Janis Voigtländer

Description of Properties of Attributed Tree Transducers by means of (Monadic) Second–Order Logic

Project in Computational Logic

20.10.1999 – 19.04.2000

Chair: Grundlagen der Programmierung

Supervisor: Prof. Dr.–Ing. habil. Heiko Vogler

Contents

1 Introduction

Functional programs often use definition of functions by means of structural recursion or tree–walking on an input structure. Tree transducers, which describe tree transformations over ranked alphabets, are formal models that can be used to study properties of such program schemes. In the literature, several types of tree transducers are studied such as the classes of top–down tree transducers (cf. [1, 2, 3, 4]), attributed tree transducers (cf. [5]), macro tree transducers (cf. [6, 7, 8]) and restricted subclasses of those (e.g., in [9, 10]).

As an abstraction from Knuth's attribute grammars (c.f. [11]), by taking trees over a ranked alphabet as input instead of derivation trees of a context free grammar, and by taking the free term algebra as semantic domain, one obtains the attributed tree transducers. In [12] it was shown that every tree transformation computable by an attributed tree transducer can also be computed by a macro tree transducer. Several properties of attributed tree transducers have been studied in the literature, and there are deep results comparing the obtained subclasses of attributed tree transducers with other restricted classes of tree transducers. In [10], the properties of attributed tree transducers to be *nonnested*, *one–visit* and *absolutely noncircular*, respectively, were studied and it was shown that they correspond to certain properties of macro tree transducers. This systematic comparison was continued in [13] and [14]. In [14], important properties known from attribute grammars are transfered to attributed tree transducers. There, the properties of attributed tree transducers to be *simple κ–visit* (*–sweep*, *–alternating pass*, *–pass*) and *simple multi–visit* (*–sweep*, *–alternating pass*, *–pass*), respectively, are defined and studied. Another property, namely the *single–use* property (cf. attribute coupled grammars in [15]), was defined for attributed tree transducers in [9] and the obtained subclass of tree transformations was studied with respect to composition and decomposition results.

In the literature, all the above mentioned properties are defined in an algebraic way or by giving a decision algorithm. In order to unify reasoning about these properties, it would be desirable to define and to present them in a unified way. Basically, these properties can be described as properties of certain graphs that are obtained from the specification of an attributed tree transducer, such as dependency and partition graphs. In [17], it was studied how monadic second–order logic can be used to express graph properties. The aim of the present paper is to describe properties of attributed tree transducers in terms of monadic second–order logic. We will give an appropriate representation for attributed tree transducers by relational structures over a particular signature we will develop. All the above mentioned properties will be characterized by logical formulas over this signature, in the sense that an attributed tree transducer has a certain property iff its representing relational structure satisfies the corresponding formula. Since not all properties are expressible in monadic second–order logic, we will also consider full second–order logic.

This paper is organized as follows. In Section 2 we define necessary notations and the basic concepts of attributed tree transducers and second–order logic. In Section 3 we show how the logical framework can be used to represent attributed tree transducers. Further, the chosen representation will be justified. In Section 4 we describe several properties of attributed tree transducers by logical formulas. The properties will be defined in an algebraic way as taken from the literature. We will then show how these properties can be formalized by second–order formulas (or formulas of a weaker logic than second–order) over our representation. In Section 5 implementational aspects are discussed.

2 Preliminaries

We denote by $\mathbb{N}$ the set of natural numbers including 0 and by $[n]$ the set $\{1, \dots, n\}$. For a set S of natural numbers we denote by $max\ S$ the maximum of all its elements.
For a set S, we denote by S^* the set of finite sequences of elements of S, where ϵ will denote the empty sequence. We set $S^+ = S^* \backslash \{\epsilon\}$. $\mathcal{P}(S)$ will denote the power set of a set S. $|S|$ will denote the number of elements of a finite set S. A tuple $P = \langle S_1, \dots, S_n \rangle$ is called an *ordered n–partition* of a set S, if $S = \bigcup_{1 \le i \le n} S_i$ and for all $i \neq j$, $S_i \cap S_j = \emptyset$. For an element $s \in S$, $ind_P(s)$ denotes the unique number i for which $s \in S_i$.
For a binary relation R, we denote by R^+ and R^* its transitive closure and its reflexive, transitive closure, respectively.
A *ranked alphabet* is a pair $(\Sigma, rank_\Sigma)$, where Σ is a finite set of symbols and $rank_\Sigma$ assigns to each symbol a natural number, its *rank*. For every $n \geq 0$, we define $\Sigma^{(n)} = \{\sigma \in \Sigma \mid rank_\Sigma(\sigma) = n\}$. The rank n of a symbol σ will also be denoted by writing $\sigma^{(n)}$. A ranked alphabet Σ is called *unary* if $\Sigma = \Sigma^{(1)}$. For a unary ranked alphabet Σ and a set A, we denote by $\Sigma(A)$ the set $\{\sigma(a) \mid \sigma \in \Sigma,\, a \in A\}$.
By $rki(\Sigma) = max\{n \mid \Sigma^{(n)} \neq \emptyset\}$ we denote the maximal rank of a ranked alphabet.
For a ranked alphabet Σ and a set A disjoint with Σ we define the set $T_\Sigma(A)$ of *trees over* Σ *indexed by* A as the smallest set T such that (i) $A \in T$ and (ii) whenever $\sigma \in \Sigma^{(n)}$ and $t_1, \dots, t_n \in T$, also $\sigma(t_1, \dots, t_n) \in T$. By T_Σ we denote the set $T_\Sigma(\emptyset)$.
We will need the *set of occurrences* of a tree, which we obtain with the function $occ : T_\Sigma(A) \to \mathcal{P}(\mathbb{N}^*)$ that is defined by structural induction as follows: (i) if $t \in \Sigma^{(0)} \cup A$, then $occ(t) = \{\epsilon\}$ and (ii) if $t = \sigma(t_1, \dots, t_n)$, then $occ(t) = \{\epsilon\} \cup \{i\nu \mid 1 \le i \le n,\, \nu \in occ(t_i)\}$.
We will also need the *label at an occurrence of a tree*, given by the mapping $lab : \{(t, \omega) \mid t \in T_\Sigma(A),\, \omega \in occ(t)\} \to \Sigma \cup A$ defined by: (i) if $t \in \Sigma^{(0)} \cup A$, then $lab(t, \epsilon) = t$ and (ii) if $t = \sigma(t_1, \dots, t_n)$ then $lab(t, \epsilon) = \sigma$ and $lab(t, i\nu) = lab(t_i, \nu)$.

2.1 Attributed Tree Transducers

For attributed tree transducers we will follow the notations and definitions used in [10], with only slight changes such as demanding input and output ranked alphabets to be disjoint. In order to obtain a readable definition of attributed tree transducers, we first define the set of possible right–hand sides of rules.

Definition 2.1 Let Att_{syn} and Att_{inh} be disjoint unary ranked alphabets, let Δ be a ranked alphabet with $\Delta \cap (Att_{syn} \cup Att_{inh}) = \emptyset$ and let $k \in \mathbb{N}$.
The set $RHS(Att_{syn}, Att_{inh}, \Delta, k)$ of *right–hand sides over* Att_{syn}*,* Att_{inh}*,* Δ *and* k is the smallest subset RHS of $T_\Delta((Att_{syn} \cup Att_{inh})(\{\pi 0, \pi 1, \dots, \pi k\}))$ such that:

- for every $a \in Att_{syn}$ and $1 \le i \le k$ we have $a(\pi i) \in RHS$
- for every $b \in Att_{inh}$ we have $b(\pi 0) \in RHS$
- for every $\delta \in \Delta^{(n)}$ and $\theta_1, \dots, \theta_n \in RHS$ we have $\delta(\theta_1, \dots, \theta_n) \in RHS$ □

The symbol π is called an occurrence variable and plays a role when defining the semantics of attributed tree transducers by a derivation relation. Using this set of right–hand sides, we can now define attributed tree transducers.

Definition 2.2 An *attributed tree transducer* is a tuple $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ such that the following conditions hold:

1. Att is a unary ranked alphabet of *attributes* that is partitioned into the sets Att_{syn} and Att_{inh}, whose elements are called *synthesized* and *inherited* attributes, respectively.

2. Σ and Δ are disjoint ranked alphabets of *input* and *output* symbols, respectively, where $Att \cap (\Sigma \cup \Delta) = \emptyset$.

3. $a_0 \in Att_{syn}$ is a designated synthesized attribute, called the *initial* attribute.

4. R, the *set of rules*, is partitioned into sets R_σ for all $\sigma \in \Sigma$ such that:

   (a) for every $a \in Att_{syn}$ and $\sigma \in \Sigma^{(k)}$, the set R_σ contains exactly one rule of the form
   $$a(\pi 0) \to \theta$$
   where $\theta \in RHS(Att_{syn}, Att_{inh}, \Delta, k)$

   (b) for every $b \in Att_{inh}$, $\sigma \in \Sigma^{(k)}$ and $1 \leq i \leq k$, the set R_σ contains exactly one rule of the form
   $$b(\pi i) \to \theta$$
   where $\theta \in RHS(Att_{syn}, Att_{inh}, \Delta, k)$.

5. $E : Att_{inh} \to T_\Delta$ is a mapping, called the *environment*. □

Note that these syntactic restrictions are all we require from an attributed tree transducer. Especially, we do not require attributed tree transducers to be noncircular.
We now give an example for an attributed tree transducer which will be used to illustrate further definitions and later our representation of attributed tree transducers in the logical framework.

Example 2.3 Define $ATT_{leafs} = (Att, \Sigma, \Delta, a_0, R, E)$ as follows:

- $Att = Att_{syn} \cup Att_{inh}$, where $Att_{syn} = \{a\}$ and $Att_{inh} = \{b\}$

- $\Sigma = \Sigma^{(2)} \cup \Sigma^{(0)}$, where $\Sigma^{(2)} = \{\sigma\}$ and $\Sigma^{(0)} = \{\alpha, \beta\}$;
  $\Delta = \Delta^{(1)} \cup \Delta^{(0)}$, where $\Delta^{(1)} = \{\alpha', \beta'\}$ and $\Delta^{(0)} = \{\omega\}$

- $a_0 = a$

- $R = R_\sigma \cup R_\alpha \cup R_\beta$, where
  $$\begin{aligned} R_\sigma = \{\ & a(\pi 0) \to a(\pi 2), \\ & b(\pi 1) \to b(\pi 0), \\ & b(\pi 2) \to a(\pi 1)\ \} \\ R_\alpha = \{\ & a(\pi 0) \to \alpha'(b(\pi 0))\ \} \\ R_\beta = \{\ & a(\pi 0) \to \beta'(b(\pi 0))\ \} \end{aligned}$$

- $E(b) = \omega$ □

Note that we have used α' and β' in the output ranked alphabet to fulfill the restriction of Σ and Δ to be disjoint. Although we will not define formally the derivation relation that would allow us to compute the tree transformation induced by an attributed tree transducer, it should be intuitively clear that ATT_{leafs} computes the list of leafs of a binary tree from right to left. For the formal definition of the induced tree transformation, consult [10].

Now we define some notions that we will later use to access certain characteristics of an attributed tree transducer $ATT = (Att, \Sigma, \Delta, a_0, R, E)$:

- for every $\sigma \in \Sigma^{(n)}$, the set $inner(\sigma)$ of *inside attribute occurrences at* σ is defined by $\{a(\pi 0) \mid a \in Att_{syn}\} \cup \{b(\pi j) \mid b \in Att_{inh},\ 1 \leq j \leq n\}$
- for every $\sigma \in \Sigma^{(n)}$, the set $outer(\sigma)$ of *outside attribute occurrences at* σ is defined by $\{b(\pi 0) \mid b \in Att_{inh}\} \cup \{a(\pi j) \mid a \in Att_{syn},\ 1 \leq j \leq n\}$
- for every $\sigma \in \Sigma^{(n)}$ and every inside attribute occurrence $c(\pi\eta)$ at σ with $\eta \in \{0\} \cup [n]$, we denote by $rhs(c(\pi\eta), \sigma)$ the right–hand side of the rule $c(\pi\nu) \to \theta$ in R_σ.

Note that the inside attribute occurrences are exactly the synthesized attributes at the root and the inherited attributes at the child nodes, while the outside attribute occurrences are the inherited attributes at the root and the synthesized attributes at the child nodes. Further note that for every inside attribute occurrence at a symbol σ, there is exactly one rule in R_σ with this attribute occurrence as left–hand side, and thus the function rhs is well–defined. Only outside attribute occurrences at σ appear in right–hand sides of rules in R_σ.

Example 2.4 For ATT_{leafs} we have:

$$\begin{aligned} inner(\sigma) &= \{a(\pi 0),\ b(\pi 1),\ b(\pi 2)\} \\ outer(\alpha) &= \{b(\pi 0)\} \\ rhs(a(\pi 0), \beta) &= \beta'(b(\pi 0)) \end{aligned}$$

□

Further, we define the concepts of the dependency graph of an input symbol with respect to ATT, and is–graphs together with operations on them.

Definition 2.5 Let $\sigma \in \Sigma^{(n)}$. The *dependency graph of* σ *with respect to* ATT, denoted by $D_{ATT}(\sigma)$, is a directed graph, having as nodes the elements of the set $inner(\sigma) \cup outer(\sigma)$, and an edge from $c(\pi\eta) \in outer(\sigma)$ to $c'(\pi\eta') \in inner(\sigma)$ iff $c(\pi\eta)$ appears in $rhs(c'(\pi\eta'), \sigma)$. In that case we say that $c'(\pi\eta')$ *depends directly* on $c(\pi\eta)$ at σ. □

Note that the direction of edges in the dependency graph is reversed with respect to the direction of arrows in the corresponding rules. We will illustrate dependency graphs as shown in Figure 1. To the left of the σ– or every child–node in the σ–pattern, we draw the inherited attribute occurrences and to the right the synthesized ones. The inside attribute occurrences are drawn inside the trapezoid of the pattern, while the outside attribute occurrences are drawn outside. The dependency graph of an input tree is obtained by putting the dependency graphs of the symbols occurring in the tree together. A computation of the attributed tree transformation for an input tree is only possible, if its dependency graph does not contain a cycle.

Definition 2.6 An *is–graph* over Att_{syn} and Att_{inh} is a directed graph that has as nodes the elements of the set $Att_{syn} \cup Att_{inh}$ and only edges from elements of Att_{inh} to elements of Att_{syn}. □

We will view is–graphs as subsets of $Att_{inh} \times Att_{syn}$ and thus reason about is–graphs just as about sets, using intersection, inclusion etc. Further, we have two operations on is–graphs, namely the composition of a dependency graph with is–graphs and building the is–graph of such a composition.

Definition 2.7 Let $\sigma \in \Sigma^{(n)}$, let $D_{ATT}(\sigma)$ be the dependency graph of σ for ATT and let $is_1, \ldots, is_n$ be is–graphs over Att_{syn} and Att_{inh}. We define:

- the *composition of* $D_{ATT}(\sigma)$ *and* $is_1, \ldots, is_n$, denoted by $D_{ATT}(\sigma)[is_1, \ldots, is_n]$, is the directed graph obtained from $D_{ATT}(\sigma)$ by adding for every $k \in [n]$ an edge from $b(\pi k)$ to $a(\pi k)$ whenever there is an edge from b to a in is_k.
- the *is–graph of* $D_{ATT}(\sigma)[is_1, \ldots, is_n]$, denoted by $is(D_{ATT}(\sigma)[is_1, \ldots, is_n])$, is the is–graph, in which there is an edge from $b \in Att_{inh}$ to $a \in Att_{syn}$ iff there is a path from $b(\pi 0)$ to $a(\pi 0)$ in $D_{ATT}(\sigma)[is_1, \ldots, is_n]$. □

Example 2.8 We show the dependency graph of σ for ATT_{leafs} and its composition with is_1, is_2, where $is_1 = is_2$ is the is–graph containing an edge from b to a.

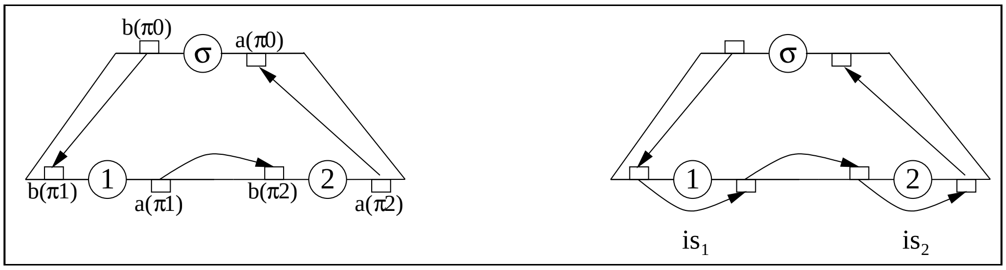

Figure 1: the dependency graph $D_{ATT_{leafs}}(\sigma)$

The is–graph $is(D_{ATT_{leafs}}(\sigma)[is_1, is_2])$ is again is_1. □

2.2 Second–Order Logic

Our notions and notations in the logical framework are based on [17], where logical formulas are used to describe properties of graphs. In order to obtain more readable formulas, we use sorted signatures and thus get typed formulas. It is well known (for example from [20]) that this does not influence the expressive power of our logic.

A signature $SIG = (S, R, \alpha)$ is formed by a finite set S of sorts and a finite set R of relation symbols, together with a *type* mapping $\alpha : R \to S^+$. To show the type of relations, we will use the notation $r : \alpha(r)$.

A SIG–structure is an object $T = \langle (T_s)_{s\in S}, (r_T)_{r\in R}\rangle$, where for every s in S, T_s is a finite, nonempty set, called the *domain of sort* s of T, and for every $r \in R$, r_T is a relation: $r_T \subseteq T_{\alpha(r)}$ (for a nonempty sequence μ of sorts $s_1 \dots s_n$, we let $T_\mu := T_{s_1} \times \cdots \times T_{s_n}$). We assume that $T_s \cap T_{s'} = \emptyset$ for $s \neq s'$. The set of all SIG–structures will be denoted by $STR(SIG)$.

Let Var be a countable set of variables, where lowercase variables such as $x, x_1, x_2, y, \dots$ will be used as *individual variables* and uppercase variables such as $X, Y, R, \dots$ as *relation variables*, respectively. Given a set S of sorts, to every individual variable x, a sort $s \in S$ is associated, denoted by x^s; and to every relation variable R, a sequence $\mu \in S^+$ is associated, denoted by R^μ. The set of these typed variables is denoted by Var^S.

