute instances S, we write $u.S$ for the set $\{(\alpha, uv) \mid (\alpha, v) \in S\}$. Then the following identity holds $u.A_v(N\restriction_u) = A_{uv}(N)$.

The attribute transducer associates an equation with every attribute (α, u) of $A(N)$ as follows. If an equation $E_{(\alpha,i)} \in R(a)$ is of the form $(\alpha, i) = M(\alpha_1, i_1) \dots (\alpha_n, i_n)$ then, for all path $u \in V_N$ such that $lab_N(u) = a$, the equation $(\alpha, u.i) = M(\alpha_1, u.i_1) \dots (\alpha_n, u.i_n)$ is the equation for the attribute $(\alpha, u.i)$ and is noted $u.E_{(\alpha,i)}$. The operation $u.$ on equations naturally extends to sets of equations. We note $Eq_u(N)$ the set of equations $u.R(lab_N(u))$, and $Eq_{u\downarrow}(N)$ the set of equations $\bigcup_{v \in V_N\restriction_u} Eq_{uv}(N)$. Then the set of equations associated with N (noted $Eq(N)$) is $Eq(N) = Eq_{\epsilon\downarrow}(N) = \bigcup_{u \in V_N} Eq_u(N)$. The *complete* set of equations of N (noted $CEq(N)$) is $CEq(N) = root(lab_N(\epsilon)) \cup Eq(N)$. We will also use the notation $CEq_{u\uparrow}(N)$ for the set $CEq(N) \setminus Eq_{u\downarrow}(N)$ for all $u \in V_N$.

We represent the way attributes depend on each other using graph as follows. With an equation $E_{(\alpha,i)} \in R(a)$ of the form $(\alpha, i) = M(\alpha_1, i_1) \dots (\alpha_n, i_n)$ we associate the directed graph $G(E_{(\alpha,i)})$ whose set of vertices is $V = \{(\alpha, i), (\alpha_1, i_1), \dots, (\alpha_n, i_n)\}$ and set of edges is $E = \{((\alpha, i), (\alpha_j, i_j)) \mid j \in [1, n]\}$. Define the operation of non-disjoint union of graphs whose sets of vertices are not necessarity disjoint as follows: for all graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. For all set Eq of equations we define the graph $G(Eq)$ associated with the set of equations Eq as $\bigcup_{E \in Eq} G(E)$. We define the operation $u.$ on such graphs by: $u.G(Eq) = G(u.Eq)$. The *dependency graph* and *complete dependency graph* of N are $G(Eq(N))$ and $G(CEq(N))$ respectively, and they are noted $D(N)$ and $CD(N)$ respectively. Similarly, we will use the notations $D_{u\downarrow}(N)$ for the graph $G(Eq_{u\downarrow}(N))$ and $CD_{u\uparrow}(N)$ for the graph $G(Eq_{u\uparrow}(N))$.

Note that in $D(N)$, there are no edges pointing to inherited attributes of the root node of N (attributes in $I \times \{\epsilon\}$).

When $CD(N)$ is acyclic, the attribute grammar is said *non-circular* on N and we note $\mathrm{Ord}(CD(N))$ the set of its topological sorts (i.e. the total orders which embed into the partial order on nodes induced by the acyclic graph $CD(N)$). In that case, we can associate with every attribute of N a tree built on Δ by applying the equations in $CEq(N)$. Indeed, a topological sort of the acyclic graph $CD(N)$ gives an order in which we can evaluate the attributes of N, i.e. associate with them a term built on Δ. Then the tree associated with the attribute (out, ϵ) is the result of the attribute tree transducer. An attribute tree transducer is said *non-circular* when for every N, $CD(N)$ is acyclic. We note ATT the class of transductions that are defined by Attribute Tree Transducers. When moreover for every N the dependency graph is a tree, the Attribute Tree Transducer is said *single use restricted*. We note $\mathrm{ATT}_{\mathrm{sur}}$ the class of transductions that are defined by single use restricted Attribute Tree Transducers.

▶ **Theorem 10.** *[2]*

We have the following equivalences:

- $\mathrm{REL} \circ ATT = MSOTS$,
- $\mathrm{REL} \circ ATT_{sur} = MSOT$,

B.2 $\mathsf{REL} \circ \mathsf{ATT} \subseteq \mathsf{HODTR}_{\mathsf{al}}$ and $\mathsf{REL} \circ \mathsf{ATT}_{sur} \subseteq \mathsf{HODTR}_{\mathsf{lin}}$

In this part we want to prove that the composition of a relabeling attribute grammar with an attributed tree transducer can be modeled by a $\mathrm{HODTR}_{\mathrm{al}}$, and that if the attributed tree transducer is single use restricted then the translated $\mathrm{HODTR}_{\mathrm{al}}$ is a $\mathrm{HODTR}_{\mathrm{lin}}$.

The order in which the attributes are computed is important, in that regard we need a few more definitions.

B.2.1 Definitions and notations

For all tree N, we note $CD^{\top}(N)$ the graph obtained from $CD(N)$ by adding a vertex noted $\top$ and an edge $((out, \epsilon), \top)$, and, for all $u \in V_N$, we note $CD^{\top}_{u\uparrow}(N)$ the graph obtained similarly from $CD_{u\uparrow}(N)$ by adding a vertex $\top$ and an edge $((out, \epsilon), \top)$. We note $A^{\top}(N) = \{\top\} \cup A(N)$ and, for all path $u \in V_N$, $A^{\top}_u(N) = \{\top\} \cup A_u(N)$.

We use the convention that $u.\top = u^{-1}.\top = \top$.

For all graph $G = (V, E)$ and set V', we note $\mathrm{tr}(G)_{|V'}$ the subgraph of the transitive closure of G induced by $V' \cap V$.

▶ **Lemma 11.** *For all graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ and set V, if $V_1 \cap V_2 \subseteq V$, then $\mathrm{tr}(G_1 \cup G_2)_{|V} = \mathrm{tr}(\mathrm{tr}(G_1)_{|V} \cup G_2)_{|V}$.*

Proof. The set of vertices of both $\mathrm{tr}(G_1 \cup G_2)_{|V}$ and $\mathrm{tr}(\mathrm{tr}(G_1)_{|V} \cup G_2)_{|V}$ is $V \cap (V_1 \cup V_2)$.

For all vertices x and y, if there is in $\mathrm{tr}(G_1)_{|V}$ a path from x to y there exists a path from x to y in G_1. Then for all path in $\mathrm{tr}(G_1)_{|V} \cup G_2$ from x to y there is a path from x to y in $G_1 \cup G_2$. So, for all edge (x, y) in the graph $\mathrm{tr}(\mathrm{tr}(G_1)_{|V} \cup G_2)_{|V}$ there is an edge (x, y) in $\mathrm{tr}(G_1 \cup G_2)_{|V}$.

Let $(x, y) \in V^2$ be an edge of the graph $\mathrm{tr}(G_1 \cup G_2)_{|V}$, then there is a path w from x to y in $G_1 \cup G_2$. This path can be written $w = w_1 \dots w_n$ where $w_1, \dots, w_n$ are paths in either G_1 or G_2 and, for all $i \leq n-1$, if w_i is a path in G_1 then w_{i+1} is a path in G_2 and if w_i is a path in G_2 then w_{i+1} is a path in G_1. Then, for all $i \leq n-1$, the end vertex of path w_i is in $V_1 \cap V_2$. Since $V_1 \cap V_2 \subseteq V$ and x and y are in V, all start and end vertices of paths $w_1, \dots, w_n$ are in V. Then for all path w_i in $\mathrm{tr}(G_1)_{|V}$ there is a path w'_i with same start and end vertices in the graph G_1. Therefore, noting $w'_i = w_i$ if w_i is a path in G_2 but not G_1 for all $i \leq n$, $w'_1, \dots, w'_n$ is a path from x to y in $\mathrm{tr}(G_1)_{|V} \cup G_2$. So there is an edge (x, y) in the graph $\mathrm{tr}(\mathrm{tr}(G_1)_{|V} \cup G_2)_{|V}$. ◀

▶ **Definition 12.** *For all tree path $u \in V_N$, we define the* synthesis graph *of path u in N, noted $GS_u(N)$, as the graph $u^{-1}.(\mathrm{tr}(D_{u\downarrow}(N))_{|A_u(N)})$.*

For all tree path $u \in V_N$, we call the inheritance graph *of path u in N, noted $GI_u(N)$, the graph $u^{-1}.(\mathrm{tr}(CD^{\top}_{u\uparrow}(N))_{|V'})$ where V' is the subset of $A^{\top}_u(N)$ of vertices connected to the vertex $\top$ in the graph $CD^{\top}_{u\uparrow}(N)$.*

For every tree N and path $u \in V_N$, the sets of nodes of $GS_u(N)$ and $GI_u(N)$ are $A_\epsilon(N \lfloor_u)$ and $A^{\top}_\epsilon(N \lfloor_u)$ respectively, since these sets are not dependent on the tree N or the path u we simply note them $A_\epsilon = (S \cup I) \times \{\epsilon\}$ and $A^{\top}_\epsilon = \{\top\} \cup A_\epsilon$ respectively.

▶ **Lemma 13.** *For all $u \in V_N$, the edges of the graph $GS_u(N)$ are of the form $((\alpha, \epsilon), (\gamma, \epsilon))$ with $\alpha \in S \cup I$ and $\gamma \in S$.*

▶ **Lemma 14.** *For all $u \in V_N$, $GS_u(N) = \mathrm{tr}(G(R(a)) \bigcup_{1 \leq i \leq n} i.GS_{ui}(N))_{|A^{\top}_\epsilon}$ where n is the arity of the node at path u in N.*

Proof. We note G_0 the graph $\mathrm{tr}(G(R(a)) \bigcup_{1 \leq i \leq n} i.GS_{ui}(N))_{|A_\epsilon^\top}$. The graphs G_0 and $GS_u(N)$ have the same set of vertices A_ϵ.

Let (x, y) be an edge of the graph G_0, then, by definition of G_0, there is a path from $u.x$ to $u.y$ in the graph $\bigcup_{i \leq n} ui.GS_{ui}(N) \cup u.G(R(a))$ (this works because $u.$ is only a renaming of the attributes). By definition, any edge in $u.G(R(a))$ is in $D_{u\downarrow}(N)$. For all $i \leq n$ and for all edge (x_i, y_i) in $ui.GS_{ui}(N)$ there is a path in $D_{ui\downarrow}(N)$ from x_i to y_i, then this path also exists in the graph $D_{u\downarrow}(N)$. Then there is in the graph $D_{u\downarrow}(N)$ a path from $u.a$ to $u.b$. So the set of edges of G_0 is included in the set of edges of $GS_u(N)$.

Let (x, y) be an edge of $GS_u(N)$, then there is in the graph $D_{u\downarrow}(N)$ a path from $u.x$ to $u.y$. This path is of the form $w_1 e_1 w_2 \dots w_m e_m w_{m+1}$ where $e_1, \dots, e_m$ are edges in $u.G(R(a))$ and $w_1, \dots, w_{m+1}$ are paths with no edges in $u.G(R(a))$. Since $D_{u\downarrow}(N) = u.G(R(a)) \cup \bigcup_{1 \leq i \leq n} D_{ui\downarrow}(N)$ and the graphs $D_{ui\downarrow}(N)$ have disjoint sets of vertices, for all $j \leq m+1$ there is an index $i_j \leq n$ such that the path w_j is in the graph $D_{ui_j\downarrow}(N)$. Then for all $j \leq m+1$, noting x_j and y_j the respective start and end of path w_j, there is an edge (x_j, y_j) in the graph $ui_j.GS_{ui_j}(N)$. Then the path $(x_1, y_1)e_1 \dots e_m(x_{m+1}, y_{m+1})$ is in the graph $\bigcup_{i \leq n} ui.GS_{ui}(N) \cup u.G(R(a))$, with $u.x = x_1$ and $u.y = y_{m+1}$. So there is a path from x to y in the graph $\bigcup_{i \leq n} i.GS_{ui}(N) \cup G(R(a))$, therefore there is an edge (x, y) in the graph $\mathrm{tr}(\bigcup_{i \leq n} i.GS_{ui}(N) \cup G(R(a)))_{|A_\epsilon}$, so that edge is in G_0.

