A Comparison of Tree Transductions defined by Monadic Second Order Logic and by Attribute Grammars

Roderick Bloem* and Joost Engelfriet**

Department of Computer Science, Leiden University
P.O.Box 9512, 2300 RA Leiden, The Netherlands
e-mail: engelfri@wi.leidenuniv.nl

Abstract. Formulas from monadic second order (MSO) logic with one and two free variables can be used to define the nodes and edges (respectively) of a graph, in terms of a given graph. Such MSO definable graph transductions play a role in the theory of graph grammars. Here we investigate the special case of trees. The main result is that the MSO definable tree transductions are exactly those tree transductions that can be computed by attributed tree transducers with look-ahead, which are a specific type of two-stage attribute grammar: in the first (look-ahead) stage all attributes have finitely many values, in the second stage all attributes are trees, and the second stage satisfies the single use restriction (i.e., each attribute is used at most once). Moreover, if we allow the MSO transductions to produce trees with shared subtrees (i.e., term graphs, that have to be unfolded), then the single use restriction can be dropped.

1 Introduction

Formulas of monadic second order (MSO) logic can be used to express properties of labeled graphs, and, in particular, trees and strings. Thus, monadic second order logic is a convenient language for the specification of sets of graphs, relations between graphs, relations between the nodes of graphs, etcetera. Several results are known that provide formal models for the implementation of such specifications (see, e.g., [Tho, Eng5]). The classical one, proved in [Büc, Elg], is that a set of strings can be specified in monadic second order logic (by a closed formula) if and only if it can be recognized by a finite-state automaton. This was generalized in [Don, ThaWri] to sets of node-labeled ordered trees, with an appropriate generalization of the finite-state automaton to the bottom-up finite-state tree automaton. The trees considered are the usual representations of terms over a finite set of operators. In [Cou1] one direction of the result was generalized to

* The present address of the first author is: Department of Computer Science, University of Colorado at Boulder, P.O.Box 430, Boulder, CO 80303, email: Roderick.Bloem@colorado.edu

** The second author was supported by ESPRIT BRWG No.7183 COMPUGRAPH II and TMR Network GETGRATS

graphs: every set of graphs that can be specified in monadic second order logic is recognizable in the algebraic sense (see [MezWri]), for a specific algebra of graphs. This was used to show that the class of context-free graph languages, generated by context-free graph grammars, is closed under intersection with MSO definable sets of graphs.

Inspired by the ideas of [Cou1], a characterization of the context-free graph languages in terms of monadic second order logic was presented in [Oos, Eng4, EngOos], based on the following natural logical way to specify graph transductions, i.e., functions from graphs to graphs. The idea is that the output graph is specified by formulas that are interpreted on the input graph (see [ArnLagSee] for the history of the concept of interpreting one logical structure in another). More precisely, for a given input graph g_1, the nodes of the output graph g_2 are a subset of the nodes of g_1, which is specified by a unary MSO formula, i.e., a formula with one free variable ranging over the nodes of g_1; the edges of g_2 are specified by a binary MSO formula, i.e., a formula with two free variables ranging over the nodes of g_1. In fact, to define the labels of the nodes and edges of g_2, these formulas are indexed by node and edge labels, respectively. It was shown in [Oos, Eng4, EngOos] that a set of graphs is context-free iff it is the image of an MSO definable set of trees under an MSO definable graph transduction. Since it is easy to see that the MSO definable graph transductions are closed under composition, this characterization implies that the context-free graph languages are closed under MSO definable graph transductions (comparable to the closure of the context-free string languages under finite state transductions). [3]

In [Cou3] the MSO definable graph transductions were generalized, in a natural way, such that the above characterization of the context-free graph languages still holds. As a result, the context-free graph languages are even closed under this larger class of functions on graphs. The main idea of the generalization is to allow the output graph to contain a fixed number k of copies of each node of the input graph. Accordingly, the MSO formulas specifying the graph transduction are additionally indexed by numbers from 1 to k. This is the type of MSO definable function on graphs that we consider in this paper. The MSO definable functions in [EngOos, Cou3] are partial, and in [Cou3] also MSO definable relations are considered. Here, we investigate total functions only. For a survey on MSO definable graph transductions, see [Cou4].

The results discussed above show how to specify the context-free graph languages by MSO definable graph transductions, but, the other way around, they do not answer the question of how to implement MSO specifications of graph transductions, which is the topic of this paper. Since there do not seem to be

[3] It should be noted here that there are (at least) two natural classes of context-free graph languages, those generated by (hyper)edge replacement (HR, see, e.g., [DreHabKre]) and those generated by node replacement (NR, see, e.g., [EngRoz]). They are related to MSO logic with and without variables ranging over edges, respectively. The original result of [Cou1] was proved for HR (and, later, for NR in [CouEngRoz]). The characterization of [Oos, Eng4, EngOos] was proved for NR (and, later, for HR in [CouEng]). For a survey of the relationships between context-free graph grammars and monadic second order logic, see [Cou2, Cou5, Eng6].

suitable formal models for the implementation of functions on graphs, we consider in this paper the special case of trees: both input and output graph are trees. This choice is also motivated by the fact that tree transductions form a well-known and well-investigated model of syntax-directed semantics. A partial answer to the question of how to implement the MSO definable tree transductions was given in [BloEng], where it was shown how to implement unary and binary MSO formulas for trees: unary formulas can be implemented by attribute grammars of which all attributes have finitely many values (proved independently in [NevBus]), and binary formulas can be implemented by a particular type of finite-state tree-walking automaton. Since the specification of a graph transduction consists of a collection of unary and binary formulas, we can use these characterizations to obtain a model for the implementation of MSO specifications of tree transductions. The main idea is that the tree-walking automata can be turned into an attribute grammar. Thus, we obtain a characterization of the MSO definable tree transductions in terms of attribute grammars, one of the best known formal models for defining syntax-directed semantics. Of course, attribute grammars are still a specification language, but they are much closer to implementation than MSO logic, and their implementation has been studied extensively (see, e.g., [DerJouLor, Eng3, KühVog2]).

To be precise, we prove that a tree transduction is MSO definable iff it can be computed by an "attributed tree transducer with look-ahead" ($\mathrm{att}^{\mathrm{R}}$ for short). This is a transducer which consists of two attribute grammars, each computing a tree transduction, the composition of which is the tree transduction computed by the transducer. The two attribute grammars work in different ways. The first attribute grammar is a so-called relabeling attribute grammar (introduced in [BloEng]) which just preprocesses the input tree by relabeling its nodes: all attributes have finitely many values and one of the attributes holds the new label of each node. This is the look-ahead part of the $\mathrm{att}^{\mathrm{R}}$; the R stands for 'relabeling' or for 'regular look-ahead', where 'regular' is used to refer to finite-state devices (in this case an attribute grammar of which all attributes have finitely many values). The second attribute grammar is an attributed tree transducer (introduced in [Fül], see also [EngFil]) which performs the actual computation: the values of all attributes are trees, with substitution as the only operation on trees, and the output tree is the value of a designated attribute at the root of the input tree. Moreover, the second attribute grammar satisfies the so-called single use restriction (investigated in [Gan, GanGie, Gie]), which means that each attribute is used at most once. An $\mathrm{att}^{\mathrm{R}}$ can also be viewed as one attribute grammar of which the attributes can be evaluated in two phases: in the first phase only attributes with finitely many values are evaluated, called "flags" in [Eng2, Blo], and in the second phase the value of each attribute is a tree which is defined by a conditional rule, depending on the values of the flags. Intuitively, this is a more understandable model. Technically, due to the presence of conditional rules in such two-phase attribute grammars, it is more convenient to work with compositions of attribute grammars as above.

Since trees are terms and terms can be evaluated in a semantic domain, at-

tributed tree transducers are a schematic model of attribute grammars: every attribute grammar can be viewed as an attributed tree transducer followed by an evaluation of terms (cf. [Eng2]). The attributed tree transducers seem to have certain undesirable properties, see, e.g., [FülVag]. Based on the results of this paper, we think that the attributed tree transducer with look-ahead is a more attractive and robust formal model of attribute grammars. By the result of [FülVag] it is more powerful than the attributed tree transducer. This can be compared with the addition of regular look-ahead to top-down tree transducers (see [Eng1, GécSte]), which are, in fact, attributed tree transducers with synthesized attributes only.

When implementing tree transductions (as in term rewriting systems) it is natural, and efficient, to allow trees with shared subtrees: so-called term graphs (see, e.g., [SlePleEek, CorMon]). Thus, we also investigate MSO definable graph transductions of which the input graph is a tree, but the output graph is a term graph. Such a graph transduction defines a tree transduction, obtained by unfolding the output graph into the tree it represents. We prove that a tree transduction can be defined by such an MSO definable term graph transduction iff it can be computed by an attributed tree transducer with look-ahead, without the single use restriction. In fact, one of the reasons that attribute grammars are a popular tool for the implementation of syntax-directed semantics is the fact that attributes can be used several times, but are computed only once. Note that the single use restriction naturally correspond to the requirement that the term graph has no sharing, i.e., is a tree.

The structure of this paper is as follows. Section 2 contains the basic terminology on graphs, term graphs, trees, MSO logic, and attribute grammars. In Section 3 we recall the definition of MSO definable graph transduction, and define the two classes of MSO definable tree transductions that we investigate, with and without sharing of subtrees. The notion of an MSO relabeling is introduced, which is an MSO definable tree transduction that just relabels the nodes of the input tree. In Section 4 we define the attributed tree transducer with look-ahead, by recalling the definitions of a relabeling attribute grammar and an attributed tree transducer. This section also contains some basic properties of attributed tree transducers, which are used in Section 5 to show that every attributed tree transduction is MSO definable (without sharing of subtrees if the single use restriction is satisfied). In Section 6 we recall the characterization of unary and binary formulas on trees, proved in [BloEng], and we show that relabeling attribute grammars and MSO relabelings have the same power. With the result of the previous section, this shows half of our main results: every tree transduction computed by an attributed tree transducer with look-ahead can be specified in MSO logic. The other half is proved in Section 7: every MSO definable tree transduction can be computed by an $\mathrm{att}^{\mathrm{R}}$. After the proof, a detailed example is given of this implementation. The attributed tree transducer constructed in the proof is only guaranteed to be noncircular when restricted to the output trees of the relabeling attribute grammar. It is shown in Section 8 that it can be turned into a noncircular attributed tree transducer. Finally, the main results are stated and

discussed in Section 9.

Most of the results of this paper were proved as part of [Blo], the Master's Thesis of the first author.

2 Preliminaries

In this section we recall some well-known concepts concerning graphs and trees, monadic second order logic on graphs, and attribute grammars.

$\mathbb{N} = \{0, 1, 2, \dots\}$, and for $m, n \in \mathbb{N}$, $[m, n] = \{i \mid m \leq i \leq n\}$. For a set S, $\mathcal{P}(S)$ is its powerset and $\#S$ its cardinality. For binary relations R_1 and R_2, their composition is $R_1 \circ R_2 = \{(x, z) \mid \exists y : (x, y) \in R_1 \text{ and } (y, z) \in R_2\}$; note that the order of R_1 and R_2 is nonstandard. For sets of relations $\mathcal{R}_1$ and $\mathcal{R}_2$, $\mathcal{R}_1 \circ \mathcal{R}_2 = \{R_1 \circ R_2 \mid R_1 \in \mathcal{R}_1, R_2 \in \mathcal{R}_2\}$. The transitive reflexive closure of a binary relation R is denoted R^*. A binary relation R is said to be *functional* if it is a partial function, i.e., if $(x, y), (x, z) \in R$ implies $y = z$.

2.1 Graphs, term graphs, and trees

We view trees and term graphs as finite, directed graphs with labeled nodes and edges, in the usual way. Let Σ and Γ be alphabets (of node labels and edge labels, respectively). A *graph* over (Σ, Γ) is a triple (V, E, lab), with V a finite set of nodes, $E \subseteq V \times \Gamma \times V$ the set of labeled edges, and $\text{lab} : V \to \Sigma$ the node-labeling function. For a given graph g, its nodes, edges, and node-labeling function are denoted V_g, E_g, and lab_g, respectively. The set of all graphs over (Σ, Γ) is denoted $G_{\Sigma,\Gamma}$.

Let $g = (V, E, \text{lab})$ be a graph. An edge (u, γ, v) is an edge with label γ from u to v, it is an outgoing edge of u, and an incoming edge of v. We also say that u is a *parent* of v and that v is a *child* of u. Instead of $(u, \gamma, v) \in E$ we also write $u \xrightarrow{\gamma} v$. For nodes $u, v \in V$, a (directed, possibly empty) *path* from u to v is an alternating sequence $u_0 e_1 u_1 e_2 u_2 \cdots e_n u_n$ of nodes u_i and edges e_i, with $n \geq 0$ (the length of the path), $u_0 = u$, $u_n = v$, and e_i is an edge from u_{i-1} to u_i. We also say that u is an *ancestor* of v and that v is a *descendant* of u. The path is also written $u_0 \xrightarrow{\gamma_1} u_1 \xrightarrow{\gamma_2} u_2 \cdots \xrightarrow{\gamma_n} u_n$, if $e_i = (u_{i-1}, \gamma_i, u_i)$. It is a *cycle* if $n \geq 1$ and $u_0 = u_n$. A graph is *noncircular* if it has no cycles, and circular otherwise.

The trees we consider are the usual graphical representations of terms, which form the free algebra over a set of operators. An *operator alphabet* Σ is an alphabet Σ together with a *rank function* $\text{rk} : \Sigma \to \mathbb{N}$. For all $k \in \mathbb{N}$, $\Sigma_k = \{\sigma \in \Sigma \mid \text{rk}(\sigma) = k\}$ is the set of operators of rank k, i.e., with k arguments. The *rank interval* of the operator alphabet Σ is $\text{rki}(\Sigma) = [1, m]$ where m is the maximal rank of the elements of Σ.

The nodes of a tree or term graph over Σ are labeled by operators. To indicate the order of the arguments of an operator, we label the edges by natural numbers. Since a term graph may contain "garbage", the root of the tree it represents has

to be marked. Here it is technically convenient to add the mark # to the label of the root. This is only necessary if the term graph itself has no unique root.

Let Σ be an operator alphabet. By $\Sigma_{\#}$ we denote the ranked alphabet $\Sigma \cup (\Sigma \times \{\#\})$ in which, for every $\sigma \in \Sigma$, both σ and $(\sigma, \#)$ have the same rank as σ in Σ. A *term graph* over Σ is a graph t over $(\Sigma_{\#}, \mathrm{rki}(\Sigma))$ such that

(1) t is noncircular,

(2) for every node u of t, if $\mathrm{lab}_t(u) = \sigma$ or $\mathrm{lab}_t(u) = (\sigma, \#)$, with $\sigma \in \Sigma_k$, then all outgoing edges of u have labels in $[1, k]$, and, for every $i \in [1, k]$, u has exactly one outgoing edge with label i, and

(3) either there is a unique node with label in $\Sigma \times \{\#\}$, or there is no node with label in $\Sigma \times \{\#\}$ and there is a (unique) node which is an ancestor of all nodes.

The unique node mentioned in condition (3) is called the root of t, denoted $\mathrm{root}(t)$. Thus, either $\mathrm{root}(t)$ is the unique node that is marked with #, or no node is marked with # and $\mathrm{root}(t)$ is the unique node that is an ancestor of all nodes of t.

A node without outgoing edges is called a leaf of t. For nodes u and v of t, if $(u, i, v) \in E_t$, then v is called the i-th child of u, denoted by $u \cdot i$. For technical convenience, we also define $u \cdot 0 = u$. All nodes of t that are not descendants of $\mathrm{root}(t)$ are referred to as *garbage*. Note that, by condition (3), there is no garbage if the root is not marked by #.

A *forest* is a term graph such that no node has more than one incoming edge. A *tree* is a forest such that all nodes have their labels in Σ. The set of all trees over Σ is denoted T_Σ. A function from T_Σ to T_Δ (where Δ is an operator alphabet too) is called a *tree transduction* or a *term transduction*. As usual, trees will also be denoted by the corresponding terms over Σ. Thus, for $\sigma \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$, the term $\sigma(t_1, \ldots, t_k)$ denotes the tree t with $\mathrm{lab}_t(\mathrm{root}(t)) = \sigma$ and $\mathrm{root}(t) \cdot i = \mathrm{root}(t_i)$ for every $i \in [1, k]$.

Term graphs can be unfolded into trees. For a term graph t over Σ, the *unfolding* of t, denoted $\mathrm{unfold}(t)$, is the tree over Σ defined by $\mathrm{unfold}(t) = \mathrm{unf}_t(\mathrm{root}(t))$, where, for $u \in V_t$, $\mathrm{unf}_t(u) = \sigma(\mathrm{unf}_t(u \cdot 1), \ldots, \mathrm{unf}_t(u \cdot k))$ with $\mathrm{lab}_t(u) = \sigma$ or $\mathrm{lab}_t(u) = (\sigma, \#)$, $\sigma \in \Sigma_k$.

A term graph over Σ is *clean* if all its node labels are in Σ (and hence all its nodes are descendants of the root). Garbage can be removed from term graphs as follows. For a term graph t over Σ, the *cleaning* of t, denoted $\mathrm{clean}(t)$, is the subgraph of t induced by the descendants of $\mathrm{root}(t)$, in which # is removed from the label of $\mathrm{root}(t)$, if present. Obviously, $\mathrm{clean}(t)$ is a term graph over Σ of which all nodes have their labels in Σ. Moreover, $\mathrm{root}(\mathrm{clean}(t)) = \mathrm{root}(t)$ and $\mathrm{unfold}(\mathrm{clean}(t)) = \mathrm{unfold}(t)$.

We also need terms with variables. These variables are treated as constants. Let Σ be an operator alphabet and Ξ a finite set (of variables), disjoint with Σ. Let $\Sigma(\Xi)$ be the operator alphabet with $\Sigma(\Xi)_0 = \Sigma_0 \cup \Xi$ and $\Sigma(\Xi)_k = \Sigma_k$ for $k \geq 1$. Then $T_\Sigma(\Xi) = T_{\Sigma(\Xi)}$ is the set of trees over Σ with variables in Ξ. A tree $t \in T_\Sigma(\Xi)$ is *linear* if every variable appears at most once in t. If $\Xi = \{\xi_1, \ldots, \xi_k\}$ and $t_i \in T_\Sigma(\Xi)$ for $i \in [0, k]$, then $t_0[\xi_1 \mapsto t_1, \ldots, \xi_k \mapsto t_k]$

denotes the result of (simultaneously) substituting t_i for ξ_i in t_0, $i \in [1,k]$.

2.2 Monadic second order logic

Monadic second order logic can be used to describe properties of graphs (see, e.g., [Cou2, Cou5, Eng4, Eng6, EngOos]), and hence in particular to describe properties of trees and term graphs. For alphabets Σ and Γ, we use the language $\mathrm{MSOL}(\Sigma,\Gamma)$ of monadic second order (MSO) formulas over (Σ,Γ). Formulas over (Σ,Γ) describe properties of graphs over (Σ,Γ). This logical language has node variables $x, y, \ldots$, and node-set variables $X, Y, \ldots$. For a given graph g over (Σ,Γ), node variables range over the elements of V_g, and node-set variables range over the subsets of V_g.

There are three types of atomic formulas in $\mathrm{MSOL}(\Sigma,\Gamma)$: $\mathrm{lab}_\sigma(x)$, for every $\sigma \in \Sigma$, denoting that x has label σ; $\mathrm{edg}_\gamma(x,y)$, for every $\gamma \in \Gamma$, denoting that there is an edge labeled γ from x to y; and $x \in X$, denoting that x is an element of X. The formulas are built from the atomic formulas using the connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$, as usual. Both node variables and node-set variables can be quantified with $\exists$ and $\forall$. We will use $\mathrm{edg}(x,y)$ to abbreviate the disjunction of all $\mathrm{edg}_\gamma(x,y)$, $\gamma \in \Gamma$; it denotes that there is an edge from x to y. We will use $x = y$ for $\forall X(x \in X \leftrightarrow y \in X)$, denoting that x equals y. Finally, we will use the formula $\mathrm{path}(x,y)$ which expresses the existence of a directed path from x to y:

$$\mathrm{path}(x,y) = \forall X((\mathrm{closed}(X) \wedge x \in X) \rightarrow y \in X)$$

where $\mathrm{closed}(X) = \forall x,y((\mathrm{edg}(x,y) \wedge x \in X) \rightarrow y \in X)$.

For every $k \in \mathbb{N}$, the set of MSO formulas over (Σ,Γ) with k free node variables and no free node-set variables is denoted $\mathrm{MSOL}_k(\Sigma,\Gamma)$. For $k = 1, 2$, the elements of $\mathrm{MSOL}_k(\Sigma,\Gamma)$ are also called *unary* and *binary* formulas, respectively.

Since we are predominantly interested in trees and term graphs, over an operator alphabet Σ, an MSO formula over $(\Sigma_\#, \mathrm{rki}(\Sigma))$ will simply be called an MSO formula over Σ. Also, $\mathrm{MSOL}(\Sigma_\#, \mathrm{rki}(\Sigma))$ will be abbreviated to $\mathrm{MSOL}(\Sigma)$, and similarly for $\mathrm{MSOL}_k(\Sigma)$. Note that, in $\mathrm{MSOL}(\Sigma)$, $\mathrm{edg}_i(x,y)$ means that y is the i-th child of x, and $\mathrm{edg}(x,y)$ means that x is a parent of y. We will additionally use $\mathrm{root}(x)$ for a formula that expresses that x is the root of a term graph, e.g., the formula $\mathrm{lab}_\#(x) \vee \forall y(\neg\,\mathrm{lab}_\#(y) \wedge \mathrm{path}(x,y))$, where $\mathrm{lab}_\#(x)$ is the disjunction of all $\mathrm{lab}_{(\sigma,\#)}(x)$, for $\sigma \in \Sigma$. Moreover, we will use $\mathrm{leaf}(x)$ for $\neg\exists y(\mathrm{edg}(x,y))$, which denotes that x is a leaf. In case we consider trees only, the component $\Sigma \times \{\#\}$ can of course be dropped, and, e.g., $\mathrm{root}(x)$ can be simplified to $\forall y(\mathrm{path}(x,y))$.

For a closed formula $\phi \in \mathrm{MSOL}_0(\Sigma,\Gamma)$ and a graph $g \in G_{\Sigma,\Gamma}$, we write $g \models \phi$ if g satisfies ϕ. Given a graph g, a valuation ν is a function that assigns to each node variable an element of V_g, and to each node-set variable a subset of V_g. We write $(g,\nu) \models \phi$, if ϕ holds in g, where the free variables of ϕ are assigned values according to the valuation ν. If a formula ϕ has free variables, say, x, X, y and no others, we also write $\phi(x,X,y)$. Moreover, we write $(g,u,U,v) \models \phi(x,X,y)$ for $(g,\nu) \models \phi(x,X,y)$, where $\nu(x) = u$, $\nu(X) = U$, and $\nu(y) = v$. As a very

simple example, $(g, u, v) \models \mathrm{path}(x, y)$ means that there is a path from u to v in g, and $g \models \forall x, y(\mathrm{path}(x, y))$ means that g is strongly connected.

2.3 Attribute Grammars

In this subsection we recall some terminology concerning attribute grammars (see, e.g., [Knu, DerJouLor, Eng3, KühVog2]). In order to allow the attribute grammar to work on arbitrary trees over an operator alphabet, rather than on derivation trees of an underlying context-free grammar, we consider a slight variation of the attribute grammar that was introduced in [Fül]. The semantic rules of the attribute grammar are grouped by operator rather than by grammar production, and there are special semantic rules for the inherited attributes of the root. All operators have the same attributes.

Let Σ be an operator alphabet. An *attribute grammar* over Σ is a six-tuple $G = (\Sigma, S, I, \Omega, W, R, \alpha_{\mathrm{m}})$ where

- Σ is the *input alphabet*.
- S is a finite set, the set of *synthesized attributes*.
- I, disjoint with S, is a finite set, the set of *inherited attributes*.
- Ω is a finite set of sets, the *semantic domains* of the attributes.
- $W : (S \cup I) \to \Omega$ is the *domain assignment*.
- R describes the *semantic rules*; it is a function associating a set of rules with every $\sigma \in \Sigma \cup \{\mathrm{root}\}$:
  - For $\sigma \in \Sigma$, $R(\sigma)$ is the set of *internal rules* for σ; $R(\sigma)$ contains one rule

    $$\langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_k, i_k \rangle)$$

    for every pair $\langle \alpha_0, i_0 \rangle$, where either α_0 is a synthesized attribute and $i_0 = 0$, or α_0 is an inherited attribute and $i_0 \in [1, \mathrm{rk}(\sigma)]$. Furthermore, $k \geq 0$, $\alpha_1, \ldots, \alpha_k \in S \cup I$, $i_1, \ldots, i_k \in [0, \mathrm{rk}(\sigma)]$, f is a function from $W(\alpha_1) \times \cdots \times W(\alpha_k)$ to $W(\alpha_0)$, and the $\langle \alpha_j, i_j \rangle$ are all distinct.
  - $R(\mathrm{root})$ is the set of *root rules*; $R(\mathrm{root})$ contains one rule

    $$\langle \alpha_0, 0 \rangle = f(\langle \alpha_1, 0 \rangle, \ldots, \langle \alpha_k, 0 \rangle)$$

    for every $\alpha_0 \in I$, where $k \geq 0$, $\alpha_1, \ldots, \alpha_k \in S \cup I$, f is a function from $W(\alpha_1) \times \cdots \times W(\alpha_k)$ to $W(\alpha_0)$, and the $\langle \alpha_j, i_j \rangle$ are all distinct.
- $\alpha_{\mathrm{m}} \in S$ is the *meaning* attribute.

Usually, for a semantic rule $\langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_k, i_k \rangle)$ the function f is given as $f = \lambda x_1, \ldots, x_k.e$ for some expression e with variables in $\{x_1, \ldots, x_k\}$. We will then informally denote the rule by $\langle \alpha_0, i_0 \rangle = e'$ where e' is obtained from e by substituting $\langle \alpha_j, i_j \rangle$ for x_j, for all $j \in [1, k]$.

If $I = \emptyset$, i.e., G has synthesized attributes only, then G is said to be *Only Synthesized* (OS). Note that an OS attribute grammar has no root rules.

If all sets in Ω are finite, then G is said to be *finite-valued*; this means that each attribute has finitely many values.

Let t be a tree over Σ. The set of attributes of t is $A(t) = (S \cup I) \times V_t$. The set $R(t)$ of *semantic instructions* of t is defined as follows. Recall that, for $u \in V_t$, $u \cdot 0 = u$. For every node $u \in V_t$, if $\mathrm{lab}_t(u) = \sigma$, and $\langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_k, i_k \rangle)$ is a rule in $R(\sigma)$, then

$$\langle \alpha_0, u \cdot i_0 \rangle = f(\langle \alpha_1, u \cdot i_1 \rangle, \ldots, \langle \alpha_k, u \cdot i_k \rangle)$$

is an *internal instruction* of t. Analogously, if $\langle \alpha_0, 0 \rangle = f(\langle \alpha_1, 0 \rangle, \ldots, \langle \alpha_k, 0 \rangle)$ is a root rule, then

$$\langle \alpha_0, \mathrm{root}(t) \rangle = f(\langle \alpha_1, \mathrm{root}(t) \rangle, \ldots, \langle \alpha_k, \mathrm{root}(t) \rangle)$$

is a *root instruction* of t. The set of all internal instructions and root instructions of t is $R(t)$. Note that for every $\langle \alpha, u \rangle \in A(t)$ there is exactly one semantic instruction with left-hand side $\langle \alpha, u \rangle$ in $R(t)$.