The *atomic* formulas over SIG are: $x^s = y^s$, $r({x_1}^{s_1}, \dots, {x_n}^{s_n})$, $X^\mu({x_1}^{s_1}, \dots, {x_n}^{s_n})$, where $s, s_1, \dots, s_n \in S, x^s, y^s, {x_1}^{s_1}, \dots, {x_n}^{s_n}, X^\mu \in Var^S$, $r \in R$ and $\mu = \alpha(r) = s_1 \dots s_n$. The *second–order* formulas (over SIG) are constructed from the atomic formulas with the propositional connectives $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$, and the quantifications $\exists x^s, \forall x^s, \exists X^\mu, \forall X^\mu$ over individual and relation variables in the usual way, where parentheses may be omitted using the following precedence hierarchy: $\neg > \wedge > \vee > \Rightarrow > \Leftrightarrow$.
By restricting relation variables to be *unary*, i.e., having a type sequence of length one, we get *monadic second–order* formulas. Since a unary relation is nothing but a set, these relation variables will be called set variables. Instead of $X^s(x^s)$ we will then just write $x^s \in X^s$.
By forbidding the use of relation variables and not allowing $x^s = y^s$ as atomic formula, we get *first–order* formulas. The sets of all second–order formulas (monadic second–order formulas, and first–order formulas) are denoted by **SO**, **MSO**, and **FO**, respectively.

We will also use some abbreviations to make formulas more readable. $x^s \neq y^s$ will stand for $\neg(x^s = y^s)$ and $\phi \not\Leftrightarrow \psi$ for $\neg(\phi \Leftrightarrow \psi)$. If ϕ is a formula with no occurrence of y^s, we will write $\exists! x^s.(\phi)$ for $\exists x^s.((\phi) \wedge \forall y^s.((\phi[x^s \leftarrow y^s]) \Rightarrow y^s = x^s))$, meaning "exists exactly one" (see below for substitution in formulas).

If rel is a formula defining a binary relation on elements of the same sort s, we will use

$$\begin{aligned} rel^+(x^s, y^s) &= \forall X^s.(\forall u^s, v^s.(u^s \in X^s \wedge rel(u^s, v^s) \Rightarrow v^s \in X^s) \\ &\qquad \wedge \forall u^s.(rel(x^s, u^s) \Rightarrow u^s \in X^s) \Rightarrow y^s \in X^s) \\ rel^*(x^s, y^s) &= (x^s = y^s) \vee rel^+(x^s, y^s) \end{aligned}$$

for the transitive and the reflexive, transitive closure of rel, respectively (coming from [17]). Note that rel^+ is a monadic second–order formula if rel is one.

When defining formulas, we will characterize to which type of logic they belong, by writing **SO** for second–order, **MSO** for monadic second–order and **FO** for first–order formulas. We will also use **TC** when the above mentioned construction of the transitive closure of a binary formula is used and **EQ** for indicating the occurrence of an atomic formula of the form $x^s = y^s$. By, e.g., **FO+TC+EQ** we will then characterize the set of all formulas that do not make use of relation variables, but may contain the transitive closure of a formula and make use of equality as atomic formula. Note that **FO+TC+EQ** $\subseteq$ **MSO** in the obvious sense.

In order to indicate free variables of a formula explicitly, we will write $\phi({x_1}^{s_1}, \dots, {x_n}^{s_n}, {X_1}^{\mu_1}, \dots, {X_m}^{\mu_m})$ in the definition of a formula ϕ, if only the variables in $\{{x_1}^{s_1}, \dots, {x_n}^{s_n}, {X_1}^{\mu_1}, \dots, {X_m}^{\mu_m}\}$ may occur free in ϕ (note that not all of them have to). It is a well–known concept to rename bound variables, so we can make use of that in defining substitution in formulas.

Let ϕ be a formula, let $v_1, \dots, v_n$ be pairwise distinct variables and let $u_1, \dots, u_n$ be variables of corresponding kinds (individual or relational, same sort or type, respectively). By $\phi[v_1 \leftarrow u_1, \dots, v_n \leftarrow u_n]$ we denote the formula obtained by renaming all bound variables in ϕ, such that no variable in $\{u_1, \dots, u_n\}$ occurs bound, and then substituting u_i for every occurrence of v_i, $i \in [n]$. If a formula ϕ is of the form $\phi({x_1}^{s_1}, \dots, {x_n}^{s_n}, {X_1}^{\mu_1}, \dots, {X_m}^{\mu_m})$, then we will write $\phi({y_1}^{s_1}, \dots, {y_n}^{s_n}, {Y_1}^{\mu_1}, \dots, {Y_m}^{\mu_m})$ for $\phi[{x_1}^{s_1} \leftarrow {y_1}^{s_1}, \dots, {x_n}^{s_n} \leftarrow {y_n}^{s_n}, {X_1}^{\mu_1} \leftarrow {Y_1}^{\mu_1}, \dots, {X_m}^{\mu_m} \leftarrow {Y_m}^{\mu_m}]$.

2.2.1 Semantics of second–order logic

To define the semantics of second–order formulas over a signature $SIG = (S, R, \alpha)$, we introduce the notion of a state. Let T be a SIG–structure, then a *state* over T is a mapping $\sigma : Var^S \to \bigcup_{s \in S} T_s \cup \bigcup_{\mu \in S^+} \mathcal{P}(T_\mu)$ such that for every $x^s \in Var^S$, we have $\sigma(x^s) \in T_s$, and for every $R^\mu \in Var^S$, we have $\sigma(R^\mu) \subseteq T_\mu$.
Let $x^s, R^\mu \in Var^S$, $d \in T_s$, $D \subseteq T_\mu$ and let σ be a state, then by $\sigma\{x^s \to d\}$ and $\sigma\{R^\mu \to D\}$ we denote the states obtained from σ by assigning d to x^s and D to R^μ, respectively, while leaving all other assignments unchanged.

Definition 2.9 Given a SIG–structure T, we define for every state σ and every formula ϕ the relation $T \models_\sigma \phi$ (in words: state σ satisfies ϕ in T) as follows:

$$
\begin{array}{ll}
T \models_\sigma x^s = y^s & \text{iff } \sigma(x^s) = \sigma(y^s), \\
T \models_\sigma r({x_1}^{s_1}, \dots, {x_n}^{s_n}) & \text{iff } (\sigma({x_1}^{s_1}), \dots, \sigma({x_n}^{s_n})) \in r_T, \\
T \models_\sigma X^\mu({x_1}^{s_1}, \dots, {x_n}^{s_n}) & \text{iff } (\sigma({x_1}^{s_1}), \dots, \sigma({x_n}^{s_n})) \in \sigma(X^\mu), \\
T \models_\sigma \phi \wedge \psi & \text{iff } T \models_\sigma \phi \text{ and } T \models_\sigma \psi, \\
T \models_\sigma \phi \vee \psi & \text{iff } T \models_\sigma \phi \text{ or } T \models_\sigma \psi, \\
T \models_\sigma \neg\phi & \text{iff } T \not\models_\sigma \phi, \\
T \models_\sigma \phi \Rightarrow \psi & \text{iff } T \not\models_\sigma \phi \text{ or } T \models_\sigma \psi, \\
T \models_\sigma \phi \Leftrightarrow \psi & \text{iff } T \models_\sigma \phi \Rightarrow \psi \text{ and } T \models_\sigma \phi \Rightarrow \psi, \\
T \models_\sigma \exists x^s.(\phi) & \text{iff there exists } d \in T_s \text{ such that } T \models_{\sigma\{x^s \to d\}} \phi, \\
T \models_\sigma \forall x^s.(\phi) & \text{iff for every } d \in T_s, \text{ we find } T \models_{\sigma\{x^s \to d\}} \phi, \\
T \models_\sigma \exists X^\mu.(\phi) & \text{iff there exists } D \subseteq T_\mu \text{ such that } T \models_{\sigma\{X^\mu \to D\}} \phi, \\
T \models_\sigma \forall X^\mu.(\phi) & \text{iff for every } D \subseteq T_\mu, \text{ we find } T \models_{\sigma\{X^\mu \to D\}} \phi
\end{array}
$$
□

A SIG–structure T is called a *model* for ϕ (T satisfies ϕ), in symbols $T \models \phi$, iff for every state σ, we find $T \models_\sigma \phi$. Note that if ϕ contains no free variables, then for every structure T, we find $T \models \phi$ iff $T \models_\sigma \phi$, where σ is any state.
For formulas that do contain free variables, we will need the notion of a structure together with determined elements of and relations over its domains, respectively (each one of corresponding sort or type for every free variable), to satisfy a formula with its free variables given the values of the determined elements and relations. Formally, if the formula with

free variables has the form $\phi(x_1{}^{s_1}, \dots, x_n{}^{s_n}, X_1{}^{\mu_1}, \dots, X_m{}^{\mu_m})$ and there is a structure T and $d_1 \in T_{s_1}, \dots, d_n \in T_{s_n}, D_1 \subseteq T_{\mu_1}, \dots, D_m \subseteq T_{\mu_m}$, we write $(T, d_1, \dots, d_n, D_1, \dots, D_m) \models \phi$ iff $T \models_{\sigma\{x_1{}^{s_1} \to d_1\} \cdots \{x_n{}^{s_n} \to d_n\}\{X_1{}^{\mu_1} \to D_1\} \cdots \{X_m{}^{\mu_m} \to D_m\}} \phi$, where σ is any state.

Note that, since we are only dealing with finite domains, for every SIG–structure we can effectively decide (by following the above definitions), whether it is a model for any given formula. However, because of its complexity, this decision procedure poses problems that will be discussed in Section 5.

Example 2.10 For a ranked alphabet Σ, we define the signature $SIG_\Sigma = (\{n\}, \{lab_\sigma \mid \sigma \in \Sigma\} \cup \{edg_i \mid i \in [rki(\Sigma)]\}, \alpha)$ with the sort n of nodes, where α maps all the lab–relations to type n and all the edg–relations to nn. Every tree t over the ranked alphabet Σ can be represented as a structure over the signature SIG_Σ, by taking for every occurrence in $occ(t)$ exactly one corresponding node into the domain of sort n, and defining the lab– and edg–relations in the obvious way.

Thus, with $\Sigma = \{\sigma^{(2)}, \alpha^{(0)}\}$, the tree $\sigma(\sigma(\alpha, \alpha), \alpha)$, for example, corresponds to a structure $T \in STR(SIG_\Sigma)$ as follows:

$$\begin{array}{ll} T_n & = \{v_\epsilon, v_1, v_2, v_{11}, v_{12}\} \\ lab_{\sigma T} & = \{v_\epsilon, v_1\} \\ lab_{\alpha T} & = \{v_2, v_{11}, v_{12}\} \\ edg_{1T} & = \{(v_\epsilon, v_1), (v_1, v_{11})\} \\ edg_{2T} & = \{(v_\epsilon, v_2), (v_1, v_{12})\} \end{array}$$

On structures of this type, we can now interpret monadic second–order formulas over SIG_Σ such as:

$$\begin{array}{l} \exists X, Y.(\forall x.(\neg \exists y.(edg_1(y, x) \vee edg_2(y, x)) \Rightarrow x \in X) \\ \quad \wedge \forall x.(x \in X \not\Leftrightarrow x \in Y) \\ \quad \wedge \forall x, y.(edg_1(x, y) \vee edg_2(x, y) \Rightarrow (x \in X \Leftrightarrow y \in Y)) \\ \quad \wedge \forall x.(lab_\alpha(x) \Rightarrow x \in X)), \end{array}$$

intuitively describing that every leaf of a tree has an even distance from the tree´s root. Since our signature has only the one sort n of nodes and only unary relation variables are used, we may omit superscripts to the variables in this case. □

3 Representing attributed tree transducers by structures

In order to describe properties of attributed tree transducers by logical formulas, we need a representation of attributed tree transducers as structures over a suitable signature. That is, we should find a suitable collection of sorts and relation symbols over these sorts, together with a way to encode an attributed tree transducer $(Att, \Sigma, \Delta, a_0, R, E)$ into a structure over this signature. Of course, there are arbitrarily many possibilities to choose such a signature. However, there are some properties we would reasonably expect from a suitable representation.
For example, we want a representation which is "simple" in the sense that it does not formalize things that have nothing to do with and are not needed in the context of attributed tree transducers. Also, the signature should be chosen such that the encoding from attributed tree transducers to structures is "immediate" and does not need complicated calculations. On the other hand, we have to ensure that all information contained in the description of an attributed tree transducer is still available in its representation as a structure over the chosen signature, because otherwise we would be doomed from the beginning to fail in expressing certain properties. To guarantee this, we should have a kind of decoding that gives us from a structure over the chosen signature again a tuple of the form $(Att, \Sigma, \Delta, a_0, R, E)$, which corresponds to the original attributed tree transducer in that it has all the same properties. Of course, in our representation we should try to follow the structure of a given attributed tree transducer tuple as far as possible, because that will simplify the task of transferring algebraic definitions of properties of attributed tree transducers to logical formulas.
Another question of interest will be, whether all structures over our signature can be seen as attributed tree transducers. That will not be the case, but we will try to distinguish by a logical formula the structures that do correspond to an attributed tree transducer and define our decoding for exactly these structures.

In the Example 2.10 at the end of the previous section we already represented trees over a fixed ranked alphabet by structures using a sort n of nodes together with *edg*– and *lab*–relations. Since for attributed tree transducers we will also need to formalize trees, namely the right–hand sides of rules and the environment, we will follow this natural approach. However, as the ranked alphabets over which these trees are built, are not fixed but dependent on the particular attributed tree transducer to be represented, we will have more general *edg*– and *lab*–relations with additional arguments. For the same reason, besides the sort of nodes we will also have sorts for attributes and symbols, representing the particular alphabets *Att*, Σ and Δ of an attributed tree transducer. In order to distinguish inherited from synthesized attributes, and input from output symbols, respectively, as well as for distinguishing the initial attribute, we will use unary relation symbols. For several purposes, such as expressing the ranks of symbols, distinguishing the children of a node in a tree as well as the occurrences of attributes, we need a concept of natural numbers. Thus, we will have another sort for integer values, together with an ordering successor relation. The trees that form the right–hand sides of rules and the environment, respectively, will be distinguished by their root nodes in two relations $rule$ and env.

Formally, we propose the following signature SIG_{ATT} for representing attributed tree transducers.

Definition 3.1 SIG_{ATT} has four sorts $\{a, s, i, n\}$ for attributes, symbols, integers and nodes, respectively. The signature has 12 relations with the following types and "intuitive meanings":

inh	$: a$	$inh(x^a)$ means: x^a is an inherited attribute
syn	$: a$	$syn(x^a)$ means: x^a is a synthesized attribute
in	$: s$	$in(x^s)$ means: x^s is an input symbol
out	$: s$	$out(x^s)$ means: x^s is an output symbol
rk	$: si$	$rk(x^s, y^i)$ means: y^i denotes the rank of x^s
suc	$: ii$	$suc(x^i, y^i)$ means: y^i is the successor of x^i
$initial$	$: a$	$initial(x^a)$ means: x^a is the initial attribute
$rule$	$: sain$	$rule(x^s, y^a, z^i, u^n)$ means: u^n is the root node of the right–hand side of the rule for y^a at the z^ith child of x^s
lab_{att}	$: nai$	$lab(x^n, y^a, z^i)$ means: x^n is labeled by the attribute occurrence $y^a(\pi z^i)$
lab_Δ	$: ns$	$lab_\Delta(x^n, y^s)$ means: x^n is labeled by symbol y^s
$edg_\#$	$: nin$	$edg_\#(x^n, y^i, z^n)$ means: z^n is the y^i–th child of x^n
env	$: an$	$env(x^a, y^n)$ means: y^n is the root of the environment tree for x^a

□

Note that we have two different lab–relations for nodes labeled by an attribute occurrence, and nodes labeled by an output symbol, respectively. Further note that the $edg_\#$–relation has an additional argument to distinguish the children of a node by numbers and thereby to formalize ordered trees.

Now we will define the encoding Φ from attributed tree transducers to structures over our chosen signature SIG_{ATT}. The domains for sorts a and s are obvious. As domain of sort i we take an initial fragment of the natural numbers which has to be large enough to denote all ranks from the input and output ranked alphabets and to count the attributes. For the domain of sort n we take all the nodes of all trees that occur as right–hand sides of rules or in the environment. Every node will have the form of either $v_{(\sigma,c,\eta),p}$, if it occurs in the right–hand side of some inside attribute occurrence $c(\pi\eta)$ at σ, or $v_{b,p}$, if it occurs in the environment of some inherited attribute b, where in both cases p is some occurrence of the tree under question. In the following definition note that $Att = Att_{inh} \cup Att_{syn}$ and $\Sigma \cap \Delta = \emptyset$.

Definition 3.2 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer. The corresponding structure $A = \Phi(ATT) \in STR(SIG_{ATT})$ is defined as follows:

$$
\begin{array}{ll}
A_a & = Att \\
A_s & = \Sigma \cup \Delta \\
A_i & = \{0\} \cup [max\{rki(\Sigma \cup \Delta), |Att|\}] \\
A_n & = \{v_{(\sigma,c,\eta),p} \mid \sigma \in \Sigma, c(\pi\eta) \in inner(\sigma), p \in occ(rhs(c(\pi\eta), \sigma))\} \\
 & \quad \cup \{v_{b,p} \mid b \in Att_{inh}, p \in occ(E(b))\} \\
inh_A & = Att_{inh} \\
syn_A & = Att_{syn} \\
initial_A & = \{a_0\} \\
suc_A & = \{(x, y) \in A_i \times A_i \mid y = x + 1\} \\
in_A & = \Sigma \\
out_A & = \Delta \\
rk_A & = \{(\sigma, n) \in A_s \times A_i \mid \sigma \in \Sigma^{(n)} \text{ or } \sigma \in \Delta^{(n)}\} \\
rule_A & = \{(\sigma, c, i, v_{(\sigma,c,i),\epsilon}) \in A_s \times A_a \times A_i \times A_n\} \\
lab_{att\,A} & = \{(v_{(\sigma,c,i),p}, b, j) \in A_n \times A_a \times A_i \mid lab(rhs(c(\pi i), \sigma), p) = b(\pi j)\} \\
lab_{\Delta A} & = \{(v_{(\sigma,c,i),p}, \delta) \in A_n \times A_s \mid lab(rhs(c(\pi i), \sigma), p) = \delta\} \\
 & \quad \cup \{(v_{b,p}, \delta) \in A_n \times A_s \mid b \in Att_{inh}, lab(E(b), p) = \delta\} \\
edg_{\# A} & = \{(v_{t,p}, i, v_{t,pi}) \in A_n \times A_i \times A_n \mid t \text{ of the form } (\sigma, c, i) \text{ or } b \\
 & \qquad \text{for some } \sigma, c, i \text{ or } b\} \\
env_A & = \{(b, v_{b,\epsilon}) \in A_a \times A_n \mid b \in Att_{inh}\}
\end{array}
$$

□

How the encoding works, can best be seen with an example.