So $G_0 = GS_u(N)$. ◀

▶ **Lemma 15.** *There exists a bottom-up tree automaton* $\mathtt{A}$*, whose set of states is the set of directed acyclic graphs with set of vertices* A_ϵ*, which associates with any node in a tree* N *the graph* $GS_u(N)$*.*

Proof. We define the bottom-up tree automaton $\mathtt{A} = (\Sigma_P, \Sigma_1, R_\mathtt{A})$ where P is the set of states of the form p_G where $G = (V, E)$ is a directed acyclic graph with $V = A_\epsilon$ and $E \subseteq \{((\alpha, \epsilon), (\gamma, \epsilon)) \mid \alpha \in S \cup I, \gamma \in S\}$, i.e. potential synthesis graphs according to lemma 13; and $R_\mathtt{A}$ is the set of rules of the form $a(p_{G_1} \dots p_{G_n}) \to p_{G_0}$ where a is a tree constant in Σ_1 of arity n, and G_0 is the graph $\mathrm{tr}(\bigcup_{i \leq n} i.G_i \cup G(R(a)))_{|A_\epsilon}$ where $G(R(a))$ is the graph induced by the equations of the attribute transducer associated with the tree constant a.

Lemma 14 implies by induction that automaton $\mathtt{A}$ indeed associates with any node at path u in N the synthesis graph $GS_u(N)$ of N at path u. ◀

▶ **Definition 16.** *For all tree path* $u \in V_N$*, The* interface graph *of* N *at path* u *(noted* $G_u(N)$*) is the directed acyclic graph* $u^{-1}.(\mathrm{tr}(CD^\top(N))_{|V'})$ *where* V' *is the subset of* $A_u^\top(N)$ *of vertices connected to the vertex* $\top$ *in the graph* $CD^\top(N)$*.*

▶ **Lemma 17.** *For all path* $u \in V_N$*,* $G_u(N) = \mathrm{tr}(GS_u(N) \cup GI_u(N))_{|V'}$ *where* V' *is the subset of* $A_\epsilon^\top$ *of vertices connected to the vertex* $\top$ *in the graph* $GS_u(N) \cup GI_u(N)$*.*

Proof. We note $G = \mathrm{tr}(GS_u(N) \cup GI_u(N))_{|V'}$. We first prove the following claim:

▷ Claim 18. For all $x, y \in A_\epsilon^\top$, there is a path from $u.x$ to $u.y$ in the graph $CD^\top(N)$ if and only if there is a path from x to y in the graph $GS_u(N) \cup GI_u(N)$.

Proof. Assume there is a path from $u.x$ to $u.y$ in $CD^\top(N)$. Since $CD^\top(N) = CD_{u\uparrow}^\top(N) \cup D_{u\downarrow}(N)$, this path can be seen as a sequence of paths $w_1 \dots w_m$ alternating between graphs $CD_{u\uparrow}^\top(N)$ and $D_{u\downarrow}(N)$ (if w_i is a path in the graph $CD_{u\uparrow}^\top(N)$ then w_{i+1} is a path in $D_{u\downarrow}(N)$ and conversely). We note x_i and y_i the respective start and end of path w_i for all $i \leq m$. For all $i \leq m-1$, since the vertex $y_i = x_{i+1}$ is in both graphs $CD_{u\uparrow}^\top(N)$ and $D_{u\downarrow}(N)$, it must be in the set $A_u(N)$. Then there is an edge (x_i, y_i) in either $\mathrm{tr}(CD_{u\uparrow}^\top(N))_{|A_u(N)}$ or

$\mathrm{tr}(D_{u\downarrow}(N))_{|_{A_u(N)}}$ for all $i \leq m$. Because $x = u^{-1}.x_1$ and $y = u^{-1}.y_m$, there is in the graph $GS_u(N) \cup GI_u(N)$ a path from x to y.

Assume there is a path from x to y in $CD^\top(N)$. That path is of the form $(x_1, x_2)(x_2, x_3)\dots(x_m, x_{m+1})$ where, for each $i \leq m$, (x_i, x_{i+1}) is an edge of either $GS_u(N)$ or $GI_u(N)$. So, for all $i \leq m$, there is either in $CD^\top_{u\uparrow}(N)$ or in $D_{u\downarrow}(N)$ a path from $u.x_i$ to $u.x_{i+1}$. Therefore we have in the graph $CD^\top(N)$ a path from $u.x = u.x_1$ to $u.y = u.x_{m+1}$. ◀

This claim applied with $y = \top$ implies that G and $G_u(N)$ have the same sets of vertices. The claim also implies that (x, y) is an edge of G if and only if (x, y) is an edge of $G_u(N)$. So $G = G_u(N)$ for all path $u \in V_N$. ◀

▶ **Lemma 19.** *For all directed acyclic graph $G = (V, E)$, and subset $V' \subseteq V$ of vertices, and for all two vertices $x, y \in V'$, noting $\mathrm{tr}(G)_{|_{V'}} = (V', E')$ the subgraph of the transitive closure of G induced by the subset V' of vertices, if the graph $(V', E' \cup \{(x, y)\})$ is acyclic then the graph $(V, E \cup \{(x, y)\})$ is also acyclic.*

Proof. We use ad absurdum reasoning. We assume that the graph $(V', E' \cup \{(x, y)\})$ is acyclic and that there is a cycle in the graph $(V, E \cup \{(x, y)\})$. Since (V, E) is acyclic the edge (x, y) is part of the cycle, so the cycle is of the form $(x, y)(y, x_1)\dots(x_n, x)$ with vertices $x_1, \dots, x_n \in V$. Then there is a path from y to x in G, therefore there is an edge (y, x) in $\mathrm{tr}(G)_{|_{V'}}$, so $(y, x) \in E'$. Then $(V', E' \cup \{(x, y)\})$ is not acyclic, which leads to a contradiction. ◀

We will use the notations $A_{[1,n]} = \bigcup_{1 \leq j \leq n}(S \cup I) \times \{j\}$, $A_{[0,n]} = \bigcup_{0 \leq j \leq n}(S \cup I) \times \{j\}$ (with the convention that $0 = \epsilon$), $A^\top_{[1,n]} = \{\top\} \cup A_{[1,n]}$ and $A^\top_{[0,n]} = \{\top\} \cup A_{[0,n]}$.

▶ **Definition 20.** *For all path $u \in V_N$, we define the* local dependency graph *of N at path u, noted $G_{u.[0,n]}(N)$ where n is the arity of $lab_N(u)$, as the graph $u^{-1}.\mathrm{tr}(CD^\top(N))_{|_{V'}}$ where V' is the set of vertices in $u.A^\top_{[0,n]}$ that are connected to the vertex $\top$ in the graph $CD^\top(N)$.*

▶ **Lemma 21.** *For all tree N and path $u \in V_N$, noting $a = lab_N(u)$ the constant of the node at path u in N and n its arity, the local dependency graph $G_{u.[0,n]}(N)$ of N at path u is $\mathrm{tr}(GI_u(N) \cup G(R(a)) \cup \bigcup_{1 \leq j \leq n} j.GS_{uj}(N))_{|_{V'}}$ where V' is the set of vertices in $A^\top_{[0,n]}$ that are connected to the vertex $\top$ in the graph $GI_u(N) \cup G(R(a)) \cup \bigcup_{1 \leq j \leq n} j.GS_{uj}(N)$.*

Proof. We first prove the following claim:

▷ Claim 22. For all vertices $x, y \in A^\top_{[0,n]}$, there is in the graph $CD^\top(N)$ a path from $u.x$ to $u.y$ if and only if there is a path from $u.x$ to $u.y$ in the graph $G = \mathrm{tr}(CD^\top_{u\uparrow}(N))_{|_{A^\top_u(N)}} \cup u.G(R(a)) \cup \bigcup_{1 \leq i \leq n} \mathrm{tr}(D_{ui\downarrow}(N))_{|_{A_{uj}(N)}}$.

Proof. If there is a path from $u.x$ to $u.y$ in G then, because $CD^\top(N) = CD^\top_{u\uparrow}(N) \cup u.G(R(a)) \cup \bigcup_{1 \leq i \leq n} D_{ui\downarrow}(N)$, there must be a path from $u.x$ to $u.y$ in $CD^\top(N)$.

If there is a path from $u.x$ to $u.y$ in $CD^\top(N)$, then this path can be seen as a sequence $w_1 \dots w_m$ of paths where each w_j with $j \leq m$ is a path in either one of the following $n + 2$ graphs: $CD^\top_{u\uparrow}(N), D_{u1\downarrow}(N), \dots D_{un\downarrow}(N), u.G(R(a))$, and, for each $j \leq m-1$, w_{j+1} is a path in a different graph than w_j. Noting x_j the end of path w_j or start of path w_{j+1}, since w_j and w_{j+1} are paths of a different graph among $CD^\top_{u\uparrow}(N), D_{u1\downarrow}(N), \dots D_{un\downarrow}(N)$ and $u.G(R(a))$, x_j is in the intersection of the sets of vertices of these two graphs, which is necessarily included in the set $u.A^\top_{[0,n]}(N)$. $x_0 = u.x$ and $x_m = u.y$ are also in the set $u.A^\top_{[0,n]}(N)$. This implies that if w_j is a path in $CD^\top_{u\uparrow}(N)$ then there is in $\mathrm{tr}(CD^\top_{u\uparrow}(N))_{|_{A^\top_u(N)}}$ a path w'_j from x_{j-1} to x_j. Also if w_j is a path in $D_{ui\downarrow}(N)$ then there is in $\mathrm{tr}(D_{ui\downarrow}(N))_{|_{A_{uj}(N)}}$ a path w'_j from x_{j-1} to

x_j. So there is in the graph $G = \mathrm{tr}(CD^{\top}_{u\uparrow}(N))_{|_{A^{\top}_u(N)}} \cup u.G(R(a)) \cup \bigcup_{1\leq i\leq n} \mathrm{tr}(D_{ui\downarrow}(N))_{|_{A_{uj}(N)}}$ a path from $u.x$ to $u.y$. ◀

Since $G_{u.[0,n]}(N) = u^{-1}.\mathrm{tr}(CD^{\top}(N))_{|_{u.A^{\top}_{[0,n]}}}$ and $\mathrm{tr}(GI_u(N) \cup G(R(a)) \cup \bigcup_{1\leq j\leq n} j.GS_{uj}(N))_{|_{A^{\top}_{[0,n]}}} = u^{-1}.\mathrm{tr}(G)_{|_{V'}}$, the claim implies that the set of vertices of the graph $G_{u.[0,n]}(N)$ is the set V' of vertices in $A^{\top}_{[0,n]}$ that are connected to the vertex $\top$ in the graph $GI_u(N) \cup G(R(a)) \cup \bigcup_{1\leq j\leq n} j.GS_{uj}(N)$.

It also entails that, for all vertices $x, y \in V'$, there is in the graph $G_{u.[0,n]}(N)$ an edge (x, y) if and only if (x, y) is an edge in the graph $\mathrm{tr}(GI_u(N) \cup G(R(a)) \cup \bigcup_{1\leq j\leq n} j.GS_{uj}(N))_{|_{V'}}$.

Therefore $G_{u.[0,n]}(N) = \mathrm{tr}(GI_u(N) \cup G(R(a)) \cup \bigcup_{1\leq j\leq n} j.GS_{uj}(N))_{|_{V'}}$. ◀

▶ **Corollary 23.** *The local dependency graph $G_{u.[0,n]}(N)$ can be computed using only the constant $lab_N(u)$, the inheritance graph of N at path u and the synthesis graphs of N at paths $u1, \ldots, un$.*

▶ **Lemma 24.** *If $CD^{\top}(N)$ is a tree then, for all path $u \in V_N$, $G_{u.[0,n]}(N)$ is a tree.*

Proof. We use *ad absurdum* reasoning. We assume that $G_{u.[0,n]}(N)$ is not a tree, so there exists two nodes x, y and two distinct paths from x to y in $G_{u.[0,n]}(N) = u^{-1}.\mathrm{tr}(CD^{\top}(N))_{|_{V'}}$. Then there are two distinct paths from $u.x$ to $u.y$ in $CD^{\top}(N)$, then $CD^{\top}(N)$ is not a tree. ◀

▶ **Corollary 25.** *If the ATT is* single use restricted *then for all input tree N and path $u \in V_N$, the graph $G_{u.[0,n]}(N)$ is a tree.*

B.2.2 Topological sorts

In order to sequentialize the computation of attributes, we use topological sorts of the graphs of dependency prevously defined. We will later need to use induction on the sorted attributes, in order to facilitate that we define our topological sorts as sequences of attributes:

▶ **Definition 26.** *We call a total order $<$ on a finite set V* compatible with *a directed acyclic graph $G = (V, E)$ if, for all edge $(v, v') \in E$, $v < v'$.*

Noting n the size of the set V, for all sequence $\tau = v_1 \ldots v_n \in V^*$ of length n such that $i \neq j \Rightarrow v_i \neq v_j$ for all $1 \leq i, j \leq n$, we associate with τ the unique total order $<$ on V such that $i < j \Leftrightarrow v_i < v_j$ for all $1 \leq i, j \leq n$.