With the help of $R(t)$, the dependencies between the attributes of t can be represented in the usual way by a graph. The *dependency graph* of a tree t over Σ is the unlabeled directed graph $D(t) = (V, E)$, where $V = A(t)$ and E consists of all edges $(\langle \alpha, u \rangle, \langle \alpha', u' \rangle)$ such that there is a semantic instruction $\langle \alpha', u' \rangle = f(\langle \alpha_1, u_1 \rangle, \ldots, \langle \alpha_k, u_k \rangle) \in R(t)$ with $\langle \alpha, u \rangle = \langle \alpha_i, u_i \rangle$ for some $i \in [1, k]$. An attribute grammar G is *noncircular* on a tree $t \in T_\Sigma$ if $D(t)$ is noncircular, and G is noncircular if it is noncircular on every tree $t \in T_\Sigma$. An attribute grammar G is *single use restricted* (SUR) on a tree $t \in T_\Sigma$ if no node of $D(t)$ has more than one outgoing edge, and G is SUR if it is SUR on every tree $t \in T_\Sigma$. The single use restriction was investigated in [Gan, GanGie, Gie]; it received its name in [Gie].

We now define how to give the correct values to the attributes of the tree t. Let dec be a function from $A(t)$ to $\cup\Omega$, such that $\mathrm{dec}(\langle \alpha, u \rangle) \in W(\alpha)$ for every $\langle \alpha, u \rangle \in A(t)$. The function dec is a *decoration* of t if all instructions are satisfied, i.e., for every instruction $\langle \alpha_0, u_0 \rangle = f(\langle \alpha_1, u_1 \rangle, \ldots, \langle \alpha_k, u_k \rangle) \in R(t)$, $\mathrm{dec}(\langle \alpha_0, u_0 \rangle) = f(\mathrm{dec}(\langle \alpha_1, u_1 \rangle), \ldots, \mathrm{dec}(\langle \alpha_k, u_k \rangle))$. It is well known that if $D(t)$ is noncircular, then t has a unique decoration; this decoration will be denoted by $\mathrm{dec}_{G,t}$.

We will not use the meaning attribute α_m here. It will be used in later sections, in different ways, to define the translation realized by an attribute grammar.

Finally, we define the dependency graphs of symbols (and the root), which are similar to the dependency graphs of productions in the usual type of attribute grammar [Knu]. They are the atomic graphs from which all dependency graphs $D(t)$ of trees $t \in T_\Sigma$ are built. For $\sigma \in \Sigma_k$, the *dependency graph of* σ is the unlabeled directed graph $D(\sigma) = (V, E)$ where $V = A \times [0, k]$ and E consists of all edges $(\langle \alpha_j, i_j \rangle, \langle \alpha_0, i_0 \rangle)$ such that there is a rule $\langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_r, i_r \rangle) \in R(\sigma)$, with $j \in [1, r]$. The *dependency graph of the root* is the unlabeled directed graph $D(\mathrm{root}) = (V, E)$ where $V = A \times \{0\}$ and E consists of all edges $(\langle \alpha_j, 0 \rangle, \langle \alpha_0, 0 \rangle)$ such that there is a rule $\langle \alpha_0, 0 \rangle = f(\langle \alpha_1, 0 \rangle, \ldots, \langle \alpha_r, 0 \rangle) \in R(\mathrm{root})$, with $j \in [1, r]$.

3 MSO definable graph transductions

The main concept in this paper is that of an MSO definable function f on graphs, introduced in [Oos, Eng4, EngOos], and independently in [Cou3] (for a recent survey see [Cou4]). The idea is that, for a given input graph g_1, the nodes, edges, and labels of the output graph $g_2 = f(g_1)$ are described in terms of MSO formulas on g_1. For the simplest type of MSO definable function (see [EngOos]), the nodes of g_2 are a subset of the nodes of g_1. In fact, for each node label σ of g_2 there is a unary formula $\psi_\sigma(x)$ expressing that node x of g_1 will be a node of g_2 with label σ (provided no other such formula is true for x). The edges of g_2 are specified by a binary formula $\chi_\gamma(x, y)$, for every edge label γ of g_2, expressing that there will be a γ-labeled edge from x to y in g_2. Although this type of MSO definable function is sufficient for many purposes, its power is restricted by the fact that the size of the output graph cannot be larger than the size of the input graph. Thus, in [Cou3] the above idea was extended by allowing the output graph to contain (a fixed number k of) copies of each node of the input graph. Now, for each $i \in [1, k]$ there is a formula $\psi_{\sigma,i}(x)$ expressing that the i-th copy of node x of g_1 will be a node of g_2 with label σ, and, similarly, for $i, j \in [1, k]$ there are formulas $\chi_{\gamma,i,j}(x, y)$ for the edges of g_2. It will be convenient to use arbitrary names for the copies, rather than numbers.

We will only consider total functions f (see [Cou4] for the extension to partial functions and to relations). Since our aim is to compare the defining power of monadic second order logic with the power of certain types of tree transducers, we will view an MSO specification of f as a "graph transducer"; this is then a total deterministic transducer. We now define the syntax and semantics of such MSO graph transducers.

An MSO *graph transducer* from (Σ_1, Γ_1) to (Σ_2, Γ_2) is a triple $T = (C, \Psi, X)$ where

- C is a finite set of *copy names*,
- $\Psi = \{\psi_{\sigma,c}(x)\}_{\sigma \in \Sigma_2, c \in C}$, with $\psi_{\sigma,c}(x) \in \mathrm{MSOL}_1(\Sigma_1, \Gamma_1)$, is the family of *node formulas*, and
- $X = \{\chi_{\gamma,c,c'}(x, y)\}_{\gamma \in \Gamma_2, c, c' \in C}$, with $\chi_{\gamma,c,c'}(x, y) \in \mathrm{MSOL}_2(\Sigma_1, \Gamma_1)$, is the family of *edge formulas*.

The *copy number* of T is $\#C$.

The *graph transduction* $\mathcal{T}_{\mathrm{gr}} : G_{\Sigma_1,\Gamma_1} \to G_{\Sigma_2,\Gamma_2}$ *defined by* T is defined as follows. For every graph g_1 over (Σ_1, Γ_1), $\mathcal{T}_{\mathrm{gr}}(g_1)$ is the graph g_2 over (Σ_2, Γ_2) with

- $V_{g_2} = \{(c, u) \mid c \in C, u \in V_{g_1},$ and there is exactly one $\sigma \in \Sigma_2$ such that $(g_1, u) \models \psi_{\sigma,c}(x)\}$,
- $E_{g_2} = \{((c, u), \gamma, (c', u')) \mid (c, u), (c', u') \in V_{g_2}, \gamma \in \Gamma_2,$ and $(g_1, u, u') \models \chi_{\gamma,c,c'}(x, y)\}$,
- $\mathrm{lab}_{g_2} = \{((c, u), \sigma) \mid (c, u) \in V_{g_2}, \sigma \in \Sigma_2,$ and $(g_1, u) \models \psi_{\sigma,c}(x)\}$.

The set of all graph transductions defined by MSO graph transducers will be denoted MSO-GT. They are the MSO *definable graph transductions*.

If the copy number of an MSO graph transducer is 1, i.e., the copy set C is a singleton $\{c\}$, we will drop the subscripts c from the node and edge formulas, i.e., we write $\psi_\sigma(x)$ and $\chi_\gamma(x,y)$ instead of $\psi_{\sigma,c}(x)$ and $\chi_{\gamma,c,c}(x,y)$, respectively.

Note that a copy (c,u) of a node u of g_1 may *not* be a node of $g_2 = \mathcal{T}_{\mathrm{gr}}(g_1)$ for two reasons: either there is no σ such that $(g_1,u) \models \psi_{\sigma,c}(x)$, or there is more than one such σ. However, it is easy to see that we may always assume, for fixed $c \in C$, the formulas $\psi_{\sigma,c}(x)$ to be mutually exclusive, in which case only the first reason remains and

- $V_{g_2} = \{(c,u) \in C \times V_{g_1} \mid \exists \sigma \in \Sigma_2 : (g_1,u) \models \psi_{\sigma,c}(x)\}$, and
- $\mathrm{lab}_{g_2} = \{((c,u),\sigma) \in (C \times V_{g_1}) \times \Sigma_2 \mid (g_1,u) \models \psi_{\sigma,c}(x)\}$.

In fact, the formula $\psi_{\sigma,c}(x)$ can be replaced by its conjunction with all $\neg\psi_{\sigma',c}(x)$, for $\sigma' \in \Sigma_2 - \{\sigma\}$.

Similarly, it can always be assumed that an edge formula $\chi_{\gamma,c,c'}(x,y)$ only holds for nodes $u,u' \in V_{g_1}$ if $(c,u),(c',u')$ are nodes of the output graph g_2, i.e., that $(g_1,u,u') \models \chi_{\gamma,c,c'}(x,y)$ implies $(c,u),(c',u') \in V_{g_2}$. In that case

- $E_{g_2} = \{((c,u),\gamma,(c',u')) \mid (c,u),(c',u') \in C \times V_{g_1}, \gamma \in \Gamma_2,$
  $(g_1,u,u') \models \chi_{\gamma,c,c'}(x,y)\}$.

Assuming the node formulas to be mutually exclusive (for fixed c), this is achieved by changing the formula $\chi_{\gamma,c,c'}(x,y)$ into its conjunction with the disjunction of all formulas $\psi_{\sigma,c}(x) \wedge \psi_{\sigma',c'}(y)$, for $\sigma,\sigma' \in \Sigma_2$.

These two assumptions will be made whenever convenient, without mentioning. They allow us, e.g., to check fewer conditions in the simulation of MSO graph transducers by attribute grammars (in Section 7).

Terms and trees We now introduce the specific MSO graph transducers that we are interested in. They have trees as input, and they also produce trees as output, either by directly defining the output tree or by defining a term graph of which the unfolding is the output tree.

Let Σ and Δ be operator alphabets. An MSO *term graph transducer* from Σ to Δ is an MSO graph transducer T from $(\Sigma, \mathrm{rki}(\Sigma))$ to $(\Delta_\#, \mathrm{rki}(\Delta))$ such that $\mathcal{T}_{\mathrm{gr}}(t)$ is a term graph over Δ for every $t \in T_\Sigma$. It defines the tree transduction $\mathcal{T} : T_\Sigma \to T_\Delta$ with $\mathcal{T}(t) = \mathrm{unfold}(\mathcal{T}_{\mathrm{gr}}(t))$ for every $t \in T_\Sigma$.

An MSO *tree transducer* from Σ to Δ is an MSO term graph transducer T from Σ to Δ such that $\mathcal{T}_{\mathrm{gr}}(t) \in T_\Delta$ for every $t \in T_\Sigma$. Note that $\mathcal{T}(t) = \mathcal{T}_{\mathrm{gr}}(t)$ for every $t \in T_\Sigma$, because unfolding has no effect on trees.

The set of all tree transductions defined by MSO term graph transducers is denoted MSO-TGT, and the set of all tree transductions defined by MSO tree transducers is denoted MSO-TT. To distinguish between MSO-TT and MSO-TGT, the transductions in MSO-TT will be called MSO *definable tree transductions*, and those in MSO-TGT will be called MSO *definable term transductions*. Note that,

by definition, MSO-TT $\subseteq$ MSO-TGT. Properness of this inclusion will be shown in Example 1(5, binary); thus, in general, the unfolding of term graphs is not MSO definable.

Special cases Two special cases of interest are the following. First, an MSO graph transducer is said to be "direction preserving" if edges of the output graph correspond to (directed) paths in the input graph. If, in particular, the input graph is a tree, then all edges in the output graph lead from (a copy of) a node of the input tree to (a copy of) one of its descendants. Formally, an MSO graph transducer T from (Σ_1, Γ_1) to (Σ_2, Γ_2) is *direction preserving* if, for every graph $g_1 \in G_{\Sigma_1,\Gamma_1}$, if $((c,u),\gamma,(c',u'))$ is an edge of $\mathcal{T}_{\mathrm{gr}}(g_1)$ then there is a directed path from u to u' in g_1. We will indicate the direction preserving property by a subscript 'dir'. Thus, MSO-TGT$_{\mathrm{dir}}$ denotes the class of all term transductions defined by direction preserving MSO term graph transducers. We will show that these transducers are related to Only Synthesized attribute grammars.

Second, an MSO tree transducer is said to be a "relabeling" if it just relabels all nodes of the input tree. Formally, an MSO tree transducer $T = (C, \Psi, X)$ from Σ to Δ is an MSO *relabeling* if (1) the copy number of T is 1, (2) $\chi_i(x,y) = \mathrm{edg}_i(x,y)$ for every $i \in \mathrm{rki}(\Delta)$, and (3) for every $t \in T_\Sigma$ and $u \in V_t$ there is a unique $\delta \in \Delta$ such that $(t,u) \models \psi_\delta(x)$. The class of all tree transductions defined by MSO relabelings will be denoted MSO-REL. Note that MSO-REL $\subseteq$ MSO-TT$_{\mathrm{dir}}$.

An inclusion diagram of the classes of tree transductions defined above is given in Fig. 1. Its correctness will follow from Example 1 below, in particular Example 1(3, stars), Example 1(5, binary), and Example 1(6, yield).

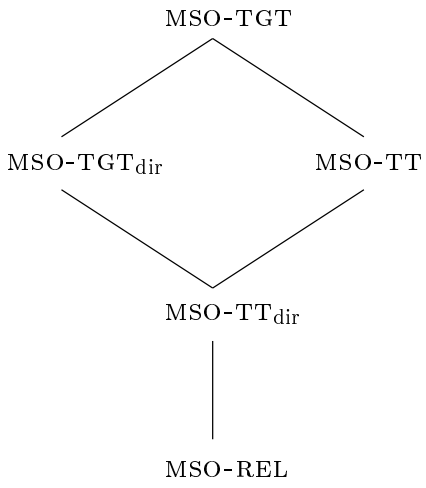


Fig. 1. An inclusion diagram of classes of tree transductions

We observe here that, restricting attention to input trees over Σ, all the above properties of MSO graph transducers T are decidable: whether or not T is an MSO term graph transducer, is an MSO tree transducer, is direction preserving, or is an MSO relabeling. This is because for each of these properties a closed formula $\phi \in \mathrm{MSOL}(\Sigma)$ can be found such that T has the property iff $t \models \phi$ for every tree $t \in T_\Sigma$. Decidability now follows from the classical fact that $\{t \in T_\Sigma \mid t \models \neg\phi\}$ is a regular tree language ([Don, ThaWri]), and the well-known fact that emptiness of regular tree languages is decidable (see, e.g., [GécSte]). As an example, for the direction preserving property the formula is $\phi = \forall x, y(\phi'(x, y))$, where $\phi'(x, y)$ is the conjunction of all formulas $\chi_{i,c,c'}(x, y) \to \mathrm{path}(x, y)$, for every edge formula $\chi_{i,c,c'}(x, y)$.

Examples We now give some examples of MSO definable graph transductions.

Example 1. (1, clean) The cleaning of term graphs (see Section 2.1) is MSO definable, i.e., for every operator alphabet Σ there is an MSO graph transducer T from $(\Sigma_\#, \mathrm{rki}(\Sigma))$ to $(\Sigma, \mathrm{rki}(\Sigma))$ such that for every term graph t over Σ, $\mathcal{T}_{\mathrm{gr}}(t) = \mathrm{clean}(t)$. The copy number of T is 1. For every $\sigma \in \Sigma$ it has node formula

$$\psi_\sigma(x) = (\mathrm{lab}_\sigma(x) \vee \mathrm{lab}_{(\sigma,\#)}(x)) \wedge \forall y(\mathrm{root}(y) \to \mathrm{path}(y, x)),$$

and for every $i \in \mathrm{rki}(\Sigma)$ it has edge formula $\chi_i(x, y) = \mathrm{edg}_i(x, y)$. Through the node formulas, only those nodes of the input term graph t are copied to the output graph that are descendants of $\mathrm{root}(t)$. The labels of those nodes stay the same, except that $\mathrm{root}(t)$ is unmarked (if it was marked by #), and the edges between them are just copied to the output graph. This shows that the output graph is $\mathrm{clean}(t)$.

(2, relab) If $\phi_1(x), \ldots, \phi_n(x)$ is a sequence of formulas in $\mathrm{MSOL}_1(\Sigma)$, then there is an MSO relabeling T from Σ to $\Sigma \times \{\mathrm{true}, \mathrm{false}\}^n$ such that, for every $t \in T_\Sigma$ and $u \in V_t$, $\mathrm{lab}_{\mathcal{T}(t)}(u) = (\sigma, b_1, \ldots, b_n)$ where $\sigma = \mathrm{lab}_t(u)$ and, for every $i \in [1, n]$, $b_i = \mathrm{true}$ iff $(t, u) \models \phi_i(x)$. In fact, for $\sigma' = (\sigma, b_1, \ldots, b_n)$, the node formula $\psi_{\sigma'}(x)$ of T is the conjunction of the formula $\mathrm{lab}_\sigma(x)$ with all formulas $\phi_i(x)$ for which $b_i = \mathrm{true}$ and all formulas $\neg\phi_i(x)$ for which $b_i = \mathrm{false}$.

(3, stars) We give an MSO tree transducer T from Σ to Δ, where $\Sigma = \Sigma_0 \cup \Sigma_2$, with $\Sigma_0 = \{a\}$, and $\Sigma_2 = \{\sigma\}$, and $\Delta = \Delta_0 \cup \Delta_1 \cup \Delta_2$, with $\Delta_0 = \Sigma_0$, $\Delta_2 = \Sigma_2$, and $\Delta_1 = \{*\}$. Note that $\mathrm{rki}(\Delta) = \mathrm{rki}(\Sigma) = \{1, 2\}$. The transducer transforms a tree by inserting a node with label $*$ on each of its edges. See Fig. 2 for an example. The transducer $T = (C, \Psi, X)$ is defined as follows.

- The copy set C is $\{\mathrm{o}, \mathrm{n}\}$, where o stands for 'old', and n for 'new'. A node (o, u) of $\mathcal{T}(t)$ is an old copy of the node u of t (with the same label). A node (n, u) is a new copy of the node u; it has label $*$, and it has node (o, u) as its child. Two copies are made of every node except the root.

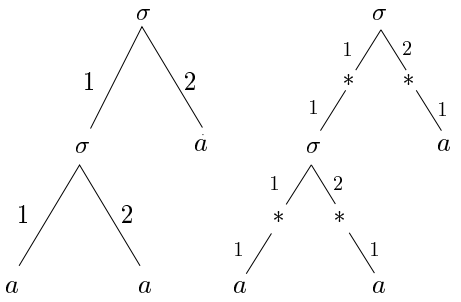


Fig. 2. Example of a tree t and its transduction $\mathcal{T}(t)$

– Ψ consists of the node formulas

$$\begin{aligned}
\psi_{\delta,\mathrm{o}}(x) &= \mathrm{lab}_\delta(x) && \text{for all } \delta \in \Sigma,\\
\psi_{*,\mathrm{o}}(x) &= \mathrm{false}, &&\\
\psi_{\delta,\mathrm{n}}(x) &= \mathrm{false}, && \text{for all } \delta \in \Sigma, \text{ and}\\
\psi_{*,\mathrm{n}}(x) &= \neg\,\mathrm{root}(x). &&
\end{aligned}$$

– X consists of the edge formulas

$$\begin{aligned}
\chi_{i,\mathrm{o},\mathrm{o}}(x,y) &= \mathrm{false} && \text{for } i \in \{1,2\},\\
\chi_{i,\mathrm{n},\mathrm{n}}(x,y) &= \mathrm{false} && \text{for } i \in \{1,2\},\\
\chi_{i,\mathrm{o},\mathrm{n}}(x,y) &= \mathrm{edg}_i(x,y) && \text{for } i \in \{1,2\},\\
\chi_{1,\mathrm{n},\mathrm{o}}(x,y) &= (x=y), && \text{and}\\
\chi_{2,\mathrm{n},\mathrm{o}}(x,y) &= \mathrm{false}. &&
\end{aligned}$$

Note that T is a direction preserving MSO tree transducer. Thus, $\mathcal{T}$ is in $\text{MSO-TT}_{\mathrm{dir}}$, but not in MSO-REL because for an MSO relabeling the output tree has the same size as the input tree.

(4, path) The next example is adapted from [FülVag]. It is a direction preserving MSO tree transducer T from Σ to Δ, where the input alphabet Σ is $\{\sigma, *, a\}$, with $\mathrm{rk}_\Sigma(\sigma) = 2$, and $\mathrm{rk}_\Sigma(*) = \mathrm{rk}_\Sigma(a) = 0$, and the output alphabet Δ is $\{1, 2, *, a\}$, with $\mathrm{rk}_\Delta(1) = \mathrm{rk}_\Delta(2) = 1$, and $\mathrm{rk}_\Delta(*) = \mathrm{rk}_\Delta(a) = 0$. If the input tree t contains exactly one leaf labeled $*$, the transducer transforms it into a tree over Δ, which codes the path leading from the root of t to the leaf labeled $*$ in the obvious way. Otherwise, the output is a. For example, $\mathcal{T}(\sigma(\sigma(a,\sigma(a,*)),a)) = 1(2(2(*)))$, and $\mathcal{T}(\sigma(*,\sigma(a,*))) = \mathcal{T}(\sigma(a,\sigma(a,a))) = a$. The copy number of T is 1. The node and edge formulas of T are

$$\begin{aligned}
\psi_1(x) &= \mathrm{us} \wedge \exists y,z(\mathrm{edg}_1(x,y) \wedge \mathrm{path}(y,z) \wedge \mathrm{lab}_*(z)),\\
\psi_2(x) &= \mathrm{us} \wedge \exists y,z(\mathrm{edg}_2(x,y) \wedge \mathrm{path}(y,z) \wedge \mathrm{lab}_*(z)),\\
\psi_*(x) &= \mathrm{us} \wedge \mathrm{lab}_*(x),\\
\psi_a(x) &= \neg\mathrm{us} \wedge \mathrm{root}(x),\\
\chi_1(x,y) &= \mathrm{edg}(x,y),
\end{aligned}$$

where 'us' is the closed formula $\exists z(\mathrm{lab}_*(z) \wedge \forall y(\mathrm{lab}_*(y) \to y = z))$, expressing that the input tree has a unique star.

(5, binary) Consider the MSO term graph transducer T from Σ to Δ, where $\Sigma = \Delta = \{\sigma, a\}$, with $\mathrm{rk}_\Sigma(\sigma) = 1$, $\mathrm{rk}_\Delta(\sigma) = 2$, and $\mathrm{rk}_\Sigma(a) = \mathrm{rk}_\Delta(a) = 0$. The transducer transforms a linear input tree into a full binary output tree, of the same height. To do this, it turns the input tree into a term graph by changing every edge (with label 1) into two edges (with labels 1 and 2). This term graph is then unfolded into a full binary tree. The copy number of T is 1, it has node formulas $\psi_\sigma(x) = \mathrm{lab}_\sigma(x)$ and $\psi_a(x) = \mathrm{lab}_a(x)$, and edge formulas $\chi_1(x, y) = \mathrm{edg}_1(x, y)$ and $\chi_2(x, y) = \mathrm{edg}_1(x, y)$. Note that, again, T is direction preserving. Thus, $\mathcal{T}$ is in MSO-TGT$_{\mathrm{dir}}$, but not in MSO-TT; in fact, it should be obvious that, for every $\mathcal{T} \in$ MSO-TT and every input tree t, the size of $\mathcal{T}(t)$ is linear in the size of t, where the constant is the copy number of T.

(6, yield) Finally, we consider an example of an MSO tree transducer that is not direction preserving. Let the input alphabet be $\Sigma = \Sigma_0 \cup \Sigma_2$, for some Σ_0 and Σ_2. The output alphabet is $\Delta = \Delta_0 \cup \Delta_1$, with $\Delta_1 = \Sigma_0$ and $\Delta_0 = \{\hat{\sigma} \mid \sigma \in \Sigma_0\}$.

We present an MSO tree transducer T such that for any tree $t \in T_\Sigma$, $\mathcal{T}(t)$ is equal to the yield of t as a monadic tree (a string can be seen as a monadic tree, with its first symbol as root, the second symbol as child of the first, etcetera, up to the last symbol, which is the leaf of the tree). We have to beware, because all labels in the monadic tree that constitutes the yield have rank 1, except for the last one, which has rank 0. We will therefore use elements of Δ_1 for all of the labels, except the last one, for which we will use a corresponding label from Δ_0. To do this we need the following MSO formula over Σ, which, for a node x of a (binary) tree, checks that it is on the path from the root to the rightmost leaf:

$$\mathrm{rm}(x) = \forall y(\mathrm{root}(y) \to \mathrm{path}_2(y, x))$$

where the formula $\mathrm{path}_2(x, y)$ expresses that there is a path from x to y of which the edges are all labeled 2 (obtained by changing $\mathrm{edg}(x, y)$ into $\mathrm{edg}_2(x, y)$ in the formula $\mathrm{path}(x, y)$, see Section 2.2).

Let $T = (\{c\}, \{\psi_\sigma\}_{\sigma \in \Sigma_0} \cup \{\psi_{\hat{\sigma}}\}_{\sigma \in \Sigma_0}, \{\chi_1\})$ be the MSO tree transducer from Σ to Δ, with

$$\begin{aligned}
\psi_\sigma(x) &= \mathrm{lab}_\sigma(x) \wedge \neg\, \mathrm{rm}(x) \\
\psi_{\hat{\sigma}}(x) &= \mathrm{lab}_\sigma(x) \wedge \mathrm{rm}(x) \\
\chi_1(x, y) &= \mathrm{leaf}(x) \wedge \chi(x, y) \wedge \mathrm{leaf}(y)
\end{aligned}$$

where

$$\chi(x, y) = \exists z\, (\exists z_l(\mathrm{edg}_1(z, z_l) \wedge \mathrm{path}_2(z_l, x)) \wedge \exists z_r(\mathrm{edg}_2(z, z_r) \wedge \mathrm{path}_1(z_r, y)))\,.$$

For leaves x and y, the formula $\chi(x, y)$ checks that y directly follows x in the left-to-right order of leaves. Note that in this formula, z is the least common ancestor of x and y, and z_l and z_r are its two children. The formula $\mathrm{path}_1(x, y)$ is analogous to $\mathrm{path}_2(x, y)$, as explained above.