Example 3.3 For our example attributed tree transducer ATT_{leafs}, the construction yields $A_{leafs} = \Phi(ATT_{leafs})$ as follows:

$$
\begin{array}{ll}
A_{leafs_a} & = \{a,\, b\} \\
A_{leafs_s} & = \{\sigma,\, \alpha,\, \beta,\, \alpha',\, \beta',\, \omega\} \\
A_{leafs_i} & = \{0,\, 1,\, 2\} \\
A_{leafs_n} & = \{v_{(\sigma,a,0),\epsilon},\, v_{(\sigma,b,1),\epsilon},\, v_{(\sigma,b,2),\epsilon},\, v_{(\alpha,a,0),\epsilon},\, v_{(\alpha,a,0),1},\, v_{(\beta,a,0),\epsilon},\, v_{(\beta,a,0),1},\, v_{b,\epsilon}\} \\
inh_{A_{leafs}} & = \{b\} \\
syn_{A_{leafs}} & = \{a\} \\
initial_{A_{leafs}} & = \{a\} \\
suc_{A_{leafs}} & = \{(0,1),\, (1,2)\} \\
in_{A_{leafs}} & = \{\sigma,\, \alpha,\, \beta\} \\
out_{A_{leafs}} & = \{\alpha',\, \beta',\, \omega\} \\
rk_{A_{leafs}} & = \{(\sigma,2),\, (\alpha,0),\, (\beta,0),\, (\alpha',1),\, (\beta',1),\, (\omega,0)\} \\
rule_{A_{leafs}} & = \{(\sigma,a,0,v_{(\sigma,a,0),\epsilon}),\, (\sigma,b,1,v_{(\sigma,b,1),\epsilon}),\, (\sigma,b,2,v_{(\sigma,b,2),\epsilon}), \\
 & \quad\ (\alpha,a,0,v_{(\alpha,a,0),\epsilon}),\, (\beta,a,0,v_{(\beta,a,0),\epsilon})\} \\
lab_{att\,A_{leafs}} & = \{(v_{(\sigma,a,0),\epsilon},a,2),\, (v_{(\sigma,b,1),\epsilon},b,0),\, (v_{(\sigma,b,2),\epsilon},a,1),\, (v_{(\alpha,a,0),1},b,0), \\
 & \quad\ (v_{(\beta,a,0),1},b,0)\} \\
lab_{\Delta\,A_{leafs}} & = \{(v_{(\alpha,a,0),\epsilon},\alpha'),\, (v_{(\beta,a,0),\epsilon},\beta'),\, (v_{b,\epsilon},\omega)\} \\
edg_{\#\,A_{leafs}} & = \{(v_{(\alpha,a,0),\epsilon},1,v_{(\alpha,a,0),1}),\, (v_{(\beta,a,0),\epsilon},1,v_{(\beta,a,0),1})\} \\
env_{A_{leafs}} & = \{(b,v_{b,\epsilon})\}
\end{array}
$$

$\square$

It should be intuitively clear that the above structure A_{leafs} over SIG_{ATT} contains all information necessary for reconstructing the original attributed tree transducer.
However, this can not work for all structures over SIG_{ATT}, because obviously there are such structures that do not correspond to an attributed tree transducer. For example, we can construct a structure, where the interpretation of the relation symbol *rule* is the empty relation. Clearly, this contradicts the restriction from Definition 2.2 of attributed tree transducers that there must be a rule for every inside attribute occurrence at every input symbol.

Now we want to construct a formula that characterizes those structures over SIG_{ATT}, which correspond to an attributed tree transducer (in the sense in which A_{leafs} corresponds to ATT_{leafs}) and for which this corresponding attributed tree transducer can be obtained in a constructive way. Intuitively, such a formula ϕ_{att} has to express all the syntactic restrictions (such as the one above) from Definitions 2.1 and 2.2. Clearly, every structure obtained from an attributed tree transducer by the construction Φ should satisfy ϕ_{att}.
We claim that the following monadic second–order formula ϕ_{att} does the job. To make this claim plausible, we will present ϕ_{att} by several sub–formulas and explain, which restrictions are expressed by every sub–formula.

Let ϕ_{att} be the closed formula $\phi_1 \wedge \phi_2 \wedge \phi_3 \wedge \phi_4 \wedge \phi_5 \wedge \phi_6 \wedge \phi_7 \wedge \phi_8 \wedge \phi_9$, where:

- $\phi_1 = \forall x^a.((syn(x^a) \not\Leftrightarrow inh(x^a)) \wedge (initial(x^a) \Rightarrow syn(x^a))) \wedge \exists! x^a.(initial(x^a))$ expresses that each attribute (element of the domain of sort a) is either a synthesized or an inherited one, that there is exactly one initial attribute and that this initial attribute is a synthesized one

- $\phi_2 = \forall x^i.(\neg suc^+(x^i, x^i) \wedge \forall y^i.(x^i \neq y^i \Rightarrow (suc^+(x^i, y^i) \not\Leftrightarrow suc^+(y^i, x^i)))$
$\wedge \forall y^i, z^i.(suc(x^i, y^i) \wedge suc(x^i, z^i)$
$\vee suc(y^i, x^i) \wedge suc(z^i, x^i) \Rightarrow y^i = z^i))$
expresses that the transitive closure of *suc* defines an irreflexive, total order on the domain of sort i, and that the relation *suc* as well as its inverse are partial mappings, i.e., the domain of sort i together with *suc* as successor relation is indeed isomorphic to an initial fragment of the set of natural numbers

- $\phi_3 = \forall x^s.((in(x^s) \not\Leftrightarrow out(x^s)) \wedge \exists! y^i.(rk(x^s, y^i)))$
expresses that each symbol (element of the domain of sort s) is either an input or an output symbol and has an unique rank

- $\phi_4 = \forall x^n.(\neg edg^+(x^n, x^n) \wedge \forall y^n, z^i, u^n, v^i.(edg_\#(y^n, z^i, x^n) \wedge edg_\#(u^n, v^i, x^n)$
$\Rightarrow (y^n = u^n \wedge z^i = v^i)))$
expresses that the nodes (elements of the domain of sort n) together with the $edg_\#$–relation form a *forest*, i.e., a directed, acyclic graph, where every node has at most one incoming edge

- $\phi_5 = \forall x^n.((\exists y^s.(lab_\Delta(x^n, y^s)) \not\Leftrightarrow \exists y^a, z^i.(lab_{att}(x^n, y^a, z^i)))$
$\wedge \forall y^s, z^s.(lab_\Delta(x^n, y^s) \wedge lab_\Delta(x^n, z^s) \Rightarrow (y^s = z^s \wedge out(y^s)))$
$\wedge \forall y^a, z^i, u^a, v^i.(lab_{att}(x^n, y^a, z^i) \wedge lab_{att}(x^n, u^a, v^i) \Rightarrow (y^a = u^a \wedge z^i = v^i)))$
expresses that each node is uniquely labeled by either an output symbol or an attribute occurrence (fixed by an attribute and an integer)

- $\phi_6 = \forall x^n.((\exists y^a, z^i.(lab_{att}(x^n, y^a, z^i)) \Rightarrow leaf(x^n))$
$\wedge \forall z^i, u^n, v^n.(edg_\#(x^n, z^i, u^n) \wedge edg_\#(x^n, z^i, v^n) \Rightarrow u^n = v^n)$
$\wedge \forall y^s.(lab_\Delta(x^n, y^s) \Rightarrow \forall z^i.(\exists u^n.(edg_\#(x^n, z^i, u^n)) \Leftrightarrow sub(y^s, z^i))))$
expresses that only leaf nodes (nodes without outgoing edges) can be labeled by attribute occurrences and that a node labeled by an output symbol of rank k has exactly k outgoing edges, each one for every integer between 1 and k (expressed by *sub*, see below)

- $\phi_7 = \forall x^n.(root(x^n) \Rightarrow (\exists y^a.(env(y^a, x^n)) \not\Leftrightarrow \exists y^s, z^a, u^i.(rule(y^s, z^a, u^i, x^n)))$
$\wedge \forall y^a, z^a.(env(y^a, x^n) \wedge env(z^a, x^n) \Rightarrow y^a = z^a)$
$\wedge \forall y^s, z^a, u^i, v^s, w^a, t^i.(rule(y^s, z^a, u^i, x^n) \wedge rule(v^s, w^a, t^i, x^n)$
$\Rightarrow (y^s = v^s \wedge z^a = w^a \wedge u^i = t^i)))$
expresses that every root node (node without incoming edge) does either belong to the environment of a unique attribute or to the right–hand side of a unique rule

- $\phi_8 = \forall x^s, y^a, z^i.((\exists u^n.(rule(x^s, y^a, z^i, u^n)) \Rightarrow in(x^s) \wedge inner(x^s, y^a, z^i))$
$\wedge (in(x^s) \wedge inner(x^s, y^a, z^i) \Rightarrow \exists! u^n.(rule(x^s, y^a, z^i, u^n))$
$\wedge \forall u^n.(rule(x^s, y^a, z^i, u^n)$
$\Rightarrow (root(u^n) \wedge \forall v^n.(edg^*(u^n, v^n)$
$\Rightarrow \exists t^s.(lab_\Delta(v^n, t^s))$
$\vee \exists t^a, w^i.(lab_{att}(v^n, t^a, w^i) \wedge outer(x^s, t^a, w^i)))))))$
expresses that only inside attribute occurrences at input symbols (expressed by *inner*) have a right–hand side, that for every inside attribute occurrence at an input symbol there is an unique node, which is the root of its right–hand side, and that in the tree rooted by this node only output symbols and outside attribute occurrences at the input symbol (expressed by *outer*) occur

- $\phi_9 = \forall x^a.((\exists y^n.(env(x^a, y^n)) \Rightarrow inh(x^a))$
  $\wedge(inh(x^a) \Rightarrow \exists! y^n.(env(x^a, y^n))$
  $\wedge\forall y^n.(env(x^a, y^n) \Rightarrow root(y^n)$
  $\wedge\forall z^n.(edg^*(y^n, z^n) \Rightarrow \exists u^s.(lab_\Delta(z^n, u^s))))))$

  expresses that only inherited attributes have an environment, that for every inherited attribute there is a unique node, which is the root of its environment, and that in the tree rooted by this node only output symbols occur

In ϕ_{att} we used several auxiliary formulas, which we have to define yet. The formulas *edg*, *leaf*, *root* and *null* as given below should be self–explaining. By *sub* we want to check, whether a node labeled by a symbol x^s should have an y^i–th child node, that is, whether y^i is greater than zero and smaller or equal (expressed by suc^*) than the rank of x^s. The formulas *inner* and *outer* test the property of an attribute occurrence to be an inside or outside attribute occurrence, respectively. This is done by following the definitions of the sets $inner(\sigma)$ and $outer(\sigma)$, respectively, for an input symbol σ (page 4).

The definitions of the auxiliary formulas are as follows:

$$
\begin{array}{ll}
edg(x^n, y^n) & = \exists z^i.(edg_\#(x^n, z^i, y^n)) \\
leaf(x^n) & = \neg\exists y^n.(edg(x^n, y^n)) \\
root(x^n) & = \neg\exists y^n.(edg(y^n, x^n)) \\
null(x^i) & = \neg\exists y^i.(suc(y^i, x^i)) \\
sub(x^s, y^i) & = \neg null(y^i) \wedge \forall z^i.(rk(x^s, z^i) \Rightarrow suc^*(y^i, z^i)) \\
inner(x^s, y^a, z^i) & = (syn(y^a) \wedge null(z^i)) \vee (inh(y^a) \wedge sub(x^s, z^i)) \\
outer(x^s, y^a, z^i) & = (inh(y^a) \wedge null(z^i)) \vee (syn(y^a) \wedge sub(x^s, z^i))
\end{array}
$$

Since we claimed that for every structure over SIG_{ATT} satisfying ϕ_{att}, the corresponding attributed tree transducer can be constructed effectively, we still have to give such a construction.

Definition 3.4 Let A be a structure over SIG_{ATT} satisfying ϕ_{att}. From A we define the attributed tree transducer $ATT = \Psi(A)$ by applying the following construction:

$$
\begin{array}{ll}
ATT & = (Att, \Sigma, \Delta, a_0, R, E), \text{ with:} \\
Att & = A_a \\
Att_{inh} & = inh_A \\
Att_{syn} & = syn_A \\
\Sigma & = \{\sigma^{(int_A(k))} \mid \sigma \in in_A, (\sigma, k) \in rk_A\} \\
\Delta & = \{\delta^{(int_A(k))} \mid \delta \in out_A, (\delta, k) \in rk_A\} \\
a_0 & = \text{the single element in } initial_A \\
R_\sigma & = \{c(\pi\ int_A(i)) \to unfold_A(n) \mid (\sigma, c, i, n) \in rule_A\} \text{ for every } \sigma \in in_A \\
E(b) & = unfold_A(n) \text{ for every } (b, n) \in env_A,
\end{array}
$$

where the function $int_A : A_i \to \mathbb{N}$ is defined by:

$$
\begin{array}{lll}
int_A(k) & = 0 & \text{if there is no } j \in A_i \text{ with } (j, k) \in suc_A \\
int_A(k) & = 1 + int_A(j) & \text{if } (j, k) \in suc_A,
\end{array}
$$

and the function $unfold_A : A_n \to T_\Delta(Att(\{\pi\, int_A(k) \mid k \in A_i\}))$ is defined by:

$$
\begin{array}{lll}
unfold_A(n) & = c(\pi\, int_A(i)) & \text{if } (n, c, i) \in lab_{att\,A} \\
unfold_A(n) & = \delta(unfold_A(n_1), \ldots, unfold_A(n_{int_A(k)})) & \text{if } (n, \delta) \in lab_{\Delta A}, (\delta, k) \in rk_A \\
 & & \text{and } (n, i, n_{int_A(i)}) \in edg_{\# A}
\end{array}
$$

□

Now we want to explain informally and by arguing with the help of the formula ϕ_{att}, why this construction is well–defined and indeed gives an attributed tree transducer:

By ϕ_1 we have that Att is partitioned into Att_{inh} and Att_{syn} and that the attribute a_0 is well–defined and a synthesized one. From ϕ_2 we know that the function int_A is well–defined (for every $k \in A_i$ there is at most one $j \in A_i$ with $(j,k) \in suc_A$). By ϕ_3 we know that we can construct the ranked alphabets Σ and Δ ($rank_\Sigma$ and $rank_\Delta$ are well–defined) and that they will be disjoint. From ϕ_4, ϕ_5 and ϕ_6 we can deduce that the function $unfold_A$ is well–defined (for every $(n,\delta) \in lab_{\Delta A}$ and $(\delta,k) \in rk_A$, the $n_1,\ldots,n_{int_A(k)}$ exist and are unique) and indeed gives the intended trees built over Δ and possibly labeled by attribute occurrences at the leafs. By ϕ_7 we know that there are no superfluous nodes (every node belongs to a right–hand side or an environment) and every node is used only in one right–hand side or environment tree. By ϕ_8 we have that for every $\sigma \in \Sigma$, the rules R_σ are well–defined and that for every inside attribute occurrence, there is a rule with valid right–hand side, in the sense of Definition 2.1. From ϕ_9 we know that the function E is well–defined and gives for every inherited attribute a tree over the ranked alphabet Δ.

Since every structure obtained from an attributed tree transducer by the encoding Φ clearly satisfies ϕ_{att}, we can always apply the decoding Ψ to such a structure. In order to justify our choice of the signature SIG_{ATT} and of how to represent attributed tree transducers as structures over this signature, we can now formulate the following claim, as proposed in the introduction of this section.

Claim 3.5 For every attributed tree transducer ATT, the two transducers ATT and $\Psi(\Phi(ATT))$ are literally the same. $\square$

Note that for the dual claim, namely that for a structure A satisfying ϕ_{att}, A and $\Phi(\Psi(A))$ are in some sense the same, we would have to put an additional restriction ϕ_{10} on A to ensure that $|A_i| = 1 + max\{|A_a|, rki(\Sigma)\}$, which can not be expressed in the monadic fragment of second–order logic.

4 Describing properties of attributed tree transducers by formulas

Definition 4.1 Given some property P of attributed tree transducers and a closed formula ϕ_P over SIG_{ATT}, we say that ϕ_P *describes* P iff for every attributed tree transducer ATT holds:

$$ATT \text{ has the property } P \qquad \text{iff} \qquad \Phi(ATT) \text{ is a model for } \phi_P \qquad \square$$

In the following, we will recall algebraic definitions of well–known attributed tree transducer properties from the literature, and claim that they are described by certain closed formulas. Therefore, we will define various auxiliary formulas with free variables, describing certain concepts in the context of attributed tree transducers, like dependencies, brother graphs etc. We will then use these auxiliary results to show how our desired properties of attributed tree transducers can be described.

One of the notions, which we will need to express again and again, is the notion of an attribute occurrence directly depending on another. Recall that we defined an inside attribute occurrence $c'(\pi\eta')$ to depend directly on an outside attribute occurrence $c(\pi\eta)$ at an input symbol σ iff $c(\pi\eta)$ appears in $rhs(c'(\pi\eta'),\sigma)$. Since in our representation of attributed tree transducers right–hand sides are represented by their root node occurring in the interpretation of the relation $rule$, we get the following claim.

Claim 4.2 Consider the formula:
$direct(x^s, y^a, z^i, u^a, v^i) = \exists p^n, q^n.(rule(x^s, y^a, z^i, p^n) \wedge edg^*(p^n, q^n) \wedge lab_{att}(q^n, u^a, v^i))$
Then for every attributed tree transducer ATT, input symbol σ and attribute occurrences $c'(\pi\eta') \in inner(\sigma)$ and $c(\pi\eta) \in outer(\sigma)$:
$(ATT, \sigma, c', \eta', c, \eta) \models direct$ iff $c'(\pi\eta')$ depends directly on $c(\pi\eta)$ at σ. $\square$

From Figure 2, visualizing the roles of the variables, it should be clear that the above claim holds.

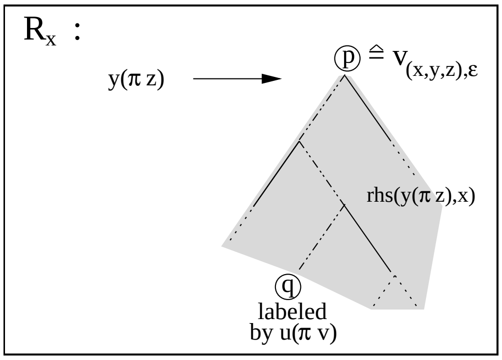

Figure 2: visualization of $direct(x^s, y^a, z^i, u^a, v^i)$

The nodes p (representing the right–hand side of the rule for $y(\pi z)$ at symbol x) and its descendant q (some leaf of this right–hand side) are shown in small circles. The formula $direct$ simply tests, whether there is such a node q labeled by $u(\pi v)$.

4.1 single–use

As a first example, we describe the property single–use (coming from [15], transfered to attributed tree transducers in [9]). Intuitively, an attributed tree transducer is called single–use if every attribute occurrence in every tree is used at most once, that is, every attribute occurrence appears at most once in every R_σ.

Definition 4.3 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer. ATT is *single–use* if for every $\sigma \in \Sigma^{(n)}$ and $c(\pi\eta) \in outer(\sigma)$, $c(\pi\eta)$ appears in a right–hand side of at most one rule in R_σ and there exactly once. □

Example 4.4 From the sets of rules it is obvious that ATT_{leafs} is single–use. □

We can think of Definition 4.3 as expressing that, if in the set of rules for some input symbol we have two occurrences (in right–hand sides at this input symbol) labeled by the same outside attribute occurrence, then these two occurrences are the same. This we can express by a monadic second–order formula.