We call a sequence $\tau \in V^*$ a topological sort *of a directed acyclic graph $G = (V, E)$ if it is of length n and the total order $<$ associated with it is compatible with G.*

▶ **Lemma 27.** *For all directed acyclic graph G we can build a topological sort τ of G.*

Proof. We build τ inductively. We note $G = (V, E)$.

Since G is acyclic there exists a vertex x of G which has no incoming edges. We use induction and assume we can build a topological sort τ' of the subgraph of G induced by the set $V \setminus \{x\}$ of vertices. Then $\tau = x\tau'$ is a topological sort of G. ◀

▶ **Lemma 28.** *For all path $u \in V_N$ any topological sort τ of $G_u(N)$ is of the form $\tau = \tau'(\alpha, \epsilon)\top$ with $\alpha \in S$.*

Proof. By definition of $G_u(N)$, from any vertex of $G_u(N)$ there is a path to $\top$, so a topological sort of $G_u(N)$ must end with $\top$. The form of the rules of the attribute transducer imply that if there is a path in $G_u(N)$ from (γ, ϵ) to $\top$ with $\gamma \in I$ then there must exists $\alpha \in S$

and a path in the graph $G_u(N)$ from (α, ϵ) to (γ, ϵ). So any topological sort of $G_u(N)$ ends with $(\alpha, \epsilon)\top$ for some $\alpha \in S$. ◀

▶ **Definition 29.** *For all sets V and V' such that $V' \subset V$, for all graph $G = (V, E)$ and topological sort τ of G, we call* topological subsort induced *by the subset V', and we note $\tau|_{V'}$, the biggest subsequence of τ included in V'^*.*

▶ **Lemma 30.** *For all directed acyclic graph $G = (V, E)$, topological sort τ of G and subset V' of V, $\tau|_{V'}$ is a topological sort of $\mathrm{tr}(G)_{|_{V'}}$.*

Proof. We note $G' = \mathrm{tr}(G)_{|_{V'}} = (V', E')$. Let (a, b) be an edge in E', then there is a path in the graph G from a to b of the form $a\, v_1 \dots v_m\, b$. So, noting $<_\tau$ the total order on V associated with τ, $a <_\tau v_1 <_\tau \cdots <_\tau v_m <_\tau b$. Therefore $a <_\tau b$, and a appears in the sequence τ strictly before b. Then a appears in the sequence $\tau|_{V'}$ strictly before b, and $a <_{\tau'} b$ where $<_{\tau'}$ is the total order on V' associated with $\tau|_{V'}$.

We have shown that $<_{\tau'}$ is compatible with G', so $\tau|_{V'}$ is a topological sort of $\mathrm{tr}(G)_{|_{V'}}$. ◀

▶ **Lemma 31.** *For all directed acyclic graph $G = (V, E)$, subset V' of V and topological sort τ' of $\mathrm{tr}(G)_{|_{V'}}$, there exists a topological sort τ of G such that $\tau' = \tau|_{V'}$.*

Proof. We note $x_1, \dots, x_n$ the vertices in V' such that $\tau' = x_1 \dots x_n$, and $\mathrm{tr}(G)_{|_{V'}} = (V', E')$. We note $E_{\tau'}$ the set of edges $E_{\tau'} = \{(x_i, x_j)\}_{1 \le i < j \le n}$, then we show that the graph $G' = (V', E' \cup E_{\tau'})$ is acyclic.

If G' contained a cycle, it would imply that there was in (V', E') a path from x_j to x_i with $i < j$, which is contradicts the fact that $\tau' = x_1 \dots x_n$ is a topological sort of (V', E').

Since $(V', E' \cup E_{\tau'})$ is acyclic, we can use lemma 19 and deduce that $(V, E \cup E_{\tau'})$ is also acyclic. Then there exists a topological sort τ of $(V, E \cup E_{\tau'})$. Because $E_{\tau'} = \{(x_i, x_j)\}_{1 \le i < j \le n}$ and by definition of topological sorts: $\tau|_{V'} = \tau'$. Also τ is a topological sort of G. ◀

▶ **Definition 32.** *For all graphs G and $\tilde{G}$ with the same set of vertices, we say that $\tilde{G}$ is an* over-specification *of G, and we note $\tilde{G} \trianglerighteq G$, if all topological sort of $\tilde{G}$ is a topological sort of G.*

▶ **Lemma 33.** *The relation $\trianglerighteq$ has the following properties:*

1. *for all graphs G_1, G_2 and G_3, $G_1 \trianglerighteq G_2 \trianglerighteq G_3 \Rightarrow G_1 \trianglerighteq G_3$ (transitivity),*
2. *for all graphs $G = (V, E)$ and $\tilde{G} = (V, \tilde{E})$, $E \subseteq \tilde{E} \Rightarrow \tilde{G} \trianglerighteq G$,*
3. *for all graph $G = (V, E)$, $G \trianglerighteq \mathrm{tr}(G)_{|_V} \trianglerighteq G$,*
4. *for all graphs G and $\tilde{G}$ and set V', $\tilde{G} \trianglerighteq G \Rightarrow \mathrm{tr}(\tilde{G})_{|_{V'}} \trianglerighteq \mathrm{tr}(G)_{|_{V'}}$*
5. *for all graphs $G_1, G_2, \tilde{G}_1$ and $\tilde{G}_2$, $\tilde{G}_1 \trianglerighteq G_1$ and $\tilde{G}_2 \trianglerighteq G_2 \Rightarrow \tilde{G}_1 \cup \tilde{G}_2 \trianglerighteq G_1 \cup G_2$*

Proof.

1. Implied by the definition of $\trianglerighteq$.
2. Implied by the definition of topological sorts.
3. The previous point implies that $\mathrm{tr}(G)_{|_V} \trianglerighteq G$. For all topological sort τ of G, by transitivity of the order associated with τ, τ is also a topological order of the transitive closure $\mathrm{tr}(G)_{|_V}$ of G. So $G \trianglerighteq \mathrm{tr}(G)_{|_V}$.
4. For all topological sort τ' of $\mathrm{tr}(\tilde{G})_{|_{V'}}$, according to lemma 31, there is a topological sort τ of $\tilde{G}$ such that $\tau|_{V'} = \tau'$. Then τ is also a topological sort of G and, according to lemma 30, $\tau' = \tau|_{V'}$ is a topological sort of $\mathrm{tr}(G)_{|_{V'}}$.
5. Let us assume that $\tilde{G}_1 \trianglerighteq G_1$ and $\tilde{G}_2 \trianglerighteq G_2$ with $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. For all topological sort τ of $\tilde{G}_1 \cup \tilde{G}_2$, according to lemma 30, $\tau|_{V_1}$ and $\tau|_{V_2}$ are topological sorts of $\tilde{G}_1$ and $\tilde{G}_2$ respectively. So $\tau|_{V_1}$ and $\tau|_{V_2}$ respectively are topological sorts of G_1 and G_2. So τ is a topological sort of $G_1 \cup G_2$. Therefore $\tilde{G}_1 \cup \tilde{G}_2 \trianglerighteq G_1 \cup G_2$.

◀

▶ **Lemma 34.** *For all graphs G_1 and G_2 such that $G_2 \trianglerighteq G_1$ and G_2 is closed by transitivity, then G_2 can be obtained from G_1 by adding edges.*

Proof. We note $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$. Then G_2 can be obtained from G_1 by adding edges if and only if $E_1 \subseteq E_2$. We use *ad absurdum* reasoning and assume there is an edge $(x, y) \in E_1 \setminus E_2$. We note $V_{y\downarrow} = \{z \mid z \in V, (z, y) \in E_2\}$ and $V_x = V \setminus (\{y\} \cup V_{y\downarrow})$. So $x \in V_x$. Let $\tau_{y\downarrow}$ and τ_x be topological sorts of the acyclic graphs $\mathrm{tr}(G_2)_{|_{V_{y\downarrow}}}$ and $\mathrm{tr}(G_2)_{|_{V_x}}$ respectively. We now prove that $\tau = \tau_{y\downarrow} y \tau_x$ is a topological sort of G_2: for all $z_1, z_2 \in E_2$,

- if $(z_1, z_2) \in V_{y\downarrow} \times \{y\}$ then $z_1 <_\tau z_2$ because we put $\tau_{y\downarrow}$ before y in τ,
- if $(z_1, z_2) \in V_{y\downarrow} \times V_x$ then $z_1 <_\tau z_2$ because we put $\tau_{y\downarrow}$ before τ_x in τ,
- if $(z_1, z_2) \in \{y\} \times V_x$ then $z_1 <_\tau z_2$ because we put y before τ_x in τ,
- if $(z_1, z_2) \in V_{y\downarrow}^2$ then $z_1 <_{\tau_{y\downarrow}} z_2$ entails $z_1 <_\tau z_2$,
- the case $(z_1, z_2) \in \{y\}^2$ is impossible because G_2 is acyclic,
- if $(z_1, z_2) \in V_x^2$ then $z_1 <_{\tau_x} z_2$ entails $z_1 <_\tau z_2$.
- the case $(z_1, z_2) \in \{y\} \times V_{y\downarrow}$ is impossible because $z_2 \in V_{y\downarrow} \Rightarrow (z_2, y) \in E_2$ and G_2 is acyclic,
- the case $(z_1, z_2) \in V_x \times V_{y\downarrow}$ is impossible because the transitivity of G_2 would imply that $(z_1, y) \in E_2$, which contradicts the fact that $z_1 \notin V_{y\downarrow}$,
- the case $(z_1, z_2) \in V_x \times \{y\}$ also contradicts $z_1 \notin V_{y\downarrow}$.

So τ is a topological sort of G_2. But since $(x, y) \in E_1$ and $y <_\tau x$, τ is not a topological sort of G_1. That is in contradiction with the fact that $G_2 \trianglerighteq G_1$. ◀

▶ **Lemma 35.** *There exists a constructive function f such that, for all path $u \in V_N$ where the tree constant $a = lab_N(u)$ is of arity n and for all topological sort τ_0 of $G_u(N)$, $\tau = f(a, \tau_0, (GS_{u1}(N), \ldots, GS_{un}(N)))$ is a topological sort of $G_{u.[0,n]}(N)$ and, for each $j \leq n$, $j^{-1}.(\tau|_{A_j^\top})$ is a topological sort of $G_{uj}(N)$.*

Proof. We note V' the set of vertices of the graph $G_u(N)$. For all tree constant a of arity n, for all topological sort τ_0 over a subset of $A_\epsilon^\top$ and for all synthesis graphs $G_1, \ldots, G_n$ (acyclic graphs with vertices in $A_\epsilon^\top$ and edges included in $((S \cup I) \times \{\epsilon\}) \times (S \times \{\epsilon\})$ as per lemma 13), we define $f(a, \tau_0, (G_1, \ldots, G_n))$ as the topological sort τ, obtained using lemma 27, of the graph $G = \mathrm{tr}(\bigcup_{1 \leq i \leq n} i.G_i \cup G(R(a)) \cup G_{\tau_0})_{|_{V''}}$ where G_{τ_0} is the graph with set of vertices V' and set of edges $E_{\tau_0} = \{(x, y) \mid x <_{\tau_0} y\}$, and V'' is the set of vertices in $A_{[0,n]}^\top$ that are connected to the vertex $\top$ in the graph $\bigcup_{1 \leq i \leq n} i.G_i \cup G(R(a)) \cup G_{\tau_0}$.

In order to use lemma 27 we need to prove that G is acyclic. By construction, τ_0 is the only topological sort of G_{τ_0}. Since τ_0 is a topological sort of $G_u(N)$, $G_{\tau_0} \trianglerighteq G_u(N)$. According to lemma 17 $G_u(N) = \mathrm{tr}(GS_u(N) \cup GI_u(N))_{|_{A_\epsilon^\top}}$, so $G_u(N) \trianglerighteq GS_u(N) \cup GI_u(N) \trianglerighteq GI_u(N)$. Then, according to lemma 34, G_{τ_0} can be obtained from $GI_u(N)$ by adding edges. So G can be obtained from $\bigcup_{1 \leq i \leq n} i.G_i \cup G(R(a)) \cup GI_u(N)$ by adding edges. By adding these same edges to $G_u(N) = \mathrm{tr}(\bigcup_{1 \leq i \leq n} i.G_i \cup G(R(a)) \cup GI_u(N))_{|_{A_\epsilon^\top}}$ we get G_{τ_0}, which is acyclic, so according to lemma 19 G is acyclic too.