Note that we could as well have taken $\chi_1(x,y) = \chi(x,y)$, because, by definition, an edge in the output graph is only drawn if both nodes it is incident to exist, so this need not be checked explicitly. However, as observed before, such a check can always be added to $\chi_1(x,y)$, and that is what we have done here. Note also that $\mathcal{T}$ is in MSO-TT, but not in $\text{MSO-TGT}_{\text{dir}}$; in fact, it is not difficult to see that, for every $\mathcal{T} \in \text{MSO-TGT}_{\text{dir}}$ and every input tree t, the height of $\mathcal{T}(t)$ is linear in the height of t, where the constant is the copy number of T (cf. the similar statement on size in the previous example).

Composition We end this section by stating a well-known and basic property of MSO definable graph transductions, and its consequences for the specific transductions considered in this paper. It is proved, e.g., in Proposition 3.2(2) of [Cou3].

Proposition 1. MSO-GT *is closed under composition.*

An immediate consequence of this proposition is that MSO-TT is closed under composition, and that $\text{MSO-TT} \circ \text{MSO-TGT} \subseteq \text{MSO-TGT}$. To see that also $\text{MSO-GT}_{\text{dir}}$, $\text{MSO-TT}_{\text{dir}}$, and MSO-REL are closed under composition, it suffices to know the following about the proof of Proposition 1. If T' and T'' are MSO graph transducers with copy sets C' and C'', then their composition is realized by an MSO graph transducer T with copy set $C'' \times C'$. Moreover, for every input graph g_1, the output graphs $\mathcal{T}_{\text{gr}}(g_1)$ and $\mathcal{T}''_{\text{gr}}(\mathcal{T}'_{\text{gr}}(g_1))$ are isomorphic, with node $((c'',c'),u)$ corresponding to node $(c'',(c',u))$.

Proposition 2. $\text{MSO-GT}_{\text{dir}}$, MSO-TT, $\text{MSO-TT}_{\text{dir}}$, *and* MSO-REL *are closed under composition;* $\text{MSO-TT} \circ \text{MSO-TGT} \subseteq \text{MSO-TGT}$ *and* $\text{MSO-TT}_{\text{dir}} \circ \text{MSO-TGT}_{\text{dir}} \subseteq \text{MSO-TGT}_{\text{dir}}$.

It will follow from our results that $\text{MSO-TGT}_{\text{dir}}$ is also closed under composition (see Section 9), thus strengthening the last inclusion in Proposition 2. It is straightforward to show that MSO-TGT is *not* closed under composition, by a size argument similar to the one in Example 1(5, binary).

4 Attributed tree transducers

We will show that MSO definable term transductions can be computed by attribute grammars, and that MSO definable tree transductions can be computed by SUR attribute grammars. To obtain precise characterizations, we consider two types of tree transducers that are based on attribute grammars: the relabeling attribute grammar (see [BloEng]) and the attributed tree transducer (see [Fül, EngFil, FHVV, KühVog1]). We prove that MSO term graph transducers have the same power as two-stage attribute grammars, of which the first stage is a relabeling attribute grammar, and the second stage is an attributed tree transducer. For MSO tree transducers the second phase is SUR. Such two-stage

attribute grammars will be called attributed tree transducers with look-ahead ($\mathrm{att}^{\mathrm{R}}$).

A relabeling attribute grammar is a quite restricted type of tree transducer: it only changes the labels of the nodes of the input tree. All its attributes have finitely many values, and, for each node of a given input tree, the new label of the node is the value of its meaning attribute. Since it is finite-valued, a relabeling attribute grammar can be viewed as a finite-state tree automaton that relabels the nodes of the tree. It is the look-ahead stage of an $\mathrm{att}^{\mathrm{R}}$.

An attributed tree transducer is a much more powerful device. It is an attribute grammar in which all attribute values are trees and the semantic rules are limited to involve substitution only. For a given input tree, the output tree is the value of the meaning attribute at the root. It is the computation stage of an $\mathrm{att}^{\mathrm{R}}$.

In this section we first define relabeling attribute grammars, attributed tree transducers, and attributed tree transducers with look-ahead. Then we discuss a normal form for attributed tree transducers, and we define the notion of a semantic graph for attributed tree transducers in this normal form.

4.1 Formal definitions

Let Σ and Δ be operator alphabets. A *relabeling attribute grammar* from Σ to Δ is a finite-valued noncircular attribute grammar $G = (\Sigma, S, I, \Omega, W, R, \alpha_{\mathrm{m}})$ with $W(\alpha_{\mathrm{m}}) = \Delta$, such that for every $t \in T_\Sigma$ and $u \in V_t$, $\mathrm{dec}_{G,t}(\langle \alpha_{\mathrm{m}}, u\rangle)$ has the same rank as $\mathrm{lab}_t(u)$. The *tree transduction computed by* G is the total function $r(G) = \{(t, t') \in T_\Sigma \times T_\Delta \mid V_{t'} = V_t, E_{t'} = E_t, \text{ and } \mathrm{lab}_{t'}(u) = \mathrm{dec}_{G,t}(\langle \alpha_{\mathrm{m}}, u\rangle) \text{ for all } u \in V_t\}$; $r(G)$ is called an *attributed relabeling*. The set of all attributed relabelings is denoted ATT-REL. In Section 6.1 we will show that ATT-REL equals the class MSO-REL of MSO relabelings.

Let Σ, Δ be operator alphabets (the input and output alphabet, respectively). An *attributed tree transducer* (*att*, for short) from Σ to Δ is an attribute grammar $G = (\Sigma, S, I, \Omega, W, R, \alpha_{\mathrm{m}})$ with the following two properties:

- $\Omega = \{T_\Delta\}$, and
- every semantic rule is of the form

$$\langle \alpha_0, i_0\rangle = f(\langle \alpha_1, i_1\rangle, \ldots, \langle \alpha_k, i_k\rangle),$$

  where for all $t_1, \ldots, t_k \in T_\Delta$,

$$f(t_1, \ldots, t_k) = r[\xi_1 \mapsto t_1, \ldots, \xi_k \mapsto t_k]$$

  for some linear $r \in T_\Delta(\{\xi_1, \ldots, \xi_k\})$.

For the notation and terminology used for r see the end of Section 2.1.

Note that we did not require G to be noncircular. Although we are only interested in noncircular attributed tree transducers, we need circular ones as a technical tool (see Sections 7 and 8). A circular attributed tree transducer only translates input trees with a noncircular dependency graph.

The *tree transduction computed by* attributed tree transducer G is the partial function $\mathcal{G}$ from T_Σ to T_Δ such that for every tree $t \in T_\Sigma$ with noncircular dependency graph $D(t)$, $\mathcal{G}(t) = \mathrm{dec}_{G,t}(\langle \alpha_\mathrm{m}, \mathrm{root}(t)\rangle)$.

The class of all tree transductions that can be computed by *noncircular* attributed tree transducers is denoted ATT. Note that ATT contains total functions only. The class of tree transductions computed by noncircular attributed tree transducers that satisfy the single use restriction (SUR), is denoted $\mathrm{ATT}_{\mathrm{sur}}$. The corresponding classes of tree transductions computed by OS attributed tree transducers are denoted $\mathrm{ATT}_{\mathrm{os}}$ and $\mathrm{ATT}_{\mathrm{os,sur}}$, respectively.

From here on, we will identify the right-hand side $f(\langle \alpha_1, i_1\rangle, \ldots, \langle \alpha_k, i_k\rangle)$ of a semantic rule as above, with the tree $r[\xi_1 \mapsto \langle \alpha_1, i_1\rangle, \ldots, \xi_k \mapsto \langle \alpha_k, i_k\rangle]$. Note that, since the $\langle \alpha_j, i_j\rangle$ are all distinct (see Section 2.3), this is a linear term in $T_\Delta((I \cup S) \times \mathbb{N})$. As an example, with $\delta \in \Delta_2$ and $c \in \Delta_0$, the semantic rule

$$\langle \alpha, 0\rangle = f(\langle \beta, 1\rangle, \langle \alpha, 2\rangle), \quad \text{with} \quad f(t_1, t_2) = \delta(t_1, \delta(c, t_2))$$

is identified with

$$\langle \alpha, 0\rangle = \delta(\langle \beta, 1\rangle, \delta(c, \langle \alpha, 2\rangle)).$$

Note that $f(t_1, t_2) = r[\xi_1 \mapsto t_1, \xi_2 \mapsto t_2]$ for $r = \delta(\xi_1, \delta(c, \xi_2)))$. As a second example, the semantic rule $\langle \alpha, 0\rangle = f(\langle \beta, 1\rangle)$ with $f(t) = t = \xi_1[\xi_1 \mapsto t]$, is identified with $\langle \alpha, 0\rangle = \langle \beta, 1\rangle$.

In exactly the same way, for an input tree t, the right-hand sides of semantic instructions in $R(t)$ will be identified with trees in $T_\Delta(A(t))$. Thus, for a decoration dec of t and an instruction $\langle \alpha, u\rangle = r$, $\mathrm{dec}(\langle \alpha, u\rangle)$ is the result of substituting $\mathrm{dec}(\langle \beta, v\rangle)$ for every $\langle \beta, v\rangle$ in r.

Linearity of the terms used in the semantic rules is a convenient technical detail which is intuitively required for SUR att's. For att's that are not SUR it is an inessential restriction, because it can always be achieved by duplicating attributes. To see this, we give an example of a rule that is not linear, and transform it to two linear rules. With $\delta \in \Delta_2$, a typical nonlinear rule would be $\langle \alpha, 0\rangle = \delta(\langle \alpha, 1\rangle, \langle \alpha, 1\rangle)$. We can make it linear by adding an attribute α' with rule $\langle \alpha', 0\rangle = \langle \alpha, 1\rangle$, and changing the rule for α into $\langle \alpha, 0\rangle = \delta(\langle \alpha', 0\rangle, \langle \alpha, 1\rangle)$. Note that this transformation does not preserve the SUR because $\langle \alpha, 1\rangle$ is used twice.

An *attributed tree transducer with look-ahead* (*att*^{R}, for short) from Σ to Δ is a pair $G = (G_1, G_2)$ where G_1 is a relabeling attribute grammar from Σ to Ω, and G_2 is an attributed tree transducer from Ω to Δ, for some operator alphabet Ω. The *tree transduction computed by* G is $r(G_1) \circ \mathcal{G}_2$. The att$^{\mathrm{R}}$ G is noncircular, OS, or SUR, if G_2 is (with no restrictions on G_1 except that it is noncircular). Obviously, the class of all tree transductions that can be computed by noncircular attributed tree transducers with look-ahead is ATT-REL $\circ$ ATT. Restricting the att$^{\mathrm{R}}$'s to be SUR, or OS, or both, they compute the classes ATT-REL $\circ$ $\mathrm{ATT}_{\mathrm{sur}}$, ATT-REL $\circ$ $\mathrm{ATT}_{\mathrm{os}}$, and ATT-REL $\circ$ $\mathrm{ATT}_{\mathrm{os,sur}}$, respectively.

We now give some examples, numbered (3)–(6), corresponding to the MSO term graph transducers of Example 1(3)–(6). All examples are noncircular.

Example 2. (3, stars) The tree transduction $\mathcal{T}$ of Example 1(3, stars) is computed by an att G with one synthesized attribute α, which is also the meaning attribute, and semantic rules $R(a) = \{\langle\alpha,0\rangle = a\}$ and $R(\sigma) = \{\langle\alpha,0\rangle = \sigma(*(\langle\alpha,1\rangle), *(\langle\alpha,2\rangle))\}$. Note that G is OS and SUR, and so $\mathcal{T} \in \text{ATT}_{\text{os,sur}}$.

(4, path) The tree transduction $\mathcal{T}$ of Example 1(4, path) cannot be computed by an attributed tree transducer, as proved in [FülVag]. We now show that it can be computed by an attributed tree transducer with look-ahead, i.e., that it is the composition of an attributed relabeling $r(G_1)$ and an attributed tree transduction $\mathcal{G}_2$. The relabeling attribute grammar G_1 has a synthesized attribute 'ns' that counts the number of descendants that are labeled by $*$, with $W(\text{ns}) = \{0, 1, \text{many}\}$, where 'many' means 'more than one'. It has meaning attribute α that adds to the label of each node the values of the ns-attribute of its children, with $W(\alpha) = \{a, *\} \cup \{(\sigma, i, j) \mid i, j \in W(\text{ns})\}$. The semantic rules of G_1 are

$$R(\sigma) = \{\ \langle\text{ns},0\rangle = \langle\text{ns},1\rangle + \langle\text{ns},2\rangle, \quad \langle\alpha,0\rangle = (\sigma, \langle\text{ns},1\rangle, \langle\text{ns},2\rangle)\ \},$$

where addition is defined in the obvious way ($1 + 1 =$ many, and many plus anything equals many),

$$\begin{aligned} R(a) &= \{\ \langle\text{ns},0\rangle = 0, \quad \langle\alpha,0\rangle = a\ \}, \text{ and} \\ R(*) &= \{\ \langle\text{ns},0\rangle = 1, \quad \langle\alpha,0\rangle = *\ \}. \end{aligned}$$

The att G_2 has one synthesized attribute β, with semantic rules

$$\begin{aligned} R(a) &= \{\ \langle\beta,0\rangle = a\ \}, \\ R(*) &= \{\ \langle\beta,0\rangle = *\ \}, \end{aligned}$$

and, for $i, j \in W(\text{ns})$,

$$R(\sigma,i,j) = \begin{cases} \{\ \langle\beta,0\rangle = 1(\langle\beta,1\rangle)\ \} & \text{if } i = 1 \text{ and } j = 0, \\ \{\ \langle\beta,0\rangle = 2(\langle\beta,2\rangle)\ \} & \text{if } i = 0 \text{ and } j = 1, \\ \{\ \langle\beta,0\rangle = a\ \} & \text{otherwise.} \end{cases}$$

It should be clear that $\mathcal{T} = r(G_1) \circ \mathcal{G}_2$, and so $\mathcal{T} \in \text{ATT-REL} \circ \text{ATT}_{\text{os,sur}}$.

(5, binary) The term transduction $\mathcal{T}$ of Example 1(5, binary) is computed by an att G with two synthesized attributes α and α' (where α is the meaning attribute), and semantic rules $R(a) = \{\langle\alpha,0\rangle = a, \langle\alpha',0\rangle = a\}$ and $R(\sigma) = \{\langle\alpha,0\rangle = \sigma(\langle\alpha,1\rangle, \langle\alpha',1\rangle), \langle\alpha',0\rangle = \sigma(\langle\alpha,1\rangle, \langle\alpha',1\rangle)\}$. Thus, $\mathcal{T} \in \text{ATT}_{\text{os}}$. Note that $\mathcal{T}$ cannot be computed by a SUR attributed tree transducer with look-ahead, because for a transduction in $\text{ATT-REL} \circ \text{ATT}_{\text{sur}}$ the size of the output tree is linear in the size of the input tree, where the constant is the number of attributes of the att times the maximal size of the right-hand sides of its semantic rules. Note also that if semantic rules with nonlinear right-hand sides were allowed, G could do without α', and have rules $\langle\alpha,0\rangle = a$ for a, and $\langle\alpha,0\rangle = \sigma(\langle\alpha,1\rangle, \langle\alpha,1\rangle)$ for σ.

(6, yield) Finally, we show that the tree transduction $\mathcal{T}$ of Example 1(6, yield) is in ATT-REL $\circ$ $\mathrm{ATT}_{\mathrm{sur}}$, i.e., can be computed by a SUR $\mathrm{att}^{\mathrm{R}}$ (G_1, G_2). The obvious idea is to construct an att G_2 with a synthesized attribute 'up' and an inherited attribute 'down', that makes a depth-first right-to-left walk through the input tree, adding one symbol to the output tree at each leaf. The only problem is that it has to treat the rightmost leaf differently from the other leaves. Thus, the relabeling attribute grammar G_1 will mark the rightmost leaf. The output alphabet of G_1 is $\Sigma \cup \Delta_0$. It has an inherited attribute 'rm' with $W(\mathrm{rm}) = \{\mathrm{true}, \mathrm{false}\}$, root rule $\langle \mathrm{rm}, 0\rangle = \mathrm{true}$, and, for $\sigma \in \Sigma_2$, internal rules $\langle \mathrm{rm}, 1\rangle = \mathrm{false}$ and $\langle \mathrm{rm}, 2\rangle = \langle \mathrm{rm}, 0\rangle$. It has meaning attribute α with internal rule $\langle \alpha, 0\rangle = \sigma$ for $\sigma \in \Sigma_2$, and the following internal rule for $\sigma \in \Sigma_0$:

$$\langle \alpha, 0\rangle = \begin{cases} \hat{\sigma} & \text{if } \langle \mathrm{rm}, 0\rangle = \mathrm{true}, \\ \sigma & \text{otherwise.} \end{cases}$$

Now the semantic rules of G_2 are: for $\sigma \in \Sigma_2$,

$$R(\sigma) = \{\ \langle \mathrm{down}, 2\rangle = \langle \mathrm{down}, 0\rangle, \quad \langle \mathrm{down}, 1\rangle = \langle \mathrm{up}, 2\rangle, \quad \langle \mathrm{up}, 0\rangle = \langle \mathrm{up}, 1\rangle\ \};$$

for $\sigma \in \Sigma_0$, $R(\hat{\sigma}) = \{\langle \mathrm{up}, 0\rangle = \hat{\sigma}\}$ and $R(\sigma) = \{\langle \mathrm{up}, 0\rangle = \sigma(\langle \mathrm{down}, 0\rangle)\}$; and $R(\mathrm{root})$ contains a "dummy" rule $\langle \mathrm{down}, 0\rangle = \bot$, where $\bot$ is any element of Δ_0.

It should be clear that $\mathcal{T} = r(G_1) \circ \mathcal{G}_2$. Thus, $\mathcal{T} \in$ ATT-REL $\circ$ $\mathrm{ATT}_{\mathrm{sur}}$. Note that $\mathcal{T}$ cannot be computed by an OS attributed tree transducer with look-ahead, because for a transduction in ATT-REL $\circ$ $\mathrm{ATT}_{\mathrm{os}}$ the height of the output tree is linear in the height of the input tree, where the constant is the number of attributes of the att times the maximal height of the right-hand sides of its semantic rules. □

4.2 Operator form

To simplify proofs we will consider attributed tree transducers for which every right-hand side of a semantic rule contains exactly one output symbol. Here we show that this is, in a certain sense, a normal form for attributed tree transducers.

An attributed tree transducer G from Σ to Δ is in *operator form* if every semantic rule of G is of the form $\langle \alpha_0, i_0\rangle = \delta(\langle \alpha_1, i_1\rangle, \ldots, \langle \alpha_k, i_k\rangle)$ for some $\delta \in \Delta_k$. Thus, for a tree $t \in T_\Sigma$, the semantic instructions in $R(t)$ are of the form $\langle \alpha_0, u_0\rangle = \delta(\langle \alpha_1, u_1\rangle, \ldots, \langle \alpha_k, u_k\rangle)$, and a decoration dec of t should satisfy $\mathrm{dec}(\langle \alpha_0, u_0\rangle) = \delta(\mathrm{dec}(\langle \alpha_1, u_1\rangle), \ldots, \mathrm{dec}(\langle \alpha_k, u_k\rangle))$, accordingly.

We first show that for every att there is an equivalent one for which every right-hand side of a semantic rule contains at most one output symbol.

Lemma 3. *For every att G there is an att G_1 such that $\mathcal{G}_1 = \mathcal{G}$ and every semantic rule of G_1 is either of the form $\langle \alpha_0, i_0\rangle = \delta(\langle \alpha_1, i_1\rangle, \ldots, \langle \alpha_k, i_k\rangle)$, for some $\delta \in \Delta_k$, or of the form $\langle \alpha_0, i_0\rangle = \langle \alpha_1, i_1\rangle$. If G is* SUR *or* OS*, then so is G_1.*

Proof. The straightforward idea is to introduce new attributes for the subtrees of the right-hand side of a semantic rule. As an example, the semantic rule $\langle\alpha,0\rangle = \delta(\langle\beta,1\rangle,\delta(c,\langle\alpha,2\rangle))$ can be changed into the semantic rules $\langle\alpha,0\rangle = \delta(\langle\beta,1\rangle,\langle\alpha',0\rangle)$, $\langle\alpha',0\rangle = \delta(\langle\alpha'',0\rangle,\langle\alpha,2\rangle)$, and $\langle\alpha'',0\rangle = c$, where α' and α'' are new synthesized attributes.

Formally we transform G into G_1 by iterating the following procedure. Let $\langle\alpha,i\rangle = r$ be a semantic rule such that $r = \delta(r_1,\ldots,r_j,\ldots,r_k)$ and r_j is not an attribute occurrence, i.e., $r_j \notin (S \cup I) \times \mathbb{N}$. Replace this semantic rule by the two semantic rules $\langle\alpha,i\rangle = \delta(r_1,\ldots,r_{j-1},\langle\alpha',0\rangle,r_{j+1},\ldots,r_k)$ and $\langle\alpha',0\rangle = r_j$, where α' is a new, synthesized, attribute. Moreover, since the new attribute is not used in the $R(\sigma)$'s to which $\langle\alpha,i\rangle = r$ does not belong, it is defined there by a "dummy" rule $\langle\alpha',0\rangle = \bot$, where $\bot$ is any element of Δ_0.

It should be clear that one such replacement does not change $\mathcal{G}$ and preserves the SUR and OS properties. Iteration of these replacements leads to an att of the required form. □

A semantic rule of the form $\langle\alpha_0,i_0\rangle = \langle\alpha_1,i_1\rangle$ is called a *copy rule*. To put an att G from Σ to Δ into operator form, we just replace such a copy rule by the rule $\langle\alpha_0,i_0\rangle = \mathrm{id}(\langle\alpha_1,i_1\rangle)$, where 'id' is a new output symbol of rank 1, the *identity operator*. The new att outputs trees over $\Delta \cup \{\mathrm{id}\}$ from which then all occurrences of id have to be removed to obtain the output tree of G.

For an operator alphabet Δ and a symbol 'id' of rank 1, not in Δ, let Δ^{id} denote the operator alphabet $\Delta \cup \{\mathrm{id}\}$. For a tree t over Δ^{id}, the *pruning* of t is the tree $\mathrm{prune}(t)$ over Δ which is obtained from t by pruning all occurrences of id. In other words, $\mathrm{prune}(\mathrm{id}(t)) = \mathrm{prune}(t)$, and, for $\delta \in \Delta_k$, $\mathrm{prune}(\delta(t_1,\ldots,t_k)) = \delta(\mathrm{prune}(t_1),\ldots,\mathrm{prune}(t_k))$.

The effect of the identity operator, in general, is stated in the following lemma. In the statement of the lemma we apply the function 'prune' to right-hand sides r of semantic rules of an att from Σ to Δ^{id}. This is well defined because r is (identified with) a tree over the operator alphabet $\Delta \cup \{\mathrm{id}\} \cup ((I \cup S) \times \mathbb{N})$.

Lemma 4. *Let G be an att from Σ to Δ^{id}, and let G_{prune} be the att from Σ to Δ which is obtained from G by changing every semantic rule $\langle\alpha,i\rangle = r$ into $\langle\alpha,i\rangle = \mathrm{prune}(r)$. Then $\mathcal{G}_{\mathrm{prune}} = \mathcal{G} \circ \mathrm{prune}$. Moreover, G is* OS *iff G_{prune} is* OS*, and, for $t \in T_\Sigma$, G is* SUR *on t iff G_{prune} is* SUR *on t.*

Proof. Obviously, a tree $t \in T_\Sigma$ has the same dependency graph $D(t)$ in both G and G_{prune}. Hence, G_{prune} is SUR on t iff G is SUR on t, and G_{prune} is noncircular on t iff G is noncircular on t. For such a noncircular t, it is straightforward to show that the mapping $\mathrm{dec}_{G,t} \circ \mathrm{prune}$ is a decoration of t for G_{prune}. Thus, since t has a unique decoration, $\mathrm{dec}_{G_{\mathrm{prune}},t} = \mathrm{dec}_{G,t} \circ \mathrm{prune}$, and so $\mathcal{G}_{\mathrm{prune}} = \mathcal{G} \circ \mathrm{prune}$. □

We now show in which sense operator form is a normal form for attributed tree transducers.

Lemma 5. *For every att G from Σ to Δ there is an att G' in operator form from Σ to Δ^{id} such that $\mathcal{G} = \mathcal{G}' \circ \mathrm{prune}$. If G is* SUR *or* OS*, then so is G'.*

Proof. Let G_1 be the att obtained from G by Lemma 3, and change G_1 into G' by replacing every copy rule $\langle \alpha_0, i_0 \rangle = \langle \alpha_1, i_1 \rangle$ by $\langle \alpha_0, i_0 \rangle = \mathrm{id}(\langle \alpha_1, i_1 \rangle)$. Then $G'_{\mathrm{prune}} = G_1$, and so, by Lemma 4, $\mathcal{G} = \mathcal{G}_1 = \mathcal{G}'_{\mathrm{prune}} = \mathcal{G}' \circ \mathrm{prune}$. □

The operator form is used in Section 5 to facilitate the proof that every attributed tree transduction is an MSO definable term transduction. In fact, for an att in operator form the output tree is the unfolding of a term graph which is closely related to the dependency graph of the input tree, and which is therefore MSO definable in a straightforward way. This term graph is defined in the next subsection.

4.3 Semantic graphs

If G is an att in operator form, and t is an input tree with noncircular dependency graph $D(t)$, then the nodes and edges of $D(t)$ can be labeled in a straightforward way such that, after reversing the direction of all its edges, it turns into a term graph of which the unfolding is the output tree $\mathcal{G}(t)$. This term graph will be called the "semantic graph" of t. In fact, an attribute of $D(t)$ is labeled with the single operator $\delta \in \Delta$ used in the semantic instruction that defines the attribute, and an edge of $D(t)$ is labeled according to the order of the attributes in the instruction that defines the dependency. Moreover, the mark # is added to the label of $\langle \alpha_{\mathrm{m}}, \mathrm{root}(t) \rangle$. We now turn to the formal definition.

Let G be an att from Σ to Δ in operator form. The *semantic graph* $S_G(t)$ of a tree $t \in T_\Sigma$ is the graph $S_G(t) = (V, E, \mathrm{lab})$ over $(\Delta_\#, \mathrm{rki}(\Delta))$, where $V = A(t)$, and E and lab are determined as follows. If $\langle \alpha_0, u_0 \rangle = \delta(\langle \alpha_1, u_1 \rangle, \ldots, \langle \alpha_k, u_k \rangle)$ is a semantic instruction in $R(t)$, then $(\langle \alpha_0, u_0 \rangle, j, \langle \alpha_j, u_j \rangle)$ is in E for every $j \in [1, k]$, $\mathrm{lab}(\langle \alpha_0, u_0 \rangle) = \delta$ if $\langle \alpha_0, u_0 \rangle \neq \langle \alpha_{\mathrm{m}}, \mathrm{root}(t) \rangle$, and $\mathrm{lab}(\langle \alpha_0, u_0 \rangle) = (\delta, \#)$ if $\langle \alpha_0, u_0 \rangle = \langle \alpha_{\mathrm{m}}, \mathrm{root}(t) \rangle$.