Theorem 4.5 The following formula $\phi_{single-use}$ (**FO+TC+EQ**) describes the property of an attributed tree transducer to be single–use.

$$\begin{aligned}&\forall x^s, y^a, z^i, u^n.(rule(x^s, y^a, z^i, u^n) \Rightarrow \forall w^a, t^i, p^n.(rule(x^s, w^a, t^i, p^n)\\ &\qquad \Rightarrow \forall v^n, q^n.((edg^*(u^n, v^n) \wedge edg^*(p^n, q^n)\\ &\qquad\qquad \wedge \exists r^a, s^i.(lab_{att}(v^n, r^a, s^i) \wedge lab_{att}(q^n, r^a, s^i))) \Rightarrow v^n = q^n)))\end{aligned}$$

□

Proof: Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer. We have:

$\Phi(ATT) \models \phi_{single-use}$

$\Updownarrow$ (by definition of $\models$, no free variables)

$\Phi(ATT) \models_{\sigma_0} \phi_{single-use}$, where σ_0 is an arbitrary state

$\Updownarrow$ (by semantics of SO logic)

for every $\delta \in \Phi(ATT)_s$, $c \in \Phi(ATT)_a$, $\eta \in \Phi(ATT)_i$, $v \in \Phi(ATT)_n$:
$\Phi(ATT) \models_{\sigma_1} \phi_1$, where

$\sigma_1 = \sigma_0\{x^s \to \delta\}\{y^a \to c\}\{z^i \to \eta\}\{u^n \to v\}$ and

$$\begin{aligned}\phi_1 = {}& rule(x^s, y^a, z^i, u^n) \Rightarrow \forall w^a, t^i, p^n.(rule(x^s, w^a, t^i, p^n)\\ &\quad \Rightarrow \forall v^n, q^n.((edg^*(u^n, v^n) \wedge edg^*(p^n, q^n)\\ &\qquad \wedge \exists r^a, s^i.(lab_{att}(v^n, r^a, s^i) \wedge lab_{att}(q^n, r^a, s^i))) \Rightarrow v^n = q^n))\end{aligned}$$

$\Updownarrow$ (by definition of Φ, semantics of SO logic)

for every $\delta \in \Sigma \cup \Delta$, $c \in Att$, $\eta \in \Phi(ATT)_i$, $v \in \Phi(ATT)_n$:
$\Phi(ATT) \not\models_{\sigma_1} rule(x^s, y^a, z^i, u^n)$ or $\Phi(ATT) \models_{\sigma_1} \phi_2$, where

$$\begin{aligned}\phi_2 = {}& \forall w^a, t^i, p^n.(rule(x^s, w^a, t^i, p^n)\\ &\quad \Rightarrow \forall v^n, q^n.((edg^*(u^n, v^n) \wedge edg^*(p^n, q^n)\\ &\qquad \wedge \exists r^a, s^i.(lab_{att}(v^n, r^a, s^i) \wedge lab_{att}(q^n, r^a, s^i))) \Rightarrow v^n = q^n))\end{aligned}$$

$\Updownarrow$ (by semantics of SO logic)

for every $\delta \in \Sigma \cup \Delta$, $c \in Att$, $\eta \in \Phi(ATT)_i$, $v \in \Phi(ATT)_n$:
$(\sigma_1(x^s), \sigma_1(y^a), \sigma_1(z^i), \sigma_1(u^n)) \notin rule_{\Phi(ATT)}$ or $\Phi(ATT) \models_{\sigma_1} \phi_2$

$\Updownarrow$ (by using $\sigma_1 = \sigma_0\{x^s \to \delta\}\{y^a \to c\}\{z^i \to \eta\}\{u^n \to v\}$)

for every $\delta \in \Sigma \cup \Delta$, $c \in Att$, $\eta \in \Phi(ATT)_i$, $v \in \Phi(ATT)_n$:
if $(\delta, c, \eta, v) \in rule_{\Phi(ATT)}$ then $\Phi(ATT) \models_{\sigma_1} \phi_2$

$\Updownarrow$ (by definition of Φ)

for every $\delta \in \Sigma \cup \Delta$, $c \in Att$, $\eta \in \Phi(ATT)_i$, $v \in \Phi(ATT)_n$:
if $\delta \in \Sigma$, $c(\pi\eta) \in inner(\delta)$ and $v = v_{(\delta,c,\eta),\epsilon}$, then $\Phi(ATT) \models_{\sigma_1} \phi_2$

$\Updownarrow$ (by semantics of SO logic)

for every $\delta \in \Sigma$, $c(\pi\eta) \in inner(\delta)$, $v = v_{(\delta,c,\eta),\epsilon}$: for every $c' \in \Phi(ATT)_a$, $\eta' \in \Phi(ATT)_i$, $v' \in \Phi(ATT)_n$:
$\Phi(ATT) \models_{\sigma_2} \phi_3$, where

$\sigma_2 = \sigma_1\{w^a \to c'\}\{t^i \to \eta'\}\{p^n \to v'\}$ and

$$\phi_3 = rule(x^s, w^a, t^i, p^n) \Rightarrow \forall v^n, q^n.((edg^*(u^n, v^n) \wedge edg^*(p^n, q^n) \wedge \exists r^a, s^i.(lab_{att}(v^n, r^a, s^i) \wedge lab_{att}(q^n, r^a, s^i))) \Rightarrow v^n = q^n)$$

$\Updownarrow$ (by definition of Φ, semantics of SO logic)

for every $\delta \in \Sigma$, $c(\pi\eta) \in inner(\delta)$, $v = v_{(\delta,c,\eta),\epsilon}$, $c' \in Att$, $\eta' \in \Phi(ATT)_i$, $v' \in \Phi(ATT)_n$:
$\Phi(ATT) \not\models_{\sigma_2} rule(x^s, w^a, t^i, p^n)$ or $\Phi(ATT) \models_{\sigma_2} \phi_4$, where

$$\phi_4 = \forall v^n, q^n.((edg^*(u^n, v^n) \wedge edg^*(p^n, q^n) \wedge \exists r^a, s^i.(lab_{att}(v^n, r^a, s^i) \wedge lab_{att}(q^n, r^a, s^i))) \Rightarrow v^n = q^n)$$

$\Updownarrow$ (by semantics of SO logic)

for every $\delta \in \Sigma$, $c(\pi\eta) \in inner(\delta)$, $v = v_{(\delta,c,\eta),\epsilon}$, $c' \in Att$, $\eta' \in \Phi(ATT)_i$, $v' \in \Phi(ATT)_n$:
if $(\sigma_2(x^s), \sigma_2(w^a), \sigma_2(t^i), \sigma_2(p^n)) \in rule_{\Phi(ATT)}$ then $\Phi(ATT) \models_{\sigma_2} \phi_4$

$\Updownarrow$ (by using σ_2, semantics of SO logic)

for every $\delta \in \Sigma$, $c(\pi\eta) \in inner(\delta)$, $v = v_{(\delta,c,\eta),\epsilon}$, $c' \in Att$, $\eta' \in \Phi(ATT)_i$, $v' \in \Phi(ATT)_n$:
if $(\delta, c', \eta', v') \in rule_{\Phi(ATT)}$ then for every $l, l' \in \Phi(ATT)_n$: $\Phi(ATT) \models_{\sigma_3} \phi_5$, where

$\sigma_3 = \sigma_2\{v^n \to l\}\{q^n \to l'\}$ and

$$\phi_5 = (edg^*(u^n, v^n) \wedge edg^*(p^n, q^n) \wedge \exists r^a, s^i.(lab_{att}(v^n, r^a, s^i) \wedge lab_{att}(q^n, r^a, s^i))) \Rightarrow v^n = q^n$$

$\Updownarrow$ (by definition of Φ)

for every $\delta \in \Sigma$, $c(\pi\eta) \in inner(\delta)$, $v = v_{(\delta,c,\eta),\epsilon}$, $c'(\pi\eta') \in inner(\delta)$, $v' = v_{(\delta,c',\eta'),\epsilon}$ and $l, l' \in \Phi(ATT)_n$:
$\Phi(ATT) \models_{\sigma_3} \phi_5$

$\Updownarrow$ (by semantics of SO logic)

for every $\delta \in \Sigma$, $c(\pi\eta) \in inner(\delta)$, $v = v_{(\delta,c,\eta),\epsilon}$, $c'(\pi\eta') \in inner(\delta)$, $v' = v_{(\delta,c',\eta'),\epsilon}$ and $l, l' \in \Phi(ATT)_n$:
if $\Phi(ATT) \models_{\sigma_3} (edg^*(u^n, v^n) \wedge edg^*(p^n, q^n) \wedge \exists r^a, s^i.(lab_{att}(v^n, r^a, s^i) \wedge lab_{att}(q^n, r^a, s^i)))$,
then $\Phi(ATT) \models_{\sigma_3} v^n = q^n$

$\Updownarrow$ (by semantics of SO logic)

for every $\delta \in \Sigma$, $c(\pi\eta) \in inner(\delta)$, $v = v_{(\delta,c,\eta),\epsilon}$, $c'(\pi\eta') \in inner(\delta)$, $v' = v_{(\delta,c',\eta'),\epsilon}$ and $l, l' \in \Phi(ATT)_n$:
if $\Phi(ATT) \models_{\sigma_3} edg^*(u^n, v^n)$, $\Phi(ATT) \models_{\sigma_3} edg^*(p^n, q^n)$ and
$\Phi(ATT) \models_{\sigma_3} \exists r^a, s^i.(lab_{att}(v^n, r^a, s^i) \wedge lab_{att}(q^n, r^a, s^i))$, then $\sigma_3(v^n) = \sigma_3(q^n)$

$\Updownarrow$ (by using $\sigma_3(u^n) = v_{(\delta,c,\eta),\epsilon}$, $\sigma_3(p^n) = v_{(\delta,c',\eta'),\epsilon}$ and **Claim 4.6 below**)

for every $\delta \in \Sigma$, $c(\pi\eta) \in inner(\delta)$, $v = v_{(\delta,c,\eta),\epsilon}$, $c'(\pi\eta') \in inner(\delta)$, $v' = v_{(\delta,c',\eta'),\epsilon}$ and $l, l' \in \Phi(ATT)_n$:
if $\sigma_3(v^n) = v_{(\delta,c,\eta),w}$, $\sigma_3(q^n) = v_{(\delta,c',\eta'),w'}$ for some $w, w' \in \mathbb{N}^*$ and
$\Phi(ATT) \models_{\sigma_3} \exists r^a, s^i.(lab_{att}(v^n, r^a, s^i) \wedge lab_{att}(q^n, r^a, s^i))$, then $\sigma_3(v^n) = \sigma_3(q^n)$

$\Updownarrow$ (by definition of Φ, using $\sigma_3(v^n) = l$ and $\sigma_3(q^n) = l'$)

for every $\delta \in \Sigma$, $c(\pi\eta) \in inner(\delta)$, $v = v_{(\delta,c,\eta),\epsilon}$, $c'(\pi\eta') \in inner(\delta)$, $v' = v_{(\delta,c',\eta'),\epsilon}$,
$w \in occ(rhs(c(\pi\eta), \delta))$, $w' \in occ(rhs(c'(\pi\eta'), \delta))$, $l = v_{(\delta,c,\eta),w}$ and $l' = v_{(\delta,c',\eta'),w'}$:
if $\Phi(ATT) \models_{\sigma_3} \exists r^a, s^i.(lab_{att}(v^n, r^a, s^i) \wedge lab_{att}(q^n, r^a, s^i))$, then $l = l'$

$\Updownarrow$ (by semantics of SO logic)

for every $\delta \in \Sigma$, $c(\pi\eta) \in inner(\delta)$, $v = v_{(\delta,c,\eta),\epsilon}$, $c'(\pi\eta') \in inner(\delta)$, $v' = v_{(\delta,c',\eta'),\epsilon}$, $w \in occ(rhs(c(\pi\eta),\delta))$, $w' \in occ(rhs(c'(\pi\eta'),\delta))$, $l = v_{(\delta,c,\eta),w}$ and $l' = v_{(\delta,c',\eta'),w'}$:
if there exist $a \in \Phi(ATT)_a$ and $i \in \Phi(ATT)_i$, such that
$\Phi(ATT) \models_{\sigma_3\{r^a \to a\}\{s^i \to i\}} (lab_{att}(v^n, r^a, s^i) \wedge lab_{att}(q^n, r^a, s^i))$, then $l = l'$

$\Updownarrow$ (by semantics of SO logic, using $\sigma_3\{r^a \to a\}\{s^i \to i\}$)

for every $\delta \in \Sigma$, $c(\pi\eta) \in inner(\delta)$, $v = v_{(\delta,c,\eta),\epsilon}$, $c'(\pi\eta') \in inner(\delta)$, $v' = v_{(\delta,c',\eta'),\epsilon}$, $w \in occ(rhs(c(\pi\eta),\delta))$, $w' \in occ(rhs(c'(\pi\eta'),\delta))$, $l = v_{(\delta,c,\eta),w}$ and $l' = v_{(\delta,c',\eta'),w'}$:
if there exist $a \in \Phi(ATT)_a$ and $i \in \Phi(ATT)_i$, such that
$(l, a, i) \in lab_{att_{\Phi(ATT)}}$ and $(l', a, i) \in lab_{att_{\Phi(ATT)}}$, then $l = l'$

$\Updownarrow$ (by definition of Φ)

for every $\delta \in \Sigma$, $c(\pi\eta), c'(\pi\eta') \in inner(\delta)$, $w \in occ(rhs(c(\pi\eta),\delta))$, $w' \in occ(rhs(c'(\pi\eta'),\delta))$, $l = v_{(\delta,c,\eta),w}$ and $l' = v_{(\delta,c',\eta'),w'}$:
if there exist $a \in Att$ and $i \in \{0\} \cup [max\{rki(\Sigma \cup \Delta), |Att|\}]$, such that
$lab(rhs(c(\pi\eta),\delta), w) = a(\pi i)$ and $lab(rhs(c'(\pi\eta'),\delta), w') = a(\pi i)$, then $l = l'$

$\Updownarrow$ (by reformulation)

for every $\delta \in \Sigma$, $c(\pi\eta), c'(\pi\eta') \in inner(\delta)$, $w \in occ(rhs(c(\pi\eta),\delta))$ and $w' \in occ(rhs(c'(\pi\eta'),\delta))$:
if the occurrence w in the right–hand side of $c(\pi\eta)$ at δ and the occurrence w' in the right–hand side of $c'(\pi\eta')$ at δ are labeled by the same attribute occurrence,
then $c = c'$, $\eta = \eta'$ and $w = w'$

$\Updownarrow$ (by remark above Theorem 4.5)

ATT is single–use

□

In the proof we used the following auxiliary claim, which follows from our construction of the reflexive, transitive closure.

Claim 4.6 $\Phi(ATT) \models_\sigma edg^*(x^n, y^n)$ iff $\sigma(x^n) = v_{t,p}$ and there is a $w \in \mathbb{N}^*$ such that $\sigma(y^n) = v_{t,pw}$. □

We will not give such lengthy proofs for the other describing formulas of properties of attributed tree transducers. Anyway, the maybe most interesting point in the proof as given is the last step, which comes from the remark above Theorem 4.5 and gives a definition of single–use that is very close to our logical representation. While here, in the case of the property single–use, this step is rather trivial, in the formalization of most of the other properties, this will be the crucial point: to find and prove, from the definitions as given in the various literature, characterizations that are suitable to be expressed in our representation of attributed tree transducers. That these characterizations are formalized by certain given formulas, will then be plausible without proofs such as the one above.

4.2 nonnested, one–visit and absolutely noncircular

In this subsection we want to study the properties of attributed tree transducers to be nonnested, one–visit or absolutely noncircular. These kinds of attributed tree transducers (as studied in [10]) are of special interest, because they correspond to certain classes of macro tree transducers, namely basic well–presented, super well–presented and well–presented macro tree transducers, respectively.

Intuitively, an attributed tree transducer is called nonnested if no inherited attribute depends on a synthesized attribute. In other words, in every dependency graph, there are no edges from one child node to another.

Definition 4.7 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer. ATT is *nonnested* if for every $\sigma \in \Sigma^{(n)}$, $1 \leq k \leq n$ and $b \in Att_{inh}$, there are no $a \in Att_{syn}$ and $1 \leq i \leq n$ such that $b(\pi k)$ depends directly on $a(\pi i)$ at σ. □

Example 4.8 ATT_{leafs} is not nonnested, because in the dependency graph of σ (see next page) there is an edge from the first to the second child node. □

This property can easily be described by a formula, which in our representation has to express that in the right–hand side of every inherited attribute occurrence at every symbol, there is no occurrence, which is labeled by a synthesized attribute occurrence.