Since $\top$ is not a vertex in the graph $\bigcup_{1 \leq i \leq n} i.G_i \cup G(R(a))$ and the vertices of G_{τ_0} are the vertices of $G_u(N)$, the set V'' of vertices connected to $\top$ in $\bigcup_{1 \leq i \leq n} i.G_i \cup G(R(a)) \cup G_{\tau_0}$ is also the set of vertices connected to $\top$ in the graph $\bigcup_{1 \leq i \leq n} i.G_i \cup G(R(a)) \cup GI_u(N)$. Then, according to lemma 21, $G_{u.[0,n]}(N) = \mathrm{tr}(\bigcup_{1 \leq i \leq n} i.G_i \cup G(R(a)) \cup GI_u(N))_{|_{V''}}$. So G can be obtained from $G_{u.[0,n]}(N)$ by adding edges, therefore $G \trianglerighteq G_{u.[0,n]}(N)$. So τ is a topological sort of $G_{u.[0,n]}(N)$. ◀

▶ **Lemma 36.** *For all path $u \in V_N$ any topological sort τ of $G_{u.[0,n]}(N)$ is of the form $\tau = \tau'(\alpha, \epsilon)\top$ with $\alpha \in S$.*

Proof. Similar to proof of lemma 28. ◀

From now on, when we introduce a topological sort τ over a subset of $A^{\top}_{\epsilon}$ or $A^{\top}_{[0,n]}$, we assume it is of the form described in lemmas 28 and 36.

B.2.3 Sequentializing the computation of attributes

For all input tree N and path $u \in V_N$, a topological sort of the interface graph $G_u(N)$ gives an order in which the attibutes can be computed. The type of the output λ-term of the subtree $N \restriction_u$ then depends on the topological sort of $G_u(N)$ which gives the computation order of the attributes. That type is defined as follows:

▶ **Definition 37.** *For all topological sort τ over a subset of $A^{\top}_{\epsilon}$ (of the form described in lemma 28), we associate with τ the type $t(\tau)$ inductively defined by:*

- *if τ is of the form $(\alpha, \epsilon)\tau'$ with $\alpha \in S$ then $t(\tau) \triangleq o \times t(\tau')$,*
- *if τ is of the form $(\alpha, \epsilon)\tau'$ with $\alpha \in I$ then $t(\tau) \triangleq o \to t(\tau')$,*
- *if $\tau = (\alpha, \epsilon)\top$ where $\alpha \in S$, then $t(\tau) \triangleq o$.*

For all input tree N and path $u \in V_N$, we want to associate a λ-term with the subtree $N \restriction_u$ of N which sequentializes the computation of the attributes of the node at path u, in order to do so we use a topological sort of the interface graph at path u in N, with the following semantics:

▶ **Definition 38.** *For all topological sort τ over the set $A^{\top}_{\epsilon}$, term N and path $u \in V_N$, noting $Att(N, (\alpha, u))$ the tree associated with the attribute (α, u) in the ATT, we define $\mathcal{R}_\tau(N, u)$ by induction on τ:*

- $\mathcal{R}_{(\alpha,u)\tau'}(N, u) \triangleq \{(M_1, M_2) \mid M_1 \to^*_{\beta\eta} Att(N, (\alpha, u)), M_2 \in \mathcal{R}_{\tau'}(N, u)\}$ *if $\alpha \in S$,*
- $\mathcal{R}_{(\alpha,u)\tau'}(N, u) \triangleq \{M \mid M(Att(N, (\alpha, u))) \in \mathcal{R}_{\tau'}(N, u)\}$ *if $\alpha \in I$.*
- $\mathcal{R}_{(\alpha,\epsilon)\top}(N, u) \triangleq \{M \mid M \to^*_{\beta\eta} Att(N, (\alpha, u))\}$ *where $\alpha \in S$.*

Notice that terms in $\mathcal{R}_\tau(N, u)$ have type $t(\tau)$.

▶ **Lemma 39.** *For all terms M and M' that are $\beta\eta$-equivalent,*

$$M \in \mathcal{R}_\tau(N, u) \Leftrightarrow M' \in \mathcal{R}_\tau(N, u)$$

Proof. Straightforward induction on τ. ◀

For the purpose of clarity, we will use a special notation for the binding of variables: for binding a variable x to a term M inside a term M', in place of $(\lambda x.M')\, M$ we will write $\mathsf{let}\ x = M\ \mathsf{in}\ M'$. We want to build a λ-term which computes the term associated with a node depending on the terms associated with its child nodes. That will depend on a topological sort of the local dependency graph, which gives an order to compute the attributes of the nodes and its child nodes. We use the following definition:

▶ **Definition 40.** *For all tree constant a of arity n in Σ, for all topological sort τ over a subset of $A^{\top}_{[0,n]}$, injective substitution var which associates variables of type o with attributes and injective substitution $Cont$ which associates variables with indices between 1 and n such that for all $i \in [1, n]$, $Cont(i)$ is of type $t(\tau|_{A^{\top}_i})$, we define the term $\mathbb{M}_a(\tau, var, Cont)$ by induction on τ as follows:*

- *if* $\tau = (\alpha, 0)\top$ *with* $\alpha \in S$ *then :* $\mathbb{M}_a(\tau, var, Cont) \triangleq var(R(a)((\alpha, 0)))$
- *if* $\tau = (\alpha, 0)\tau'$ *with* $\alpha \in S$ *and* $\tau' \neq \top$ *then :*
  $\mathbb{M}_a(\tau, var, Cont) \triangleq \mathsf{let}\ y_{(\alpha,0)} = var(R(a)((\alpha, 0)))\ \mathsf{in}$
  $(y_{(\alpha,0)}, \mathbb{M}_a(\tau', var \uplus [(\alpha, 0) \to y_{(\alpha,0)}], Cont))$
- *if* $\tau = (\gamma, 0)\tau'$ *with* $\gamma \in I$ *then :*
  $\mathbb{M}_a(\tau, var, Cont) \triangleq \lambda y_{(\gamma,0)}.\mathbb{M}_a(\tau', var \uplus [(\gamma, 0) \to y_{(\gamma,0)}], Cont)$
- *if* $\tau = (\alpha, i)\tau'$ *with* $\alpha \in S$, $i \neq 0$ *and* $\tau|_{A_i^\top} \neq (\alpha, i)\top$ *then :*
  $\mathbb{M}_a(\tau, var, Cont) \triangleq \mathsf{let}\ (y_{(\alpha,i)}, X_i') = Cont(i)\ \mathsf{in}$
  $\mathbb{M}_a(\tau', var \uplus [(\alpha, i) \to y_{(\alpha,i)}], Cont \circ [i \to X_i'])$
  with X_i' *a fresh variable of type* $t(\tau'|_{A_i^\top})$.
- *if* $\tau = (\alpha, i)\tau'$ *with* $\alpha \in S$, $i \neq 0$ *and* $\tau|_{A_i^\top} = (\alpha, i)\top$ *then :*
  $\mathbb{M}_a(\tau, var, Cont) \triangleq \mathsf{let}\ y_{(\alpha,i)} = Cont(i)\ \mathsf{in}\ \mathbb{M}_a(\tau', var \uplus [(\alpha, i) \to y_{(\alpha,i)}], Cont')$
  where $Cont'$ *is* $Cont$ *from which we removed the association* $[i \to Cont(i)]$.
- *if* $\tau = (\gamma, i)\tau'$ *with* $\gamma \in I$ *and* $i \neq 0$ *then :*
  $\mathbb{M}_a(\tau, var, Cont) \triangleq \mathsf{let}\ y_{(\gamma,i)} = var(R(a)((\gamma, i)))\ \mathsf{and}\ X_i' = Cont(i)\, y_{(\gamma,i)}\ \mathsf{in}$
  $\mathbb{M}_a(\tau', var \uplus [(\gamma, i) \to y_{(\gamma,i)}], Cont \circ [i \to X_i'])$
  where X_i' *is a fresh variable of type* $t(\tau'|_{A_i^\top})$.

Then we prove that $\mathbb{M}_a$ fits the semantics we have chosen:

▶ **Lemma 41.** *For all constant* a *of arity* n *in* Σ*, and for all topological sort* τ *over a subset of* $A^\top_{[0,n]}$*, noting* $\tau_i = \tau|_{A_i^\top}$ *for* $i \leq n$*, noting* $M = \mathbb{M}_a(\tau, var, Cont)$ *where* var *is the empty substitution and for all* $i \in [1, n]$*,* $Cont(i) = X_i$ *with* X_i *a free variable of type* $t(\tau_i)$*, then* M *is of type* $t(\tau_0)$ *and, for all tree* N *and path* $u \in V_N$ *such that* $lab_N(u) = a$ *and* τ *is a topological sort of* $G_{u.[0,n]}(N)$*, for all terms* $M_1 \in \mathcal{R}_{\tau_1}(N, u1), \ldots, M_n \in \mathcal{R}_{\tau_n}(N, un)$*:*

$$M[X_1/M_1, \ldots, X_n/M_n] \in \mathcal{R}_{\tau_0}(N, u)$$

Proof. We first prove a more general claim by induction on τ:

▷ Claim 42. For all topological sort τ over a subset of $A^\top_{[0,n]}$, for all tree N and path $u \in V_N$ such that $lab_N(u) = a$ and τ is a topological sort of $G_{u.[0,n]}(N)$, for all injective mapping var from attributes to variables such that, for all $(\alpha, i) \in \tau$, all attribute appearing in $R(a)((\alpha, i))$ is either in τ or in the domain of var, for all function $Cont$ associating variables with indices $i \in [1, n]$ and for all substitution σ of the variables in $Cont$ such that $\forall i \in [1, n], \sigma(Cont(i)) \in \mathcal{R}_{\tau_i}(N, ui)$ with $\tau_i = \tau|_{A_i^\top}$:

$$\sigma \circ \nu(\mathbb{M}_a(\tau, var, Cont)) \in \mathcal{R}_{\tau_0}(N, u)$$

where ν is the variable substitution such that for all attribute (α, i) in $\mathsf{dom}(var)$: $\nu(var((\alpha, i))) = Att(N, (\alpha, ui))$.

Proof. We fix a topological sort τ over a subset of $A^\top_{[0,n]}$, an input tree N, a path $u \in V_N$ such that $lab_N(u) = a$ and τ is a topological sort of $G_{u.[0,n]}(N)$, an injective mapping var from attributes to variables such that, noting $\mathsf{dom}(var)$ its domain, for all $(\alpha, i) \in \tau$, all attribute appearing in $R(a)((\alpha, i))$ is either in τ or in $\mathsf{dom}(var)$. We note ν the variable substitution such that for all attribute $(\alpha, i) \in \mathsf{dom}(var)$, $\nu(var((\alpha, i))) = Att(N, (\alpha, ui))$ (exists because var is injective), we also fix a function $Cont$ associating variables with indices in $[1, n]$, and a substitution σ of the free variables in $Cont$ such that $\forall i \in [1, n], \sigma(Cont(i)) \in \mathcal{R}_{\tau_i}(N, ui)$ where $\tau_i = \tau|_{A_i^\top}$.

We assume the induction hypothesis for all topological sort τ' shorter (with a smaller number of elements) than τ.

As in the definition of $\mathbb{M}_a$ we have 6 cases:

- if $\tau = (\alpha, 0)\,\top$ with $\alpha \in S$ then $\mathbb{M}_a(\tau, var, Cont) \triangleq var(R(a)((\alpha, 0)))$. In this case $\sigma \circ \nu(\mathbb{M}_a(\tau, var, Cont)) = \nu \circ var(R(a)((\alpha, 0)))$. Since all attributes appearing in $R(a)((\alpha, 0))$ are in $\mathsf{dom}(var)$, and $\forall(\alpha, i) \in \mathsf{dom}(var)$, $\nu(var((\alpha, i))) = Att(N, (\alpha, ui))$. Then by definition of $Att(N, (\alpha, u))$ with $lab_N(u) = a$: $\sigma \circ \nu(\mathbb{M}_a(\tau, var, Cont)) = Att(N, (\alpha, u)) \in \mathcal{R}_{(\alpha,0)\top}(N, u)$.
- if $\tau = (\alpha, 0)\,\tau'$ with $\alpha \in S$ and $\tau' \neq \top$ then $\mathbb{M}_a(\tau, var, Cont) \triangleq$ $\mathsf{let}\ y_{(\alpha,0)} = var(R(a)((\alpha, 0)))\ \mathsf{in}\ (y_{(\alpha,0)}, \mathbb{M}_a(\tau', var \uplus [(\alpha, 0) \to y_{(\alpha,0)}], Cont))$. The induction hypothesis implies that $\sigma \circ \nu'(\mathbb{M}_a(\tau', var, Cont)) \in \mathcal{R}_{\tau'_0}(N, u)$ where $\nu' = \nu \uplus [y_{(\alpha,0)} \to Att(N, (\alpha, u))]$. Similarly to the case $\tau = (\alpha, 0)\,\top$: $\nu(var(R(a)((\alpha, 0)))) = Att(N, (\alpha, u))$. Therefore $\sigma \circ \nu(\mathbb{M}_a((\alpha, 0)\tau', var, Cont)) \in \mathcal{R}_{(\alpha,0)\tau'_0}(N, u)$.
- if $\tau = (\gamma, 0)\,\tau'$ with $\gamma \in I$ then $\mathbb{M}_a(\tau, var, Cont) \triangleq \lambda y_{(\gamma,0)}.\mathbb{M}_a(\tau', var \uplus [(\gamma, 0) \to y_{(\gamma,0)}], Cont)$. The induction hypothesis entails that $\sigma \circ \nu'(\mathbb{M}_a(\tau', var \uplus [(\gamma, 0) \to y_{(\gamma,0)}], Cont)) \in \mathcal{R}_{\tau'_0}(N, u)$ where $\nu' = \nu \uplus [y_{(\gamma,0)} \to Att(N, (\gamma, u))]$. Then, by definition of $\mathcal{R}_{(\gamma,0)\tau'_0}(N, u)$ for $\gamma \in I$, $\sigma \circ \nu(\mathbb{M}_a(\tau, var, Cont)) \in \mathcal{R}_{(\gamma,0)\tau'_0}(N, u)$.
- if $\tau = (\alpha, i)\,\tau'$ with $\alpha \in S$, $i \neq 0$ and $\tau_i \neq (\alpha, i)\,\top$ then $\mathbb{M}_a(\tau, var, Cont) \triangleq$ $\mathsf{let}\ (y_{(\alpha,i)}, X'_i) = Cont(i)\ \mathsf{in}\ \mathbb{M}_a(\tau', var \uplus [(\alpha, i) \to y_{(\alpha,i)}], Cont \circ [i \to X'_i])$ where X'_i is a fresh variable of type $t(\tau'_i)$. Noting $(M_1, M_2) = \sigma(Cont(i)) \in \mathcal{R}_{(\alpha,i)\tau'_i}(N, ui)$, we have $M_1 \to^*_{\beta\eta} Att(N, (\alpha, ui))$ and $M_2 \in \mathcal{R}_{\tau'_i}(N, ui)$. So we apply the induction hypothesis on $\sigma' \circ \nu'(\mathbb{M}_a(\tau', var \uplus [(\alpha, i) \to y_{(\alpha,i)}], Cont \circ [i \to X'_i]))$ where $\nu' = \nu \uplus [y_{(\alpha,i)} \to Att(N, (\alpha, ui))]$ and σ' is obtained from σ by removing the association $[Cont(i) \to \sigma(Cont(i))]$ and adding $[X'_i \to M_2]$. So $\sigma \circ \nu(\mathbb{M}_a(\tau, var, Cont)) =_{\beta\eta} \sigma' \circ \nu'(\mathbb{M}_a(\tau', var \uplus [(\alpha, i) \to y_{(\alpha,i)}], Cont \circ [i \to X'_i]))$ and therefore $\sigma \circ \nu(\mathbb{M}_a(\tau, var, Cont)) \in \mathcal{R}_{\tau_0}(N, u)$.
- if $\tau = (\alpha, i)\,\tau'$ with $\alpha \in S$, $i \neq 0$ and $\tau_i = (\alpha, i)\,\top$ then $\mathbb{M}_a(\tau, var, Cont) \triangleq$ $\mathsf{let}\ y_{(\alpha,i)} = Cont(i)\ \mathsf{in}\ \mathbb{M}_a(\tau', var \uplus [(\alpha, i) \to y_{(\alpha,i)}], Cont')$ where $Cont'$ is $Cont$ from which we removed the association $i \to Cont(i)$. This case is analogous to the previous one, and with the same arguments we reach the conclusion that $\sigma \circ \nu(\mathbb{M}_a(\tau, var, Cont)) \in \mathcal{R}_{\tau_0}(N, u)$.
- if $\tau = (\gamma, i)\,\tau'$ with $\gamma \in I$ and $i \neq 0$ then $\mathbb{M}_a(\tau, var, Cont) \triangleq$ $\mathsf{let}\ y_{(\gamma,i)} = var(R(a)((\gamma, i)))\ \mathsf{and}\ X'_i = Cont(i)\ y_{(\gamma,i)}\ \mathsf{in}$ $\mathbb{M}_a(\tau', var \uplus [(\gamma, i) \to y_{(\gamma,i)}], Cont \circ [i \to X'_i])$ where X'_i is a fresh variable of type $t(\tau'_i)$. We have $\sigma(Cont(i)) \in \mathcal{R}_{(\gamma,i)\tau'_i}(N, ui)$ and $\nu(var(R(a)((\gamma, i)))) =_{\beta\eta} Att(N, (\gamma, ui))$, then $\sigma(Cont(i))\,\nu(var(R(a)((\gamma, i)))) \in \mathcal{R}_{\tau'_i}(N, ui)$. We apply the induction hypothesis on $\sigma' \circ \nu'(\mathbb{M}_a(\tau', var \uplus [(\gamma, i) \to y_{(\gamma,i)}], Cont \circ [i \to X'_i]))$ where $\nu' = \nu \uplus [y_{(\gamma,i)} \to Att(N, (\gamma, ui))]$ and σ' is obtained from σ by removing the association $[Cont(i) \to \sigma(Cont(i))]$ and adding $[X'_i \to \sigma(Cont(i))\ \nu(var(R(a)((\gamma, i))))]$. Therefore $\sigma \circ \nu(\mathbb{M}_a(\tau, var, Cont)) = \sigma' \circ \nu'(\mathbb{M}_a(\tau', var \uplus [(\gamma, i) \to y_{(\gamma,i)}], Cont \circ [i \to X'_i])) \in \mathcal{R}_{\tau_0}(N, u)$.

This ends the inductive proof of the claim. ◀

Since τ is a topological sort of the graph $G_{u.[0,n]}(N)$, for all $(\alpha, i) \in \tau$, all attribute appearing in $R(a)((\alpha, i))$ is in τ. Therefore we can apply the claim on τ with $Cont(i)$ the substitution such that $Cont(i) = X_i$ for $i \in [1, n]$, σ the substitution such that $\sigma(X_i) = M_i$ for $i \in [1, n]$ and var and ν empty substitutions. So :

$$\mathbb{M}_a(\tau, var, Cont)[X_1/M_1, \ldots, X_n/M_n] = \sigma(\mathbb{M}_a(\tau, var, Cont)) \in \mathcal{R}_{\tau_0}(N, u)$$

◀

Now that we have shown that $\mathbb{M}$ computes terms correctly, we need to prove that it is *almost linear* in general, and *linear* if our ATT is *single use restricted*.

▶ **Lemma 43.** *For all tree constant a of arity n in Σ, for all topological sort τ over a subset of $A^{\top}_{[0,n]}$, injective substitution var which associates variables of type o with attributes and injective substitution $Cont$ which associates variables with indices between 1 and n such that, for all $i \in [1,n]$, $Cont(i)$ is of type $t(\tau|_{A^{\top}_i})$, the term $\mathbb{M}_a(\tau, var, Cont)$ is* almost linear.

Proof. In the inductive definition of $\mathbb{M}_a(\tau, var, Cont)$, the variables we use are either in var or in $Cont$. Variables in var are of atomic type so copying them does not prevent almost linearity. Each time a variable of $Cont$ is used, it occurs once and is removed from $Cont$ in the inductive call to $\mathbb{M}_a(\tau', var', Cont')$. So $\mathbb{M}_a(\tau, var, Cont)$ is *almost linear*. ◀

▶ **Lemma 44.** *Assumming the ATT is* single use restricted*, for all tree constant a of arity n in Σ, for all topological sort τ over a subset of $A^{\top}_{[0,n]}$, injective substitution var which associates variables of type o with attributes and injective substitution $Cont$ which associates variables with indices between 1 and n such that, for all $i \in [1,n]$, $Cont(i)$ is of type $t(\tau|_{A^{\top}_i})$, the term $\mathbb{M}_a(\tau, var, Cont)$ is* linear.

Proof. As we saw in the previous lemma, variables in $Cont$ are never copied, so we only need to prove that variables in var are not copied.

According to corollary 25, since the ATT is *single use restricted*, the graph $G_{u.[0,n]}(N)$ is a tree. For all attribute (α, i) in $G_{u.[0,n]}(N)$ there exists a unique attribute x in $G_{u.[0,n]}(N)$ such that there is an edge $((\alpha, i), x)$ in $G_{u.[0,n]}(N)$. So x is the only attribute in $G_{u.[0,n]}(N)$ such that (α, i) occurs in $R(a)(x)$.

A straightforward induction on τ proves that for all τ, var and $Cont$ such that $var((\alpha, i)) = y_{(\alpha,i)}$, the number of occurrences of $y_{(\alpha,i)}$ in $\mathbb{M}_a(\tau, var, Cont)$ is 1 if x is in τ and 0 otherwise.

Therefore the term $\mathbb{M}_a(\tau, var, Cont)$ is *linear*. ◀

Then we define the term that will compute the inherited attributes of the root node of an input tree by applying the *root* equations:

▶ **Definition 45.** *With $G(root)$ the graph whose set of vertices is $A^{\top}_{\epsilon}$ an edges represent dependencies in the root equations; for all subsort τ of a topological sort of $G(root)$, injective substitution var which associates variables of type o with attributes, and variable X_0 of type $t(\tau)$, we define the term $\mathbb{M}_{root}(\tau, var, X_0)$ of type o by induction on τ as follows:*

- *if $\tau = (\alpha, 0)\,\top$ with $\alpha \in S$ then :* $\mathbb{M}_{root}(\tau, var, X_0) \triangleq X_0$
- *if $\tau = (\alpha, 0)\,\tau'$ with $\alpha \in S$ and $\tau' \neq \top$ then :*
  $\mathbb{M}_{root}(\tau, var, X_0) \triangleq \mathsf{let}\ (y_{(\alpha,0)}, X'_0) = X_0\ \mathsf{in}\ \mathbb{M}_{root}(\tau', var \uplus [(\alpha, 0) \to y_{(\alpha,0)}], X'_0)$
- *if $\tau = (\gamma, 0)\,\tau'$ with $\gamma \in I$ then :* $\mathbb{M}_{root}(\tau, var, X_0) \triangleq$
  $\mathsf{let}\ y_{(\gamma,0)} = var(root((\gamma, 0)))\ \mathsf{and}\ X'_0 = X_0\ y_{(\gamma,0)}\ \mathsf{in}\ \mathbb{M}_{root}(\tau', var \uplus [(\gamma, 0) \to y_{(\gamma,0)}], X'_0)$
  where X'_0 is a fresh variable of type $t(\tau')$.

For all subsort τ of a topological sort of $G(root)$ we define the term $\mathbb{M}_{root}(\tau)$ as the term $\lambda X_0.\mathbb{M}_{root}(\tau, var, X_0)$ where var is the empty substitution and X_0 is a free variable of type $t(\tau)$.

Then we prove that $\mathbb{M}_{root}$ computes the right output:

▶ **Lemma 46.** *For all subsort τ of a topological sort of $G(root)$, for all tree N such that τ is a topological sort of $G_{\epsilon}(N)$ and for all term $M_0 \in \mathcal{R}_{\tau}(N, \epsilon)$, the term $\mathbb{M}_{root}(\tau)\,M_0$ β-reduces to the output of the ATT on input N.*

Proof. Similar to lemma 41. ◀

▶ **Lemma 47.** *For all subsort τ of a topological sort of $G(root)$, injective substitution var which associates variables of type o with attributes and variable X_0 of type $t(\tau)$, the term $\mathbb{M}_{root}(\tau, var, X_0)$ is* almost linear *in general and* linear *if the ATT is* single use restricted.