It should be clear that $S_G(t)$ satisfies all requirements of a term graph over Δ, except that it may be circular. Hence, for every $t \in T_\Sigma$, $S_G(t)$ is a term graph over Δ if and only if $S_G(t)$ is noncircular if and only if $D(t)$ is noncircular. Moreover, if $S_G(t)$ is a term graph, then $\mathrm{root}(S_G(t)) = \langle \alpha_{\mathrm{m}}, \mathrm{root}(t) \rangle$. Also, $S_G(t)$ is a forest if and only if G is SUR on t.

We now show that the output tree is the unfolding of the semantic graph of the input tree.

Lemma 6. *Let G be an att from Σ to Δ in operator form, and let $t \in T_\Sigma$ be an input tree with noncircular $D(t)$. Then $\mathcal{G}(t) = \mathrm{unfold}(S_G(t))$. If, moreover, G is* SUR *on t, then $\mathcal{G}(t) = \mathrm{clean}(S_G(t))$.*

Proof. Let $S = S_G(t)$. Since $D(t)$ is noncircular, S is a term graph over Δ. Define the mapping $\mathrm{dec} : A(t) \to T_\Delta$ such that $\mathrm{dec}(\langle \alpha, u \rangle) = \mathrm{unf}_S(\langle \alpha, u \rangle)$. Since t has a unique decoration $\mathrm{dec}_{G,t}$, it suffices to show that dec is a decoration of t. When this is established, it follows that $\mathrm{dec}_{G,t} = \mathrm{dec}$, and hence that $\mathcal{G}(t) = \mathrm{dec}_{G,t}(\langle \alpha_{\mathrm{m}}, \mathrm{root}(t) \rangle) = \mathrm{unf}_S(\langle \alpha_{\mathrm{m}}, \mathrm{root}(t) \rangle) = \mathrm{unf}_S(\mathrm{root}(S)) = \mathrm{unfold}(S)$.

A mapping is a decoration of t if all semantic instructions in $R(t)$ are obeyed. If $R(t)$ contains a semantic instruction $\langle \alpha_0, u_0 \rangle = \delta(\langle \alpha_1, u_1 \rangle, \ldots, \langle \alpha_k, u_k \rangle)$, then

$\mathrm{lab}_S(\langle\alpha_0, u_0\rangle) = \delta$ (or $(\delta, \#)$) and $(\langle\alpha_0, u_0\rangle, j, \langle\alpha_j, u_j\rangle) \in E_S$ for all $j \in [1, k]$, and hence $\mathrm{unf}_S(\langle\alpha_0, u_0\rangle) = \delta(\mathrm{unf}_S(\langle\alpha_1, u_1\rangle), \ldots, \mathrm{unf}_S(\langle\alpha_k, u_k\rangle))$, i.e., dec obeys this semantic instruction.

If G is SUR on t, then $S_G(t)$ is a forest. Clearly, $\mathrm{unfold}(\tau) = \mathrm{clean}(\tau)$ for any forest τ. □

Example 3. To illustrate Lemma 6, we give a simple example of a (SUR) attributed tree transducer G in operator form. Given a tree t, it reproduces the monadic tree that constitutes the path from the root of t to its leftmost leaf, cf. Example 2(4, path). The input alphabet of G is $\Sigma = \Sigma_2 \cup \Sigma_0$, with $\Sigma_2 = \{\sigma\}$ and $\Sigma_0 = \{a, b\}$, and the output alphabet is $\Delta = \Delta_1 \cup \Delta_0$, with $\Delta_1 = \{\sigma\}$ and $\Delta_0 = \{a, b\}$. There is only one, synthesized, attribute α. The rules are simple: the single rule in $R(\sigma)$ is $\langle\alpha, 0\rangle = \sigma(\langle\alpha, 1\rangle)$, $R(a)$ contains the single rule $\langle\alpha, 0\rangle = a$, and $R(b)$ the single rule $\langle\alpha, 0\rangle = b$. For the tree $t = \sigma(\sigma(a, a), \sigma(b, a))$,

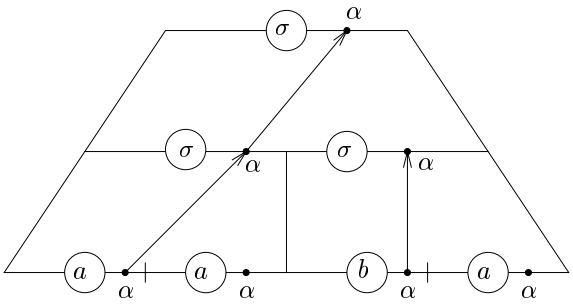


Fig. 3. Dependency graph

the dependency graph $D(t)$ is given in Fig. 3, the semantic graph $S_G(t)$ in Fig. 4, and the output tree $\mathcal{G}(t) = \sigma(\sigma(a))$ in Fig. 5. Note that since G is SUR, $\mathrm{unfold}(S_G(t)) = \mathrm{clean}(S_G(t))$. □

5 From ATT to MSO

In this section we will prove that every noncircular attributed tree transducer can be simulated by an MSO term graph transducer, i.e., that ATT ⊆ MSO-TGT. By Lemma 5 it suffices to show this for att's in operator form, provided we also show that MSO-TGT ∘ {prune} ⊆ MSO-TGT.

Recall that for an MSO term graph transducer T and an input tree t, $\mathcal{T}(t) = \mathrm{unfold}(\mathcal{T}_{\mathrm{gr}}(t))$; thus, the output tree is obtained by unfolding the term graph $\mathcal{T}_{\mathrm{gr}}(t)$, which is the output graph of the MSO graph transducer T.

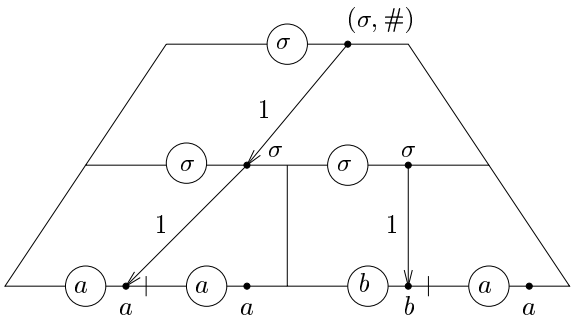


Fig. 4. Semantic graph

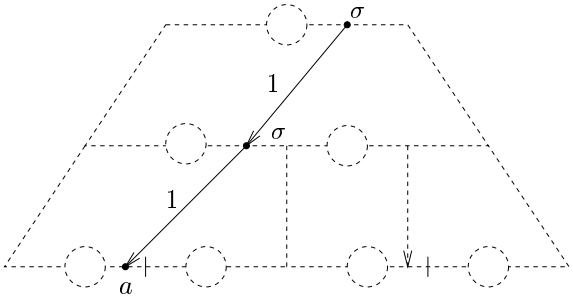


Fig. 5. Output tree

Lemma 7. *For every noncircular attributed tree transducer G in operator form there is an* MSO *term graph transducer T such that $\mathcal{T} = \mathcal{G}$. Moreover, if G is* SUR*, then T is an* MSO *tree transducer, and if G is* OS*, then T is direction preserving.*

Proof. Let $G = (\Sigma, S, I, \Omega, W, R, \alpha_{\mathrm{m}})$ be a noncircular att in operator form with $\Omega = \{T_\Delta\}$. We will abbreviate $S \cup I$ by A.

By Lemma 6, $\mathcal{G}(t) = \mathrm{unfold}(S_G(t))$ for every $t \in T_\Sigma$. Thus, it suffices to define an MSO graph transducer T such that, for every $t \in T_\Sigma$, $\mathcal{T}_{\mathrm{gr}}(t) = S_G(t)$, the semantic graph of t. To simplify the description of this T, we introduce semantic graphs $S(\sigma)$, for $\sigma \in \Sigma$, and $S(\mathrm{root})$. These graphs are (partially) labeled versions of the usual dependency graphs $D(\sigma)$ and $D(\mathrm{root})$, as defined in Section 2.3, with the edges reversed. They are the atomic graphs from which all semantic graphs $S_G(t)$ are built, just as the dependency graphs $D(t)$ are built from the $D(\sigma)$ and $D(\mathrm{root})$.

For $x \in \Sigma \cup \{\mathrm{root}\}$, the *semantic graph of x* is the graph $S(x) = (V, E, \mathrm{lab})$ over $(\Delta, \mathrm{rki}(\Delta))$ where $V = A \times [0, \mathrm{rk}(\sigma)]$ if $x = \sigma \in \Sigma$ and $V = A \times \{0\}$ if $x =$

root, and E and lab are determined as follows. If $\langle \alpha_0, i_0 \rangle = \delta(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_k, i_k \rangle)$ is a rule in $R(x)$, then $(\langle \alpha_0, i_0 \rangle, j, \langle \alpha_j, i_j \rangle)$ is in E for every $j \in [1, k]$, and $\mathrm{lab}(\langle \alpha_0, i_0 \rangle) = \delta$. Note that lab is a partial function.

The MSO graph transducer defining the semantic graphs of the input trees of G is

$$T = (A, \{\psi_{\delta,\alpha}\}_{\delta \in \Delta_{\#}, \alpha \in A}, \{\chi_{j,\alpha,\alpha'}\}_{j \in \mathrm{rki}(\Delta), \alpha, \alpha' \in A}),$$

where the copy set, edge formulas, and node formulas are defined as follows (in that order).

- The set of copy names is $A = S \cup I$, the set of attributes of G. For each node u of an input tree t, T makes a copy (α, u) for every attribute α, corresponding to node $\langle \alpha, u \rangle$ of $S_G(t)$. Thus, all these copies are nodes of the output graph $S_G(t)$.
- The edge formulas check for a dependency between attributes. In particular, the edge formula $\chi_{j,\alpha,\alpha'}(x, y)$ checks the semantic rules to see if there is a semantic instruction that defines $\langle \alpha, x \rangle$ in terms of $\langle \alpha', y \rangle$. In what follows $\mathrm{edg}_0(x, y)$ stands for $x = y$.

  For all $j \in \mathrm{rki}(\Delta)$ and $\alpha, \alpha' \in A$, the edge formula $\chi_{j,\alpha,\alpha'}(x, y)$ is defined to be the disjunction of all formulas $\exists z(\mathrm{lab}_\sigma(z) \wedge \mathrm{edg}_i(z, x) \wedge \mathrm{edg}_{i'}(z, y))$, for all $\sigma \in \Sigma$ and $i, i' \in [0, \mathrm{rk}(\sigma)]$ such that $(\langle \alpha, i \rangle, j, \langle \alpha', i' \rangle)$ is an edge of $S(\sigma)$, and the formula $\mathrm{root}(x) \wedge x = y$ in case $(\langle \alpha, 0 \rangle, j, \langle \alpha', 0 \rangle)$ is an edge of $S(\mathrm{root})$.
- The node formulas determine the operators in Δ by which the attributes are defined, and they determine the root of the semantic graph $S_G(t)$. Again, $\mathrm{edg}_0(x, y)$ stands for $x = y$.

  Let $\delta \in \Delta$ and $\alpha \in A$. We first consider the case that $\alpha \neq \alpha_{\mathrm{m}}$. Then the node formula $\psi_{(\delta,\#),\alpha}(x)$ is defined to be false, and the node formula $\psi_{\delta,\alpha}(x)$ is defined to be the disjunction of all formulas $\exists z(\mathrm{lab}_\sigma(z) \wedge \mathrm{edg}_i(z, x))$, for all $\sigma \in \Sigma$ and $i \in [0, \mathrm{rk}(\sigma)]$ such that $\langle \alpha, i \rangle$ has label δ in $S(\sigma)$, and the formula $\mathrm{root}(x)$ in case $\langle \alpha, 0 \rangle$ has label δ in $S(\mathrm{root})$.

  To define the node formulas for α_{m}, let $\psi'_{\delta,\alpha_{\mathrm{m}}}(x)$ be the disjunction of all formulas $\exists z(\mathrm{lab}_\sigma(z) \wedge \mathrm{edg}_i(z, x))$, for all σ, i such that $\langle \alpha_{\mathrm{m}}, i \rangle$ has label δ in $S(\sigma)$. Then $\psi_{(\delta,\#),\alpha_{\mathrm{m}}}(x) = \mathrm{root}(x) \wedge \psi'_{\delta,\alpha_{\mathrm{m}}}(x)$, and $\psi_{\delta,\alpha_{\mathrm{m}}}(x) = \neg\, \mathrm{root}(x) \wedge \psi'_{\delta,\alpha_{\mathrm{m}}}(x)$.

It should be obvious that the above formulas closely correspond to the definition of $S_G(t)$ in Section 4.3. More exactly, for $\delta \in \Delta$ and $\alpha \neq \alpha_{\mathrm{m}}$, $(t, u) \models \psi_{\delta,\alpha}(x)$ iff there is a semantic instruction of the form $\langle \alpha, u \rangle = \delta(\ldots) \in R(t)$, and similarly for the other node formulas. Also, $(t, u, v) \models \chi_{j,\alpha,\alpha'}(x, y)$ iff there is a semantic instruction in $R(t)$ of the form $\langle \alpha, u \rangle = \delta(\ldots, \langle \alpha', v \rangle, \ldots)$ where $\langle \alpha', v \rangle$ is the j-th argument of δ. This implies that $\mathcal{T}_{\mathrm{gr}}(t) = S_G(t)$ for every $t \in T_\Sigma$.

If G is OS and $(\langle \alpha, u \rangle, j, \langle \beta, v \rangle)$ is an edge of $S_G(t)$, then v is a descendant of u (in fact, either $v = u$ or v is a child of u). Hence T is direction preserving in that case.

Now assume that G is SUR. Then, for every tree $t \in T_\Sigma$, $S_G(t)$ is a forest. Thus, T need not be a tree transducer. However, in this case $\mathcal{G}(t) = \mathrm{clean}(S_G(t))$

for every $t \in T_\Sigma$, by Lemma 6. Since 'clean' is definable by a direction preserving MSO graph transducer, as shown in Example 1(1, clean), we can replace T by the MSO tree transducer T' such that $\mathcal{T}' = \mathcal{T}'_{\mathrm{gr}} = \mathcal{T}_{\mathrm{gr}} \circ$ clean which can be found by Proposition 1 (MSO-GT is closed under composition). Then $\mathcal{G}(t) = \mathcal{T}'(t)$ for every $t \in T_\Sigma$. If, moreover, G is OS, then T' is direction preserving by Proposition 2 (MSO-$\mathrm{GT}_{\mathrm{dir}}$ is closed under composition). □

It now remains to be shown that for every term graph transducer T from Σ to Δ^{id} there is a term graph transducer T' from Σ to Δ such that $\mathcal{T}'(t) = \mathrm{prune}(\mathcal{T}(t))$ for every $t \in T_\Sigma$. Recall from Section 4.2 that the mapping 'prune' removes all occurrences of 'id' from a tree. Thus, by Proposition 1, it suffices to construct an MSO graph transducer which defines a mapping that extends 'prune' to term graphs, in such a way that $\mathrm{unfold}(\mathrm{prune}(\tau)) = \mathrm{prune}(\mathrm{unfold}(\tau))$ for every term graph τ.

We first define the extension of 'prune' to term graphs. It is convenient to define it for clean term graphs only. Let t be a clean term graph over Δ^{id}. Then the (clean) term graph $\mathrm{prune}(t)$ over Δ is defined as follows. For a node u of t, let the "forward node" of u, denoted $\mathrm{fw}(u)$, be the first descendant of u that does not have label id. Formally, if $\mathrm{lab}_t(u) \in \Delta$ then $\mathrm{fw}(u) = u$, and if $\mathrm{lab}_t(u) = \mathrm{id}$ then $\mathrm{fw}(u) = \mathrm{fw}(u \cdot 1)$. Now we define $V_{\mathrm{prune}(t)} = \{\mathrm{fw}(u) \mid u \in V_t\} = \{u \in V_t \mid \mathrm{lab}_t(u) \in \Delta\}$, $E_{\mathrm{prune}(t)} = \{(u, i, \mathrm{fw}(v)) \mid u \in V_{\mathrm{prune}(t)}, (u, i, v) \in E_t\}$, and $\mathrm{lab}_{\mathrm{prune}(t)}(u) = \mathrm{lab}_t(u)$ for every $u \in V_{\mathrm{prune}(t)}$. Obviously this defines the old $\mathrm{prune}(t)$ for trees over Δ^{id}.

It should be clear that $\mathrm{root}(\mathrm{prune}(t)) = \mathrm{fw}(\mathrm{root}(t))$ for every clean term graph t; this is because, in t, every node u with $\mathrm{lab}_t(u) \in \Delta$ is a descendant of $\mathrm{fw}(\mathrm{root}(t))$. It should also be clear that $\mathrm{unfold}(\mathrm{prune}(t)) = \mathrm{prune}(\mathrm{unfold}(t))$. In fact, it is straightforward to show from the definitions that $\mathrm{unf}_{\mathrm{prune}(t)}(\mathrm{fw}(u)) = \mathrm{prune}(\mathrm{unf}_t(u))$ for every $u \in V_t$, by induction on u. The result then follows by taking $u = \mathrm{root}(t)$ and using the fact that $\mathrm{fw}(\mathrm{root}(t)) = \mathrm{root}(\mathrm{prune}(t))$.

The next lemma shows that the mapping 'prune' is an MSO definable graph transduction.

Lemma 8. *Let Δ be an operator alphabet. There is a direction preserving* MSO *graph transducer T such that $\mathcal{T}_{\mathrm{gr}}(t) = \mathrm{prune}(t)$ for every clean term graph t over Δ^{id}.*

Proof. It is not difficult to find a formula $\phi(x, y)$ in $\mathrm{MSOL}_2(\Delta^{\mathrm{id}})$ such that, for every term graph t and nodes $u, v \in V_t$, $(t, u, v) \models \phi(x, y)$ iff v is the forward node of u, i.e., $\mathrm{fw}(u) = v$. In fact, $\phi(x, y) = \phi'(x, y) \wedge \neg\, \mathrm{lab}_{\mathrm{id}}(y)$, where $\phi'(x, y)$ is obtained from the formula for $\mathrm{path}(x, y)$ in Section 2.2, by changing $\mathrm{edg}(x, y)$ into $\mathrm{lab}_{\mathrm{id}}(x) \wedge \mathrm{edg}_1(x, y)$.

The required MSO graph transducer T has copy number 1, node formulas $\psi_\delta(x) = \mathrm{lab}_\delta(x)$ for every $\delta \in \Delta$, and edge formulas $\chi_i(x, y) = \exists z(\mathrm{edg}_i(x, z) \wedge \phi(z, y))$ for every $i \in \mathrm{rki}(\Delta)$. □

We now prove the simulation of attributed tree transducers by MSO term graph transducers.

Theorem 9. *For every noncircular attributed tree transducer G there is an* MSO *term graph transducer T such that $\mathcal{T} = \mathcal{G}$. Moreover, if G is* SUR*, then T is an* MSO *tree transducer, and if G is* OS*, then T is direction preserving.*

Proof. Let G be a noncircular att from Σ to Δ. By Lemma 5 there is a noncircular att G' in operator form from Σ to Δ^{id} such that $\mathcal{G} = \mathcal{G}' \circ \mathrm{prune}$. Moreover, if G is SUR or OS, then so is G'. By Lemma 7 there is an MSO term graph transducer T' such that $\mathcal{T}' = \mathcal{G}'$. Moreover, if G' is SUR, then T' is an MSO tree transducer, and if G' is OS, then T' is direction preserving.

We may assume that, for every $t \in T_\Sigma$, $\mathcal{T}'_{\mathrm{gr}}(t)$ is clean. In fact, if this is not the case, then, since $\mathrm{unfold}(\mathrm{clean}(\tau)) = \mathrm{unfold}(\tau)$ for every term graph τ, we can replace T' by an MSO graph transducer that defines $\mathcal{T}'_{\mathrm{gr}} \circ \mathrm{clean}$ which can be found by Propositions 1 and 2, and the fact that cleaning is definable by a direction preserving MSO graph transducer, as shown in Example 1(1, clean).

Let T be the MSO term graph transducer with $\mathcal{T}_{\mathrm{gr}} = \mathcal{T}'_{\mathrm{gr}} \circ \mathrm{prune}$ which can be found by Proposition 1 and Lemma 8. Since $\mathrm{unfold}(\mathrm{prune}(\tau)) = \mathrm{prune}(\mathrm{unfold}(\tau))$ for every clean term graph τ (cf. the discussion before Lemma 8), we obtain, for $t \in T_\Sigma$, that

$$\mathcal{T}(t) = \mathrm{unfold}(\mathcal{T}_{\mathrm{gr}}(t)) = \mathrm{unfold}(\mathrm{prune}(\mathcal{T}'_{\mathrm{gr}}(t))) = \mathrm{prune}(\mathrm{unfold}(\mathcal{T}'_{\mathrm{gr}}(t))) = \mathrm{prune}(\mathcal{T}'(t)) = \mathrm{prune}(\mathcal{G}'(t)) = \mathcal{G}(t).$$

Note that if T' is an MSO tree transducer then so is T (because prune transforms trees into trees), and if T' is direction preserving then so is T (by Proposition 2 and Lemma 8). □

6 Characterizing unary and binary MSO formulas

To show that MSO definable term transductions can be computed by attribute grammars, we will use the characterizations proved in [BloEng] of MSO formulas $\psi(x) \in \mathrm{MSOL}_1(\Sigma)$ and $\chi(x, y) \in \mathrm{MSOL}_2(\Sigma)$, for trees in T_Σ. These characterizations are recalled in this section. The characterization of unary formulas is already in terms of attribute grammars, and we use this characterization to show that the class MSO-REL of MSO relabelings (defined in Section 3 as a "special case") is equal to the class ATT-REL of attributed relabelings (defined in Section 4.1). The characterization of binary formulas is in terms of tree-walking automata that employ unary formulas as tests.

6.1 Unary MSO formulas

Let $\psi(x)$ be a formula in $\mathrm{MSOL}_1(\Sigma)$. It is proved in Theorem 18 of [BloEng] (and independently in [NevBus]) that there exists a finite-valued noncircular attribute grammar $G = (\Sigma, S, I, \Omega, W, R, \alpha_{\mathrm{m}})$ with $W(\alpha_{\mathrm{m}}) = \{\mathrm{true}, \mathrm{false}\}$ such that

$$\text{for every } t \in T_\Sigma \text{ and } u \in V_t,\ \mathrm{dec}_{G,t}(\langle \alpha_{\mathrm{m}}, u \rangle) = \mathrm{true} \text{ iff } (t, u) \models \psi(x).$$

This also holds the other way around: for every such attribute grammar there is a formula $\psi(x) \in \mathrm{MSOL}_1(\Sigma)$ satisfying the above condition. We now use this characterization to show that the class MSO-REL of MSO relabelings is equal to the class ATT-REL of attributed relabelings. This implies that the look-ahead stage of an $\mathrm{att}^{\mathrm{R}}$ can as well be realized by an MSO relabeling.

Theorem 10. MSO-REL = ATT-REL.

Proof. We first show MSO-REL $\subseteq$ ATT-REL. Let T be an MSO relabeling from Σ to Δ. By the above mentioned characterization of unary formulas, there exists for every node formula $\psi_\delta(x)$ of T a finite-valued noncircular attribute grammar G_δ over Σ such that $\mathrm{dec}_{G_\delta,t}(\langle \alpha_{\mathrm{m},\delta}, u\rangle) = \mathrm{true}$ iff $(t,u) \models \psi_\delta(x)$, where $\alpha_{\mathrm{m},\delta}$ is the meaning attribute of G_δ. Let G be the relabeling attribute grammar obtained by taking the union of all G_δ, in the obvious way, and adding a new meaning attribute α_{m} with $W(\alpha_{\mathrm{m}}) = \Delta$ and the semantic rule

$$\langle \alpha_{\mathrm{m}}, 0\rangle = \text{ the unique } \delta \in \Delta \text{ such that } \langle \alpha_{\mathrm{m},\delta}, 0\rangle = \mathrm{true}$$

in every $R(\sigma)$. It should be clear that $r(G) = \mathcal{T}$. In fact, for every $t \in T_\Sigma$, $u \in V_t$, and $\delta \in \Delta$, $\mathrm{lab}_{r(G)(t)}(u) = \delta$ iff $\mathrm{dec}_{G,t}(\langle \alpha_{\mathrm{m}}, u\rangle) = \delta$ iff $\mathrm{dec}_{G_\delta,t}(\langle \alpha_{\mathrm{m},\delta}, u\rangle) = \mathrm{true}$ iff $(t,u) \models \psi_\delta(x)$ iff $\mathrm{lab}_{\mathcal{T}(t)}(u) = \delta$. This also shows that G is indeed a relabeling attribute grammar, because $\mathrm{dec}_{G,t}(\langle \alpha_{\mathrm{m}}, u\rangle)$ has the same rank as $\mathrm{lab}_{\mathcal{T}(t)}(u)$, which has the same rank as $\mathrm{lab}_t(u)$.

Next we show ATT-REL $\subseteq$ MSO-REL. Let G be a relabeling attribute grammar from Σ to Δ. We now claim (as in the first part of the proof of Theorem 19 of [BloEng]) that for every $\delta \in \Delta$ there is a formula $\psi_\delta(x)$ in $\mathrm{MSOL}_1(\Sigma)$ such that, for every $t \in T_\Sigma$ and $u \in V_t$, $(t,u) \models \psi_\delta(x)$ iff $\mathrm{lab}_{r(G)(t)}(u) = \delta$. In fact, let G_δ be the attribute grammar that is obtained from G by adding a new meaning attribute $\alpha_{\mathrm{m},\delta}$ with $W(\alpha_{\mathrm{m},\delta}) = \{\mathrm{true}, \mathrm{false}\}$ which has, for every $\sigma \in \Sigma$, the internal rule $\langle \alpha_{\mathrm{m},\delta}, 0\rangle = (\langle \alpha_{\mathrm{m}}, 0\rangle = \delta)$. By the characterization of unary formulas mentioned above, there is a formula $\psi_\delta(x)$ such that $(t,u) \models \psi_\delta(x)$ iff $\mathrm{dec}_{G_\delta,t}(\langle \alpha_{\mathrm{m},\delta}, u\rangle) = \mathrm{true}$ iff $\mathrm{dec}_{G,t}(\langle \alpha_{\mathrm{m}}, u\rangle) = \delta$ iff $\mathrm{lab}_{r(G)(t)}(u) = \delta$. This proves the claim. Clearly, the formulas $\psi_\delta(x)$, $\delta \in \Delta$, uniquely determine an MSO relabeling T such that $\mathrm{lab}_{\mathcal{T}(t)}(u) = \mathrm{lab}_{r(G)(t)}(u)$, i.e., $\mathcal{T} = r(G)$. □

6.2 Binary MSO formulas

It is shown in [BloEng] that a formula $\chi(x,y) \in \mathrm{MSOL}_2(\Sigma)$ can be computed by a tree-walking automaton. A tree-walking automaton (see, e.g., [AhoUll, EngRozSlu, KamSlu]) is a finite-state automaton that walks on a tree from node to node, following the edges of the tree (downwards or upwards). Here, and in [BloEng], we allow the tree-walking automaton to test any MSO definable property of the current node, using formulas from $\mathrm{MSOL}_1(\Sigma)$. Let A be a tree-walking automaton corresponding to $\chi(x,y)$. Then, for every tree $t \in T_\Sigma$ and nodes $u, v \in V_t$, $(t,u,v) \models \chi(x,y)$ if and only if A can walk from u to v in t, starting in an initial state and ending in a final state.