Claim 4.9 The following formula $\phi_{nonnested}$ (**FO+TC+EQ**) describes the property of an attributed tree transducer to be nonnested.
$\forall x^s.(in(x^s) \Rightarrow \neg\exists y^a, z^i, u^a, v^i.(inh(y^a) \wedge syn(u^a) \wedge direct(x^s, y^a, z^i, u^a, v^i)))$ □

Example 4.10 We can show that ATT_{leafs} is not nonnested, by showing that the structure $A_{leafs} = \Phi(ATT_{leafs})$ does not model $\phi_{nonnested}$, as follows by contradiction.
Assume $A_{leafs} \models \phi_{nonnested}$, then we reason:

$A_{leafs} \models \phi_{nonnested}$

$\Updownarrow$ (by definition of $\models$, no free variables)

$A_{leafs} \models_{\delta_0} \phi_{nonnested}$, where δ_0 is an arbitrary state

$\Downarrow$ (by semantics of SO logic)

for the particular input symbol σ in A_{leafs_s}:
$A_{leafs} \models_{\delta_1} (in(x^s) \Rightarrow \neg\exists y^a, z^i, u^a, v^i.(inh(y^a) \wedge syn(u^a) \wedge direct(x^s, y^a, z^i, u^a, v^i)))$, where $\delta_1 = \delta_0\{x^s \to \sigma\}$

$\Updownarrow$ (by semantics of SO logic)

$A_{leafs} \not\models_{\delta_1} in(x^s)$ or $A_{leafs} \models_{\delta_1} \neg\exists y^a, z^i, u^a, v^i.(inh(y^a) \wedge syn(u^a) \wedge direct(x^s, y^a, z^i, u^a, v^i))$

$\Updownarrow$ (by semantics of SO logic)

$\delta_1(x^s) \notin in_{A_{leafs}}$ or $A_{leafs} \not\models_{\delta_1} \exists y^a, z^i, u^a, v^i.(inh(y^a) \wedge syn(u^a) \wedge direct(x^s, y^a, z^i, u^a, v^i))$

$\Updownarrow$ (by $\delta_1(x^s) = \sigma \in in_{A_{leafs}}$)

not $A_{leafs} \models_{\delta_1} \exists y^a, z^i, u^a, v^i.(inh(y^a) \wedge syn(u^a) \wedge direct(x^s, y^a, z^i, u^a, v^i))$

$\Updownarrow$ (by semantics of SO logic)

there are no $c \in A_{leafs_a}$, $n \in A_{leafs_i}$, $d \in A_{leafs_a}$, $m \in A_{leafs_i}$ such that:
$A_{leafs} \models_{\delta_1\{y^a \to c\}\{z^i \to n\}\{u^a \to d\}\{v^i \to m\}} inh(y^a) \wedge syn(u^a) \wedge direct(x^s, y^a, z^i, u^a, v^i)$

$\Updownarrow$

for every $c \in A_{leafs_a}$, $n \in A_{leafs_i}$, $d \in A_{leafs_a}$, $m \in A_{leafs_i}$:
not $A_{leafs} \models_{\delta_1\{y^a \to c\}\{z^i \to n\}\{u^a \to d\}\{v^i \to m\}} inh(y^a) \wedge syn(u^a) \wedge direct(x^s, y^a, z^i, u^a, v^i)$

$\Downarrow$ (instantiation)

for the particular $b \in A_{leafs_a}$, $2 \in A_{leafs_i}$, $a \in A_{leafs_a}$, $1 \in A_{leafs_i}$:
not $A_{leafs} \models_{\delta_2} inh(y^a) \wedge syn(u^a) \wedge direct(x^s, y^a, z^i, u^a, v^i)$, where
$\delta_2 = \delta_1\{y^a \to b\}\{z^i \to 2\}\{u^a \to a\}\{v^i \to 1\}$

$\Updownarrow$ (by semantics of SO logic)

not $A_{leafs} \models_{\delta_2} inh(y^a)$ or not $A_{leafs} \models_{\delta_2} syn(u^a)$ or not $A_{leafs} \models_{\delta_2} direct(x^s, y^a, z^i, u^a, v^i)$

$\Updownarrow$ (by semantics of SO logic)

$\delta_2(y^a) \notin inh_{A_{leafs}}$ or $\delta_2(u^a) \notin syn_{A_{leafs}}$ or not $A_{leafs} \models_{\delta_2} direct(x^s, y^a, z^i, u^a, v^i)$

$\Updownarrow$ (by $\delta_2(y^a) = b \in inh_{A_{leafs}}$ and $\delta_2(u^a) = a \in syn_{A_{leafs}}$, definition of *direct*)

not $A_{leafs} \models_{\delta_2} \exists p^n, q^n.(rule(x^s, y^a, z^i, p^n) \wedge edg^*(p^n, q^n) \wedge lab_{att}(q^n, u^a, v^i))$

$\Updownarrow$ (by semantics of SO logic)

there are no $v, v' \in A_{leafs_n}$ such that:
$A_{leafs} \models_{\delta_2\{p^n \to v\}\{q^n \to v'\}} rule(x^s, y^a, z^i, p^n) \wedge edg^*(p^n, q^n) \wedge lab_{att}(q^n, u^a, v^i)$

$\Updownarrow$

for every $v, v' \in A_{leafs_n}$:
not $A_{leafs} \models_{\delta_2\{p^n \to v\}\{q^n \to v'\}} rule(x^s, y^a, z^i, p^n) \wedge edg^*(p^n, q^n) \wedge lab_{att}(q^n, u^a, v^i)$

$\Downarrow$ (instantiation)

for the particular $v_{(\sigma,b,2),\epsilon} \in A_{leafs_n}$:
not $A_{leafs} \models_{\delta_3} rule(x^s, y^a, z^i, p^n) \wedge edg^*(p^n, q^n) \wedge lab_{att}(q^n, u^a, v^i)$, where
$\delta_3 = \delta_2\{p^n \to v_{(\sigma,b,2),\epsilon}\}\{q^n \to v_{(\sigma,b,2),\epsilon}\}$

$\Updownarrow$ (by semantics of SO logic)

not $A_{leafs} \models_{\delta_3} rule(x^s, y^a, z^i, p^n)$ or not $A_{leafs} \models_{\delta_3} edg^*(p^n, q^n)$ or not $A_{leafs} \models_{\delta_3} lab_{att}(q^n, u^a, v^i)$

$\Updownarrow$ (by semantics of SO logic, definition of edg^*)

$(\delta_3(x^s), \delta_3(y^a), \delta_3(z^i), \delta_3(p^n)) \notin rule_{A_{leafs}}$ or $(\delta_3(q^n), \delta_3(u^a), \delta_3(v^i)) \notin lab_{att\,A_{leafs}}$
or not $A_{leafs} \models_{\delta_3} (p^n = q^n) \vee edg^+(p^n, q^n)$

$\Updownarrow$ (by $(\sigma, b, 2, v_{(\sigma,b,2),\epsilon}) \in rule_{A_{leafs}}$ and $(v_{(\sigma,b,2),\epsilon}, a, 1) \in lab_{att\,A_{leafs}}$)

not $A_{leafs} \models_{\delta_3} (p^n = q^n) \vee edg^+(p^n, q^n)$

$\Updownarrow$ (by semantics of SO logic)

not $A_{leafs} \models_{\delta_3} (p^n = q^n)$ and not $A_{leafs} \models_{\delta_3} edg^+(p^n, q^n)$

$\Updownarrow$ (by semantics of SO logic)

$\delta_3(p^n) \neq \delta_3(q^n)$ and not $A_{leafs} \models_{\delta_3} edg^+(p^n, q^n)$

$\Updownarrow$ (by $\delta_3(p^n) = \delta_3(q^n) = v_{(\sigma,b,2),\epsilon}$)

$v_{(\sigma,b,2),\epsilon} \neq v_{(\sigma,b,2),\epsilon}$ **contradiction !**

$\square$

The next property relaxes a bit the dependencies that may occur between child nodes in the dependency graphs. Whereas in a nonnested attributed tree transducer no such dependencies are allowed, for one–visit attributed tree transducers we only demand that these dependencies do not form cycles. Therefore, we need the concept of brother graphs.

Definition 4.11 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer and let $\sigma \in \Sigma^{(n)}$. The *brother graph of* σ, denoted by BG_σ, is a directed graph, having as nodes the elements of $[n]$, and an edge from $i \in [n]$ to $j \in [n]$ iff there exist $a \in Att_{syn}$ and $b \in Att_{inh}$ such that $b(\pi j)$ depends directly on $a(\pi i)$ at σ. $\square$

Example 4.12 We show again the dependency graph of σ for ATT_{leafs} and the corresponding brother graph:

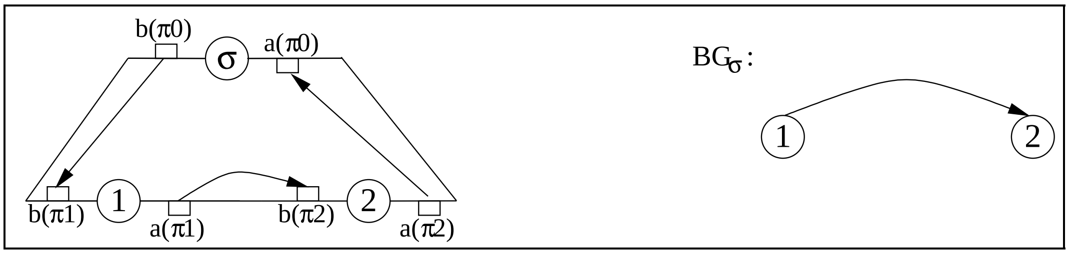

Figure 3: $D_{ATT_{leafs}}(\sigma)$ and the brother graph BG_σ

$\square$

To represent brother graphs in our logical framework, we define a formula *brother* with one free variable for the input symbol and two free variables representing the offsets of two children.

Claim 4.13 Consider the formula:
$brother(x^s, y^i, z^i) = \exists u^a, v^a.(syn(u^a) \wedge inh(v^a) \wedge direct(x^s, v^a, z^i, u^a, y^i))$
Then for every attributed tree transducer ATT, input symbol $\sigma \in \Sigma^{(n)}$ and $i, j \in [n]$:
$(\Phi(ATT), \sigma, i, j) \models brother$ iff the brother graph BG_σ contains an edge from i to j. $\square$

It is easy to see that this claim holds, as it follows the Definition 4.11 of brother graphs.

Having defined brother graphs, we can now give the definition of one–visit attributed tree transducers.

Definition 4.14 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer. ATT is *one–visit* if for every $\sigma \in \Sigma$, the brother graph BG_σ is acyclic. $\square$

Example 4.15 ATT_{leafs} is one–visit, because the brother graphs BG_α and BG_β are empty and above we have seen that BG_σ is acyclic. $\square$

To specify a formula describing the property one–visit, we thus need to test, whether the transitive closure of the relation expressed by *brother* with respect to a fixed input symbol x^s, contains a cycle on elements of the sort of integers. For this, we can use the same construction as was used in the definition of the transitive closure of a binary formula.

Claim 4.16 The following formula $\phi_{one-visit}$ (**MSO**) describes the property of an attributed tree transducer to be one–visit:

$$\neg\exists x^s.(in(x^s) \wedge \exists y^i \forall X^i.(\forall u^i, v^i.(u^i \in X^i \wedge brother(x^s, u^i, v^i) \Rightarrow v^i \in X^i)$$
$$\wedge \forall u^i.(brother(x^s, y^i, u^i) \Rightarrow u^i \in X^i) \Rightarrow y^i \in X^i))$$

$\square$

As was already mentioned in the preliminaries, we can only compute the tree transformation induced by an attributed tree transducer for a certain input tree, if the dependency graph of that tree is acyclic. There is an algorithm, which for a given attributed tree transducer decides, whether the dependency graph of every input tree is acyclic. If the answer is yes, then the attributed tree transducer is called noncircular. Here we will study a property that is more restricted, called *absolutely noncircular*. In order to introduce this notion formally, we have to deal with is–graphs. Intuitively, an attributed tree transducer is absolutely noncircular, if a) there is an is–graph that captures all dependencies that may occur between inherited and synthesized attribute occurrences at some node in some input tree, and b) the composition of every dependency graph with this specific is–graph is acyclic.

Definition 4.17 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer. ATT is *absolutely noncircular* if there is an is–graph IS such that:

1. for every $\sigma \in \Sigma^{(n)}$, the graph $D_{ATT}(\sigma)[IS, \ldots, IS]$ is acyclic, where there are n occurrences of IS and
2. for every $\sigma \in \Sigma^{(n)}$, if there is an edge from $b \in Att_{inh}$ to $a \in Att_{syn}$ in the graph $is(D_{ATT}(\sigma)[\underbrace{IS, \ldots, IS}_{\text{n times}}])$, then there is also an edge from b to a in IS.

□

Example 4.18 ATT_{leafs} is absolutely noncircular, because for the is–graph IS containing an edge from b to a obviously point 2 is fulfilled and we have seen in Example 2.8 that $D_{ATT_{leafs}}(\sigma)[IS, IS]$ is acyclic. □

Note that it is an open problem, whether there are tree transformations, which can be computed by a noncircular attributed tree transducer, but not by an absolutely noncircular one.

If an is–graph IS satisfies point 2 of Definition 4.17, we will call it *closed*. Now we reason that there is a least is–graph among all closed is–graphs and that ATT is absolutely noncircular iff this least closed is–graph satisfies point 1 of Definition 4.17.

Lemma 4.19

a) There is a least closed is–graph IS_{ATT}.

b) ATT is absolutely noncircular iff for every $\sigma \in \Sigma^{(n)}$, $D_{ATT}(\sigma)[IS_{ATT}, \ldots, IS_{ATT}]$ is acyclic.

□

Proof:

a) This is an easy consequence of the fact that there are finitely many is–graphs and that for two closed is–graphs is_1 and is_2, also their intersection $is_1 \cap is_2$ is closed, because for every $\sigma \in \Sigma^{(n)}$ we have:

there is an edge from b to a in $is(D_{ATT}(\sigma)[is_1 \cap is_2, \ldots, is_1 \cap is_2])$

$\Updownarrow$ (by definition)

there is a path from $b(\pi 0)$ to $a(\pi 0)$ in $D_{ATT}(\sigma)[is_1 \cap is_2, \ldots, is_1 \cap is_2]$

$\Downarrow$ ($is_1 \cap is_2 \subseteq is_1$ and $is_1 \cap is_2 \subseteq is_2$)

there is a path from $b(\pi 0)$ to $a(\pi 0)$ in $D_{ATT}(\sigma)[is_1, \ldots, is_1]$ and
there is a path from $b(\pi 0)$ to $a(\pi 0)$ in $D_{ATT}(\sigma)[is_2, \ldots, is_2]$

$\Updownarrow$ (by definition)

there is an edge from b to a in $is(D_{ATT}(\sigma)[is_1, \ldots, is_1])$ and
there is an edge from b to a in $is(D_{ATT}(\sigma)[is_2, \ldots, is_2])$

$\Downarrow$ (by assumption)

there is an edge from b to a in is_1 and there is an edge from b to a in is_2

$\Downarrow$

there is an edge from b to a in $is_1 \cap is_2$

b) The if–part is trivial, because IS_{ATT} is then just a closed is–graph satisfying point 1. The only–if–part holds, because from $IS_{ATT} \subseteq IS$ follows $D_{ATT}(\sigma)[IS_{ATT}, \ldots, IS_{ATT}] \subseteq D_{ATT}(\sigma)[IS, \ldots, IS]$ and thus the former graph is acyclic when the latter is. $\Box$

From the proof of point **a)** it is obvious that IS_{ATT} is the intersection of all closed is–graphs. This observation gives us an approach to describe IS_{ATT} by a logical formula, namely an edge is included in IS_{ATT} iff it is included in every closed is–graph. However, this can not be formalized in monadic second–order logic, because MSO gives us no means for representing is–graphs and quantifying over them. In second–order logic this can be done by using binary relation variables over attributes to represent is–graphs.

In order to describe closed is–graphs, we need to express the existence of certain paths in the composition of a dependency graph with is–graphs. As a first step, we define a formula $dep(x^s, R^{aa}, y^n, z^n)$, which describes that an inside attribute occurrence (represented by its root node z^n) depends on an inherited attribute occurrence at some child node (represented by y^n), if it depends directly on some synthesized attribute occurrence at the same child node and, in turn, this one depends on y^n via some is–graph (represented by R^{aa}).

Claim 4.20 Consider the formula (**SO**):

$$dep(x^s, R^{aa}, y^n, z^n) = \exists p^a, q^i.(rule(x^s, p^a, q^i, y^n) \wedge inh(p^a) \wedge \exists u^a, v^i.(rule(x^s, u^a, v^i, z^n) \\ \wedge \exists w^n, t^a.(edg^*(z^n, w^n) \wedge lab_{att}(w^n, t^a, q^i) \wedge syn(t^a) \wedge R^{aa}(p^a, t^a))))$$

Then, for every attributed tree transducer ATT, $\sigma \in \Sigma^{(n)}$, is–graph IS, and inside attribute occurrences $b(\pi\eta), c'(\pi\eta') \in inner(\sigma)$:
$(\Phi(ATT), \sigma, IS, v_{(\sigma,b,\eta),\epsilon}, v_{(\sigma,c',\eta'),\epsilon}) \models dep$ iff $c'(\pi\eta')$ at σ depends directly on some synthesized attribute occurrence $c(\pi\eta)$ such that there is an edge from b to c in IS. $\Box$

To make this claim more plausible and the formula understandable, we visualize the situation and the roles of the variables in Figure 4 (where, in this example, the inside attribute occurrence $u(\pi v)$ is the occurrence of a synthesized attribute u at the root, i.e., $v = 0$).

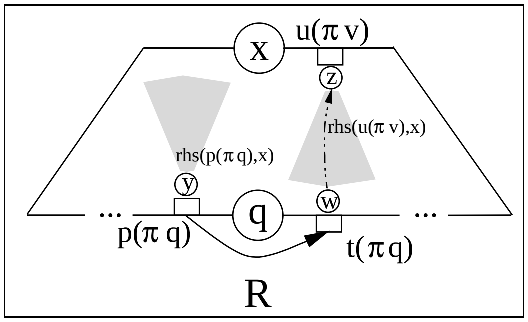

Figure 4: visualization of $dep(x^s, R^{aa}, y^n, z^n)$

The nodes are again visualized in small circles. The shaded areas represent the right–hand sides rooted by the nodes z and y. The node w is a leaf of $rhs(u(\pi v), x)$, labeled by an attribute occurrence $t(\pi q)$. Note that $c'(\pi\eta')$ (or $u(\pi v)$) can be any inside attribute occurrence, not necessarily a synthesized attribute occurrence at the root node as visualized.

By applying the usual construction to obtain the reflexive, transitive closure of *dep* for fixed x^s and R^{aa}, we get a formula describing exactly the dependencies between inside attribute occurrences in the composition of a dependency graph with copies of the same is–graph.

Claim 4.21 Consider the formula:
$dep^*(x^s, R^{aa}, y^n, z^n) = \forall X^n.(\forall u^n, v^n.(u^n \in X^n \wedge dep(x^s, R^{aa}, u^n, v^n) \Rightarrow v^n \in X^n) \wedge y^n \in X^n \Rightarrow z^n \in X^n)$
Then for every attributed tree transducer ATT, $\sigma \in \Sigma^{(n)}$, is–graph IS and inside attribute occurrences $c(\pi\eta), c'(\pi\eta') \in inner(\sigma)$:
$(\Phi(ATT), \sigma, IS, v_{(\sigma,c,\eta),\epsilon}, v_{(\sigma,c',\eta'),\epsilon}) \models dep^*$ iff there is a (possibly empty) path from $c(\pi\eta)$ to $c'(\pi\eta')$ in $D_{ATT}(\sigma)[IS, \ldots, IS]$. $\square$

Note that dep^* is not a monadic second–order formula, because it uses dep.

Now we are in the position to express that an is–graph (represented as a binary relation over the sort of attributes) is closed.

Claim 4.22 Consider the following formula:
$closed(R^{aa}) = \forall r^a, u^a.(inh(r^a) \wedge syn(u^a) \wedge \exists x^s.(in(x^s) \wedge \exists v^i, z^n.(rule(x^s, u^a, v^i, z^n) \wedge \exists p^a, q^i, y^n.(rule(x^s, p^a, q^i, y^n) \wedge \exists s^n.(edg^*(y^n, s^n) \wedge lab_{att}(s^n, r^a, v^i) \wedge dep^*(x^s, R^{aa}, y^n, z^n))))) \Rightarrow R^{aa}(r^a, u^a))$
Then for every attributed tree transducer ATT and is–graph $IS \subseteq Att_{inh} \times Att_{syn}$:
$(\Phi(ATT), IS) \models closed$ iff IS is a closed is–graph for ATT. $\square$

This is achieved by expressing that, for every possible inherited attribute r and synthesized attribute u, there is an edge in the is–graph R, if for some input symbol x, there is a path from $r(\pi 0)$ to $u(\pi 0)$ in the composition of the corresponding dependency graph with copies of the is–graph R; the latter property is fulfilled if there is some inside attribute occurrence $p(\pi q)$ (represented by the root node y of its right–hand side) that depends directly on $r(\pi 0)$ (a descendant s of y is labeled by $r(\pi v)$), and $u(\pi 0)$ (represented by z) depends on $p(\pi q)$ by dep^*. Note that v^i will always be the 0 of the sort of integers, because synthesized attributes occur as inner attributes only at the root and an outer inherited attribute may also only occur at the root.