Proof. Similar to lemmas 43 and 44. ◀

▶ **Definition 48.** *Let $T = (\Sigma_1, \Sigma_2, S, I, out, R, root)$ be an ATT.*
We define the HODTR$_{al}$ $\mathbb{HO}(T) \triangleq (\Sigma_Q, \Sigma_1, \Sigma_2, q_0, R', \mathtt{A})$ by:

- $\mathtt{A}$*, the look-ahead automaton, is the bottom-up tree automaton given by lemma 15,*
- Σ_Q *is the signature of the set of states, which is* $Q \triangleq \{q_0\} \cup \{q_{\tau\,(\alpha,\epsilon)\,\top} \mid \tau\,(\alpha,\epsilon)\,\top$ *is a topological sort on a subset of $A^{\top}_{\epsilon}$ and $\alpha \in S\}$, the number of states is $|Q| = |S \cup I|!$. The type of a state q_τ is $o_1 \to t(\tau)$, where $t(\tau)$ is defined in definition 37,*
- Σ_1 *and* Σ_2 *are respectively the input and output tree signatures from the ATT T,*
- R' *is the set of rules, it includes the rules the form:*

$$q_{\tau_0}(a\,\overrightarrow{x})\langle\overrightarrow{\ell}\rangle \to M(q_{\tau_1}x_1)\dots(q_{\tau_n}x_n)$$

where $\overrightarrow{\ell} = \ell_1, \dots, \ell_n$ are the states of look-ahead associated with the subtrees $\overrightarrow{x} = x_1, \dots, x_n$ respectively and, noting $\tau = f(a, \tau_0, (\ell_1, \dots, \ell_n))$ the topological sort computed in lemma 35, for all $1 \le j \le n$: τ_j is the topological sort $\tau_j = j^{-1}.(\tau|_{A^{\top}_j})$. And with $M =$ let $X_1 = q_{\tau_1}(x_1)$ and $\dots X_n = q_{\tau_n}(x_n)$ in $\mathbb{M}_a(\tau, var, Cont)$ where var is the empty substitution, $Cont = [i \to X_i]_{i\in[1,n]}$ and $\mathbb{M}_a$ is defined in definition 40.
To that first set of rules we add special rules for the initial state q_0 : for all rule already in R' of the form $q_{\tau_0}(a\,\overrightarrow{x})\langle\overrightarrow{\ell}\rangle \to M$ where τ_0 is a subsort of a topological sort of $G(root)$, we add to R' the rule:

$$q_0(a\,\overrightarrow{x})\langle\overrightarrow{\ell}\rangle \to \mathbb{M}_{root}(\tau_0)\,M$$

A complexity analysis on the size of $\mathbb{HO}(T)$ reveals that, noting $m = |S| + |I|$ the number of attributes, n the maximum arity of a symbol in Σ_1 and p the number of symbols in Σ_1, the number of states in the look-ahead automaton of $\mathbb{HO}(T)$ grows in e^{m^2} (graphs with attributes as vertices), the number of states of $\mathbb{HO}(T)$ grows with $m!$ (orderings on the set of attributes). Then the number of rules of $\mathbb{HO}(T)$ grows in $m! * p * e^{m^2 * n}$ and the size of these rules grows linearly with the size of the rules of T and the number m of attributes. Note that the only non-linear factor is $m! * e^{m^2 * n}$ and comes from the potentially big numbers of accessible synthesis graphs and topological sorts of synthesis graphs, which could be smaller in practical cases.

▶ **Theorem 49.** *For all ATT T, the HODTR$_{al}$ $T' = \mathbb{HO}(T)$ is equivalent to T, and T' is linear if T is* single use restricted.

Proof. Let N be an input tree of T'.

For all path $u \in V_N$, according to lemma 15, the look-ahead state associated with the node at path u in N is the synthesis graph $GS_u(N)$ of N at path u.

Then a straighforward downward induction using lemma 35 shows that for all non-ϵ path $u \in V_N$ the node at path u in N is processed by a state of the form q_τ where τ is a topological sort of $G_u(N)$.

A straighforward upward induction using lemma 41 proves that for all non-ϵ path $u \in V_N$ the result of the computation of $q_\tau(N\ \lfloor_u)$ is a term in $\mathcal{R}_\tau(N, u)$.

Finally, using lemma 46, we conclude that $q_0(N)$ computes exactly the output of the ATT T on the input tree N. Thus we have shown that T' computes the same transduction as T.

Furthermore, lemmas 43, 44 and 47 imply that T' is almost linear in general and linear if T is *single use restricted*. ◀

▶ **Theorem 50.** *For all ATT T and relabeling attribute grammar P there exists a HODTR$_{al}$ T' equivalent to $P \circ T$ and if T is* single use restricted *then T' is linear.*

Proof. The relabeling P can be modeled by a simple HODTR$_{\text{lin}}$. Then we can compose it with $\mathbb{HO}(T)$ in order to obtain a HODTR$_{\text{al}}$ T' equivalent to $P \circ T$ such that if T is *single use restricted* then $\mathbb{HO}(T)$ is linear and therefore T' is also linear. ◀

▶ **Corollary 51.** *The class MSOT is included in the class HODTR$_{lin}$ and the class MSOTS is included in the class HODTR$_{al}$.*

B.3 HODTR$_{\text{al}} \subseteq$ REL $\circ$ ATT and HODTR$_{\text{lin}} \subseteq$ REL $\circ$ ATT$_{sur}$

▶ **Theorem 52.** *For all HODTR$_{al}$ $T = (\Sigma_Q, \Sigma_1, \Sigma_2, q_0, R, \mathtt{A})$ there exists a relabeling attribute grammar P and an ATT T' such that T is equivalent to $P \circ T'$ and, if T is linear, then T' is* single use restricted.

Proof. First we assume that T is the result of the order reduction procedure described in the proof of theorem 2, so the result of applying a state $q \in Q$ to an input tree N is a tuple of tree contexts: $q(N) \to_T (C_1, \ldots, C_n)$.

The relabeling attribute grammar framework is powerful enough to simulate the bottom-up look-ahead automaton and the top-down finite state structure of T. Therefore we can build a relabeling attribute grammar P that computes, for each node of an input tree N, which rule of T would be applied to it. Then T' will compute the actual results of applying these rules.

Since each state q of T computes a tuple of contexts, we need attributes to simulate tree contexts. We can do this by mapping the free variables of a tree context to inherited attributes, and mapping the tree context to a synthesized attribute. For example a tree context $C_1 = f\, y_1\, y_2$, where f is a tree constant of arity 2 and y_1 and y_2 are free variables, will be represented by one synthesized attribute α_1 linked to two inherited attributes β_1 and β_2 by the equation: $(\alpha_1, \epsilon) = f\,(\beta_1, \epsilon)\,(\beta_2, \epsilon)$. This way we can build an ATT T' such that $P \circ T'$ is equivalent to T.

Furthermore, if T is linear, then each tree context is used exactly once, so attributes are never used twice and T' is *single use restricted*. ◀

▶ **Corollary 53.** *HODTR$_{al}$* $\subseteq$ REL $\circ$ *ATT and HODTR$_{lin}$* $\subseteq$ REL $\circ$ *ATT$_{sur}$.*

Finally we can conclude, thanks to theorem 10, that HODTR$_{\text{al}}$ = MSOTS and HODTR$_{\text{lin}}$ = MSOT (theorem 3).

C Composition

C.1 Proof of theorem 5

In order to prove that for all term M of type A and all token $f \in [\![A]\!]$ there is at most one derivation of the judgement $\vdash M : f$, we first need to introduce known definitions and properties of coherent spaces under the framework of linear logic, as first introduced by Girard in [14].

Coherent spaces

Our main goal now is to indicate that for all term M of type A and all token $f \in [\![A]\!]$ which corresponds to a behaviour of M, there is only one possible derivation for the judgement $\vdash M : f$, which will be the key trick to preserve linearity in composition. In order to prove that, we will see that tokens form a coherent space.

First, we define a coherence relation $\frown_A \subseteq [\![A]\!] \times [\![A]\!]$ for all type A by induction on A:

▶ **Definition 54.** *For all $p, p' \in P$ and $\ell, \ell' \in L_2$,*

$$(p, \ell) \frown_{o_2} (p', \ell') \Leftrightarrow \ell = \ell'$$

For all type $A, B \in \text{types}(o_2)$, for all $f, f' \in [\![A]\!]$ and $g, g' \in [\![B]\!]$:

$$f \multimap g \frown_{A \to B} f' \multimap g' \Leftrightarrow (f \frown_A f' \Rightarrow (g \frown_B g' \wedge (f \neq f' \Rightarrow g \neq g')))$$

Intuitively, two tokens are coherent if they can both be derived from the same term. For tokens of a tree for instance, that means that they must share the same look-ahead.

We also define the corresponding incoherence relation $\asymp_A \in [\![A]\!] \times [\![A]\!]$: intuitively, two tokens are incoherent if they can not both be possible distinct tokens for the same term, so if they are either not coherent together, or if they are equal.

▶ **Definition 55.** *For all type A built on o_2:*

$$f \asymp_A f' \Leftrightarrow \neg(f \frown_A f') \vee f = f'$$

The incoherence relation allows us to give a simpler alternative definition of the coherence relation $\frown_{A \leftarrow B}$ between tokens in $[\![A \to B]\!]$: for all $f, f' \in [\![A]\!]$ and $g, g' \in [\![B]\!]$,

$$f \multimap g \frown_{A \to B} f' \multimap g' \Leftrightarrow (f \frown_A f' \Rightarrow g \frown_B g') \wedge (g \asymp_B g' \Rightarrow f \asymp_A f')$$

▶ **Theorem 56.** *For all type A and term M^A of type A, if there exists two semantic tokens $f, f' \in [\![A]\!]$ associated with M^A, i.e. the judgments $\vdash M : f$ and $\vdash M : f'$ are derivable, then f and f' are coherent: $f \frown_A f'$.*

In order to prove this theorem, we need to prove a stronger theorem, by induction on term M:

▶ **Theorem 57.** *If there exists two derivations $\mathcal{D} :: \Gamma \vdash M : f$ and $\mathcal{D}' :: \Gamma' \vdash M : f'$ then $\Gamma \multimap f \frown \Gamma' \multimap f'$.*

Here, when writing $\Gamma \multimap f$ with $\Gamma = x_1 : f_1, \ldots, x_n : f_n$, we mean by Γ the tensor product $(f_1, \ldots, f_n)$.

Proof. We prove this by induction on term M:

If $M = a$ is a constant from Σ_2 then the last rules of $\mathcal{D}$ and $\mathcal{D}'$ are:

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

$$\mathcal{D}' :: \frac{p'(a\ \overrightarrow{x})\langle \ell'_1, \ldots, \ell'_n \rangle \xrightarrow{T_2} M(p'_1\ x_1) \ldots (p'_n\ x_n) \qquad \mathtt{A}_2(a\ (\ell'_1, \ldots, \ell'_n)) = \ell'}{\vdash a : (p'_1, \ell'_1) \multimap \cdots \multimap (p'_n, \ell'_n) \multimap (p', \ell')}$$

If $((p_1,\ell_1),\dots,(p_n,\ell_n)) \subset ((p'_1,\ell'_1),\dots,(p'_n,\ell'_n))$ then $(\ell_1,\dots,\ell_n) = (\ell'_1,\dots,\ell'_n)$, therefore $\ell = \ell'$ and so $(p,\ell) \subset (p',\ell')$. If $((p_1,\ell_1),\dots,(p_n,\ell_n)) \subset ((p'_1,\ell'_1),\dots,(p'_n,\ell'_n))$ and $(p,\ell) = (p',\ell')$ then $p = p'$ and, since $\ell_i = \ell'_i$ for all i and T_2 is deterministic, $p_i = p'_i$ for all i. This shows that $((p_1,\ell_1),\dots,(p_n,\ell_n)) \subset ((p'_1,\ell'_1),\dots,(p'_n,\ell'_n))$ and $(p,\ell) = (p',\ell')$ implies that $((p_1,\ell_1),\dots,(p_n,\ell_n)) = ((p'_1,\ell'_1),\dots,(p'_n,\ell'_n))$. As a consequence, $((p_1,\ell_1),\dots,(p_n,\ell_n)) \multimap (p,\ell) \subset ((p'_1,\ell'_1),\dots,(p'_n,\ell'_n)) \multimap (p',\ell')$. So we have shown the equivalent statement: $(p_1,\ell_1) \multimap \dots (p_n,\ell_n) \multimap (p,\ell) \subset (p'_1,\ell'_1) \multimap \dots (p'_n,\ell'_n) \multimap (p',\ell')$.

If $M = N_1\, N_2$ then the last rules of $\mathcal{D}$ and $\mathcal{D}'$ respectively are of the form:

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

Through the induction hypothesis, we get that $\Gamma_1 \multimap (g \multimap f) \subset \Gamma'_1 \multimap (g' \multimap f')$ and $\Gamma_2 \multimap g \subset \Gamma'_2 \multimap g'$. Then $\Gamma_1,\Gamma_2 \subset \Gamma'_1,\Gamma'_2$ implies that $\Gamma_1 \subset \Gamma'_1$ and $\Gamma_2 \subset \Gamma'_2$, which means that $g \multimap f \subset g' \multimap f'$ and $g \subset g'$, which in turn implies that $f \subset f'$. Reciprocally, assuming that $f \asymp f'$, we have two cases depending on whether or not $g \asymp g'$. On the one hand we have that $g \asymp g'$ implies that $\Gamma_2 \asymp \Gamma'_2$ and therefore $\Gamma_1,\Gamma_2 \asymp \Gamma'_1,\Gamma'_2$, on the other hand we have that $f \asymp f'$ and $g \subset g'$ imply that $g \multimap f \asymp g' \multimap f'$ and so $\Gamma_1 \asymp \Gamma'_1$ and $\Gamma_1,\Gamma_2 \asymp \Gamma'_1,\Gamma'_2$. In either case $f \asymp f'$ implies that $\Gamma_1,\Gamma_2 \asymp \Gamma'_1,\Gamma'_2$. Finally we can conclude that $\Gamma_1,\Gamma_2 \multimap f \subset \Gamma'_1,\Gamma'_2 \multimap f'$

If $M = \lambda x^B.N$ then $f = g \multimap h$, $f' = g' \multimap h'$ and the last rules of $\mathcal{D}$ and $\mathcal{D}'$ respectively are:

$$\frac{\Gamma, x^B : g \vdash N : h}{\Gamma \vdash \lambda x^B.N : g \multimap h} \qquad \frac{\Gamma', x^B : g' \vdash N : h'}{\Gamma' \vdash \lambda x^B.N : g' \multimap h'}$$

The induction hypothesis gives $(\Gamma, x^B : g) \multimap h \subset (\Gamma', x^B : g') \multimap h'$, which we can write: $(\Gamma, g) \multimap h \subset (\Gamma', g') \multimap h'$ using the tensor product, and that is equivalent to $\Gamma \multimap (g \multimap h) \subset \Gamma' \multimap (g' \multimap h')$.