We recall the formal definition of the tree-walking automata of [BloEng]. The instructions used by tree-walking automata are called directives (as in [KlaSch]). Let Σ be an operator alphabet. The (infinite) set of *directives* over Σ is

$$D_\Sigma = \{\downarrow_i \mid i \in \mathrm{rki}(\Sigma)\} \cup \{\uparrow_i \mid i \in \mathrm{rki}(\Sigma)\} \cup \mathrm{MSOL}_1(\Sigma).$$

A directive is an instruction of how to move from one node to another: $\downarrow_i$ means "move along an edge labeled i", i.e., "move to the i-th child of the current node (provided it has one)"; $\uparrow_i$ means "move against an edge labeled i", i.e., "move to the parent of the current node (provided it has one, and it is the i-th child of its parent)"; and $\psi(x)$ means "check that ψ holds for the current node (and do not move)". Formally, we define for each $t \in T_\Sigma$ and each directive $d \in D_\Sigma$ the node relation $R_t(d) \subseteq V_t \times V_t$, as follows:

$$\begin{aligned} R_t(\downarrow_i) &= \{(u,v) \mid (u,i,v) \in E_t\}, \\ R_t(\uparrow_i) &= \{(u,v) \mid (v,i,u) \in E_t\}, \text{ and} \\ R_t(\psi(x)) &= \{(u,u) \mid (t,u) \models \psi(x)\}. \end{aligned}$$

Syntactically, a tree-walking automaton is just an ordinary finite automaton (on strings) with a finite subset of D_Σ as input alphabet. However, the symbols of D_Σ are interpreted as instructions on the input tree as explained above.

Let Σ be an operator alphabet. A *tree-walking automaton* (*with* MSO *tests*) over Σ is a quintuple $A = (Q, \Delta, \delta, I, F)$, where

- Q is a finite set of *states*,
- Δ is the *instruction set*, a finite subset of D_Σ,
- $\delta \subseteq Q \times \Delta \times Q$ is the *transition relation*, the elements of which are called *transitions*,
- $I \subseteq Q$ is the set of *initial states*, and
- $F \subseteq Q$ is the set of *final states*.

For a tree-walking automaton $A = (Q, \Delta, \delta, I, F)$ and a tree t, an element (q,u) of $Q \times V_t$ is a *configuration* of the automaton. Intuitively, it denotes that A is in state q at node u. One step of A on t is defined by the binary relation $\twoheadrightarrow_{A,t}$ on the set of configurations, as follows. For every $q, q' \in Q$ and $u, u' \in V_t$,

$$(q,u) \twoheadrightarrow_{A,t} (q',u') \text{ iff } \exists d \in \Delta : (q,d,q') \in \delta \text{ and } (u,u') \in R_t(d).$$

To indicate the directive that is executed by A in this step, we also write $(q,u) \overset{d}{\twoheadrightarrow}_{A,t} (q',u')$.

For each tree $t \in T_\Sigma$, A computes the node relation $R_t(A) = \{(u,v) \in V_t \times V_t \mid (q,u) \twoheadrightarrow^*_{A,t} (q',v)$ for some $q \in I$ and $q' \in F\}$. Thus, A computes all pairs of nodes (u,v) such that A can walk from u to v in t, starting in an initial state and ending in a final state.

Example 4. Let $\Sigma = \Sigma_0 \cup \Sigma_2$, and consider the edge formula $\chi_1(x,y)$ of the MSO tree transducer of Example 1(6, yield). For nodes x and y of a tree $t \in T_\Sigma$, it checks that x is a leaf and y is the next leaf after x, in the left-to-right order of the

leaves of t. The following tree-walking automaton $A = (Q, \Delta, \delta, I, F)$ computes $R_t(A) = \{(u, v) \mid (t, u, v) \models \chi_1(x, y)\}$, for every $t \in T_\Sigma$, i.e., it walks from one leaf to the next. The transition graph of A is depicted in Fig. 6, in the way usual for finite automata. Thus, A has states $Q = \{\text{I}, \text{II}, \text{III}, \text{IV}, \text{V}\}$, initial state $I = \{\text{I}\}$,

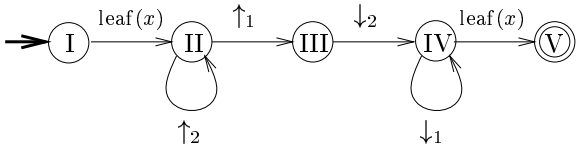


Fig. 6. The tree-walking automaton for $\chi_1(x, y)$

and final state $F = \{\text{V}\}$. The instruction set is $\Delta = \{\downarrow_1, \downarrow_2, \uparrow_1, \uparrow_2, \text{leaf}(x)\}$, and the transition relation consists of the transitions $(\text{I}, \text{leaf}(x), \text{II})$, $(\text{II}, \uparrow_2, \text{II})$, $(\text{II}, \uparrow_1, \text{III})$, $(\text{III}, \downarrow_2, \text{IV})$, $(\text{IV}, \downarrow_1, \text{IV})$, and $(\text{IV}, \text{leaf}(x), \text{V})$.

On a given tree, the automaton A first checks that it is at a leaf. If so, it walks to the next leaf along the shortest undirected path: it walks upwards over 2-labeled edges as far as possible (in state II), moves to the other child of its parent (through state III), and walks downwards along 1-labeled edges as far as possible (in state IV). □

The characterization of binary MSO formulas by tree-walking automata is stated next; it is Theorem 13 of [BloEng]. For a formula $\chi(x, y)$ in $\text{MSOL}_2(\Sigma)$ and a tree $t \in T_\Sigma$, we denote by $R_t(\chi)$ the node relation $\{(u, v) \in V_t \times V_t \mid (t, u, v) \models \chi(x, y)\}$. The characterization says that binary MSO formulas and tree-walking automata compute the same node relations. We also need a special case of this. We will say that a formula $\chi(x, y)$ in $\text{MSOL}_2(\Sigma)$ is *direction preserving* if $(t, u, v) \models \chi(x, y)$ implies that v is a descendant of u, for every $t \in T_\Sigma$ and $u, v \in V_t$. Moreover, we will say that a tree-walking automaton is *descending* if it does not have directives $\uparrow_i$ in its instruction set.

Proposition 11. *For every formula $\chi(x, y) \in \text{MSOL}_2(\Sigma)$ there is a tree-walking automaton A over Σ such that, for all $t \in T_\Sigma$, $R_t(A) = R_t(\chi)$. The other way around, for every tree-walking automaton A over Σ there is a formula $\chi(x, y) \in \text{MSOL}_2(\Sigma)$ such that, for all $t \in T_\Sigma$, $R_t(\chi) = R_t(A)$. The same two statements hold for direction preserving formulas and descending automata.*

The special case of this characterization was not explicitly stated in [BloEng]. From automata to formulas it is obvious because a descending automaton always walks from a node to one of its descendants. From formulas to automata, it was shown in the proof of Theorem 13 of [BloEng] that, in general, the automaton can always compute $(t, u, v) \models \chi(x, y)$ by walking from u to v along the shortest undirected path in t. Thus, if χ is direction preserving, the automaton walks downwards only.

Determinism The formulas $\chi_i(x,y)$ used in an MSO term graph (or tree) transducer are all *functional*, i.e., the node relation $R_t(\chi_i)$ is functional for every input tree t. This follows, of course, from the fact that every node u of a term graph has a unique i-th child $u \cdot i$. It is shown in [BloEng] that such relations can be computed by *deterministic* tree-walking automata. This will be essential in the proof of the simulation of MSO term graph transducers by attribute grammars, in the next section.

A tree-walking automaton $A = (Q, \Delta, \delta, I, F)$ over Σ is *deterministic* if the following three conditions hold: (1) I is a singleton, (2) if $(q, d, q') \in \delta$, then $q \notin F$, and (3) for every tree $t \in T_\Sigma$ and every configuration (q,u), there is at most one configuration (q',u') such that $(q,u) \twoheadrightarrow_{A,t} (q',u')$.

Note that the automaton of Example 4 is deterministic. In [BloEng] a stronger definition of determinism is given (which is not satisfied by the automaton of Example 4). The above definition suffices for our present purposes.

A tree-walking automaton A is *functional* if the node relation $R_t(A)$ is functional for every input tree t. Obviously, every deterministic tree-walking automaton is functional. The next result is Theorem 14 of [BloEng].

Proposition 12. *For every functional tree-walking automaton A over Σ there is a deterministic tree-walking automaton A' over Σ with $R_t(A') = R_t(A)$ for every $t \in T_\Sigma$. Moreover, if A is descending then so is A'.*

As for Proposition 11, the second statement was not explicitly stated in Theorem 14 of [BloEng], but is obvious from its proof.

7 From MSO to ATT

In this section we prove the more difficult part of our main result, viz. that every MSO term graph transducer can be simulated by an attributed tree transducer with look-ahead. The constructed $\mathrm{att}^{\mathrm{R}}$ is, however, not necessarily noncircular (though it is, of course, noncircular on every output tree of its relabeling attribute grammar). Moreover, for an MSO tree transducer, the $\mathrm{att}^{\mathrm{R}}$ is not necessarily SUR though it is SUR on every output tree of its relabeling attribute grammar. These problems are taken care of in the next section.

One of the essential differences between attributed tree transducers and MSO term graph transducers is that the former operate locally, i.e., attributes of a node can only be computed in terms of attributes of its neighbors (viz. its children, its parent, or itself), whereas the latter operate "globally", i.e., edges of the output graph can be established between (copies of) nodes of the input tree that are not necessarily neighbors, but may be far apart. On the other hand, attributed tree transducers have copy rules by which they can transport attribute values through the tree.

Thus, to facilitate the simulation of MSO term graph transducers by attributed tree transducers, we first prove that every MSO term graph transducer can, in a certain sense, be simulated by one that is "local", i.e., establishes edges between (copies of) neighbors only. Moreover, we want to forbid term graphs

with multiple edges; this is related to the requirement in att's that the right-hand side of a semantic rule should be linear. Finally, we require that the root of the output term graph is a copy of the root of the input tree.

An MSO term graph transducer $T = (C, \Psi, X)$ from Σ to Δ is *local* if the following three conditions hold:

- For every edge formula $\chi_{j,c,c'}(x,y)$, every $t \in T_\Sigma$, and every $u, v \in V_t$, if $(t,u,v) \models \chi_{j,c,c'}(x,y)$ then $v = u$ or v is a child of u or v is the parent of u.
- For every $t \in T_\Sigma$, $\mathcal{T}_{\mathrm{gr}}(t)$ has no multiple edges, i.e., no edges $((c,u), j, (c',v))$ and $((c,u), i, (c',v))$ with $j \neq i$.
- There exists $c_{\mathrm{m}} \in C$ such that for every $t \in T_\Sigma$, $\mathrm{root}(\mathcal{T}_{\mathrm{gr}}(t)) = (c_{\mathrm{m}}, \mathrm{root}(t))$.

To alleviate the locality restriction, we will allow a local MSO term graph transducer to use the identity operator 'id'. This corresponds to the use of copy rules in attributed tree transducers, cf. Section 4.2 where the identity operator was used to put att's into operator form. The formulation of the next lemma is similar to that of Lemma 5.

Lemma 13. *For every* MSO *term graph transducer T from Σ to Δ there is a local* MSO *term graph transducer T' from Σ to Δ^{id} such that $\mathcal{T} = \mathcal{T}' \circ \mathrm{prune}$. If T is an* MSO *tree transducer then so is T', and if T is direction preserving then so is T'.*

Proof. Let $T = (C, \Psi, X)$ be an MSO term graph transducer from Σ to Δ. We may assume, by Propositions 1 and 2 and Example 1(1, clean), that T produces clean term graphs only, i.e., $\mathcal{T}_{\mathrm{gr}}(t)$ is clean for every $t \in T_\Sigma$.

We first discuss the intuitive ideas behind the construction of T'. Let $t \in T_\Sigma$, $\tau = \mathcal{T}_{\mathrm{gr}}(t)$, and $\tau' = \mathcal{T}'_{\mathrm{gr}}(t)$. The (clean) term graph τ' will have the same nodes $(c,u) \in C \times V_t$ as τ, but it will have additional id-labeled nodes, in such a way that $\mathrm{prune}(\tau') = \tau$ (see Section 5 for the definition of 'prune'). Then $\mathcal{T}(t) = \mathrm{unfold}(\tau) = \mathrm{unfold}(\mathrm{prune}(\tau')) = \mathrm{prune}(\mathrm{unfold}(\tau')) = \mathrm{prune}(\mathcal{T}'(t))$, as required. The new, id-labeled, nodes of τ' and the edges of τ' are constructed as follows. Consider an edge $(c,u) \xrightarrow{j} (c',v)$ in τ. The difficult point for T' is that the nodes u and v may be far apart in t. Thus, in τ' the edge will be divided into small pieces, by introducing id-labeled nodes that are copies of nodes of t on an undirected path from u to v, see Fig. 7. In particular, the edge is replaced by a path in τ' of the form

$$(c,u) \xrightarrow{j} (c_1,u_1) \xrightarrow{1} \cdots \xrightarrow{1} (c_n,u_n) \xrightarrow{1} (c',v)$$

where $(c_1,u_1), \ldots, (c_n,u_n)$ are id-labeled nodes, $n \geq 1$, $c_1, \ldots, c_n$ are new copy names, and $u_1, \ldots, u_n$ are the nodes on an undirected path in t from u to v. More precisely, $u_1 = u$, $u_n = v$, and $u_{i+1} = u_i$ or u_{i+1} is a child of u_i or u_{i+1} is the parent of u_i, for every $i \in [1, n-1]$. If we view (c,u) and (c',v) as having "value" $\mathrm{unf}_\tau(c,u)$ and $\mathrm{unf}_\tau(c',v)$, respectively, then the value of (c',v) is transported backwards along the above path, from v to u, in order to be used in the value of (c,u). The problem for T' is that, to transport this value through the tree, it has

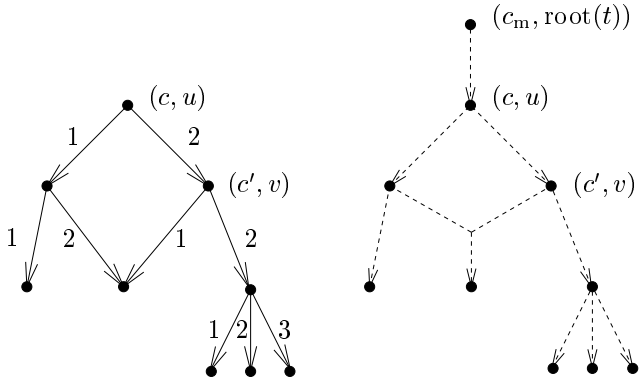


Fig. 7. To the left: $\tau = \mathcal{T}_{\mathrm{gr}}(t)$, to the right: $\tau' = \mathcal{T}'_{\mathrm{gr}}(t)$, where dashed lines symbolize paths in τ' corresponding to edges of τ, plus the root path

to know the proper route from u to v. Now, since $(c, u) \xrightarrow{j} (c', v)$ in τ, the edge formula $\chi_{j,c,c'}(x, y)$ of T is satisfied by (t, u, v). Hence, by the characterization of binary MSO formulas in Proposition 11, there is a tree-walking automaton $A = A_{j,c,c'}$ that knows how to walk from u to v. The formula $\chi_{j,c,c'}(x, y)$ is functional, because every node of τ has a unique j-th child, and so we may assume that the automaton is deterministic (see Proposition 12). Hence A has a unique walk $(q, u) \twoheadrightarrow^*_{A,t} (q', v)$, starting at u in its initial state q and ending at v in one of its final states q'. If, in detail, the walk is

$$(q_1, u_1) \twoheadrightarrow_{A,t} (q_2, u_2) \twoheadrightarrow_{A,t} \cdots \twoheadrightarrow_{A,t} (q_n, u_n) \tag{1}$$

with $(q_1, u_1) = (q, u)$ and $(q_n, u_n) = (q', v)$, then the nodes $u_1, \ldots, u_n$ are the ones that will be used in the above path in τ'. As the new copy names $c_1, \ldots, c_n$ we will use $((j, c, c'), q_1), \ldots, ((j, c, c'), q_n)$ which uniquely identify the states $q_1, \ldots, q_n$ of the automaton $A = A_{j,c,c'}$. Thus, the above path of τ' now has the following form, where $e = (j, c, c')$, $q_e = q_1 = q$ is the initial state of A, $(q_i, u_i) \twoheadrightarrow_{A,t} (q_{i+1}, u_{i+1})$ for every $i \in [1, n-1]$, and q_n is a final state of A:

$$\begin{aligned}(c, u) \xrightarrow{j} ((e, q_e), u) \xrightarrow{1} ((e, q_2), u_2) \xrightarrow{1} \cdots \\ \cdots \xrightarrow{1} ((e, q_{n-1}), u_{n-1}) \xrightarrow{1} ((e, q_n), v) \xrightarrow{1} (c', v).\end{aligned} \tag{2}$$

It should be clear that in this way the first locality requirement for T' is satisfied. The second is also satisfied because if $j \neq i$ then $e = (j, c, c')$ differs from $e' = (i, c, c'')$, and hence $((e, q_e), u) \neq ((e', q_{e'}), u)$. Finally, to satisfy the third locality requirement, viz. that $\mathrm{root}(\tau') = (c_{\mathrm{m}}, \mathrm{root}(t))$, we introduce a new copy name c_{m} and we add to τ' the "root path"

$$(c_{\mathrm{m}}, u_1) \xrightarrow{1} \cdots \xrightarrow{1} (c_{\mathrm{m}}, u_n) \xrightarrow{1} (c, u) \tag{3}$$

where $u_1 = \mathrm{root}(t)$, $u_n = u$, $(c, u) = \mathrm{root}(\tau)$, $(c_\mathrm{m}, u_1), \ldots, (c_\mathrm{m}, u_n)$ are id-labeled nodes, with $n \geq 1$, and $u_1, \ldots, u_n$ are the nodes on the path in t from $\mathrm{root}(t)$ to u, see Fig. 7. In this way, the "value" $\mathrm{unf}_\tau(c, u) = \mathrm{unfold}(\tau)$ is transported from u to $\mathrm{root}(t)$.

We now turn to the formal definitions. In what follows we will denote elements (j, c, c') of $\mathrm{rki}(\Delta) \times C \times C$ by e. We will also denote $\mathrm{rki}(\Delta) \times C \times C$ by E. Thus, T has an edge formula $\chi_e(x, y)$ for every $e \in E$. For every $e \in E$, let $A_e = (Q_e, \Delta_e, \delta_e, \{q_e\}, F_e)$ be a deterministic tree-walking automaton such that for all $t \in T_\Sigma$ and $u, v \in V_t$, $(u, v) \in R_t(A)$ iff $(t, u, v) \models \chi_e(x, y)$, according to Propositions 11 and 12 (recall that $\chi_e(x, y)$ is functional). We will write $\twoheadrightarrow_{e,t}$ instead of $\twoheadrightarrow_{A_e,t}$.

We will need an MSO formula $\mathrm{root}_c(x)$, with $c \in C$, such that, for $t \in T_\Sigma$ and $u \in V_t$, $(t, u) \models \mathrm{root}_c(x)$ iff (c, u) is the root of $\tau = \mathcal{T}_\mathrm{gr}(t)$. Since we have assumed that τ is clean, it should be clear that the root of τ is the unique node without incoming edges, and so we can take $\mathrm{root}_c(x)$ to be the conjunction of all formulas $\neg\exists y(\chi_{j,c',c}(y, x))$, for $j \in \mathrm{rki}(\Delta)$ and $c' \in C$.

The local MSO term graph transducer $T' = (C', \Psi', X')$ from Σ to Δ^id is now defined. The set of copy names of T' is $C' = C \cup \{(e, q) \mid e \in E, q \in Q_e\} \cup \{c_\mathrm{m}\}$. The node formulas of T' are defined as follows. All node formulas that are not mentioned, are defined to be false. We also give some informal comments, related to the informal explanations above.

- For all $\delta \in \Delta$ and $c \in C$, $\psi'_{\delta,c}(x) = \psi_{\delta,c}(x)$.
  This means that all nodes of τ are also nodes of τ', with the same labels.
- For all $e \in E$ and $q \in Q_e$, $\psi'_{\mathrm{id},(e,q)}(x)$ is defined in such a way that, for all $t \in T_\Sigma$ and $w \in V_t$, $(t, w) \models \psi'_{\mathrm{id},(e,q)}(x)$ iff there are nodes $u, v \in V_t$ such that $(q_e, u) \twoheadrightarrow^*_{e,t} (q, w) \twoheadrightarrow^*_{e,t} (q', v)$ for some $q' \in F_e$. By Proposition 11 such a formula exists. In fact, applying Proposition 11 to the tree-walking automaton $(Q_e, \Delta_e, \delta_e, \{q_1\}, \{q_2\})$, which is A_e with start state q_1 and final state q_2, we obtain, for any two states q_1, q_2 of A_e, a formula $\phi_{q_1,q_2}(x, y)$ which expresses that $(q_1, x) \twoheadrightarrow^*_e (q_2, y)$. Then

  $$\psi'_{\mathrm{id},(e,q)}(x) = \exists y, z(\phi_{q_e,q}(y, x) \wedge \phi_{q,F_e}(x, z)),$$

  where $\phi_{q,F_e}(x, z)$ is the disjunction of all $\phi_{q,q'}(x, z)$, for $q' \in F_e$.
  Through this node formula, we only add an id-labeled node $((e, q), w)$ to τ' when it is needed on a path (2) of τ' that replaces an edge of τ, as discussed above, i.e., when the configuration (q, w) is on the walk (1) of A_e that determines that path.
- $\psi'_{\mathrm{id},c_\mathrm{m}}(x)$ is the disjunction of all formulas $\exists y(\mathrm{path}(x, y) \wedge \mathrm{root}_c(y))$, for $c \in C$.
  Thus, an id-labeled node (c_m, w) is only added to τ' when it is needed on the root path (3).

The edge formulas of T' are now defined as follows. Again, edge formulas that are not mentioned, are defined to be false.

– For every $e = (j, c, c') \in E$, $\chi'_{j,c,(e,q_e)}(x, y) = (x = y)$.
  This edge formula establishes the first edge of a path (2) of τ' that replaces an edge of τ.
– For every $e = (j, c, c') \in E$ and $q \in F_e$, $\chi'_{1,(e,q),c'}(x, y) = (x = y)$.
  This edge formula establishes the last edge of a path (2) of τ'.
– For every $e \in E$ and $q, q' \in Q_e$,

$$\chi'_{1,(e,q),(e,q')}(x, y) = \tau_{d_1}(x, y) \vee \cdots \vee \tau_{d_n}(x, y)$$

  where $\{d_1, \ldots, d_n\} = \{d \in \Delta_e \mid (q, d, q') \in \delta_e\}$ and, for $d \in D_\Sigma$, the formula $\tau_d(x, y)$ is defined as follows:
  - if $d = \downarrow_i$ then $\tau_d(x, y) = \mathrm{edg}_i(x, y)$,
  - if $d = \uparrow_i$ then $\tau_d(x, y) = \mathrm{edg}_i(y, x)$, and
  - if $d = \phi(x)$ then $\tau_d(x, y) = (\phi(x) \wedge x = y)$.

  These edge formulas establish all other edges of a path (2) of τ'. Each such edge corresponds to one step of the automaton A_e in the walk (1). Note that, for every $t \in T_\Sigma$, $R_t(\tau_d) = R_t(d)$.
– $\chi'_{1,c_\mathrm{m},c_\mathrm{m}}(x, y) = \mathrm{edg}(x, y)$, and for every $c \in C$,
  $\chi'_{1,c_\mathrm{m},c}(x, y) = (\mathrm{root}_c(x) \wedge x = y)$.
  The second formula establishes the last edge on the root path (3) of τ', and the first formula establishes all other edges on that path.

This ends the definition of T'.

We will now show the correctness of T'. To this aim, let again $t \in T_\Sigma$, $\tau = \mathcal{T}_\mathrm{gr}(t)$, and $\tau' = \mathcal{T}'_\mathrm{gr}(t)$. We have to prove that T' is local, that τ' is a clean term graph such that $\mathrm{prune}(\tau') = \tau$, that τ' is a tree if τ is one, and that T' is direction preserving if T is.

We first make four easy observations. First, it is immediate from the definitions that the first two requirements for locality hold for T'. Second, if T is direction preserving, then every edge formula $\chi_e(x, y)$ of T is direction preserving, and so, by Propositions 11 and 12, A_e is descending. This means that A_e has no transitions $(q, \uparrow_i, q')$, which implies that all edge formulas of T' are direction preserving, i.e., that T' is direction preserving. Third, it is straightforward to see from the definition of $\twoheadrightarrow_{A,t}$ that, for every $t \in T_\Sigma$ and $u, u' \in V_t$, $(t, u, u') \models \chi'_{1,(e,q),(e,q')}(x, y)$ iff $(q, u) \twoheadrightarrow_{e,t} (q', u')$. Fourth, since all automata A_e are deterministic, it is easy to see from the definitions (and the previous observation) that each id-labeled node of τ' has at most one outgoing edge, with label 1.

To prove the remaining facts, we show that τ' has the special form that was suggested in the intuitive introduction of this proof, see Fig. 7. By the definition of $\psi'_{\delta,c}(x)$, the nodes of τ' of the form (c, u) are exactly the nodes of τ, with the same labels. All other nodes of τ' have label id. To describe the id-labeled nodes and the edges of τ', we associate with each edge $(c, u) \xrightarrow{j} (c', v)$ of τ a particular path in τ' as follows. Let $e = (j, c, c')$ and consider the walk (1) of the tree-walking automaton $A = A_e$ from $u_1 = u$ to $u_n = v$, with $q_1 = q_e$ and $q_n \in F_e$. Such a walk exists because $(t, u, v) \models \chi_e(x, y)$, and it is unique because A_e is

deterministic. Then we associate with $(c,u) \xrightarrow{j} (c',v)$ the path (2) of τ'. It is easy to check from the definition of T' that this path indeed exists in τ'. In fact, each id-labeled node $((e,q_i),u_i)$ exists because the walk $(q_e,u) \twoheadrightarrow^*_{A,t} (q_i,u_i) \twoheadrightarrow^*_{A,t} (q_n,v)$ implies that $(t,u_i) \models \psi'_{\mathrm{id},(e,q_i)}(x)$. The edges of the path are clearly established by the edge formulas of T'; in particular, $(q_i,u_i) \twoheadrightarrow_{A,t} (q_{i+1},u_{i+1})$ implies $(t,u_i,u_{i+1}) \models \chi'_{1,(e,q_i),(e,q_{i+1})}(x,y)$. Having associated a path of τ' with every edge of τ, we define one additional path of τ', the "root path". It is the path (3), where $u_1 = \mathrm{root}(t)$, $u_n = u$, $(c,u) = \mathrm{root}(\tau)$, and $u_1,\ldots,u_n$ are the nodes on the path in t from $\mathrm{root}(t)$ to u. It is easy to check from the definition of T' that this root path exists in τ'.