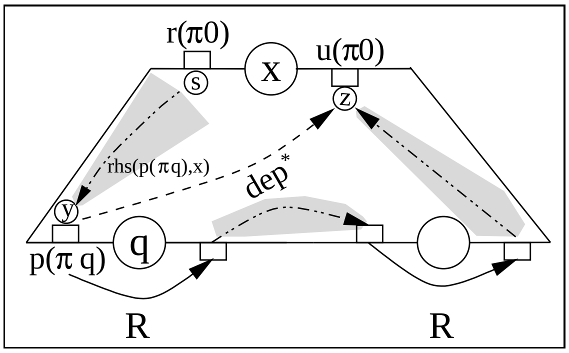

Figure 5: visualization of $closed(R^{aa})$

Using the characterization of closed is–graphs, we can now exploit our observation that the graph IS_{ATT} is exactly the intersection of all closed is–graphs.

Claim 4.23 Consider the following formula (**SO**):
$is(x^a, y^a) = \forall R^{aa}.(closed(R^{aa}) \Rightarrow R^{aa}(x^a, y^a))$
Then for every attributed tree transducer ATT and attributes b and a:
$(\Phi(ATT), b, a) \models is$ iff there is an edge from b to a in IS_{ATT}. $\square$

Note that the formula does not explicitly state that x^a and y^a represent an inherited and synthesized attribute, respectively. However, this is forced implicitly by the formula $closed$.

To complete the description of absolutely noncircularity, we need one further formula, describing dependencies in the composition of dependency graphs with IS_{ATT}. This formula is very similar to *dep* but it uses the formula *is* instead of a relation variable R^{aa} and it restricts u^a to be an inherited attribute, because inside synthesized attribute occurrences can not occur in a cycle. Again using the known construction for the transitive closure, we get the describing formula ϕ_{anc}.

Claim 4.24 The following formula ϕ_{anc} (**SO**) describes the property of an attributed tree transducer to be absolutely noncircular:

$$\neg\exists x^s.(in(x^s) \wedge \exists y^n.\forall X^n(\forall u^n, v^n.(u^n \in X^n \wedge dep_{is}(x^s, u^n, v^n) \Rightarrow v^n \in X^n) \\ \wedge\forall u^n.(dep_{is}(x^s, y^n, u^n) \Rightarrow u^n \in X^n) \Rightarrow y^n \in X^n))$$

where:

$$dep_{is}(x^s, y^n, z^n) = \exists p^a, q^i.(rule(x^s, p^a, q^i, y^n) \wedge inh(p^a) \wedge \exists u^a, v^i.(rule(x^s, u^a, v^i, z^n) \wedge inh(u^a) \\ \wedge\exists w^n, t^a.(edg^*(z^n, w^n) \wedge lab_{att}(w^n, t^a, q^i) \wedge syn(t^a) \wedge is(p^a, t^a))))$$

□

4.3 simple κ–visit, –sweep, –alternating pass and –pass

In this subsection we want to study the properties of attributed tree transducers to be simple κ–visit, simple κ–sweep, simple κ–alternating pass and simple κ–pass. These properties are defined in [14] as adoption from the corresponding subclasses of attribute grammars (cf. [16]). On attribute grammars, the properties are defined in terms of certain algorithms for computing attribute values, so–called attribute evaluators. In the context of attributed tree transducers, simple κ–visit means that there is a κ–partition $\langle A_1, \ldots, A_\kappa\rangle$ of the set of attributes and, for every input symbol $\sigma \in \Sigma^{(n)}$, with the ith component a visit sequence $i(1)\cdots i(m)$ with $1 \leq i(1), \ldots, i(m) \leq n$ can be associated, such that the set of all attribute occurrences of each input tree t can be evaluated as follows: every node n of t, labeled by σ, is visited κ times and during the ith visit to a node, 1) the inherited attributes in A_i at node n are evaluated, then 2) the node's children are visited in the order $i(1)$ to $i(m)$, and then 3) the synthesized attributes in A_i at node n are evaluated.

However, the definitions for attributed tree transducers in [14] are not based on attribute evaluators, but on partition graphs. Since we will follow these definitions, we first define the concept of partition graphs.

Definition 4.25 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer and let $P = \langle A_1, \cdots, A_\kappa\rangle$ be an ordered partition of Att. For $\sigma \in \Sigma^{(n)}$, the *P–partition graph of* σ, denoted by $D_{ATT}(\sigma)\{P\}$, is the graph defined as follows (where for a set of attributes $A \subseteq Att$, we define $IA = A \cap Att_{inh}$ and $SA = A \cap Att_{syn}$):

1. The nodes of $D_{ATT}(\sigma)\{P\}$ are the sets $IA_1(\{\pi\eta\}), SA_1(\{\pi\eta\}), \ldots, IA_\kappa(\{\pi\eta\}), SA_\kappa(\{\pi\eta\})$, where $\eta \in \{0\} \cup [n]$.
2. In $D_{ATT}(\sigma)\{P\}$, there are the sequence edges of the set S and the dependency edges of the set D:
   (a) for every $\eta \in \{0\} \cup [n]$ and every $1 \leq i \leq \kappa$, S contains an edge from $IA_i(\{\pi\eta\})$ to $SA_i(\{\pi\eta\})$, and for every $1 \leq i < \kappa$ an edge from $SA_i(\{\pi\eta\})$ to $IA_{i+1}(\{\pi\eta\})$,
   (b) for every $\eta, \eta' \in \{0\} \cup [n]$, $1 \leq i, i' \leq \kappa$ and $V, V' \in \{I, S\}$, D contains an edge from $V'A_{i'}(\{\pi\eta'\})$ to $VA_i(\{\pi\eta\})$, if there is an attribute occurrence $b(\pi\eta')$ in $V'A_{i'}(\{\pi\eta'\})$ and an attribute occurrence $a(\pi\eta)$ in $VA_i(\{\pi\eta\})$ such that $a(\pi\eta)$ depends directly on $b(\pi\eta')$ at σ. □

Note that every dependency edge goes from a set $IA_{i'}(\{\pi 0\})$ or $SA_{i'}(\{\pi\eta'\})$ to a set $SA_i(\{\pi 0\})$ or $IA_i(\{\pi\eta\})$, where $1 \le \eta, \eta'$.

Example 4.26 We show $D_{ATT_{leafs}}(\sigma)\{P\}$ in Figure 6, where P is the ordered partition $\langle\{b\},\{a\}\rangle$.

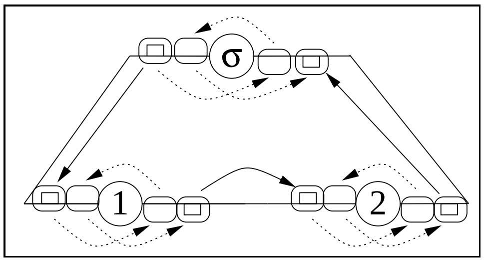

Figure 6: the partition graph $D_{ATT_{leafs}}(\sigma)\{P\}$

The nodes of the partition graph are shown as rectangles with smooth corners. For example, $IA_1(\{\pi 0\}) = \{b(\pi 0)\}$ is shown as the rectangle to the very left of σ, whereas $IA_2(\{\pi 0\})$ is empty. Sequence edges are shown by dotted, dependency edges by solid arrows. $\square$

Now we can formally define the property of an attributed tree transducer to be simple κ–visit.

Definition 4.27 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer and let $P = \langle A_1, \cdots, A_\kappa\rangle$ be an ordered partition of Att. ATT is *simple κ–visit with respect to* P if, for every $\sigma \in \Sigma$, the P–partition graph $D_{ATT}(\sigma)\{P\}$ is acyclic. ATT is *simple κ–visit* if there is a κ–partition P of Att such that ATT is simple κ–visit with respect to P. $\square$

Example 4.28 It is easy to check that ATT_{leafs} is simple 2–visit with respect to the partition $P = \langle\{b\},\{a\}\rangle$. $\square$

Note that, for the special case $\kappa = 1$, the property simple 1–visit corresponds to the already described property of an attributed tree transducer to be one–visit.

It turns out that the definitions of partition graph and simple κ–visit as given are not suitable for being expressed in our logical framework. This is mainly due to the fact that we can not easily formalize a graph of sets, because this would be a binary relation over unary relations and thus it would go beyond second–order logic. However, we can change the definitions in such a way that expressing the concept of simple κ–visit in our logical representation of attributed tree transducers becomes viable. Of course, we have to ensure that changing the definitions does not change the concept they are defining. That is, the definition obtained at the end should characterize exactly the class of simple κ–visit attributed tree transducers. To do so, we will change the Definitions 4.25 and 4.27 of partition graph and simple κ–visit in small steps, each of which we will justify and make plausible.

The general idea is to replace a partition graph edge between two sets of attribute occurrences by all edges going from an element of the first to an element of the second set in the dependency graph. However, there may be sequence edges starting or ending in an empty set. These would not be considered anymore (because no edges in the dependency graph would be produced for them) and thus the existence of a cycle (such a sequence edge **may** occur in a cycle) and the property of being simple κ–visit or not, would not be preserved. To avoid this, we consider the transitive closure of the set of sequence edges, because then we can disregard edges starting or ending in an empty set.

Claim 4.29 The property of an attributed tree transducer to be simple κ–visit does not change, if in the partition graph definition as sequence edges instead of S we take S' with: S' contains all edges of the transitive closure of S, which do not start or end in an empty set. □

Justification: Obviously, the property of the partition graph to have or not to have cycles is not changed by adding transitive edges. Further, when deleting edges that start or end in a node that is an empty set, we will clearly not create new cycles, but we will also do no harm to existing cycles, because nodes that represent empty sets can not have incoming or outgoing dependency edges, but only sequence edges. Thus, deleting them will not influence the existence or nonexistence of cycles, because they will be bridged by transitive edges. □

The transitive closure of the set of sequence edges is redundant in the sense that it also adds sequence edges that are of no importance, because the restrictions they impose on the order of attribute evaluation are already implied by other sequence edges. Basically, we are only interested in sequence edges going from inherited attribute occurrences to synthesized ones and vice versa.

Lemma 4.30 The property of an attributed tree transducer to be simple κ–visit does not change, if in the partition graph definition as sequence edges instead of S' we take S'' with: S'' contains all edges of S' that go from an I–set to a S–set or vice versa. □

Proof: We argue that without influencing the existence of a cycle, we can delete sequence edges that go from an I–set to an I–set or from a S–set to a S–set at the root or at a child node of some σ–pattern, respectively. This holds, because in all four cases we can check that if the sequence edge under question occurs in a cycle, then in the cycle there is a preceding or succeeding sequence edge such that they can be bridged by an edge from S''. We can easily see this from Figure 7, where the reasoning is as follows:
If there is a cycle containing a sequence edge of the kind **1**, then there must be a node **n** such that the cycle also contains the sequence edge **2**, and consequently we can delete edge **1** without influencing the existence of a cycle, because they are bridged by a sequence edge **3** out of S''.
For the first case (illustrated to the top–left), e.g.: Consider a sequence edge **1** from a set $IA_i(\{\pi 0\})$ to a set $IA_j(\{\pi 0\})$ $(i \leq j)$ at the root of the partition graph for σ. If edge **1** occurs in a cycle, then in the cycle it can not be preceded by a dependency edge, because there are no dependency edges going to $IA_i(\{\pi 0\})$. Consequently, there must be some $n < i$ such that the edge **2** from the set $SA_n(\{\pi 0\})$ to $IA_i(\{\pi 0\})$ occurs in the cycle preceding edge **1**. Since $n < j$, there is an edge **3** from $SA_n(\{\pi 0\})$ to $IA_j(\{\pi 0\})$ that bridges edges **2** and **1**.

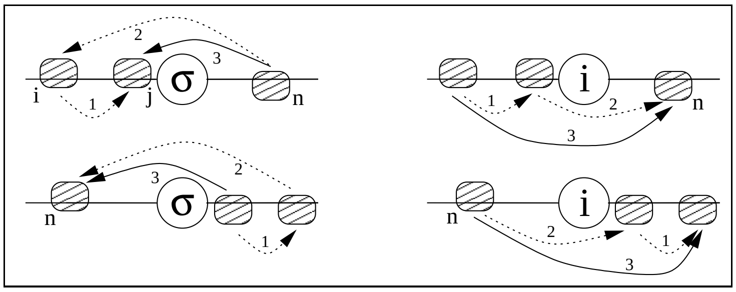

Figure 7: the four cases for the proof of Lemma 4.30

□

By constructing S'' from S and enumerating the edges in S'', we know that the class of attributed tree transducers, which are characterized as simple κ–visit, does not change, if we change point 2.(a) of Definition 4.25 into:

(a) for every $\eta \in \{0\} \cup [n]$ and every $1 \leq i \leq j \leq \kappa$, S contains an edge from $IA_i(\{\pi\eta\})$ to $SA_j(\{\pi\eta\})$ iff both are nonempty and for every $1 \leq i < j \leq \kappa$ an edge from $SA_i(\{\pi\eta\})$ to $IA_j(\{\pi\eta\})$ iff both are nonempty

This modified partition graph will be denoted by $D'_{ATT}(\sigma)\{P\}$.

Example 4.31 According to the changed definition, the partition graph of ATT_{leafs} for σ with respect to the partition $P = \langle\{b\}, \{a\}\rangle$ is the graph shown in Figure 8.

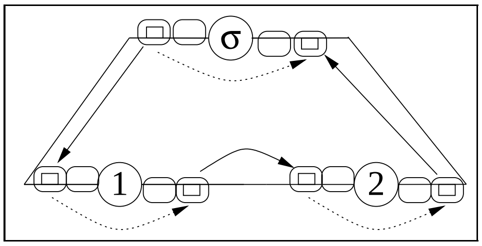

Figure 8: the modified partition graph $D'_{ATT_{leafs}}(\sigma)\{P\}$

$\square$

The definition of simple κ–visit forbids the existence of any cycles in the partition graphs. We will see that in our modified partition graphs we actually only have to look for cycles of a specific form (namely cycles with alternating sequence and dependency edges), because whenever there is any cycle, there will also be such a specific cycle. Thus, we make another change to Definition 4.27, in that we only require that there are no cycles in the modified partition graph of this particular kind.

Claim 4.32 Taking the definition of partition graphs as modified above, the property of an attributed tree transducer to be simple κ–visit does not change, if in the definition of simple κ–visit instead of "*for every $\sigma \in \Sigma$, the P–partition graph $D_{ATT}(\sigma)\{P\}$ is acyclic*", we require that "*for every $\sigma \in \Sigma$, the modified P–partition graph $D'_{ATT}(\sigma)\{P\}$ does not contain a cycle, in which sequence and dependency edges alternate*". $\square$

Justification: We show that, when the modified partition graph $D'_{ATT}(\sigma)\{P\}$ contains a cycle, it also contains a cycle, in which sequence and dependency edges alternate. Therefore, among all possible cycles consider one with minimal number of sequence edges. Obviously, no two dependency edges can occur consecutively. On the other hand, when two sequence edges occur consecutively, we can, similarly to the proof of Lemma 4.30, check in again four cases that there are indeed three consecutive sequence edges that can be bridged by one sequence edge. Consequently, a cycle with minimal number of sequence edges can **not** contain two consecutive sequence edges. To make this plausible, we visualize the four cases in Figure 9, where the reasoning is as follows:
If there is a cycle with two consecutive sequence edges of the kinds **1** and **2**, then there must be a node **n** such that the cycle also contains the sequence edge **3**, and consequently there is also a cycle without edges **1** and **2**, because we can bridge by a sequence edge **4**.
For the first case (illustrated to the top–left), e.g.: Consider two consecutive sequence edges **1** and **2** from a set $IA_i(\{\pi 0\})$ to a set $SA_j(\{\pi 0\})$ and from $SA_j(\{\pi 0\})$ to a set $IA_k(\{\pi 0\})$ $(i \leq j < k)$, respectively, at the root of the modified partition graph for σ. If the edges **1** and **2** occur in a cycle, then in the cycle edge **1** can not be preceded by a dependency edge, because there are no dependency edges going to $IA_i(\{\pi 0\})$. Consequently, there must be some $n < i$ such that the edge **3** from the set $SA_n(\{\pi 0\})$ to $IA_i(\{\pi 0\})$ occurs in the

cycle preceding edge **1**. Since $n < k$, there is an edge **4** from $SA_n(\{\pi 0\})$ to $IA_k(\{\pi 0\})$ that bridges edges **3**, **1** and **2**.

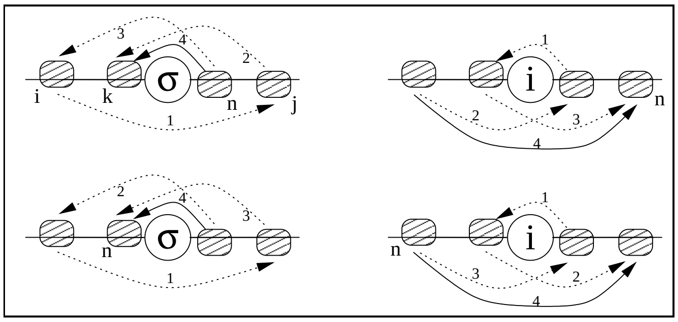

Figure 9: the four cases for the justification of Claim 4.32

□

Now we argue that we do not at all need partition graphs to characterize the property of an attributed tree transducer to be simple κ–visit, but can express the same by adding sequence edges directly to the dependency graph, thus constructing a so–called *sequence graph*. For every sequence edge of the modified partition graph, going from a set $V'A_i(\{\pi\eta\})$ to a set $VA_j(\{\pi\eta\})$ (with $V, V' \in \{I, S\}$), we add to the dependency graph a sequence edge from $c'(\pi\eta)$ to $c(\pi\eta)$ for every pair of attribute occurrences $c'(\pi\eta)$ and $c(\pi\eta)$ with $c' \in V'A_i$ and $c \in VA_j$.

Definition 4.33 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer and let $P = \langle A_1, \cdots, A_\kappa \rangle$ be an ordered partition of Att. For $\sigma \in \Sigma^{(n)}$, the *P–sequence graph of* σ, denoted by $S_{ATT}(\sigma)\{P\}$, is obtained from the dependency graph $D_{ATT}(\sigma)$ (whose edges are called dependency edges) by adding for every $\eta \in \{0\} \cup [n]$, $b \in Att_{inh}$ and $a \in Att_{syn}$ sequence edges as follows:

- an edge from $b(\pi\eta)$ to $a(\pi\eta)$ iff $ind_P(b) \leq ind_P(a)$,
- an edge from $a(\pi\eta)$ to $b(\pi\eta)$ iff $ind_P(a) < ind_P(b)$.

□

Example 4.34 The sequence graph of ATT_{leafs} for σ with respect to the partition $P = \langle \{b\}, \{a\} \rangle$ is the graph shown in Figure 10. Again, sequence edges are shown by dotted, dependency edges by solid arrows.