If $M = x^A$ then $f, f' \in [\![A]\!]$. So $\Gamma = x^A : f$ and $\Gamma' = x^A : f'$ and derivations $\mathcal{D}$ and $\mathcal{D}'$ are:

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

Trivially $f \subset f' \;\Rightarrow\; f \subset f'$ and $f \asymp f' \;\Rightarrow\; f \asymp f'$, therefore $f \multimap f \subset f' \multimap f'$. So $\Gamma \multimap f \subset \Gamma' \multimap f'$.

We have shown theorem 57, of which theorem 56 is a particular case, by induction on M. Indeed if M is a closed term and Γ and Γ' are empty substitutions then $\Gamma \multimap f$ is f and $\Gamma' \multimap f'$ is f', therefore $f \subset f'$. ◀

We have shown that any two tokens derivable for a same term are coherent. So the set of tokens derivable for a given term M^A form a clique in the coherence graph of $[\![A]\!]$, we call it the *coherent state* of term M^A in $[\![A]\!]$.

Now, using the previous theorem, we will be able to prove that there is only one way of deriving any given derivable judgement $\vdash M : f$.

Unicity of derivation for semantic token judgements

We can now prove theorem 5:

Proof. Because subterms of M may have free variables, we add a substitution Γ to the induction hypothesis:

"If there exists two derivations $\mathcal{D} :: \Gamma \vdash M : f$ and $\mathcal{D}' :: \Gamma \vdash M : f$ then $\mathcal{D}$ and $\mathcal{D}'$ are the same."

We prove this by induction on term M, so there are four distinct cases.

If $M = a$ is a constant from Σ_2 or if $M = x$ is a free variable in Γ then derivations $\mathcal{D}$ and $\mathcal{D}'$ are axioms so they must be equal.

If $M = N_1\, N_2$ then the last rules of $\mathcal{D}$ and $\mathcal{D}'$ respectively are of the form:

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

where $\Gamma_1, \Gamma_2 = \Gamma = \Gamma'_1, \Gamma'_2$. Since the variables substituted by substitutions Γ_1 and Γ'_1 must be the free variables in term N_1, $\Gamma_1 = \Gamma'_1$ (because $\mathsf{dom}(\Gamma_1) = \mathsf{FV}(N_1) = \mathsf{dom}(\Gamma'_1)$). Similarly, we deduce that $\Gamma_2 = \Gamma'_2$. Then we can apply theorem 57 to the derivations of $\Gamma_2 \vdash N_2 : g$ and $\Gamma'_2 \vdash N_2 : g'$, and to the derivations of $\Gamma_1 \vdash N_1 : g \multimap f$ and $\Gamma'_1 \vdash N_1 : g' \multimap f$. The first application yields $g \supset g'$ (since $\Gamma_2 = \Gamma'_2$), the second yields $g \multimap f \supset g' \multimap f$ (because $\Gamma_1 = \Gamma'_1$), together they imply that $g = g'$. Finally we can apply the induction hypothesis to get unicity of a derivation of $\Gamma_1 \vdash N_1 : g \multimap f$ and unicity of a derivation of $\Gamma_2 \vdash N_2 : g$, this implies that derivations $\mathcal{D}$ and $\mathcal{D}'$ are the same.

If $M = \lambda x^B.N$ then $f = g \multimap h$ and the last rule of $\mathcal{D}$ and $\mathcal{D}'$ is the same:

$$\frac{\Gamma, x^B : g \vdash N : h}{\Gamma \vdash \lambda x^B.N : g \multimap h}$$

The induction hypothesis implies the unicity of a derivation of $\Gamma, x^B : g \vdash N : h$, which entails the unicity of a derivation of $\Gamma \vdash \lambda x^B.N : g \multimap h$. ◀

C.2 Proof of theorem 7

First we need to prove that collapsed derivations of semantic tokens accurately represent the application of T_2 on terms, in order to do so we use a logical relation.

Logical relation

Our logical relation is indexed on a type A and a semantic token $f \in [\![A]\!]$, it is defined as follows:

▶ **Definition 58.** *We define the logical relation R^A_f, for all type A built on atomic type o_2 and for all semantic token $f \in [\![A]\!]$, by induction on type A:*

$$R^{o_2}_{(p,\ell)} = \{(M, N) \mid p(M\!\downarrow_\beta) \overset{T_2}{=} N\!\downarrow_\beta, \mathtt{A}_2(M\!\downarrow_\beta) = \ell\}$$

$$R^{A\to B}_{f\multimap g} = \{(M, N) \mid \forall (M', N') \in R^A_f, (M\, M', N\, N') \in R^B_g\}$$

Now we prove the *adequation* of this logical relation: for all type $A \in \mathrm{types}(o_2)$, token $f \in [\![A]\!]$ and for any closed terms M and N of respective types A and $\overline{f}$:

$$\exists \mathcal{D} ::\vdash M : f \;\text{ and }\; \overline{\mathcal{D}} =_{\beta\eta} N \;\Rightarrow\; (M, N) \in R^A_f$$

We prove a more general claim by induction on term M:

▶ **Theorem 59.** *For all type $A \in \text{types}(o_2)$, token $f \in [\![A]\!]$, terms M of type A and N of type $\overline{f}$. For all substitutions of variables Γ and σ such that $\Gamma(x) = g \Rightarrow \sigma(x) \in R_g^B$ and $\mathsf{dom}(\Gamma) = \mathsf{FV}(M)$:*

$$\exists \mathcal{D} :: \Gamma \vdash M : f \ \wedge\ \overline{\mathcal{D}} =_{\beta\eta} N \quad \Rightarrow \quad (\, M.(\pi_1 \circ \sigma)\,,\, N.(\pi_2 \circ \sigma)\,) \in R_f^A$$

In order to prove this theorem, we first need to show that the logical relation is compatible with β-reduction (and η-expansion):

▶ **Lemma 60.** *For all type A and token $f \in [\![A]\!]$, for all terms M, N, M', N' such that $M =_{\beta\eta} M'$ and $N =_{\beta\eta} N'$: $(M, N) \in R_f^A \Rightarrow (M', N') \in R_f^A$.*

Proof. We prove this lemma by induction on type A. Let M, N, M', N' be terms such that $M =_{\beta\eta} M'$, $N =_{\beta\eta} N'$ and $(M, N) \in R_f^A$.

If $A = o_2$ and $f = (p, \ell)$ then $p(M\lfloor_\beta) \overset{T_2}{=} N\lfloor_\beta$ and $\mathtt{A}_2(M\lfloor_\beta) = \ell$. So $p(M'\lfloor_\beta) = p(M\lfloor_\beta) \overset{T_2}{=} N\lfloor_\beta = N'\lfloor_\beta$ and $\mathtt{A}_2(M'\lfloor_\beta) = \mathtt{A}_2(M\lfloor_\beta) = \ell$. In that case $(M', N') \in R_f^A$.

If $A = B \to C$ and $f = g \multimap h$ then, for all $(M_1, N_1) \in R_g^B$, $(M\,M_1, N\,N_1) \in R_h^c$. Since $M =_{\beta\eta} M'$ and $N =_{\beta\eta} N'$, we have $(M\,M_1, N\,N_1) =_{\beta\eta} (M'\,M_1, N'\,N_1)$ and, by induction hypothesis on type C, $(M'\,M_1, N'\,N_1) \in R_h^C$. So $(M', N') \in R_{g\multimap h}^{B\to C}$. ◀

We can now prove theorem 59.

Proof. We use an induction on term M.

Let $A \in \text{types}(o_2)$, token $f \in [\![A]\!]$, terms M of type A and N of type $\overline{f}$. Let Γ and σ substitutions of variables such that $\Gamma(x) = g \Rightarrow \sigma(x) \in R_g^B$ and $\mathsf{dom}(\Gamma) = \mathsf{FV}(M)$. Let $\mathcal{D}$ a derivation of the judgement $\Gamma \vdash M : f$ (unique according to theorem 5). Assume that $\overline{\mathcal{D}} =_{\beta\eta} N$. We want to prove $(\, M.(\pi_1 \circ \sigma)\,,\, N.(\pi_2 \circ \sigma)\,) \in R_f^A$.

In most cases, we will show that $(M.(\pi_1 \circ \sigma), \overline{\mathcal{D}}.(\pi_2 \circ \sigma)) \in R_f^A$ and conclude using lemma 60. We distinguish four cases depending on M, one for each derivation rule as head of derivation $\mathcal{D}$:

If $M = x^A$ then the head rule of $\mathcal{D}$ is:

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

Since $\Gamma(x^A) = f$, we have $\sigma(x^A) \in R_f^A$. So $(M.(\pi_1 \circ \sigma), N.(\pi_2 \circ \sigma)) = (\pi_1(\sigma(x^A)), \pi_2(\sigma(x^A))) \in R_f^A$.

If $M = M_1\, M_2$ then the head rule of $\mathcal{D}$ is:

$$\frac{\mathcal{D}_1 :: \Gamma_1 \vdash M_1 : f' \multimap f \qquad \mathcal{D}_2 :: \Gamma_2 \vdash M_2 : f'}{\Gamma_1, \Gamma_2 \vdash M_1\, M_2 : f}$$

where $\Gamma = \Gamma_1, \Gamma_2$ such that the domains of Γ_1 and Γ_2 are the sets of free variables of M_1 and M_2 respectively. Similarly, we can split substitution σ into σ_1 and σ_2 in order to apply the induction hypothesis on $\mathcal{D}_1$ with σ_1 and on $\mathcal{D}_2$ with σ_2. Noting B the type of M_2 we get:

$$(M_1.(\pi_1 \circ \sigma_1), \overline{\mathcal{D}_1}.(\pi_2 \circ \sigma_1)) \in R_{f'\multimap f}^{B\to A} \qquad (M_2.(\pi_1 \circ \sigma_2), \overline{\mathcal{D}_2}.(\pi_2 \circ \sigma_2)) \in R_{f'}^B$$

By definition of $R_{f'\multimap f}^{B\to A}$ we get $(M_1.(\pi_1 \circ \sigma_1) M_2.(\pi_1 \circ \sigma_2), \overline{\mathcal{D}_1}.(\pi_2 \circ \sigma_1)\overline{\mathcal{D}_2}.(\pi_2 \circ \sigma_2)) \in R_f^A$. So $((M_1\, M_2).(\pi_1 \circ \sigma), (\overline{\mathcal{D}_1}\,\overline{\mathcal{D}_2}).(\pi_2 \circ \sigma)) \in R_f^A$. Since $\overline{\mathcal{D}} = \overline{\mathcal{D}_1}\,\overline{\mathcal{D}_2}$, we conclude using lemma 60.

If $M = \lambda x^B.M'$ then the head rule of $\mathcal{D}$ is:

$$\frac{\mathcal{D}' :: \Gamma, x^B : g \vdash M' : f'}{\Gamma \vdash \lambda x^B.M' : g \multimap f'}$$

where $A = B \to C$ and $f = g \multimap f'$. First we show that $(\lambda x.M'.(\pi_1 \circ \sigma), \lambda x.\overline{\mathcal{D}'}.(\pi_2 \circ \sigma)) \in R^{B\to C}_{g\multimap f'}$. Let $(M_0, N_0) \in R^B_g$. In order to use the induction hypothesis we define $\Gamma' = \Gamma, x^B : g$ and the substitution $\sigma' = \sigma \circ [x \leftarrow (M_0, N_0)]$, then: $(M'.(\pi_1 \circ \sigma'), \overline{\mathcal{D}'}.(\pi_2 \circ \sigma')) \in R^C_{f'}$. Because of the definition of σ' we have: $(\lambda x.M'.(\pi_1 \circ \sigma))\, M_0 =_{\beta\eta} M'.(\pi_1 \circ \sigma')$ and $(\lambda x.\overline{\mathcal{D}'}.(\pi_2 \circ \sigma)) =_{\beta\eta} \overline{\mathcal{D}'}.(\pi_2 \circ \sigma')$. Using lemma 60 we deduce that $((\lambda x.M'.(\pi_1 \circ \sigma))\, M_0, (\lambda x.\overline{\mathcal{D}'}.(\pi_2 \circ \sigma))\, N_0) \in R^C_{f'}$. This proves that $(\lambda x.M'.(\pi_1 \circ \sigma), \lambda x.\overline{\mathcal{D}'}.(\pi_2 \circ \sigma)) \in R^{B\to C}_{g\multimap f'}$. We conclude using lemma 60.