Having defined these special paths, we now claim that every id-labeled node and every edge of τ' is on one of these paths. Consider an id-labeled node $((e,q),w)$ with $e = (j,c,c')$. Since $(t,w) \models \psi'_{\mathrm{id},(e,q)}(x)$, there are nodes $u,v \in V_t$ such that $(q_e,u) \twoheadrightarrow^*_{e,t} (q,w) \twoheadrightarrow^*_{e,t} (q',v)$ for some $q' \in F_e$. Consequently $(t,u,v) \models \chi_e(x,y)$ and so τ has an edge $(c,u) \xrightarrow{j} (c',v)$. This means that $(q_e,u) \twoheadrightarrow^*_{e,t} (q,w) \twoheadrightarrow^*_{e,t} (q',v)$ is in fact the unique walk (1) of A_e from u to v, and hence the node $((e,q),w)$ is on the path (2) associated with this edge of τ. A similar (but easier) argument shows that all id-labeled nodes of the form (c_{m},w) are on the root path (3). Thus, every id-labeled node of τ' is on one of the special paths. Since, as observed above, an id-labeled node has at most one outgoing edge, all these edges are also on the special paths. The remaining edges of τ' are of the form $(c,u) \xrightarrow{j} ((e,q_e),u)$ with $e = (j,c,c')$; obviously, the node $((e,q_e),u)$ has at most one such incoming edge, and hence this edge is also on a special path (2). This shows that also every edge of τ' is on one of the special paths. Having shown that τ' is of the above special form, we now prove the remaining facts.

Since every node (c,u) has the same number of outgoing edges in τ' as in τ, with the same labels, τ' satisfies the second requirement for being a term graph. Moreover, the special paths of τ' do not contain cycles, because the A_e are deterministic and hence the walks (1) do not contain repetitions. A cycle of τ' would consist of consecutive special paths, and thus would give a cycle in τ. Hence τ' is noncircular. Finally, since τ is clean, τ' has root $(c_{\mathrm{m}},\mathrm{root}(t))$, which shows that τ' is a clean term graph, and that the third locality requirement holds. It should now be clear, from the special form of τ' and the definition of 'prune', that $\mathrm{prune}(\tau') = \tau$.

It remains to be shown that τ' is a tree if τ is one. It obviously suffices, in view of the special form of τ', to prove that the special paths have no id-labeled nodes in common. Suppose that the special paths associated with two distinct edges $(c_1,u_1) \xrightarrow{j} (c'_1,v_1)$ and $(c_2,u_2) \xrightarrow{j'} (c'_2,v_2)$ of τ have an id-labeled node in common. Since id-labeled nodes have exactly one outgoing edge, these paths have to end at the same node $(c'_1,v_1) = (c'_2,v_2)$. But then (c'_1,v_1) has two incoming edges in τ, contradicting the fact that τ is a tree. □

We now prove that every MSO term graph transducer can be simulated by a, possibly circular, attributed tree transducer with look-ahead. More precisely,

we show the following theorem.

Theorem 14. *For every* MSO *term graph transducer T from Σ to Δ there exist an attributed relabeling r and an attributed tree transducer G such that, for every tree $t \in T_\Sigma$, G is noncircular on $r(t)$ and $\mathcal{G}(r(t)) = \mathcal{T}(t)$. Moreover, if T is an* MSO *tree transducer, then G is* SUR *on $r(t)$ for every $t \in T_\Sigma$, and if T is direction preserving, then G is* OS.

Proof. Let $T = (C, \Psi, X)$ be an MSO term graph transducer from Σ to Δ. We may assume, by Lemmas 13 and 4, that T is local. Let c_m be the copy name that satisfies the third locality requirement.

Based on the characterization of unary MSO formulas, Theorem 10 shows that it suffices to define an MSO relabeling r instead of an attributed relabeling. The MSO relabeling r will be used to compute the truth values of the node and edge formulas of T. Since T is local, the truth values of the edge formulas can indeed be stored in the labels of the nodes. Thus, r will be of the type discussed in Example 1(2, relab), from Σ to $\Sigma' = \Sigma \times \{\mathrm{true}, \mathrm{false}\}^n$, where each symbol $\sigma' = (\sigma, b_1, \ldots, b_n)$ stores truth values for the unary formulas $\phi_1(x), \ldots, \phi_n(x)$ that we wish to consider. To avoid having to order these formulas, we will write $[\phi_i(x)]_{\sigma'}$ for b_i.

The formulas that determine r are all the node formulas $\psi_{\delta,c}(x) \in \Psi$, and all the formulas $\chi_{j,c,c'}(x \cdot i, x \cdot i')$ with $\chi_{j,c,c'}(x, y) \in X$, and $i, i' \in \{0\} \cup \mathrm{rki}(\Delta) \cup \{\uparrow\}$. What we mean by the latter formulas is the following. For $i \in \mathrm{rki}(\Delta)$, $x \cdot i$ denotes the i-th child of x, $x \cdot 0$ denotes x itself, and $x \cdot \uparrow$ denotes the parent of x. Formally, $\chi_{j,c,c'}(x \cdot i, x \cdot i')$ is the formula $\exists y, z(\mathrm{edg}_i(x, y) \wedge \mathrm{edg}_{i'}(x, z) \wedge \chi_{j,c,c'}(y, z))$, with $\mathrm{edg}_0(x, y) = (x = y)$ and $\mathrm{edg}_\uparrow(x, y) = \mathrm{edg}(y, x)$.

The att $G = (\Sigma', S, I, \Omega, W, R, \alpha_\mathrm{m})$, with $\Omega = \{T_\Delta\}$, is defined as follows. For each copy name c of T, G has a synthesized attribute α_c and an inherited attribute β_c. The meaning attribute α_m of G is α_{c_m}.

For $t \in T_\Sigma$ and $u \in V_t$, the attribute $\langle \alpha_c, u \rangle$ of $r(t)$ will have the value $\mathrm{unf}_\tau(c, u)$ where $\tau = \mathcal{T}_\mathrm{gr}(t)$. Since T is local, the first two locality requirements imply that

$$\mathrm{unf}_\tau(c, u) = \delta(\mathrm{unf}_\tau(c_1, u_1), \ldots, \mathrm{unf}_\tau(c_k, u_k))$$

where each u_i is either u itself or one of its children or its parent, and the nodes $(c_1, u_1), \ldots, (c_k, u_k)$ of τ are all distinct. Thus, the (synthesized) attribute $\langle \alpha_c, u \rangle$ depends on the α-attributes of itself, its children, and its parent. If T is direction preserving, then there is no dependency on the attributes of the parent, and the inherited attributes β_c are not needed, i.e., G is OS. Otherwise, the inherited attribute β_c is used to transport the attribute α_c of a node down to its children, by a copy rule, cf. Fig. 8.

We now define the semantic rules of G. All root rules of G are "dummy" rules, i.e., rules $\langle \beta_c, 0 \rangle = \bot$, where $\bot$ is any element of Δ_0. For $\sigma' \in \Sigma'$, the semantic rules in $R(\sigma')$ are defined as follows.

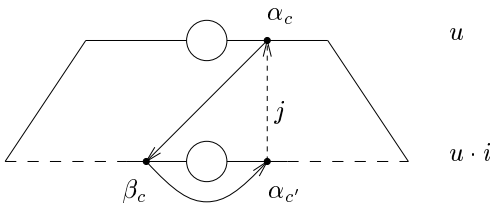


Fig. 8. Edge of the output graph (dashed line) and attribute dependencies (solid lines), for an edge from child to parent

- If $[\psi_{\delta,c}(x)]_{\sigma'} = \text{true}$ and $[\chi_{j,c,c_j}(x \cdot 0, x \cdot i_j)]_{\sigma'} = \text{true}$ for all $j \in [1,k]$ (where $k \in \text{rki}(\Delta)$, $\delta \in \Delta_k$, $c, c_1, \ldots, c_k \in C$, $i_1, \ldots, i_k \in [0, \text{rk}(\sigma')] \cup \{\uparrow\}$, and $(c_1, i_1), \ldots, (c_k, i_k)$ are all distinct), then $R(\sigma')$ contains the rule $\langle \alpha_c, 0\rangle = \delta(\langle \gamma_{c_1}, m_1\rangle, \ldots, \langle \gamma_{c_k}, m_k\rangle)$ where $\langle \gamma_{c_j}, m_j\rangle = \langle \alpha_{c_j}, i_j\rangle$ if $i_j \in [0, \text{rk}(\sigma')]$ and $\langle \gamma_{c_j}, m_j\rangle = \langle \beta_{c_j}, 0\rangle$ if $i_j = \uparrow$.

- If $[\chi_{j,c',c}(x \cdot i, x \cdot 0)]_{\sigma'} = \text{true}$ (where $i \in [1, \text{rk}(\sigma')]$, $j \in \text{rki}(\Delta)$, and $c', c \in C$), then $R(\sigma')$ contains the copy rule $\langle \beta_c, i\rangle = \langle \alpha_c, 0\rangle$.

Attributes $\langle \alpha_c, 0\rangle$ or $\langle \beta_c, i\rangle$ for which no semantic rule is defined above, are defined by dummy rules $\langle \alpha_c, 0\rangle = \bot$ or $\langle \beta_c, i\rangle = \bot$, respectively. Attributes for which more than one semantic rule is defined above, are also defined by dummy rules (this can only happen when σ' does not occur in any tree $r(t)$). Note that in the first item above, $\langle \gamma_{c_1}, m_1\rangle, \ldots, \langle \gamma_{c_k}, m_k\rangle$ are all distinct, i.e., the right-hand side of the semantic rule is linear, as required.

This ends the definition of G. To show the correctness of G, let $t \in T_\Sigma$ and $\tau = \mathcal{T}_{\text{gr}}(t)$. The definition of the attributed relabeling r implies the following facts. For a node u of $r(t)$ with label σ', $[\psi_{\delta,c}(x)]_{\sigma'} = \text{true}$ iff $(t,u) \models \psi_{\delta,c}(x)$ iff (c,u) is a node of τ with label δ. Also, $[\chi_{j,c,c'}(x \cdot i, x \cdot i')]_{\sigma'} = \text{true}$ iff $(t, u \cdot i, u \cdot i') \models \chi_{j,c,c'}(x \cdot i, x \cdot i')$ iff $(c, u \cdot i) \xrightarrow{j} (c', u \cdot i')$ is an edge of τ, where $u \cdot \uparrow$ denotes the parent of u. We will refer to these facts as "the correctness of r".

Suppose that G is noncircular on $r(t)$. Define the mapping $\text{dec} : A(r(t)) \to T_\Delta$ as follows. For $c \in C$ and $u \in V_t$,

- $\text{dec}(\langle \alpha_c, u\rangle) = \text{unf}_\tau(c,u)$ if $(c,u) \in V_\tau$, and $\bot$ otherwise, and
- $\text{dec}(\langle \beta_c, u\rangle) = \text{unf}_\tau(c,v)$ where v is the parent of u, if $((c',u), j, (c,v)) \in E_\tau$ for some $c' \in C$ and $j \in \text{rki}(\Delta)$, and $\bot$ otherwise.

The correctness of r implies that dec is a decoration of $r(t)$, and consequently $\text{dec} = \text{dec}_{G,r(t)}$, and so, using the third locality requirement, we obtain that

$$\mathcal{G}(r(t)) = \text{dec}(\langle \alpha_{\text{m}}, \text{root}(r(t))\rangle) = \text{unf}_\tau(c_{\text{m}}, \text{root}(t)) = \text{unf}_\tau(\text{root}(\tau)) = \text{unfold}(\tau) = \mathcal{T}(t).$$

To prove that G is noncircular on $r(t)$, and that G is SUR on $r(t)$ if τ is a tree, we consider the semantic graph $S_{G'}(r(t))$, where G' is the att in operator form which is obtained from G by changing every copy rule $\langle \beta_c, i\rangle = \langle \alpha_c, 0\rangle$ into $\langle \beta_c, i\rangle = \mathrm{id}(\langle \alpha_c, 0\rangle)$, cf. Lemma 5 and its proof. It now suffices to show that $S_{G'}(r(t))$ is noncircular, and that $S_{G'}(r(t))$ is a forest if τ is a tree. To prove these facts, we consider the edges of $S_{G'}(r(t))$. By the definition of G and the correctness of r, the edges of $S_{G'}(r(t))$ are of one of the following three forms

$\alpha\alpha$: $\langle \alpha_c, u\rangle \xrightarrow{j} \langle \alpha_{c'}, u \cdot i\rangle$ with $i \in \mathbb{N}$ and $(c, u) \xrightarrow{j} (c', u \cdot i)$ in τ,
$\alpha\beta$: $\langle \alpha_c, u\rangle \xrightarrow{j} \langle \beta_{c'}, u\rangle$ with $(c, u) \xrightarrow{j} (c', u{\cdot}\uparrow)$ in τ, or
$\beta\alpha$: $\langle \beta_{c'}, u\rangle \xrightarrow{1} \langle \alpha_{c'}, u{\cdot}\uparrow\rangle$,

where, again, $u{\cdot}\uparrow$ denotes the parent of u. Moreover, if edge $(\beta\alpha)$ is in $S_{G'}(r(t))$ then so is edge $(\alpha\beta)$ for some $c \in C$.

This shows that a cycle in $S_{G'}(r(t))$ is changed into a cycle in τ by changing every edge $\langle \alpha_c, u\rangle \xrightarrow{j} \langle \alpha_{c'}, u'\rangle$ into $(c, u) \xrightarrow{j} (c', u')$, and every two consecutive edges $\langle \alpha_c, u\rangle \xrightarrow{j} \langle \beta_{c'}, u\rangle \xrightarrow{1} \langle \alpha_{c'}, u{\cdot}\uparrow\rangle$ into $(c, u) \xrightarrow{j} (c', u{\cdot}\uparrow)$. Since τ is noncircular, so is $S_{G'}(r(t))$, which means that G is noncircular on $r(t)$.

It also shows that if a node $\langle \alpha_c, u\rangle$ of $S_{G'}(r(t))$ has two incoming $(\alpha\alpha)$-edges, then (c, u) has two incoming edges in τ. Similarly, if a node $\langle \beta_{c'}, u\rangle$ of $S_{G'}(r(t))$ has two incoming edges (which are necessarily $(\alpha\beta)$-edges), then $(c', u{\cdot}\uparrow)$ has two incoming edges in τ from two nodes (c_1, u) and (c_2, u). If $\langle \alpha_{c'}, v\rangle$ has two incoming $(\beta\alpha)$-edges, then (c', v) has two incoming edges from nodes (c_1, u_1) and (c_2, u_2), for two children u_1, u_2 of v. Finally, if $\langle \alpha_c, u\rangle$ has an incoming $(\alpha\alpha)$-edge and an incoming $(\beta\alpha)$-edge, then (c, u) has two incoming edges from nodes (c_1, u_1) and (c_2, u_2), where u_2 is a child of u and u_1 is not. Altogether this shows that if $S_{G'}(r(t))$ has a node with two incoming edges, then so has τ. Thus, if τ is a tree, then $S_{G'}(r(t))$ is a forest, which means that G is SUR on $r(t)$.

This proves the theorem. We finally note that τ can in fact be obtained from $S_{G'}(r(t))$ by first removing all (isolated) nodes $\langle \alpha_c, u\rangle$ of $S_{G'}(r(t))$ with $(c, u) \notin V_\tau$, and then applying 'prune' (and removing the root mark # if τ has none). □

An Example In the remainder of this section we give an example of the implementation of an MSO term graph transduction by an attributed tree transducer with look-ahead, as described in Lemma 13 and Theorem 14.

Consider the MSO tree transducer $T = (\{c\}, \Psi, X)$ of Example 1(6, yield) which transforms a binary tree into its yield, viewed as a monadic tree. Recall that T is from $\Sigma = \Sigma_0 \cup \Sigma_2$ to $\Delta = \Delta_0 \cup \Delta_1$, with $\Delta_1 = \Sigma_0$ and $\Delta_0 = \{\hat{\sigma} \mid \sigma \in \Sigma_0\}$. Recall also that T has the following node and edge formulas:

$$\begin{aligned}
\psi_{\sigma,c}(x) &= \mathrm{lab}_\sigma(x) \wedge \neg\, \mathrm{rm}(x) && \text{for } \sigma \in \Sigma_0,\\
\psi_{\hat{\sigma},c}(x) &= \mathrm{lab}_\sigma(x) \wedge \mathrm{rm}(x) && \text{for } \sigma \in \Sigma_0,\\
\chi_{1,c,c}(x, y) &= \mathrm{leaf}(x) \wedge \chi(x, y) \wedge \mathrm{leaf}(y),
\end{aligned}$$

where the formula $\chi(x,y)$ checks that y directly follows x in the left-to-right order of leaves, and the formula $\mathrm{rm}(x)$ expresses that x is a node on the path from the root to the rightmost leaf. We will also need the formula $\mathrm{lm}(x) = \forall y(\mathrm{root}(y) \to \mathrm{path}_1(y,x))$ which expresses that x is a node on the path from the root to the leftmost leaf.

In the following, let $e = (1,c,c)$. Clearly, this is the only element of $E = \mathrm{rki}(\Delta) \times C \times C$. Let $A_e = A = (Q,\Delta,\delta,I,F)$ be the deterministic tree-walking automaton of Example 4 which is equivalent with $\chi_e(x,y)$, see Fig. 6.

The local MSO tree transducer T' constructed in the proof of Lemma 13 has copy names $C' = \{c, \mathrm{I}, \mathrm{II}, \mathrm{III}, \mathrm{IV}, \mathrm{V}, c_\mathrm{m}\}$, where we have identified the copy name (e,q) with q for every $q \in Q$, because there is only one automaton. We now give the node formulas of T', or rather, node formulas that are equivalent to the ones that are actually constructed. First, the node formulas of T' include those of T, with a prime. Second, after analyzing the precise behavior of A, it can be seen that T' has the following node formulas $\psi'_{\mathrm{id},q}(x)$ for $q \in Q$:

$$\begin{aligned}
\psi'_{\mathrm{id},\mathrm{I}}(x) &= \mathrm{leaf}(x) \wedge \neg\, \mathrm{rm}(x),\\
\psi'_{\mathrm{id},\mathrm{II}}(x) &= \neg\, \mathrm{rm}(x),\\
\psi'_{\mathrm{id},\mathrm{III}}(x) &= \neg\, \mathrm{leaf}(x),\\
\psi'_{\mathrm{id},\mathrm{IV}}(x) &= \neg\, \mathrm{lm}(x), \text{ and}\\
\psi'_{\mathrm{id},\mathrm{V}}(x) &= \mathrm{leaf}(x) \wedge \neg\, \mathrm{lm}(x).
\end{aligned}$$

Third, since the root of the output tree is the leftmost leaf, $\psi'_{\mathrm{id},c_\mathrm{m}}(x) = \mathrm{lm}(x)$. Next, we give the edge formulas of T'. First, since $q_e = \mathrm{I}$, it has the edge formula $\chi'_{1,c,\mathrm{I}}(x,y) = (x = y)$. Second, since $F = \{\mathrm{V}\}$, $\chi'_{1,\mathrm{V},c}(x,y) = (x = y)$. Third, the transitions of A result in the following six edge formulas:

$$\begin{aligned}
\chi'_{1,\mathrm{I},\mathrm{II}}(x,y) &= (\mathrm{leaf}(x) \wedge x = y),\\
\chi'_{1,\mathrm{II},\mathrm{II}}(x,y) &= \mathrm{edg}_2(y,x),\\
\chi'_{1,\mathrm{II},\mathrm{III}}(x,y) &= \mathrm{edg}_1(y,x),\\
\chi'_{1,\mathrm{III},\mathrm{IV}}(x,y) &= \mathrm{edg}_2(x,y),\\
\chi'_{1,\mathrm{IV},\mathrm{IV}}(x,y) &= \mathrm{edg}_1(x,y), \text{ and}\\
\chi'_{1,\mathrm{IV},\mathrm{V}}(x,y) &= (\mathrm{leaf}(x) \wedge x = y).
\end{aligned}$$

Fourth, $\chi'_{1,c_\mathrm{m},c_\mathrm{m}}(x,y) = \mathrm{edg}(x,y)$ and, since the root of the output tree is the leftmost leaf, $\chi'_{1,c_\mathrm{m},c}(x,y) = (\mathrm{leaf}(x) \wedge \mathrm{lm}(x) \wedge x = y)$.

In what follows, it is assumed that each of the above edge formulas $\chi'_{1,c_1,c_2}(x,y)$ of T' is restricted in such a way that $(t,u,v) \models \chi'_{1,c_1,c_2}(x,y)$ implies that (c_1,u) and (c_2,v) are nodes of the output tree $\mathcal{T}'(t)$, as discussed in Section 3 and, in fact, as assumed in the proof of Theorem 14.

We now turn to the attributed relabeling r and attributed tree transducer G, constructed in the proof of Theorem 14 for the local MSO tree transducer T'. As explained in that proof, r computes the truth values of the node and edge formulas of T'. In this example, since the truth values of the formulas $\mathrm{leaf}(x)$

and $\mathrm{lab}_\sigma(x)$, for $\sigma \in \Sigma_0$, are obvious from the Σ-label of the node x, and since most of the formulas $\chi_{j,c,c'}(x \cdot i, x \cdot i')$ are trivial for similar reasons, it suffices that r computes the truth values of the formulas $\mathrm{rm}(x)$, $\mathrm{lm}(x)$, $\exists y(\mathrm{edg}_1(y,x))$, and $\exists y(\mathrm{edg}_2(y,x))$. Thus, the symbols of the output alphabet Σ' of r are of the form $\sigma' = (\sigma, b_1, b_2, b_3, b_4)$, with $\sigma \in \Sigma$ and $b_i \in \{\mathrm{true}, \mathrm{false}\}$.

The att G has synthesized attributes $\alpha_c, \alpha_{\mathrm{I}}, \ldots, \alpha_{\mathrm{V}}, \alpha_{c_{\mathrm{m}}}$, with $\alpha_{\mathrm{m}} = \alpha_{c_{\mathrm{m}}}$, and it has inherited attributes β_{II} and β_{III}. In fact, since, for $i \geq 1$ and $c_1, c_2 \in C'$, $[\chi_{j,c_1,c_2}(x \cdot i, x \cdot 0)]_{\sigma'}$ can only be true for $c_1 = \mathrm{II}$ and $c_2 \in \{\mathrm{II}, \mathrm{III}\}$, the other inherited attributes of G will be defined by dummy rules only, and hence are superfluous.

For $\sigma' \in \Sigma'_2$, if $[\mathrm{rm}(x)]_{\sigma'} = \mathrm{false}$, then $R(\sigma')$ contains the semantic rules $\langle \beta_{\mathrm{II}}, 2\rangle = \langle \alpha_{\mathrm{II}}, 0\rangle$ and $\langle \beta_{\mathrm{III}}, 1\rangle = \langle \alpha_{\mathrm{III}}, 0\rangle$, and all other β-rules are dummy rules.

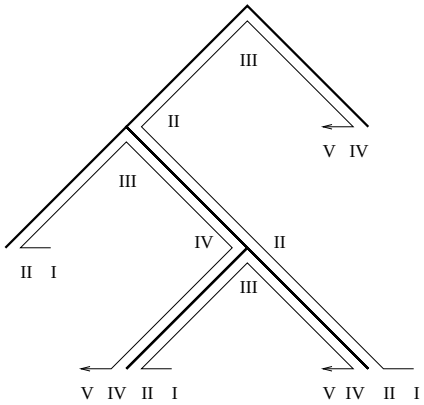


Fig. 9. All walks of the automaton on tree t

The synthesized attributes have the following semantic rules. We do not mention the dummy rules, i.e., we define the attribute α_c only for leaves, and an attribute $\alpha_{c'}$ with $c' \neq c$ only for nodes x for which $\psi'_{\mathrm{id},c'}(x)$ holds.

- For $\sigma' = (\sigma, b_1, b_2, b_3, b_4) \in \Sigma'_0$, $R(\sigma')$ contains the rule $\langle \alpha_c, 0\rangle = \sigma(\langle \alpha_{\mathrm{I}}, 0\rangle)$ if $[\mathrm{rm}(x)]_{\sigma'} = \mathrm{false}$, and the rule $\langle \alpha_c, 0\rangle = \hat{\sigma}$ otherwise.
- For $\sigma' \in \Sigma'_0$ with $[\mathrm{rm}(x)]_{\sigma'} = \mathrm{false}$, $R(\sigma')$ contains $\langle \alpha_{\mathrm{I}}, 0\rangle = \langle \alpha_{\mathrm{II}}, 0\rangle$.
- For every $\sigma' \in \Sigma'$ with $[\mathrm{rm}(x)]_{\sigma'} = \mathrm{false}$, $R(\sigma')$ contains $\langle \alpha_{\mathrm{II}}, 0\rangle = \langle \beta_{\mathrm{II}}, 0\rangle$ if $[\exists y(\mathrm{edg}_2(y,x))]_{\sigma'} = \mathrm{true}$, and $\langle \alpha_{\mathrm{II}}, 0\rangle = \langle \beta_{\mathrm{III}}, 0\rangle$ if $[\exists y(\mathrm{edg}_1(y,x))]_{\sigma'} = \mathrm{true}$.
- For $\sigma' \in \Sigma'_2$, $R(\sigma')$ contains $\langle \alpha_{\mathrm{III}}, 0\rangle = \langle \alpha_{\mathrm{IV}}, 2\rangle$.

– For every $\sigma' \in \Sigma'$ with $[\mathrm{lm}(x)]_{\sigma'} = \mathrm{false}$, $R(\sigma')$ contains $\langle \alpha_{\mathrm{IV}}, 0\rangle = \langle \alpha_{\mathrm{IV}}, 1\rangle$ if $\sigma' \in \Sigma'_2$, and $\langle \alpha_{\mathrm{IV}}, 0\rangle = \langle \alpha_{\mathrm{V}}, 0\rangle$ otherwise.

– For $\sigma' \in \Sigma'_0$ with $[\mathrm{lm}(x)]_{\sigma'} = \mathrm{false}$, $R(\sigma')$ contains $\langle \alpha_{\mathrm{V}}, 0\rangle = \langle \alpha_c, 0\rangle$.

– For every $\sigma' \in \Sigma'$ with $[\mathrm{lm}(x)]_{\sigma'} = \mathrm{true}$, $R(\sigma')$ contains $\langle \alpha_{\mathrm{m}}, 0\rangle = \langle \alpha_{\mathrm{m}}, 1\rangle$ if $\sigma' \in \Sigma'_2$, and $\langle \alpha_{\mathrm{m}}, 0\rangle = \langle \alpha_c, 0\rangle$ otherwise.