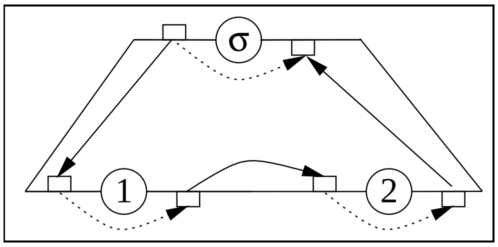

Figure 10: the sequence graph $S_{ATT_{leafs}}(\sigma)\{P\}$

□

By Claim 4.32 we know that an attributed tree transducer ATT is simple κ–visit iff the modified partition graph $D'_{ATT}(\sigma)\{P\}$ does not contain a cycle with alternating sequence and dependency edges. By the following lemma we transfer this knowledge to sequence graphs.

Lemma 4.35 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer and let $P = \langle A_1, \cdots, A_\kappa \rangle$ be an ordered partition of Att. For every $\sigma \in \Sigma$, the modified P–partition graph $D'_{ATT}(\sigma)\{P\}$ contains a cycle with alternating sequence and dependency edges iff the P–sequence graph $S_{ATT}(\sigma)\{P\}$ contains a cycle with alternating sequence and dependency edges. □

Proof: The if–part is obvious, because a cycle with alternating sequence and dependency edges in $S_{ATT}(\sigma)\{P\}$ immediately gives a corresponding cycle in $D'_{ATT}(\sigma)\{P\}$ by considering for every edge from $c'(\pi\eta')$ to $c(\pi\eta)$ in the sequence graph the corresponding edge of the modified partition graph between the two sets of attribute occurrences, to which $c'(\pi\eta')$ and $c(\pi\eta)$ belong.
For the only–if–part, consider a cycle of length $2n$ with alternating sequence and dependency edges in the modified P–partition graph $D'_{ATT}(\sigma)\{P\}$:

$$V_1 A_{i_1}(\{\pi\eta_1\}) \underbrace{-}_{s_1} V_2 A_{i_2}(\{\pi\eta_1\}) \underbrace{-}_{d_1} \cdots \underbrace{-}_{s_n} V_{2n} A_{i_{2n}}(\{\pi\eta_n\}) \underbrace{-}_{d_n} V_1 A_{i_1}(\{\pi\eta_1\}),$$

starting with sequence edge s_1 and ending with dependency edge d_n.
We see that for every $1 \le k \le n$, there is a dependency edge d_k in $D'_{ATT}(\sigma)\{P\}$ from $V_{2k} A_{i_{2k}}(\{\pi\eta_k\})$ to $V_{2k+1} A_{i_{2k+1}}(\{\pi\eta_{k+1}\})$ (indices for V and i modulo $2n$, for η modulo n). But this can only be the case, if we have attribute occurrences $c_{2k}(\pi\eta_k) \in V_{2k} A_{i_{2k}}(\{\pi\eta_k\})$ and $c_{2k+1}(\pi\eta_{k+1}) \in V_{2k+1} A_{i_{2k+1}}(\{\pi\eta_{k+1}\})$ such that $c_{2k+1}(\pi\eta_{k+1})$ depends directly on $c_{2k}(\pi\eta_k)$ at σ. Consequently, we have attribute occurrences $c_1(\pi\eta_1), c_2(\pi\eta_1), \ldots, c_{2n}(\pi\eta_n)$ and dependency edges d'_k in $S_{ATT}(\sigma)\{P\}$ with:

$$c_1(\pi\eta_1) \underbrace{-}_{?} c_2(\pi\eta_1) \underbrace{-}_{d'_1} \cdots \underbrace{-}_{?} c_{2n}(\pi\eta_n) \underbrace{-}_{d'_n} c_1(\pi\eta_1)$$

It is easy to see that from this we can construct a cycle with alternating sequence and dependency edges in the sequence graph by filling the ?–places with sequence edges $s'_1, \ldots, s'_n$, which must exist, because in the modified partition graph we have the sequence edges $s_1, \ldots, s_n$ and by Definition 4.33 there is a sequence edge from $c_{2k-1}(\pi\eta_k) \in V_{2k-1} A_{i_{2k-1}}(\{\pi\eta_k\})$ to $c_{2k}(\pi\eta_k) \in V_{2k} A_{i_{2k}}(\{\pi\eta_k\})$ in $S_{ATT}(\sigma)\{P\}$, whenever there is a sequence edge s_k from $V_{2k-1} A_{i_{2k-1}}(\{\pi\eta_k\})$ to $V_{2k} A_{i_{2k}}(\{\pi\eta_k\})$ in $D'_{ATT}(\sigma)\{P\}$. □

From Lemma 4.35 we get a characterization of simple κ–visit in terms of sequence graphs.

Definition 4.36 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer and let $P = \langle A_1, \cdots, A_\kappa \rangle$ be an ordered partition of Att. ATT is *simple κ–visit with respect to* P if, for every $\sigma \in \Sigma$, the P–sequence graph $S_{ATT}(\sigma)\{P\}$ does not contain a cycle with alternating sequence and dependency edges. ATT is *simple κ–visit* if there is a κ–partition P of Att such that ATT is simple κ–visit with respect to P. □

Using this definition, we are able to describe the property of an attributed tree transducer to be simple κ–visit by a monadic second–order formula. Before we do so, we need some auxiliary formulas and constructions. We will represent a κ–partition of the attributes by κ sets of sort a. To express that κ such sets indeed form a partition, we introduce a formula, or rather a formula scheme that, with respect to an ordered partition P, defines orders on attributes. This is done by comparing attributes by their index ind_P.

Claim 4.37 For $\kappa \in \mathbb{N}$ and a relation $R \subseteq \mathbb{N} \times \mathbb{N}$, consider the formula:
$ord_{R,\kappa}(x^a, y^a, {X_1}^a, \ldots, {X_\kappa}^a) = \bigvee \{x^a \in {X_i}^a \wedge y^a \in {X_j}^a \mid 1 \le iRj \le \kappa\}$
Then for every structure S, elements $a, b \in S_a$ and every κ–partition $\langle A_1, \ldots, A_\kappa \rangle$ of S_a:
$(S, a, b, A_1, \ldots, A_\kappa) \models ord_{R,\kappa}$ iff $ind_{\langle A_1, \ldots, A_\kappa \rangle}(a) \; R \; ind_{\langle A_1, \ldots, A_\kappa \rangle}(b)$. □

Note that the right–hand side given for $ord_{R,\kappa}$ is not really a formula according to the syntactic definition of second–order formulas, rather it is a scheme, how to construct the (finite !) formula for given R and κ. We will use this kind of abbreviations throughout the remainder of the paper; it should be clear that they well–define the formulas under question.

Example 4.38 For $\kappa = 3$ and $R =<$, we get:

$$\begin{aligned} ord_{<,3}(x^a, y^a, X_1{}^a, X_2{}^a, X_3{}^a) &= \bigvee\{x^a \in X_i{}^a \wedge y^a \in X_j{}^a \mid 1 \leq i < j \leq 3\} \\ &= x^a \in X_1{}^a \wedge y^a \in X_2{}^a \\ &\quad \vee x^a \in X_1{}^a \wedge y^a \in X_3{}^a \\ &\quad \vee x^a \in X_2{}^a \wedge y^a \in X_3{}^a \end{aligned}$$

□

From the definition of an ordered partition in the preliminaries we easily get the following formula for expressing that a collection of sets forms a partition.

Claim 4.39 For $\kappa \in \mathbb{N}$, consider the formula (**MSO**):

$$part_\kappa(X_1{}^a, \ldots, X_\kappa{}^a) = \neg\exists x^a.(ord_{<,\kappa}(x^a, x^a, X_1{}^a, \ldots, X_\kappa{}^a)) \wedge \forall x^a.(\bigvee\{x^a \in X_i{}^a \mid 1 \leq i \leq \kappa\})$$

Then for every structure S and sets $A_1, \ldots, A_\kappa \subseteq S_a$:
$(S, A_1, \ldots, A_\kappa) \models part_\kappa$ iff $\langle A_1, \ldots, A_\kappa\rangle$ is an ordered partition of S_a. □

The formula expresses that no attribute occurs in two components of the partition, but every attribute occurs in some component.

From Definition 4.33 of sequence graphs we immediately get the characterization of sequence edges by a formula.

Claim 4.40 For $\kappa \in \mathbb{N}$, consider the formula:

$$\begin{aligned} seq_\kappa(x^a, y^a, X_1{}^a, \ldots, X_\kappa{}^a) &= inh(x^a) \wedge syn(y^a) \wedge ord_{\leq,\kappa}(x^a, y^a, X_1{}^a, \ldots, X_\kappa{}^a) \\ &\quad \vee syn(x^a) \wedge inh(y^a) \wedge ord_{<,\kappa}(x^a, y^a, X_1{}^a, \ldots, X_\kappa{}^a) \end{aligned}$$

Then for every attributed tree transducer ATT, attributes $a, b \in Att$, every ordered partition $P = \langle A_1, \ldots, A_\kappa\rangle$ of Att, every P–sequence graph of ATT and every occurrence η of this sequence graph:
$(\Phi(ATT), a, b, A_1, \ldots, A_\kappa) \models seq_\kappa$ iff there is a sequence edge from $a(\pi\eta)$ to $b(\pi\eta)$ in the P–sequence graph. □

Since we have to check for cycles with alternating sequence and dependency edges, we define a formula $depseq_\kappa$ on inside attribute occurrences, which expresses that the second attribute occurrence (represented by the root node z^n of its right–hand side) depends on the first (represented by y^n) by the consecution of a sequence and a dependency edge.

Claim 4.41 For $\kappa \in \mathbb{N}$, consider the formula:

$$\begin{aligned} depseq_\kappa(x^s, y^n, z^n, X_1{}^a, \ldots, X_\kappa{}^a) = \exists u^a, v^i, p^a, q^i.(&rule(x^s, u^a, v^i, y^n) \wedge rule(x^s, p^a, q^i, z^n) \\ &\wedge \exists t^n, r^a.(edg^*(z^n, t^n) \wedge lab_{att}(t^n, r^a, v^i) \wedge seq_\kappa(u^a, r^a, X_1{}^a, \ldots, X_\kappa{}^a))) \end{aligned}$$

Then for every attributed tree transducer ATT, $\sigma \in \Sigma^{(n)}$, inside attribute occurrences $c(\pi\eta), c'(\pi\eta') \in inner(\sigma)$ and every ordered partition $P = \langle A_1, \ldots, A_\kappa\rangle$ of Att:
$(\Phi(ATT), \sigma, v_{(\sigma,c,\eta),\epsilon}, v_{(\sigma,c',\eta'),\epsilon}, A_1, \ldots, A_\kappa) \models depseq_\kappa$ iff there as an attribute occurrence $a(\pi\eta)$ at σ such that in the P–sequence graph $S_{ATT}(\sigma)\{P\}$ there is a sequence edge from $c(\pi\eta)$ to $a(\pi\eta)$ and a dependency edge from $a(\pi\eta)$ to $c'(\pi\eta')$. □

To make this claim plausible, Figure 11 visualizes what such a situation could look like. There, $u(\pi v)$ corresponds to the attribute occurrence $c(\pi\eta)$ (in this case $\eta = 0$) and $p(\pi q)$ to the attribute occurrence $c'(\pi\eta')$; the intermediate attribute occurrence $a(\pi\eta)$ is represented by $r(\pi v)$. The dotted sequence edge is implied by the test of $seq_\kappa(u^a, r^a, X_1{}^a, \ldots, X_\kappa{}^a)$, the dependency edge originates from a leaf node t in the right–hand side of $p(\pi q)$, labeled by $r(\pi v)$.

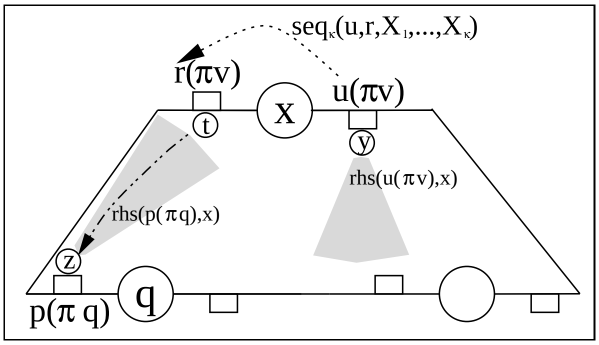

Figure 11: visualization of $depseq_\kappa(x^s, y^n, z^n, X_1{}^a, \ldots, X_\kappa{}^a)$

By applying the known construction for the transitive closure of $depseq_\kappa$ (with fixed x^s and $X_1{}^a, \ldots, X_\kappa{}^a$), we can express the restriction that there are no cycles with alternating sequence and dependency edges in the sequence graph for any input symbol by a monadic second–order formula, and thus describe the property simple κ–visit.

Claim 4.42 For $\kappa \in \mathbb{N}$, the following formula $\phi_{s\kappa v}$ (**MSO**) describes the property of an attributed tree transducer to be simple κ–visit:
$\exists X_1{}^a, \ldots, X_\kappa{}^a.(part_\kappa(X_1{}^a, \ldots, X_\kappa{}^a) \wedge visit_\kappa(X_1{}^a, \ldots, X_\kappa{}^a))$
where:

$$
\begin{aligned}
visit_\kappa(X_1{}^a, \ldots, X_\kappa{}^a) = \neg\exists x^s.(in(x^s) \wedge \exists y^n \forall Y^n.(&\\
\forall u^n, v^n.(u^n \in Y^n \wedge depseq_\kappa(x^s, u^n, v^n, X_1{}^a, \ldots, X_\kappa{}^a) \Rightarrow v^n \in Y^n)&\\
\wedge \forall u^n.(depseq_\kappa(x^s, y^n, u^n, X_1{}^a, \ldots, X_\kappa{}^a) \Rightarrow u^n \in Y^n) \Rightarrow y^n \in Y^n))&
\end{aligned}
$$

□

Note that by the remark after Example 4.28 we have that in the case $\kappa = 1$, the formula ϕ_{s1v} should characterize exactly the same attributed tree transducers as the formula $\phi_{one-visit}$ from Claim 4.16.

The properties simple κ–sweep, simple κ–alternating pass and simple κ–pass are obtained by restricting the partitions $\langle A_1, \ldots, A_\kappa\rangle$ (and associated visit sequences) with respect to which an attributed tree transducer shall be simple κ–visit. This corresponds to special kinds of attribute evaluators.

For simple κ–sweep we have that for every input symbol $\sigma \in \Sigma^{(n)}$, the visit sequence $i(1)\cdots i(m)$, which the evaluator associates with the ith component A_i, is a permutation of the numbers $1, \ldots, n$. Thus, a sweep means that every child node is visited exactly once. The formal restrictions to assure that every visit sequence can be a sweep (as taken from [14]), are summarized in the following definition.

Definition 4.43 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer and let $P = \langle A_1, \cdots, A_\kappa\rangle$ be an ordered partition of Att. ATT is *simple κ–sweep with respect to P* if it is simple κ–visit with respect to P and for every $\sigma \in \Sigma^{(n)}$, the following conditions hold:

1. for every $1 \le i \le \kappa$, $a \in SA_i$, $a' \in Att_{syn}$ and $1 \le \eta' \le n$, if $a(\pi 0)$ at σ depends directly on $a'(\pi\eta')$, then $a' \in SA_1 \cup \ldots \cup SA_i$,

2. for every $1 \le i \le \kappa$, $b \in IA_i$, $b' \in Att_{inh}$ and $1 \le \eta \le n$, if $b(\pi\eta)$ at σ depends directly on $b'(\pi 0)$, then $b' \in IA_1 \cup \ldots \cup IA_i$ and

3. for every $1 \le i \le \kappa$, $b \in IA_i$, $a \in Att_{syn}$ and $1 \le \eta, \eta' \le n$, if $b(\pi\eta)$ at σ depends directly on $a(\pi\eta')$, then:
   - if $\eta = \eta'$, then $a \in SA_1 \cup \ldots \cup SA_{i-1}$, else $a \in SA_1 \cup \ldots \cup SA_i$.

ATT is *simple κ–sweep* if there is a κ–partition P of Att such that ATT is simple κ–sweep with respect to P. □

Example 4.44 ATT_{leafs} is not simple 2–sweep with respect to the partition $P = \langle \{b\}, \{a\} \rangle$, because $b(\pi 2)$ at σ depends directly on $a(\pi 1)$, but $ind_P(a) > ind_P(b)$ and thus condition 3. of Definition 4.43 is not fulfilled. □

We can formalize the property simple κ–sweep by using our formulas for the description of simple κ–visit and express the additional restrictions on the partition P by a monadic second–order formula $sweep_\kappa({X_1}^a, \ldots, {X_\kappa}^a)$.

Claim 4.45 For $\kappa \in \mathbb{N}$, consider the formula:

$$\begin{aligned} sweep_\kappa({X_1}^a, \ldots, {X_\kappa}^a) = \forall x^s, y^a, w^i, z^a, t^i.(&in(x^s) \wedge direct(x^s, y^a, w^i, z^a, t^i) \\ &\Rightarrow (syn(y^a) \wedge syn(z^a) \Rightarrow ord_{\leq,\kappa}(z^a, y^a, {X_1}^a, \ldots, {X_\kappa}^a)) \\ &\wedge (inh(y^a) \wedge inh(z^a) \Rightarrow ord_{\leq,\kappa}(z^a, y^a, {X_1}^a, \ldots, {X_\kappa}^a)) \\ &\wedge (inh(y^a) \wedge syn(z^a) \Rightarrow (w^i = t^i \wedge ord_{<,\kappa}(z^a, y^a, {X_1}^a, \ldots, {X_\kappa}^a) \\ &\qquad \vee w^i \neq t^i \wedge ord_{\leq,\kappa}(z^a, y^a, {X_1}^a, \ldots, {X_\kappa}^a)))) \end{aligned}$$

Then for every attributed tree transducer ATT and every ordered partition $P = \langle A_1, \ldots, A_\kappa \rangle$ of Att:

$(\Phi(ATT), A_1, \ldots, A_\kappa) \models sweep_\kappa$ iff conditions 1. to 3. of Definition 4.43 hold. □

We can easily see that this claim holds by considering that x^s takes the role of σ, the variables w^i and t^i take the roles of η and η', respectively, and y^a and z^a the roles of the attributes under question in conditions 1. to 3.

Claim 4.46 For $\kappa \in \mathbb{N}$, the following formula $\phi_{s\kappa s}$ (**MSO**) describes the property of an attributed tree transducer to be simple κ–sweep:

$\exists {X_1}^a, \ldots, {X_\kappa}^a.(part_\kappa({X_1}^a, \ldots, {X_\kappa}^a) \wedge visit_\kappa({X_1}^a, \ldots, {X_\kappa}^a) \wedge sweep_\kappa({X_1}^a, \ldots, {X_\kappa}^a))$ □

To describe the next property, namely simple κ–alternating pass (short: simple κ–alt pass), the partition, with respect to which an attributed tree transducer is simple κ–visit, is further restricted. Informally, an attributed tree transducer is simple κ–alt pass, if it is simple κ–sweep such that every sweep can be either a left–to–right pass or a right–to–left pass, that is, for every $\sigma \in \Sigma^{(n)}$, the visit sequence $i(1) \cdots i(m)$ associated with the ith component A_i of the partition is either the permutation $1 \cdots n$ or $n \cdots 1$, respectively. Therefore, the third condition from Definition 4.43 is strengthened as follows (taken from [14]).