If $M = a$ then the head rule of $\mathcal{D}$ is:

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

Since Γ is the empty substitution, we only need to prove $(M, N) \in R^{o_2 \to \dots\, o_2}_{(p_1,\ell_1)\multimap \dots \multimap (p,\ell)}$. In order to do this we define the property $\mathcal{P}(i)$ for $0 \le i \le n$ by:

$$\mathcal{P}(i) = \text{"For all } (M_1, N_1) \in R^{o_2}_{(p_1,\ell_1)}, \dots, (M_i, N_i) \in R^{o_2}_{(p_i,\ell_i)},$$
$$\text{we have } (M\,M_1 \dots M_i, N'\,N_1 \dots N_i) \in R^{o_2 \to \dots \to o_2}_{(p_{i+1},\ell_{i+1})\multimap \dots\, (p_n,\ell_n)\multimap (p,\ell)}\text{"}$$

We prove $\mathcal{P}(i)$ by downward induction for $0 \le i \le n$.

We start by proving $\mathcal{P}(n)$:

let $(M_1, N_1) \in R^{o_2}_{(p_1,\ell_1)}, \dots, (M_n, N_n) \in R^{o_2}_{(p_n,\ell_n)}$. So for all $i \le n$, we have $p_i(M_i\lfloor_\beta) \overset{T_2}{\equiv} N_i\lfloor_\beta$ and $\mathtt{A}_2(M_i\lfloor_\beta) = \ell_i$. Now we look at $p(M\,M_1 \dots M_n\lfloor_\beta)$:

$$\begin{aligned} p((M\,M_1 \dots M_n)\lfloor_\beta) &= p(a\,(M_1\lfloor_\beta)\dots(M_n\lfloor_\beta)) \\ &\overset{T_2}{\equiv} N'\,(p_1(M_1\lfloor_\beta))\dots(p_n(M_n\lfloor_\beta)) \\ &\overset{T_2}{\equiv} N'\,(N_1\lfloor_\beta)\dots(N_n\lfloor_\beta) \\ &\overset{T_2}{\equiv} (N'\,N_1 \dots N_n)\lfloor_\beta \end{aligned}$$

Note that we can apply the rule of T_2 because we know that $\mathtt{A}_2(M_i\lfloor_\beta) = \ell_i$ for all $i \le n$. Then we check $\mathtt{A}_2((M\,M_1 \dots M_n)\lfloor_\beta)$:

$$\begin{aligned} \mathtt{A}_2((M\,M_1 \dots M_n)\lfloor_\beta) &= \mathtt{A}_2(a\,(M_1\lfloor_\beta)\dots(M_n\lfloor_\beta)) \\ &= \mathtt{A}_2(a\,\ell_1 \dots \ell_n) \\ &= \ell \end{aligned}$$

We have shown $\mathcal{P}(n) =$ "$(M\,M_1 \dots M_n, N'\,N_1 \dots N_n) \in R^{o_2}_{(p,\ell)}$".

Next we prove the induction step, for $1 \le j \le n$, $\mathcal{P}(j) \Rightarrow \mathcal{P}(j-1)$: we assume $\mathcal{P}(j)$ and want to prove $\mathcal{P}(j-1)$.

Let $(M_1, N_1) \in R^{o_2}_{(p_1,\ell_1)}, \dots, (M_{j-1}, N_{j-1}) \in R^{o_2}_{(p_{j-1},\ell_{j-1})}$. According to $\mathcal{P}(j)$, for all $(M_j, N_j) \in R^{o_2}_{(p_j,\ell_j)}$: $(M\,M_1 \dots M_j, N'\,N_1 \dots N_j) \in R^{o_2 \to \dots \to o_2}_{(p_{j+1},\ell_{j+1})\multimap \dots \multimap (p,\ell)}$. So $(M\,M_1 \dots M_{j-1}, N'\,N_1 \dots N_{j-1}) \in R^{o_2 \to \dots \to o_2}_{(p_j,\ell_j)\multimap (p_{j+1},\ell_{j+1})\multimap \dots \multimap (p,\ell)}$ and $\mathcal{P}(j-1)$ is true.

Therefore, by induction, $\mathcal{P}(0) =$ "$(M, N') \in R^{o_2 \to \dots\, o_2}_{(p_1,\ell_1)\multimap \dots \multimap (p,\ell)}$" is true. Since $N' = \overline{\mathcal{D}} =_{\beta\eta} N$ we can conclude that $(M, N) \in R^{o_2 \to \dots\, o_2}_{(p_1,\ell_1)\multimap \dots \multimap (p,\ell)}$ using lemma 60.

This ends the proof of theorem 59. ◀

As a corollary of theorem 59 we get that if there exists a derivation $\mathcal{D}$ of a judgement $\vdash M : f$ then $(M, \overline{\mathcal{D}}\lfloor_{\beta\eta}) \in R^A_f$.

With this corollary we can now prove theorem 7.

Proof of theorem 7

With T defined in section 4.4, we prove that $T = T_2 \circ T_1$.

Proof. We first prove the following statement by induction on a tree t of type o_1:
For all state $q \in Q$ of transducer T_1 and for all token $f \in [\![A_q]\!]$ such that $q(t) \xrightarrow{T_1} M$ and $\vdash M : f$, there exists a term N such that $(q, f)(t) \xrightarrow{T} N$ and $(M, N) \in R_f^{A_q}$.

Let $t = a\, t_1 \dots t_n$ a tree of type o_1, $q \in Q$ a state of T_1 and $f \in [\![A_q]\!]$ a token such that $q(t) \xrightarrow{T_1} M$ and $\vdash M : f$. Then there is a rule:

$$q(a\, t_1 \dots t_n) \xrightarrow{T_1} M_0\, (q_1\, t_1) \dots (q_n\, t_n)$$

If term M_0 forgets one or several of its arguments, then there exists a term M_0' which uses all its arguments such that $M_0\, (q_1\, t_1) \dots (q_n\, t_n) =_{\beta\eta} M_0'\, (q_{i_1}\, t_{i_1}) \dots (q_{i_m}\, t_{i_m})$ where $i_1, \dots, i_m$ are the indices of the arguments used by M_0. For the sake of clarity we forget this renaming of variables and proceed assuming M_0 uses all of its arguments.

Since the computation of $q(t) \xrightarrow{T_1} M$ terminates and M_0 uses all its arguments: for all $i \leq n$, the computation of $q_i(t_i)$ by T_1 terminates, we note its result M_i (a term of type A_{q_i}). Therefore $M_0\, M_1 \dots M_n \to_{\beta\eta}^* M$. So $\vdash M_0\, M_1 \dots M_n : f$ and there exists $f_1 \in [\![A_{q_1}]\!], \dots, f_n \in [\![A_{q_n}]\!]$ such that $\vdash M_0 : f_1 \multimap \dots \multimap f_n \multimap f$ and, for all $i \leq n$, $\vdash M_i : f_i$. Then we can apply the induction hypothesis to each tree t_i with state q_i and token f_i: for all $i \leq n$, there is a term N_i such that $(q_i, f_i)(t_i) \xrightarrow{T} N_i$ and $(M_i, N_i) \in R_{f_i}^{A_{q_i}}$.

Because of the rule $q(a\, t_1 \dots t_n) \xrightarrow{T_1} M_0\, (q_1 t_1) \dots (q_n t_n)$ in T_1, there must be in T a rule:

$$(q, f)(a\, t_1 \dots t_n) \xrightarrow{T} \overline{\mathcal{D}_0}\, ((q_1, f_1)(t_1)) \dots ((q_n, f_n)(t_n))$$

Where $\mathcal{D}_0$ is the derivation of the judgement $\vdash M_0 : f_1 \multimap \dots \multimap f_n \multimap f$. So $(q, f)(a\, t_1 \dots t_n) \xrightarrow{T} \overline{\mathcal{D}_0}\, N_1 \dots N_n$.

By using theorem 59 (adequation) on $\mathcal{D}_0$ we get $(M_0, \overline{\mathcal{D}_0}) \in R_{f_1 \multimap \dots f_n \multimap f}^{A_{q_1} \to \dots A_{q_n} \to A_q}$. By definition of the logical relation, we obtain $(M_0\, M_1 \dots M_n, \overline{\mathcal{D}_0}\, N_1 \dots N_n) \in R_f^{A_q}$. Finally we apply lemma 60. So, with $N = \overline{\mathcal{D}_0}\, N_1 \dots N_n$, we have $(q, f)(t) \xrightarrow{T} N$ and $(M, N) \in R_f^{A_q}$.

Let t_1 be a tree of type o_1. Assume that $T_2 \circ T_1(t_1) = t_3$. Then there is a term t_2 of type o_2 such that $q_0(t_1) \xrightarrow{T_1} t_2$ and $p_0(t_2) \xrightarrow{T_2} t_3$. Then we can derive the judgement $\vdash t_2 : (p_0, \ell)$ where ℓ is the look-ahead of T_2 on tree t_2 and p_0 is the initial state of T_2. So there exists a term N such that $(q_0, (p_0, \ell))(t_1) \xrightarrow{T} N$ and $(t_2, N) \in R_{(p_0,\ell)}^{o_2}$. By definition of the logical relation we have: $p_0(t_2) \overset{T_2}{=} N\!\downarrow_\beta$, so $t_3 = N\!\downarrow_\beta$ and $(q_0, (p_0, \ell))(t_1) \xrightarrow{T} t_3$. Thanks to the definition of R, we can conclude that $q_0'(t_1) \xrightarrow{T} t_3$. So $T_2 \circ T_1(t_1) = t_3$ implies that $T(t_1) = t_3$.

For the reverse implication, we first show by induction on tree t that, for all state $q \in Q$ and token $f \in [\![A_q]\!]$, if $(q, f)(t) \xrightarrow{T} N$ then there exists a term M such that $q(t) \xrightarrow{T_1} M$, $\vdash M : f$ and $(M, N) \in R_f^{A_q}$.

Let $t = a\ t_1 \dots t_n$ a tree of type o_1 with $(q, f)(t) \xrightarrow{T} N$. So there is a rule of T such that $(q, f)(t) \xrightarrow{T} N_0\, ((q_1, f_1)(t_1)) \dots ((q_n, f_n)(t_n))$. Then there are $N_1, \dots, N_n$ such that $(q, f)(t) \xrightarrow{T} N_0\, N_1 \dots N_n$, $N =_{\beta\eta} N_0\, N_1 \dots N_n$ and, for all $i \leq n$, $(q_i, f_i)(t_i) \xrightarrow{T} N_i$. Then we apply the induction hypothesis and get M_i such that $q_i(t_i) \xrightarrow{T_1} M_i$ and $\vdash M_i : f_i$. There is in T_1 a rule $q(t) \xrightarrow{T_1} M_0\, (q_1 t_1) \dots (q_n t_n)$, so $q(t) \xrightarrow{T_1} M_0\, M_1 \dots M_n$, with $\vdash M_0 : f_1 \multimap \dots f_n \multimap f$. So for $M = M_0\, M_1 \dots M_n$ we have $\vdash M : f$. Finally we deduce that $(M, N) \in R_f^{A_q}$ using the property we proved earlier in this proof and the lemma 60.

Now we try to show that $T(t_1) = t_3 \Rightarrow T_2 \circ T_1(t_1) = t_3$. Assume that $T(t_1) = t_3$. Then $q'_0(t_1) \xrightarrow{T} t_3$, so there exists a token $(p_0, \ell) \in [\![o_2]\!]$ such that $(q_0, (p_0, \ell))(t_1) \xrightarrow{T} t_3$. So there exists a term M such that $q(t_1) \xrightarrow{T_1} M, \vdash M : f$ and $(M, t_3) \in R^{o_2}_{(p_0,\ell)}$. Then, by definition of the logical relation: $p_0(M\downarrow_\beta) \xrightarrow{T_2} t_3$. So $T_2 \circ T_1(t_1) = t_3$.

So the transduction of T is the composition of the transductions of T_2 and T_1. ◀