This ends the description of r and G. To conclude the example, we consider how G acts on tree $t = \$(\$(a, \$(b, a)), a)$, where $a, b \in \Sigma_0$ and $\$ \in \Sigma_2$, with output $\mathcal{T}(t) = a(b(a(\hat{a})))$. Figure 9 shows all walks of the automaton on t. Figure 10 gives the dependency graph $D(r(t))$. It shows the inherited attributes to the

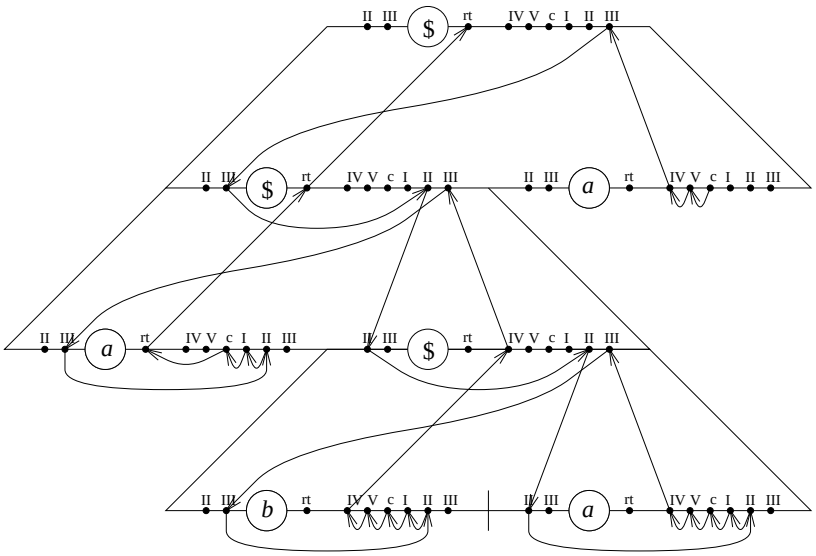


Fig. 10. The dependency graph of $r(t)$

left of each node, and the synthesized attributes to the right. The label of each node is the one in t. The attribute names have been abbreviated. Attribute α_{m} is denoted m, α_c is denoted c, and the attributes α_q and β_q are both denoted q, for all $q \in Q$. Note that the direction of the edges in $D(r(t))$, i.e., the direction in which the attributes are evaluated, is opposite to the direction in which the automaton walks on t (cf. the first part of the proof of Lemma 13).

We can easily infer the values of the attributes. Let the leaves of t be numbered u_1 through u_4, from left to right. The values of the α_c are $\mathrm{dec}_{G,t}(\langle \alpha_c, u_4\rangle) = \hat{a}$, $\mathrm{dec}_{G,t}(\langle \alpha_c, u_3\rangle) = a(\hat{a})$, $\mathrm{dec}_{G,t}(\langle \alpha_c, u_2\rangle) = b(a(\hat{a}))$, and $\mathrm{dec}_{G,t}(\langle \alpha_c, u_1\rangle) = a(b(a(\hat{a})))$, and the other occurrences of α_c have value $\perp$. The rules for all the

other attributes merely copy values, so, in particular, $\mathrm{dec}_{G,t}(\langle \alpha_{\mathrm{m}}, \mathrm{root}(t)\rangle) = a(b(a(\hat{a})))$.

It should be clear that r and G are very similar to $r(G_1)$ and G_2 of Example 2(6, yield). Roughly speaking, the attribute 'down' of that example corresponds to the attributes α_{I}, α_{II}, α_{III}, β_{II}, and β_{III}, and the attribute 'up' to the attributes α_c, α_{IV}, α_{V}, and α_{m}.

8 Noncircularity and single use

We have shown in Theorem 14 that every MSO term graph transducer can be simulated by a, possibly circular, $\mathrm{att}^{\mathrm{R}}$. We might be quite satisfied with this, but it is even more satisfactory to have a noncircular $\mathrm{att}^{\mathrm{R}}$, and, thus, to be able to state that MSO-TGT $\subseteq$ ATT-REL $\circ$ ATT. That this is possible will be proved in this section. Thus, we will show the following theorem. Note that without the last statement, the theorem says the following: for every $\mathrm{att}^{\mathrm{R}}$ that computes a total function there is an equivalent noncircular $\mathrm{att}^{\mathrm{R}}$.

Theorem 15. *Let r be an attributed relabeling and G be an att which is noncircular on $r(t)$ for every input tree t of r. Then there exist an attributed relabeling r' and a noncircular att G' such that $r' \circ \mathcal{G}' = r \circ \mathcal{G}$. Moreover, if G is* SUR *on $r(t)$ for every input tree t of r, then G' is* SUR*, and if G is* OS*, then so is G'.*

To prove this, it suffices to prove the following lemma. Note that the first sentence of this lemma says the following: for every att G there is a noncircular $\mathrm{att}^{\mathrm{R}}$ (G_1, G_2) such that $\mathcal{G} \subseteq r(G_1) \circ \mathcal{G}_2$.

Lemma 16.

1. *For every att G there exist an attributed relabeling r_0 and a noncircular att G' such that $\mathcal{G}'(r_0(t)) = \mathcal{G}(t)$ for every input tree t on which G is noncircular. Moreover, for every input tree t, if G is* SUR *on t, then G' is* SUR *on $r_0(t)$.*
2. *For every noncircular att G' there exist an attributed relabeling r_1 and a noncircular* SUR *att G'' such that $\mathcal{G}''(r_1(t)) = \mathcal{G}'(t)$ for every input tree t on which G' is* SUR*. Moreover, if G' is* OS*, then so is G''.*

In fact, assuming this lemma, the first part of Theorem 15 can be proved as follows. Take $r' = r \circ r_0$. By Proposition 2 and Theorem 10, ATT-REL is closed under composition, and hence r' is an attributed relabeling. Then, for every input tree t of r, $\mathcal{G}'(r'(t)) = \mathcal{G}'(r_0(r(t))) = \mathcal{G}(r(t))$ because G is noncircular on $r(t)$. To prove the second part of Theorem 15 (with r'' and G'' instead of r' and G', respectively), take $r'' = r' \circ r_1$. Again, r'' is an attributed relabeling. Let t be an input tree of r. Since G is SUR on $r(t)$, G' is SUR on $r_0(r(t)) = r'(t)$. Hence $\mathcal{G}''(r_1(r'(t))) = \mathcal{G}'(r'(t))$, and so $\mathcal{G}''(r''(t)) = \mathcal{G}(r(t))$. Note that for the OS-part we only need the second statement of Lemma 16, because every OS att is noncircular.

In the remainder of this section we prove Lemma 16. The lemma holds for arbitrary attribute grammars, in such a way that G' has the same semantic

domains and uses the same functions f in its semantic rules (and similarly for G''). The reader who is not interested in technical subtleties should skip the remainder of this section.

We start with the first part of Lemma 16. Let $G = (\Sigma, S, I, \Omega, W, R, \alpha_{\mathrm{m}})$ be an arbitrary att, with $\Omega = \{T_\Delta\}$. We will abbreviate $S \cup I$ by A. Note that if the dependency graph $D(\mathrm{root})$ of the root is circular, then G is circular on every input tree, and Lemma 16 holds trivially. Hence, we may assume that $D(\mathrm{root})$ is noncircular.

The relabeling r_0 will add is-graphs, well known from the circularity test for attribute grammars, to the labels of the nodes of an input tree t. This will allow G' to see whether or not G is circular on t. Let us first recall these well-known ideas from [Knu], in our notation. An *is-graph* is a subset of $I \times S$. For $\sigma \in \Sigma_k$ and is-graphs $q_1, \ldots, q_k$, $D(\sigma)[q_1, \ldots, q_k]$ is the unlabeled directed graph obtained from the dependency graph $D(\sigma)$ of σ by adding all edges $(\langle \alpha, i\rangle, \langle \beta, i\rangle)$ with $(\alpha, \beta) \in q_i$, for $i \in [1, k]$. Furthermore, $\pi_0(D(\sigma)[q_1, \ldots, q_k])$ is the is-graph consisting of all (α, β) such that there is a path from $\langle \alpha, 0\rangle$ to $\langle \beta, 0\rangle$ in $D(\sigma)[q_1, \ldots, q_k]$. For an is-graph q, $D(\mathrm{root})[q]$ is the unlabeled directed graph obtained from $D(\mathrm{root})$ by adding all edges $(\langle \alpha, 0\rangle, \langle \beta, 0\rangle)$ with $(\alpha, \beta) \in q$. For $t \in T_\Sigma$ and $u \in V_t$, the *is-graph of* u, denoted $\mathrm{is}(u)$, is defined recursively to be the is-graph $\mathrm{is}(u) = \pi_0(D(\sigma)[\mathrm{is}(u \cdot 1), \ldots, \mathrm{is}(u \cdot k)])$, where $\mathrm{lab}_t(u) = \sigma \in \Sigma_k$. The circularity test for a single tree $t \in T_\Sigma$ is now as follows: $D(t)$ is circular iff either $D(\mathrm{root})[\mathrm{is}(\mathrm{root}(t))]$ is circular or there exists $u \in V_t$ such that $D(\sigma)[\mathrm{is}(u \cdot 1), \ldots, \mathrm{is}(u \cdot k)]$ is circular, where $\mathrm{lab}_t(u) = \sigma \in \Sigma_k$.

We now define the attributed relabeling r_0. Let $Q = \mathcal{P}(I \times S)$ be the set of all is-graphs, and let $\Theta = \{\mathrm{root}, \mathrm{nonroot}\}$. The output alphabet Σ' of r_0 (which will also be the input alphabet of G') consists of all symbols

$$(\sigma, \theta, (q_0, q_1, \ldots, q_k))$$

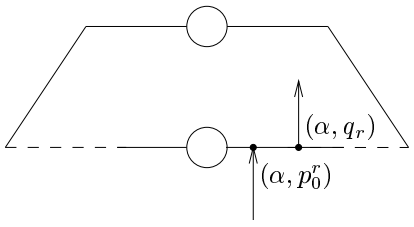
of rank k, with $k \geq 0$, $\sigma \in \Sigma_k$, $\theta \in \Theta$, $q_i \in Q$ for all $i \in [0, k]$, and $q_0 = \pi_0(D(\sigma)[q_1, \ldots, q_k])$. Note that q_0 is superfluous; it is added to simplify the description of the semantic rules of G'. The attributed relabeling r_0 relabels a node u of a tree $t \in T_\Sigma$ that has label $\sigma \in \Sigma_k$, with the label $(\sigma, \theta, (q_0, q_1, \ldots, q_k)) \in \Sigma'$, where $q_i = \mathrm{is}(u \cdot i)$ for all $i \in [0, k]$, and $\theta = \mathrm{root}$ iff $u = \mathrm{root}(t)$. It is straightforward to define a relabeling attribute grammar G_0 such that $r(G_0) = r_0$. It has a synthesized attribute 'is' with $W(\mathrm{is}) = Q$ to compute $\mathrm{is}(u)$ for every node u, an inherited attribute 'r' with $W(\mathrm{r}) = \Theta$ to indicate the root, and a (synthesized) meaning attribute 'lab' with $W(\mathrm{lab}) = \Sigma'$ to compute the new label of every node. It has one root rule: $\langle \mathrm{r}, 0\rangle = \mathrm{root}$. For every $\sigma \in \Sigma_k$ it has the internal rules

$$\begin{aligned} \langle \mathrm{is}, 0\rangle &= \pi_0(D(\sigma)[\langle \mathrm{is}, 1\rangle, \ldots, \langle \mathrm{is}, k\rangle]) \\ \langle \mathrm{r}, i\rangle &= \mathrm{nonroot} \qquad \text{for all } i \in [1, k] \\ \langle \mathrm{lab}, 0\rangle &= (\sigma, \langle \mathrm{r}, 0\rangle, (\langle \mathrm{is}, 0\rangle, \langle \mathrm{is}, 1\rangle, \ldots, \langle \mathrm{is}, k\rangle)). \end{aligned}$$

From these rules it should be clear that $r(G_0) = r_0$.

We now turn to the definition of the att G' from Σ' to Δ. For G' we will use the same notation as for G, with primes. The attributes of G' are $S' = S \cup (S \times Q)$ and $I' = I \cup (I \times Q)$, with $\alpha'_{\mathrm{m}} = \alpha_{\mathrm{m}}$. We denote $S' \cup I'$ by A'.

The idea in the construction of G' is the following. Let t' be a tree over Σ', and let t be the tree over Σ that is obtained from t' by changing every label $(\sigma, \theta, (q_0, q_1, \ldots, q_k))$ into σ. Let us first consider the case that $t' = r_0(t)$. Then the attribute $\langle \alpha, u \rangle$ of t in G is simulated in G' by the attribute $\langle (\alpha, q), u \rangle$ of t' if $\mathrm{is}(u) = q$ and u is not the root, and by the attribute $\langle \alpha, u \rangle$ if u is the root. The remaining attributes of t' are superfluous and get a dummy value. This will guarantee that $\mathcal{G}'(r_0(t)) = \mathcal{G}(t)$ if $D(t)$ is noncircular. If $D(t)$ is circular, then all cycles of $D(t)$ will be broken in $D'(t')$ because we will define $D'(\sigma, \theta, (q_0, q_1, \ldots, q_k))$ to have no edges whenever $D(\sigma)[q_1, \ldots, q_k]$ is circular (and when $D(\mathrm{root})[q_0]$ is circular, if $\theta = \mathrm{root}$). Now consider the case that $t' \neq r_0(t)$. Then a possible cycle in $D(t)$ will be broken in $D'(t')$ either for the same reason as above, or because the q_i in the labels of t' do not "fit". Roughly speaking, the semantic rules for a symbol $(\sigma, \theta, (q_0, q_1, \ldots, q_k))$ in G' will be the same as the semantic rules for σ in G, except that every $\langle \alpha, i \rangle$ is replaced by $\langle (\alpha, q_i), i \rangle$. Now, if a node u of t' has label $(\sigma, \theta, (q_0, q_1, \ldots, q_k))$, and its i-th child has a label of the form $(-, -, (p_0, \ldots))$ that does not "fit", i.e., with $p_0 \neq q_i$, then a path in $D(t)$ through $\langle \alpha, u \cdot i \rangle$ will be broken in $D'(t')$ because $\langle \alpha, u \cdot i \rangle$ is split into $\langle (\alpha, p_0), u \cdot i \rangle$ and $\langle (\alpha, q_i), u \cdot i \rangle$ between which there is no connection, see Fig. 11. It remains to specify the semantic rules of G'. For $\langle \alpha', i \rangle \in A' \times \mathbb{N}$,



$\mathrm{lab}_{t'}(u) = (\sigma, \theta, (\ldots, q_r, \ldots))$

$\mathrm{lab}_{t'}(u \cdot r) = (\sigma', \theta', (p_0^r, \ldots))$

Fig. 11. Nonfitting labels of a parent and child

a *dummy rule* for $\langle \alpha', i \rangle$ is a semantic rule $\langle \alpha', i \rangle = \perp$, where $\perp$ is an arbitrary element of Δ_0. Note that dummy rules do not produces edges in dependency graphs.

The root rules of G' are the root rules of G, together with dummy rules for the attributes in $I \times Q$.

The internal rules of G' for a symbol $\sigma' = (\sigma, \theta, (q_0, q_1, \ldots, q_k)) \in \Sigma'$ are defined as follows, distinguishing between the two possible values of θ.

Case 1: $\theta = \mathrm{nonroot}$. If $D(\sigma)[q_1, \ldots, q_k]$ is circular, then all internal rules for σ' are dummy rules. Otherwise, $R'(\sigma')$ consists of all rules of $R(\sigma)$, in which every $\langle \alpha, i \rangle$ is replaced by $\langle (\alpha, q_i), i \rangle$, and dummy rules for the remaining attributes.

Case 2: $\theta = \text{root}$. If $D(\sigma)[q_1, \ldots, q_k]$ is circular or $D(\text{root})[q_0]$ is circular, then all internal rules for σ' are dummy rules. Otherwise, $R'(\sigma')$ consists of all rules of $R(\sigma)$, in which every $\langle \alpha, i \rangle$ with $i \neq 0$ is replaced by $\langle (\alpha, q_i), i \rangle$, and dummy rules for the remaining attributes.

This ends the construction of G'. Clearly, if G is SUR on $t \in T_\Sigma$, then G' is SUR on $r_0(t)$. In fact, G' is then SUR on any tree $t' \in T_{\Sigma'}$ that is obtained from t by changing each label σ into some $(\sigma, \theta, (q_0, q_1, \ldots, q_k))$.

We will now first show that r_0 and G' compute the same tree transduction as G, for noncircular input, and then we will show that G' is noncircular.

Let $t \in T_\Sigma$ and $t' = r_0(t)$, and assume that both $D(t)$ and $D'(t')$ are noncircular. We have to prove that $\mathcal{G}'(t') = \mathcal{G}(t)$. For that purpose we define a valuation $\text{val} : A'(t') \to T_\Delta$ of the attributes of t' as follows. Let u be a node of t' with label $(\sigma, \theta, (q_0, q_1, \ldots, q_k))$. If $\theta = \text{nonroot}$, then $\text{val}(\langle (\alpha, q_0), u \rangle) = \text{dec}_{G,t}(\langle \alpha, u \rangle)$ for every $\alpha \in A$, and $\text{val}(\langle \alpha', u \rangle) = \bot$ for all remaining attributes $\alpha' \in A'$. If $\theta = \text{root}$, then $\text{val}(\langle \alpha, u \rangle) = \text{dec}_{G,t}(\langle \alpha, u \rangle)$ for every $\alpha \in A$, and $\text{val}(\langle \alpha', u \rangle) = \bot$ for every $\alpha' \in A \times Q$.

We claim that val is a decoration of t' (for G'). In fact, for every node u of t' with label $(\sigma, \theta, (q_0, q_1, \ldots, q_k))$, it follows from the definition of r_0 that $q_i = \text{is}(u \cdot i)$ for all $i \in [0, k]$, and that $\theta = \text{root}$ iff u is the root. Since $D(t)$ is noncircular, this implies that $D(\sigma)[q_1, \ldots, q_k]$ is noncircular, and that $D(\text{root})[q_0]$ is noncircular if u is the root. It now follows from the definition of the semantic rules of G' (and the fact that $\text{dec}_{G,t}$ is a decoration of t) that val is a decoration of t'. Hence, since $D'(t')$ is noncircular by assumption, $\text{val} = \text{dec}_{G',t'}$ and so $\mathcal{G}'(t') = \text{dec}_{G',t'}(\langle \alpha'_{\text{m}}, \text{root}(t') \rangle) = \text{val}(\langle \alpha_{\text{m}}, \text{root}(t) \rangle) = \text{dec}_{G,t}(\langle \alpha_{\text{m}}, \text{root}(t) \rangle) = \mathcal{G}(t)$.

Finally we show that G' is noncircular, i.e., that $D'(t')$ is noncircular for every tree $t' \in T_{\Sigma'}$. We claim that the following four statements hold for every node u of t', with label $\sigma' = (\sigma, \theta, (q_0, q_1, \ldots, q_k))$.

1. If $\theta = \text{nonroot}$ and $(\alpha', \beta') \in \text{is}'(u)$, then there exist $\alpha, \beta \in A$ such that $\alpha' = (\alpha, q_0)$, $\beta' = (\beta, q_0)$, and $(\alpha, \beta) \in q_0$.
2. If $\theta = \text{root}$ and $(\alpha', \beta') \in \text{is}'(u)$, then $\alpha', \beta' \in A$ and $(\alpha', \beta') \in q_0$.
3. If $D'(\sigma')[\text{is}'(u \cdot 1), \ldots, \text{is}'(u \cdot k)]$ is circular, then $D(\sigma)[q_1, \ldots, q_k]$ is circular.
4. If $D'(\text{root})[\text{is}'(u)]$ is circular, then $\theta = \text{root}$ and $D(\text{root})[q_0]$ is circular.

From Statements (3) and (4) the noncircularity of $D'(t')$ can be proved as follows. Assume that $D'(t')$ is circular. Then either there is a node u such that $D'(\sigma')[\text{is}'(u \cdot 1), \ldots, \text{is}'(u \cdot k)]$ is circular, or $D'(\text{root})[\text{is}'(u)]$ is circular for $u = \text{root}(t)$. In the first case, Statement (3) implies that $D(\sigma)[q_1, \ldots, q_k]$ is circular, and so, by the definition of the semantic rules of G', $D'(\sigma')$ has no edges, contradicting the circularity of $D'(\sigma')[\text{is}'(u \cdot 1), \ldots, \text{is}'(u \cdot k)]$. In the second case, Statement (4) implies that $\theta = \text{root}$ and $D(\text{root})[q_0]$ is circular, and so also in this case, by the definition of the semantic rules of G', $D'(\sigma')$ has no edges. This implies that $\text{is}'(u)$ has no edges, and so, since G' and G have the same root rules, $D(\text{root}) = D'(\text{root}) = D'(\text{root})[\text{is}'(u)]$ is circular. We have assumed, however, that $D(\text{root})$ is noncircular.

It remains to show the above four statements. First Statements (1) and (2) are proved simultaneously, by induction on u, and then Statements (3) and (4) are proved. In these proofs the following lemma is useful. Let $\pi : A' \to A$ be defined by $\pi((\alpha, q)) = \pi(\alpha) = \alpha$ for $\alpha \in A$ and $q \in Q$.

Lemma 17. *Let u be a node of $t' \in T_{\Sigma'}$ with label $\sigma' = (\sigma, \theta, (q_0, q_1, \ldots, q_k))$, and assume that Statements (1) and (2) hold for $u \cdot 1, \ldots, u \cdot k$. If*

$$\langle \alpha'_0, i_0 \rangle e_1 \langle \alpha'_1, i_1 \rangle e_2 \langle \alpha'_2, i_2 \rangle \cdots e_n \langle \alpha'_n, i_n \rangle$$

is a path in $D'(\sigma')[\mathrm{is}'(u \cdot 1), \ldots, \mathrm{is}'(u \cdot k)]$, with $e_j = (\langle \alpha'_{j-1}, i_{j-1} \rangle, \langle \alpha'_j, i_j \rangle)$, and e_1 is an edge of $D'(\sigma')$, then

$$\langle \pi(\alpha'_0), i_0 \rangle \pi(e_1) \langle \pi(\alpha'_1), i_1 \rangle \pi(e_2) \langle \pi(\alpha'_2), i_2 \rangle \cdots \pi(e_n) \langle \pi(\alpha'_n), i_n \rangle$$

is a path in $D(\sigma)[q_1, \ldots, q_k]$, with $\pi(e_j) = (\langle \pi(\alpha'_{j-1}), i_{j-1} \rangle, \langle \pi(\alpha'_j), i_j \rangle)$.

Proof. Let the label of $u \cdot r$ be $(\sigma_r, \theta_r, (p^r_0, p^r_1, \ldots, p^r_{k_r}))$, for $r \in [1, k]$. Now the essence of the whole construction of G' is that if the given path uses an edge from $\mathrm{is}'(u \cdot r)$, $r \in [1, k]$, then $p^r_0 = q_r$, see Fig. 11. The formal proof is as follows.

It is an immediate consequence of the definition of the semantic rules of G' that if e_j is an edge in $D'(\sigma')$, then $\pi(e_j)$ is an edge in $D(\sigma)$ and hence in $D(\sigma)[q_1, \ldots, q_k]$. Now consider an edge $e_{j+1} = (\langle \alpha'_j, r \rangle, \langle \alpha'_{j+1}, r \rangle)$ from $\mathrm{is}'(u \cdot r)$; or more precisely, $(\alpha'_j, \alpha'_{j+1}) \in \mathrm{is}'(u \cdot r)$ with $i_j = i_{j+1} = r \in [1, k]$. Since e_1 is an edge of $D'(\sigma')$, $j \geq 1$. Since, clearly, the edge $e_j = (\langle \alpha'_{j-1}, i_{j-1} \rangle, \langle \alpha'_j, r \rangle)$ is in $D'(\sigma')$, it follows from the definition of the semantic rules of G' that $\alpha'_j = (\alpha, q_r)$ for some inherited attribute $\alpha \in A$. Hence, by Statements (1) and (2) for $u \cdot r$, $\theta_r = \text{nonroot}$, $q_r = p^r_0$, $\alpha'_{j+1} = (\beta, p^r_0)$ for some $\beta \in A$, and $(\alpha, \beta) \in p^r_0$. So $(\alpha, \beta) \in q_r$. Since $\pi(\alpha'_j) = \alpha$ and $\pi(\alpha'_{j+1}) = \beta$, this implies that $\pi(e_{j+1}) = (\langle \pi(\alpha'_j), r \rangle, \langle \pi(\alpha'_{j+1}), r \rangle)$ is in $D(\sigma)[q_1, \ldots, q_k]$. This proves the lemma. □

The proof of Statements (1) and (2) is now straightforward. Assume by induction that the statements hold for $u \cdot 1, \ldots, u \cdot k$. Recall that $\mathrm{is}'(u)$ is equal to the graph $\pi_0(D'(\sigma')[\mathrm{is}'(u \cdot 1), \ldots, \mathrm{is}'(u \cdot k)])$. Consider a path from $\langle \alpha', 0 \rangle$ to $\langle \beta', 0 \rangle$ in the graph $D'(\sigma')[\mathrm{is}'(u \cdot 1), \ldots, \mathrm{is}'(u \cdot k)]$. Such a path must start and end with an edge in $D'(\sigma')$. Hence, by the definition of the semantic rules of G', there exist $\alpha, \beta \in A$ such that either $\theta = \text{nonroot}$, $\alpha' = (\alpha, q_0)$, and $\beta' = (\beta, q_0)$, or $\theta = \text{root}$, $\alpha' = \alpha$, and $\beta' = \beta$. According to the above lemma, there is a path from $\langle \pi(\alpha'), 0 \rangle = \langle \alpha, 0 \rangle$ to $\langle \pi(\beta'), 0 \rangle = \langle \beta, 0 \rangle$ in $D(\sigma)[q_1, \ldots, q_k]$, and so $(\alpha, \beta) \in \pi_0(D(\sigma)[q_1, \ldots, q_k]) = q_0$.

Now the proof of Statement (3) is straightforward. Consider a cycle in the graph $D'(\sigma')[\mathrm{is}'(u \cdot 1), \ldots, \mathrm{is}'(u \cdot k)]$. It must contain an edge from $D'(\sigma')$, which we can take as the first edge. Since we know that Statements (1) and (2) hold for all nodes, the above lemma gives us a cycle in $D(\sigma)[q_1, \ldots, q_k]$.

Finally we prove Statement (4). It follows from the circularity of the graph $D'(\mathrm{root})[\mathrm{is}'(u)]$, the noncircularity of $D'(\mathrm{root})$, and the fact that the root rules of G' involve attributes from A only, that $\mathrm{is}'(u)$ must have an edge that involves attributes from A. By Statements (1) and (2) this implies that $\theta = \text{root}$ and

$\mathrm{is}'(u) \subseteq q_0$. Hence, since $D'(\mathrm{root}) = D(\mathrm{root})$, all edges of $D'(\mathrm{root})[\mathrm{is}'(u)]$ are also edges of $D(\mathrm{root})[q_0]$, and so $D(\mathrm{root})[q_0]$ is circular.

This shows that G' is noncircular, and ends the proof of the first part of Lemma 16.

We now turn to the proof of the second part of that lemma, but for simplicity we use G and G' instead of G' and G'', respectively. Since the idea of the construction is similar to that of the first part of the lemma, we will not give a detailed correctness proof.

Let $G = (\Sigma, S, I, \Omega, W, R, \alpha_{\mathrm{m}})$ be an arbitrary noncircular att from Σ to Δ. We will turn semantic rules into dummy rules whenever an attribute is used more than once. To see whether an attribute of a node u is used more than once, we do not only need the dependency graph $D(\sigma)$ of the label σ of u, but also the dependency graph of the label of the parent of u (or the dependency graph of the root, if u is the root). To provide this information is the job of the relabeling r_1. Let Θ be the set of all pairs (θ, i) such that either $\theta \in \Sigma$ and $i \in [1, \mathrm{rk}(\theta)]$, or $\theta = \mathrm{root}$ and $i = 0$. The attributed relabeling r_1 has output alphabet $\Sigma' = \Sigma \times \Theta$, and r_1 relabels a node u of a tree $t \in T_\Sigma$ that has label σ, with the label (σ, θ, i) such that (1) if $u = v \cdot j$ then $\theta = \mathrm{lab}_t(v)$ and $i = j$, and (2) if $u = \mathrm{root}(t)$ then $\theta = \mathrm{root}$ and $i = 0$. This can easily be realized by a relabeling attribute grammar that has inherited attributes 'par' and 'num' with root rules $\langle \mathrm{par}, 0\rangle = \mathrm{root}$ and $\langle \mathrm{num}, 0\rangle = 0$, and internal rules $\langle \mathrm{par}, i\rangle = \sigma$ and $\langle \mathrm{num}, i\rangle = i$, $i \in [1, \mathrm{rk}(\sigma)]$, in $R(\sigma)$.

The att G', from Σ' to Δ, has attributes $S' = S \times \Theta$ and $I' = I \times \Theta$, with $\alpha'_{\mathrm{m}} = (\alpha_{\mathrm{m}}, \mathrm{root}, 0)$.

The root rules of G' are the root rules of G in which every $\langle \alpha, 0\rangle$ is replaced by $\langle (\alpha, \mathrm{root}, 0), 0\rangle$, together with dummy rules for the remaining attributes. In fact, we assume that $D(\mathrm{root})$ does not contain a node with more than one outgoing edge (otherwise G is not SUR on any input tree and Lemma 16(2) is trivial). The internal rules of G' for the symbol (σ, θ, i) are defined as follows. If $D(\sigma)$ has a node with more than one outgoing edge, or a node $\langle \alpha, 0\rangle$ with one outgoing edge such that $\langle \alpha, i\rangle$ has at least one outgoing edge in $D(\theta)$, then all internal rules of (σ, θ, i) are dummy rules. Otherwise, $R'(\sigma, \theta, i)$ consists of all rules of $R(\sigma)$, in which every $\langle \alpha, 0\rangle$ is replaced by $\langle (\alpha, \theta, i), 0\rangle$ and every $\langle \alpha, j\rangle$, $j \neq 0$, by $\langle (\alpha, \sigma, j), j\rangle$, and dummy rules for the remaining attributes.

The idea in the construction of G' is the following. Let t' be a tree over Σ', and let t be the tree over Σ that is obtained from t' by changing every label (σ, θ, i) into σ. Let us first consider the case that $t' = r_1(t)$. Then the attribute $\langle \alpha, u\rangle$ of t in G is simulated in G' by the attribute $\langle (\alpha, \theta, i), u\rangle$ of t', where (σ, θ, i) is the label of u in $r_1(t)$. This guarantees that $\mathcal{G}'(r_1(t)) = \mathcal{G}(t)$ if G is SUR on t. If G is not SUR on t, then all "bad" nodes of $D(t)$ (i.e., nodes with more than one outgoing edge) are not bad any more in $D'(t')$ because we defined $D'(\sigma, \theta, i)$ to have no edges whenever $D(\sigma)$, combined with the parent dependency graph $D(\theta)$, has bad nodes. Now consider the case that $t' \neq r_1(t)$. Then a bad node of $D(t)$ is removed from $D'(t')$ either for the same reason as above, or because the labels of t' do not "fit". In fact, if a node u of t has label θ, and its i-th child

has a label (σ, θ', i') in $r_1(t)$ that does not "fit", i.e., with $(\theta', i') \neq (\theta, i)$, then a bad node $\langle \alpha, u \cdot i \rangle$ of $D(t)$ is not bad in $D'(t')$ because it is split into the two nodes $\langle (\alpha, \theta, i), u \cdot i \rangle$ and $\langle (\alpha, \theta', i'), u \cdot i \rangle$. Note finally that G' is still noncircular: a cycle in $D'(t')$ would produce a cycle in $D(t)$ when replacing every $\langle (\alpha, \theta, i), u \rangle$ by $\langle \alpha, u \rangle$.

This ends the proof of the second part of Lemma 16, and thus also ends the proof of Theorem 15.

9 Main results

In this final section we combine all previously proved results, and derive our main results: the equivalence of MSO term graph transducers and attributed tree transducers with look-ahead, and the equivalence of MSO tree transducers and attributed tree transducers with look-ahead that satisfy the single use restriction.

Theorem 18. MSO-TGT = ATT-REL $\circ$ ATT, MSO-TT = ATT-REL $\circ$ ATT_{sur}, $\text{MSO-TGT}_{\text{dir}}$ = ATT-REL $\circ$ ATT_{os}, *and* $\text{MSO-TT}_{\text{dir}}$ = ATT-REL $\circ$ $\text{ATT}_{\text{os,sur}}$.

Proof. The $\subseteq$-inclusions are an immediate consequence of Theorems 14 and 15.

Without ATT-REL, the $\supseteq$-inclusions are stated in Theorem 9. Then, the class ATT-REL can be added because of the equality ATT-REL = MSO-REL (Theorem 10) and the composition results of Proposition 2 (recalling that MSO-REL $\subseteq$ $\text{MSO-TT}_{\text{dir}}$). □

Together with Theorem 10 this characterizes all classes of tree transductions from Fig. 1 in terms of attribute grammars. In the remainder of the section we discuss a number of issues related to these results.

Look-ahead As observed in the introduction, an attributed tree transducer with look-ahead can also be viewed as one attribute grammar of which the attributes can be evaluated in two phases (see [Blo]). Such an attribute grammar is like an att, except that it also has "flags", i.e., attributes with finitely many values. The semantic rules for flags use flags only, and therefore the flags can be evaluated in a first phase. The "tree attributes", i.e., the attributes α with $W(\alpha) = T_\Delta$, have conditional semantic rules in which the flags can be tested, i.e., a semantic rule in $R(\sigma)$ is of the form

$$\langle \alpha, i \rangle = \begin{cases} r_1 & \text{if } c_1 \\ \vdots & \vdots \\ r_n & \text{if } c_n \end{cases} \tag{4}$$

where $c_1, \ldots, c_n$ are mutually exclusive, exhaustive tests on the flags, and $r_j \in T_\Delta((I \cup S) \times [0, \text{rk}(\sigma)])$ for every $j \in [1, n]$. The root rules are of a similar form. As an example, the tree transduction of Example 2(4, path) can be computed by such an attribute grammar, with a flag 'ns' which is defined as in Example 2(4,

path), and a tree attribute β which has the same semantic rules in $R(a)$ and $R(*)$ as in Example 2(4, path) and the following conditional semantic rule in $R(\sigma)$:

$$\langle\beta,0\rangle = \begin{cases} 1(\langle\beta,1\rangle) & \text{if } \langle \mathrm{ns},1\rangle = 1 \text{ and } \langle \mathrm{ns},2\rangle = 0, \\ 2(\langle\beta,2\rangle) & \text{if } \langle \mathrm{ns},1\rangle = 0 \text{ and } \langle \mathrm{ns},2\rangle = 1, \\ a & \text{otherwise.} \end{cases}$$

The reason that we have not chosen for this model is that it leads to more complicated definitions of noncircularity (cf. [Boy]) and the single use restriction. The usual notion of dependency graph takes a worst case view on dependencies in the sense that for a conditional semantic rule (4), it would assume $\langle\alpha,i\rangle$ to depend on all attributes occurring in $r_1,\ldots,r_n$, whereas, for a particular input tree, $\langle\alpha,i\rangle$ depends of course on the attributes of one r_j only.

Another, equivalent, way of viewing an $\mathrm{att}^{\mathrm{R}}$, based on Theorem 10, is as an att which has conditional rules as above, in which the conditions $c_1,\ldots,c_n$ are unary MSO formulas $\psi_1(x),\ldots,\psi_n(x)$, where x refers to the node under consideration (usually referred to by the number 0). The tree transduction of Example 2(4, path) can be computed by such an attribute grammar, with one attribute β which, again, has the same semantic rules $R(a)$ and $R(*)$ as in Example 2(4, path), but now has the following conditional rule in $R(\sigma)$:

$$\langle\beta,0\rangle = \begin{cases} 1(\langle\beta,1\rangle) & \text{if } \forall y((\mathrm{edg}_1(x,y) \to \nu_1(y)) \wedge (\mathrm{edg}_2(x,y) \to \nu_0(y))), \\ 2(\langle\beta,2\rangle) & \text{if } \forall y((\mathrm{edg}_1(x,y) \to \nu_0(y)) \wedge (\mathrm{edg}_2(x,y) \to \nu_1(y))), \\ a & \text{if } \neg\nu_1(x), \end{cases}$$

where $\nu_1(x)$ is a formula expressing that x has exactly one descendant with label $*$, and $\nu_0(x)$ expresses that x has no descendant with label $*$. This seems to be an attractive formal model, which was in fact used in the proof of Theorem 14, in disguise.

It is natural to extend the $\mathrm{att}^{\mathrm{R}}$ by allowing semantic conditions on the flags, which are specified for each operator (and the root) in addition to the semantic rules. Such an extended $\mathrm{att}^{\mathrm{R}}$ G computes a partial function: it accepts only those input trees t for which $\mathrm{dec}_{G,t}$ satisfies the semantic conditions. Using the characterization of unary MSO formulas proved in [BloEng, NevBus] (see Section 6.1) it is straightforward to show (see [Blo]) that the domains of these functions are exactly the MSO definable tree languages. Thus, the $\mathrm{att}^{\mathrm{R}}$ with semantic conditions computes precisely the partial MSO term transductions (of which the domain is specified by a closed MSO formula, see [EngOos, Cou3]), and similarly for the SUR and OS restrictions.

Time complexity It is known from [CouMos] that MSO definable graph transductions can be computed in polynomial time. As an immediate consequence of Theorem 18 we obtain that the MSO definable tree transductions can be computed in linear time, in the size of the input tree. In fact, it is well known that

attribute grammars can be evaluated in linear time, provided each semantic rule can be evaluated in constant time. This condition is obviously satisfied for finite-valued attribute grammars (and hence for attributed relabelings), and for SUR attributed tree transducers. If one is willing to accept a term graph as the representation of the output tree, then it also holds for arbitrary att's. The next corollary also follows from the results in Section 7 of [BloEng], where it is shown that both unary formulas and functional binary formulas can be evaluated on trees in linear time.

Corollary 19. MSO *definable tree transductions can be computed in linear time. Using a term graph to represent the output tree,* MSO *definable term transductions can be computed in linear time.*

Context-free graph grammars Let REGT denote the class of regular tree languages (see, e.g., [GécSte]). According to the classical result of [Don, ThaWri], this is also the class of MSO definable tree languages. Let us now consider the class MSO-TGT(REGT) of images of the MSO definable tree languages under MSO definable term transductions. As observed in the Introduction, it is shown in [Oos, Eng4, EngOos] that MSO-GT(REGT), the class of images of MSO definable tree languages under MSO definable *graph* transductions, is equal to the class of context-free graph languages. Thus MSO-TGT(REGT) = unfold(CF-TGL), the class of all term languages unfold(L) where L is a set of term graphs that can be generated by a context-free graph grammar. This class was investigated in [EngHey], where it was shown that it equals the class ATT(REGT) of images of the regular tree languages under attributed tree transductions. Since it is straighforward to prove that REGT is closed under attributed relabelings, this result is now an immediate consequence of Theorem 18.

We have, however, skipped some details. First, as also observed in a footnote in the Introduction, there are two different types of context-free graph languages: HR (hyperedge replacement) and NR (node replacement). The result of [Oos, Eng4, EngOos] is for NR, but the result of [EngHey] is for HR. However, there is another type of MSO definable graph transductions, closely related to the one defined here, in which the incidence relation between nodes and edges has to be defined by a binary formula, see [Cou4]. It is shown in [CouEng] that MSO-GT$'$(REGT) is the class of HR context-free graph languages, where the prime indicates this other type of MSO definable graph transductions. Now it is easy to see that for term graphs there is no difference between the two types of MSO definable graph transductions, i.e., Theorem 18 holds for both types: MSO-TGT$'$ = MSO-TGT and similarly for the other three classes. Consequently, the class unfold(CF-TGL) is the same for HR and NR. Second, the notion of term graph is defined in a slightly different way in [EngHey], and called "jungle". This does not make a difference, because jungles can be transformed into term graphs, and vice versa, by MSO definable graph transductions (of both types). Third, the attributed tree transducers defined in [EngHey] are not of the type defined here (as introduced in [Fül]), but are ordinary attribute grammars of which all attributes have trees as values (as considered, e.g., in [EngFil]). The domain of

such an attributed tree transducer is the set of all derivation trees of the underlying context-free grammar, and the result of [EngHey] concerns the class of all ranges of such attributed tree transducers. It is however straightforward to prove, using the close relationship between regular tree languages and derivation tree languages of context-free grammars (see, e.g., [GécSte]), that this class of ranges equals ATT(REGT).

Consider now the class MSO-TT(REGT) of images of the MSO definable tree languages under MSO definable tree transductions. This is the class of tree languages that can be generated by context-free graph grammars. Thus, by Theorem 18, it is equal to the class $\text{ATT}_{\text{sur}}(\text{REGT})$ of images of the regular tree languages under SUR attributed tree transductions. This result is in fact also clear from the proof of [EngHey].

Corollary 20. $\text{ATT}_{\text{sur}}(\text{REGT})$ *is the class of tree languages that can be generated by (HR or NR) context-free graph grammars.*

Tree transducers In tree language theory several types of tree transducers are studied, apart from the attributed tree transducer (see, e.g., [GécSte]). In what follows we consider total deterministic transducers only. We consider three types of such transducers.

First the top-down tree transducer, which is well known to be equivalent with the OS attributed tree transducer (see [CouFra, Fül]). To ensure closure under composition, the top-down tree transducer was extended with regular look-ahead in [Eng1]. Let T^{R} denote the class of all tree transductions that are computed by top-down tree transducers with regular look-ahead. It is not difficult to understand, and is proved in [EngMan], that preprocessing the input tree with an attributed relabeling has the same effect as regular look-ahead. This implies that $\text{ATT-REL} \circ \text{ATT}_{\text{os}} = \text{T}^{\text{R}}$. Thus, by Theorem 18, $\text{MSO-TGT}_{\text{dir}} = \text{T}^{\text{R}}$, i.e., the direction preserving MSO definable term transductions are exactly the top-down tree transductions with regular look-ahead. Since T^{R} is closed under composition (see [Eng1]), this implies that $\text{MSO-TGT}_{\text{dir}}$ is closed under composition (see Proposition 2 and the discussion following it).

Second the bottom-up tree transducer, which is incomparable with the top-down tree transducer. The result of [FülVag] shows that not every bottom-up tree transducer can be simulated by an att. However, since every bottom-up tree transducer can be simulated by a top-down tree transducer with regular look-ahead, the class of bottom-up tree transductions is included in $\text{MSO-TGT}_{\text{dir}}$.

Third the macro tree transducer, introduced in [CouFra, EngVog], which extends the top-down tree transducer with parameters (and thus can be viewed as a model of denotational semantics). Every attributed tree transduction can be computed by a macro tree transducer and thus, since every macro tree transducer with regular look-ahead can be simulated by one without, the same is true for attributed tree transductions with look-ahead. Hence, by Theorem 18, every MSO definable term transduction can be implemented by a macro tree transducer. Using results from [Eng2, EngVog], it can now be shown that the class of macro tree transductions is $\text{MSO-TGT}_{\text{dir}} \circ \text{MSO-TGT}$. A precise characterization

of the MSO definable tree transductions (MSO-TT) as a subclass of the macro tree transductions is presented in [EngMan].

Closure under composition When looking for a model for the implementation of the tree transductions in MSO-TT or $\text{MSO-TT}_{\text{dir}}$, a good guideline is that these classes are closed under composition, by Proposition 2. Thus, in [EngMan], the class $\text{MSO-TT}_{\text{dir}}$ is shown to be equal to a subclass of T^{R} which is well known to be closed under composition. When searching for our main result MSO-TT = ATT-REL $\circ$ ATT_{sur}, we were also guided by closure under composition. The main reason for the introduction of the single use restriction in [Gan, GanGie, Gie] was that the SUR attribute coupled grammars are closed under composition. An attribute coupled grammar is an attribute grammar in which a distinction is made between syntactic attributes (which have trees as values) and semantic attributes (which have arbitrary values). The single use restriction is imposed on the syntactic attributes only (which is why it is actually called the syntactic single use requirement in [Gie]). Taking syntactic attributes only, the result of [Gan, GanGie, Gie] shows that ATT_{sur} is closed under composition. Adding "flags", i.e., attributes with finitely many values, as semantic attributes, it shows that the class ATT-REL $\circ$ ATT_{sur} of SUR attributed tree transductions with look-ahead is closed under composition. It is observed in [Gie] that the same construction proves that the composition of a SUR attribute coupled grammar with an arbitrary attribute coupled grammar can again be realized by an attribute coupled grammar. This corresponds to the fact that MSO-TT $\circ$ MSO-TGT $\subseteq$ MSO-TGT, as stated in Proposition 2.

References

[AhoUll] A. V. Aho, J. D. Ullman; Translations on a context-free grammar, Inf. and Control 19 (1971), 439–475

[ArnLagSee] S. Arnborg, J. Lagergren, D. Seese; Easy problems for tree-decomposable graphs, J. of Algorithms 12 (1991), 308–340

[Blo] R. Bloem; *Attribute Grammars and Monadic Second Order Logic*, Master's Thesis, Leiden University, June 1996
`http://www.wi.LeidenUniv.nl/MScThesis/bloem.html`

[BloEng] R. Bloem, J. Engelfriet; Characterization of properties and relations defined in monadic second order logic on the nodes of trees, Tech. Report 97–03, Leiden University, August 1997
`http://www.wi.LeidenUniv.nl/TechRep/tr97-03.html`

[Boy] J. T. Boyland; Conditional attribute grammars, TOPLAS 18 (1996), 73–108

[Büc] J. Büchi; Weak second-order arithmetic and finite automata, Z. Math. Logik Grundlag. Math. 6 (1960), 66–92

[CorMon] A. Corradini, U.Montanari, eds., *Proc.* SEGRAGRA'95, *Joint* COMPUGRAPH/SEMAGRAPH *Workshop on Graph Rewriting and Computation*, Electronic Notes in Theoretical Computer Science, Vol. 2, Elsevier, 1995

[Cou1] B. Courcelle; The monadic second-order logic of graphs I: Recognizable sets of finite graphs, Inform. and Comput. 85 (1990), 12–75

[Cou2] B. Courcelle; Graph rewriting: an algebraic and logic approach, in *Handbook of Theoretical Computer Science*, Vol. B (J. van Leeuwen, ed.), Elsevier, 1990, 193–242

[Cou3] B. Courcelle; The monadic second-order logic of graphs V: On closing the gap between definability and recognizability, Theor. Comput. Sci. 80 (1991), 153–202

[Cou4] B. Courcelle; Monadic second-order definable graph transductions: a survey, Theor. Comput. Sci. 126 (1994), 53–75

[Cou5] B. Courcelle; The expression of graph properties and graph transformations in monadic second-order logic, Chapter 5 of *Handbook of Graph Grammars and Computing by Graph Transformation*, Vol. 1: *Foundations* (G. Rozenberg, ed.), World Scientific, 1997

[CouEng] B. Courcelle, J. Engelfriet; A logical characterization of the sets of hypergraphs defined by hyperedge replacement grammars, Math. Systems Theory 28 (1995), 515–552

[CouEngRoz] B. Courcelle, J. Engelfriet, G. Rozenberg; Handle-rewriting hypergraph grammars, J. of Comp. Syst. Sci. 46 (1993), 218–270

[CouFra] B. Courcelle, P. Franchi-Zannettacci; Attribute grammars and recursive program schemes I, II, Theor. Comput. Sci. 17 (1982), 163–191; 235–257

[CouMos] B. Courcelle, M. Mosbah; Monadic second-order evaluations on tree decomposable graphs, Theor. Comput. Sci. 109 (1993), 49–82

[DerJouLor] P. Deransart, M. Jourdan, B. Lorho; *Attribute Grammars*, Lecture Notes in Computer Science 323, Springer-Verlag, Berlin, 1988

[Don] J. Doner; Tree acceptors and some of their applications, J. of Comp. Syst. Sci. 4 (1970), 406–451

[DreHabKre] F. Drewes, A. Habel, H.-J. Kreowski; Hyperedge replacement graph grammars, Chapter 2 of *Handbook of Graph Grammars and Computing by Graph Transformation*, Vol. 1: *Foundations* (G. Rozenberg, ed.), World Scientific, 1997

[Elg] C. C. Elgot; Decision problems of finite automata and related arithmetics, Trans. Amer. Math. Soc. 98 (1961), 21–51

[Eng1] J. Engelfriet; Top-down tree transducers with regular look-ahead, Math. Systems Theory 10 (1977), 289–303

[Eng2] J. Engelfriet; Tree transducers and syntax-directed semantics, Memorandum nr. 363, Twente University of Technology, 1981, presented at CAAP'82

[Eng3] J. Engelfriet; Attribute grammars: attribute evaluation methods, in *Methods and Tools for Compiler Construction* (B. Lorho, ed.), Cambridge University Press, 1984, 103–138

[Eng4] J. Engelfriet; A characterization of context-free NCE graph languages by monadic second-order logic on trees, in *Graph Grammars and their Application to Computer Science* (H. Ehrig, H.-J. Kreowski, G. Rozenberg, eds.), Lecture Notes in Computer Science 532, Springer-Verlag, Berlin, 1991, 311–327

[Eng5] J. Engelfriet; A regular characterization of graph languages definable in monadic second-order logic, Theor. Comput. Sci. 88 (1991), 139–150

[Eng6] J. Engelfriet; Context-free graph grammars, Chapter 3 of *Handbook of Formal Languages*, Vol. 3: *Beyond Words* (G. Rozenberg, A. Salomaa,

eds.), Springer-Verlag, 1997

[EngFil] J. Engelfriet, G. Filè; The formal power of one-visit attribute grammars, Acta Informatica 16 (1981), 275–302

[EngHey] J. Engelfriet, L. M. Heyker; Context-free hypergraph grammars have the same term-generating power as attribute grammars, Acta Informatica 29 (1992), 161–210

[EngMan] J. Engelfriet, S. Maneth; Monadic second order logic and macro tree transductions, in preparation

[EngOos] J. Engelfriet, V. van Oostrom; Logical description of context-free graph languages, Tech. Report 96–22, Leiden University, August 1996, to appear in J. of Comp. Syst. Sci.

[EngRoz] J. Engelfriet, G. Rozenberg; Node replacement graph grammars, Chapter 1 of *Handbook of Graph Grammars and Computing by Graph Transformation*, Vol. 1: *Foundations* (G. Rozenberg, ed.), World Scientific, 1997

[EngRozSlu] J. Engelfriet, G. Rozenberg, G. Slutzki; Tree transducers, L systems, and two-way machines, J. of Comp. Syst. Sci. 20 (1980), 150–202

[EngVog] J. Engelfriet, H. Vogler; Macro tree transducers, J. of Comp. Syst. Sci. 31 (1985), 71–146

[FHVV] Z. Fülöp, F. Herrmann, S. Vágvölgyi, H. Vogler; Tree transducers with external functions, Theor. Comput. Sci. 108 (1993), 185–236

[Fül] Z. Fülöp; On attributed tree transducers, Acta Cybernetica 5 (1981), 261–279

[FülVag] Z. Fülöp, S. Vágvölgyi; Attributed tree transducers cannot induce all deterministic bottom-up tree transformations, Inform. and Comput. 116 (1995), 231–240

[Gan] H. Ganzinger; Increasing modularity and language-independency in automatically generated compilers, Sci. Comput. Programming 3 (1983), 223–278

[GanGie] H. Ganzinger, R. Giegerich; Attribute coupled grammars, in *Proc.* SIGPLAN'84, SIGPLAN Notices 19 (1984), 157–170

[GécSte] F. Gécseg, M. Steinby; *Tree automata*, Akadémiai Kiadó, Budapest, 1984

[Gie] R. Giegerich; Composition and evaluation of attribute coupled grammars, Acta Informatica 25 (1988), 355–423

[KamSlu] T. Kamimura, G. Slutzki; Parallel and two-way automata on directed ordered acyclic graphs, Inf. and Control 49 (1981), 10–51

[KlaSch] N. Klarlund, M. L. Schwartzbach; Graph Types, in *Proc. of the 20th Conference on Principles of Programming Languages*, 1993, 196–205

[Knu] D. E. Knuth; Semantics of context-free languages, Math. Syst. Theory 2 (1968), 127–145. Correction: Math. Syst. Theory 5 (1971), 95–96

[KühVog1] A. Kühnemann, H. Vogler; Synthesized and inherited functions, a new computational model for syntax-directed semantics, Acta Informatica 31 (1994), 431–477

[KühVog2] A. Kühnemann, H. Vogler; *Attributgrammatiken*, Vieweg, Braunschweig, 1997

[MezWri] J. Mezei, J. B. Wright; Algebraic automata and context-free sets, Inform. and Control 11 (1967), 3–29

[NevBus] F. Neven, J. Van den Bussche; On the expressive power of Boolean-valued attribute grammars, extended abstract, University of Limburg, Belgium, 1997

[Oos] V. van Oostrom; *Graph grammars and 2nd order logic* (in Dutch), Master's Thesis, Leiden University, January 1989

[SlePleEek] M. R. Sleep, M. J. Plasmeijer, M. C. J. D. van Eekelen, eds., *Term Graph Rewriting – Theory and Practice*, John Wiley, London, 1993

[ThaWri] J. W. Thatcher, J. B. Wright; Generalized finite automata theory with an application to a decision problem of second-order logic, Math. Systems Theory 2 (1968), 57–81

[Tho] W. Thomas; Automata on infinite objects, in *Handbook of Theoretical Computer Science*, Vol. B (J. van Leeuwen, ed.), Elsevier, 1990, 133–192

This article was processed using the LaTeX macro package with LLNCS style