Definition 4.47 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer, let $P = \langle A_1, \cdots, A_\kappa \rangle$ be an ordered partition of Att and let $V_1 \cdots V_\kappa$ be a sequence of letters $V_i \in \{L, R\}$. ATT is *simple κ–alt pass with respect to P and $V_1 \cdots V_\kappa$* if it is simple κ–visit with respect to P and for every $\sigma \in \Sigma^{(n)}$, conditions 1. and 2. from Definition 4.43 hold and moreover:

3. for every $1 \leq i \leq \kappa$, $b \in IA_i$, $a \in Att_{syn}$ and $1 \leq \eta, \eta' \leq n$: if $b(\pi\eta)$ at σ depends directly on $a(\pi\eta')$, then:
   - if $V_i = L$, then [if $\eta' < \eta$, then $a \in SA_1 \cup \ldots \cup SA_i$, else $a \in SA_1 \cup \ldots \cup SA_{i-1}$],
   - if $V_i = R$, then [if $\eta' \leq \eta$, then $a \in SA_1 \cup \ldots \cup SA_{i-1}$, else $a \in SA_1 \cup \ldots \cup SA_i$].

ATT is *simple κ–alt pass* if there are a κ–partition P of Att and a sequence $V_1 \cdots V_\kappa$ such that ATT is simple κ–alt pass with respect to P and $V_1 \cdots V_\kappa$. □

Since the notion of simple κ–alt pass depends not only on a partition of the set of attributes, but also on a sequence of letters from $\{L, R\}$, we construct a monadic second order formula for every such sequence of length κ and express the existence of some sequence by a finite disjunction of such formulas.

Claim 4.48 For $\kappa \in \mathbb{N}$ and a sequence $V_1 \cdots V_\kappa$ of letters from $\{L, R\}$, consider the formula:

$$
\begin{aligned}
alt_{\kappa,V_1\cdots V_\kappa}({X_1}^a,\ldots,{X_\kappa}^a) = \forall x^s, y^a, w^i, z^a, t^i.(&in(x^s) \wedge direct(x^s, y^a, w^i, z^a, t^i)\\
&\Rightarrow (syn(y^a) \wedge syn(z^a) \Rightarrow ord_{\leq,\kappa}(z^a, y^a, {X_1}^a, \ldots, {X_\kappa}^a))\\
&\wedge(inh(y^a) \wedge inh(z^a) \Rightarrow ord_{\leq,\kappa}(z^a, y^a, {X_1}^a, \ldots, {X_\kappa}^a))\\
&\wedge(inh(y^a) \wedge syn(z^a)\\
\Rightarrow \bigvee\{y^a \in {X_i}^a \wedge (V_i = L \wedge (&suc^+(t^i, w^i) \wedge ord_{\leq,\kappa}(z^a, y^a, {X_1}^a, \ldots, {X_\kappa}^a)\\
&\vee suc^*(w^i, t^i) \wedge ord_{<,\kappa}(z^a, y^a, {X_1}^a, \ldots, {X_\kappa}^a))\\
\vee V_i = R \wedge (&suc^*(t^i, w^i) \wedge ord_{<,\kappa}(z^a, y^a, {X_1}^a, \ldots, {X_\kappa}^a)\\
&\vee suc^+(w^i, t^i) \wedge ord_{\leq,\kappa}(z^a, y^a, {X_1}^a, \ldots, {X_\kappa}^a)))\\
\mid 1 \leq i \leq \kappa\}))&
\end{aligned}
$$

Then for every attributed tree transducer ATT and every ordered partition $P = \langle A_1, \ldots, A_\kappa \rangle$ of Att:

$(\Phi(ATT), A_1, \ldots, A_\kappa) \models alt_{\kappa,V_1\cdots V_\kappa}$ iff conditions 1. to 3. from Definition 4.47 hold. □

Note that rather than a formula in strict sense, this is again a scheme for constructing a formula for specific κ and $V_1 \cdots V_\kappa$. Further note that suc^* and suc^+ are used to express the relations $\leq$ and $<$ on natural numbers.

Since for every $\kappa \in \mathbb{N}$, there is only a finite number of sequences over $\{L, R\}$ of length κ, we can express the existence of a sequence satisfying conditions 1. to 3. by a finite disjunction of $alt_{\kappa,V_1\cdots V_\kappa}$–formulas.

Claim 4.49 For $\kappa \in \mathbb{N}$, the following formula $\phi_{s\kappa a}$ (**MSO**) describes the property of an attributed tree transducer to be simple κ–alt pass:

$$
\begin{aligned}
\exists {X_1}^a, \ldots, {X_\kappa}^a.(&part_\kappa({X_1}^a, \ldots, {X_\kappa}^a) \wedge visit_\kappa({X_1}^a, \ldots, {X_\kappa}^a)\\
&\wedge \bigvee\{alt_{\kappa,V_1\cdots V_\kappa}({X_1}^a, \ldots, {X_\kappa}^a) \mid V_i \in \{L, R\}\})
\end{aligned}
$$

□

Following [14], an attributed tree transducer is simple κ–pass if it is simple κ–alt pass such that every pass is a left–to–right one. Formally, this means that the sequence $V_1 \cdots V_\kappa$ is restricted to be κ times L.

Definition 4.50 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer and let $P = \langle A_1, \cdots, A_\kappa \rangle$ be an ordered partition of Att. ATT is *simple κ–pass with respect to P* if it is simple κ–alt pass with respect to P and the sequence $L \cdots L$ of length κ. ATT is *simple κ–pass* if there is a κ–partition P of Att such that ATT is simple κ–pass with respect to P. □

We express the property simple κ–pass by "simplifying" the formula scheme for $alt_{\kappa,L\cdots L}$, thus gaining a formula $pass_\kappa$.

Claim 4.51 For $\kappa \in \mathbb{N}$, the following formula $\phi_{s\kappa p}$ (**MSO**) describes the property of an attributed tree transducer to be simple κ–pass:

$$\exists {X_1}^a, \ldots, {X_\kappa}^a.(part_\kappa({X_1}^a, \ldots, {X_\kappa}^a) \wedge visit_\kappa({X_1}^a, \ldots, {X_\kappa}^a) \wedge pass_\kappa({X_1}^a, \ldots, {X_\kappa}^a))$$

where:

$$
\begin{aligned}
pass_\kappa({X_1}^a, \ldots, {X_\kappa}^a) = \forall x^s, y^a, w^i, z^a, t^i.(&in(x^s) \wedge direct(x^s, y^a, w^i, z^a, t^i)\\
&\Rightarrow (syn(y^a) \wedge syn(z^a) \Rightarrow ord_{\leq,\kappa}(z^a, y^a, {X_1}^a, \ldots, {X_\kappa}^a))\\
&\wedge(inh(y^a) \wedge inh(z^a) \Rightarrow ord_{\leq,\kappa}(z^a, y^a, {X_1}^a, \ldots, {X_\kappa}^a))\\
&\wedge(inh(y^a) \wedge syn(z^a)\\
&\quad\Rightarrow (suc^+(t^i, w^i) \wedge ord_{\leq,\kappa}(z^a, y^a, {X_1}^a, \ldots, {X_\kappa}^a)\\
&\quad\vee suc^*(w^i, t^i) \wedge ord_{<,\kappa}(z^a, y^a, {X_1}^a, \ldots, {X_\kappa}^a))))
\end{aligned}
$$

□

4.4 simple multi–visit, –sweep, –alternating pass and –pass

In the last subsection the properties of an attributed tree transducer to be simple κ–visit, simple κ–sweep, simple κ–alt pass and simple κ–pass were defined for fixed $\kappa \in \mathbb{N}$. As suprema of these classes of attributed tree transducers, we get the corresponding classes of simple multi–visit, simple multi–sweep, simple multi–alt pass and simple multi–pass attributed tree transducers (cf. [14]).

Definition 4.52 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer.
ATT is *simple multi–visit* if there is a $\kappa \in \mathbb{N}$ such that ATT is simple κ–visit.
ATT is *simple multi–sweep* if there is a $\kappa \in \mathbb{N}$ such that ATT is simple κ–sweep.
ATT is *simple multi–alt pass* if there is a $\kappa \in \mathbb{N}$ such that ATT is simple κ–alt pass.
ATT is *simple multi–pass* if there is a $\kappa \in \mathbb{N}$ such that ATT is simple κ–pass. □

Example 4.53 ATT_{leafs} is simple multi–visit, because it is simple 2–visit. □

When we want to describe these properties by logical formulas, we can no longer make use of the representation of a κ–partition by κ set variables, because we do not know κ in advance and in our logic we have no means to quantify something like "*there is a number κ and κ sets $X_1, \ldots, X_\kappa$ such that ...*". Thus, we will make use of another representation of a partition of a finite set S, namely as a mapping from S into the natural numbers $\mathbb{N}$. If $p : S \to \mathbb{N}$ is such a mapping, this will represent the ordered n–partition $P = \langle \{s \in S \mid p(s) = 1\}, \ldots, \{s \in S \mid p(s) = n\} \rangle$, where $n = max\{i \mid p(s) = i$ for some $s \in S\}$ is the largest natural number that occurs as image of p. Note that then $ind_P = p$. Such a mapping p can be represented as a binary relation in second–order logic. Consequently, monadic second–order logic will not suffice to describe the properties simple multi–visit (simple multi–sweep, ...) using this approach. However, there is another problem, because by the definition of simple multi–visit, the $\kappa \in \mathbb{N}$, for which ATT might be simple κ–visit, is not bound, but could be arbitrarily large. This we can not capture in our representation, because we are restricted to finite domains and thus only have a finite initial fragment of the natural numbers. We overcome this problem by the following lemma.

Lemma 4.54 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer. If ATT is simple multi–visit, then there is a $\kappa \leq |Att|$ such that ATT is simple κ–visit. □

Proof: Let $\kappa \in \mathbb{N}$ be the smallest number such that ATT is simple κ–visit. If $\kappa \leq |Att|$, we are ready. Otherwise, there must be a κ–partition $P = \langle A_1, \ldots, A_\kappa \rangle$ of Att such that ATT is simple κ–visit with respect to P. Since $\kappa > |Att|$, there must be an i with $1 \leq i \leq \kappa$ and $A_i = \emptyset$. We construct a $(\kappa - 1)$–partition $P' = \langle A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_\kappa \rangle$. From Definition 4.33 of sequence graphs and since there is no attribute c with $ind_P(c) = i$, it is obvious that $S_{ATT}(\sigma)\{P\} = S_{ATT}(\sigma)\{P'\}$ for every $\sigma \in \Sigma$. Consequently, ATT would be simple $(\kappa - 1)$–visit with respect to P', but this contradicts the assumption that κ is the smallest such number. □

Similar arguments can be used for simple multi–sweep, simple multi–alt pass and simple multi–pass. Informally, the ith component A_i being empty would mean that in the ith sweep through the input tree no attribute occurrence at all would be evaluated, and consequently this sweep could be "canceled out".

Claim 4.55 Let $ATT = (Att, \Sigma, \Delta, a_0, R, E)$ be an attributed tree transducer.
If ATT is simple multi–sweep, then there is a $\kappa \leq |Att|$ such that ATT is simple κ–sweep.
If ATT is simple multi–alt pass, then there is a $\kappa \leq |Att|$ such that ATT is simple κ–alt pass.
If ATT is simple multi–pass, then there is a $\kappa \leq |Att|$ such that ATT is simple κ–pass. □

In the definition of our encoding Φ (on page 10), we included in the domain of sort i all natural numbers from 1 to $|Att|$. By Lemma 4.54 and Claim 4.55, this is indeed enough to represent all necessary partitions to be used in describing the properties simple multi–visit (simple multi–sweep, ...) by binary relations over sorts a and i.

We construct the formulas describing the "multi"–properties similarly to the formulas for simple κ–visit (simple κ–sweep, ...) but now using a binary relation variable, where before we used finitely many set variables for representing partitions. First, we define the new versions of the auxiliary formulas describing a partition and orders on attributes with respect to a partition:

$$\begin{array}{ll} part_{fun}(R^{ai}) & = \forall x^a \exists! y^i.(R^{ai}(x^a, y^i)) \\ ord_{\leq,fun}(x^a, y^a, R^{ai}) & = \forall u^i, v^i.(R^{ai}(x^a, u^i) \wedge R^{ai}(y^a, v^i) \Rightarrow suc^*(u^i, v^i)) \\ ord_{<,fun}(x^a, y^a, R^{ai}) & = \forall u^i, v^i.(R^{ai}(x^a, u^i) \wedge R^{ai}(y^a, v^i) \Rightarrow suc^+(u^i, v^i)) \end{array}$$

With the help of these auxiliary formulas, the formulas for simple multi–visit and simple multi–sweep are constructed like the corresponding simple κ–formulas, but using binary relation variables and the new $\_\__{fun}$ versions of formulas. Thus we easily get the following two claims.

Claim 4.56 The following formula ϕ_{smv} (**SO**) describes the property of an attributed tree transducer to be simple multi–visit:
$\exists R^{ai}.(part_{fun}(R^{ai}) \wedge visit_{fun}(R^{ai}))$
where:

$$\begin{array}{l} visit_{fun}(R^{ai}) = \neg\exists x^s.(in(x^s) \wedge \exists y^n \forall Y^n.(\\ \qquad\qquad \forall u^n, v^n.(u^n \in Y^n \wedge depseq_{fun}(x^s, u^n, v^n, R^{ai}) \Rightarrow v^n \in Y^n) \\ \qquad\qquad \wedge \forall u^n.(depseq_{fun}(x^s, y^n, u^n, R^{ai}) \Rightarrow u^n \in Y^n) \Rightarrow y^n \in Y^n)) \\ depseq_{fun}(x^s, y^n, z^n, R^{ai}) = \exists u^a, v^i, p^a, q^i.(rule(x^s, u^a, v^i, y^n) \wedge rule(x^s, p^a, q^i, z^n) \\ \qquad\qquad \wedge \exists t^n, r^a.(edg^*(z^n, t^n) \wedge lab_{att}(t^n, r^a, v^i) \wedge seq_{fun}(u^a, r^a, R^{ai}))) \\ seq_{fun}(x^a, y^a, R^{ai}) = inh(x^a) \wedge syn(y^a) \wedge ord_{\leq,fun}(x^a, y^a, R^{ai}) \\ \qquad\qquad \vee syn(x^a) \wedge inh(y^a) \wedge ord_{<,fun}(x^a, y^a, R^{ai}) \end{array}$$

□

Claim 4.57 The following formula ϕ_{sms} (**SO**) describes the property of an attributed tree transducer to be simple multi–sweep:
$\exists R^{ai}.(part_{fun}(R^{ai}) \wedge visit_{fun}(R^{ai}) \wedge sweep_{fun}(R^{ai}))$
where:

$$\begin{array}{l} sweep_{fun}(R^{ai}) = \forall x^s, y^a, w^i, z^a, t^i.(in(x^s) \wedge direct(x^s, y^a, w^i, z^a, t^i) \\ \qquad\qquad \Rightarrow (syn(y^a) \wedge syn(z^a) \Rightarrow ord_{\leq,fun}(z^a, y^a, R^{ai})) \\ \qquad\qquad \wedge(inh(y^a) \wedge inh(z^a) \Rightarrow ord_{\leq,fun}(z^a, y^a, R^{ai})) \\ \qquad\qquad \wedge(inh(y^a) \wedge syn(z^a) \Rightarrow (w^i = t^i \wedge ord_{<,fun}(z^a, y^a, R^{ai}) \\ \qquad\qquad\qquad \vee w^i \neq t^i \wedge ord_{\leq,fun}(z^a, y^a, R^{ai})))) \end{array}$$

□

For describing the property simple multi–alt pass, we need to formalize the sequence of letters $V_1 \cdots V_\kappa$ over $\{L, R\}$ for arbitrary κ. We can do this by representing the sequence just by a set V^i over sort i, containing all natural numbers, for which the corresponding letter in the sequence is R. By following this approach, we get a formula characterizing simple multi–alt pass attributed tree transducers.

Claim 4.58 The following formula ϕ_{sma} (**SO**) describes the property of an attributed tree transducer to be simple multi–alt pass:
$\exists R^{ai}, V^i.(part_{fun}(R^{ai}) \wedge visit_{fun}(R^{ai}) \wedge alt_{fun}(R^{ai}, V^i))$
where:

$$
\begin{aligned}
alt_{fun}(R^{ai}, V^i) = \forall x^s, y^a, w^i, z^a t^i.(in(x^s) \wedge direct(x^s, y^a, w^i, z^a, t^i) \\
\Rightarrow (syn(y^a) \wedge syn(z^a) \Rightarrow ord_{\leq, fun}(z^a, y^a, R^{ai})) \\
\wedge(inh(y^a) \wedge inh(z^a) \Rightarrow ord_{\leq, fun}(z^a, y^a, R^{ai})) \\
\wedge(inh(y^a) \wedge syn(z^a) \\
\Rightarrow \forall p^i.(R^{ai}(y^a, p^i) \Rightarrow (p^i \notin V^i \wedge (suc^+(t^i, w^i) \wedge ord_{\leq, fun}(z^a, y^a, R^{ai}) \\
\vee suc^*(w^i, t^i) \wedge ord_{<, fun}(z^a, y^a, R^{ai})) \\
\vee p^i \in V^i \wedge (suc^*(t^i, w^i) \wedge ord_{<, fun}(z^a, y^a, R^{ai}) \\
\vee suc^+(w^i, t^i) \wedge ord_{\leq, fun}(z^a, y^a, R^{ai}))))))
\end{aligned}
$$

$\square$

The formula for simple multi–pass can again be obtained from the simple multi–alt pass formula by fixing $V_1 \cdots V_\kappa$ to be a sequence of only $L \cdots L$, in this case fixing V^i to be empty in alt_{fun}. Then we can "simplify" the resulting formula and from this construct a second–order formula $pass_{fun}$.

Claim 4.59 The following formula ϕ_{smp} (**SO**) describes the property of an attributed tree transducer to be simple multi–pass:
$\exists R^{ai}.(part_{fun}(R^{ai}) \wedge visit_{fun}(R^{ai}) \wedge pass_{fun}(R^{ai}))$
where:

$$
\begin{aligned}
pass_{fun}(R^{ai}) = \forall x^s, y^a, w^i, z^a t^i.(in(x^s) \wedge direct(x^s, y^a, w^i, z^a, t^i) \\
\Rightarrow (syn(y^a) \wedge syn(z^a) \Rightarrow ord_{\leq, fun}(z^a, y^a, R^{ai})) \\
\wedge(inh(y^a) \wedge inh(z^a) \Rightarrow ord_{\leq, fun}(z^a, y^a, R^{ai})) \\
\wedge(inh(y^a) \wedge syn(z^a) \Rightarrow (suc^+(t^i, w^i) \wedge ord_{\leq, fun}(z^a, y^a, R^{ai}) \\
\vee suc^*(w^i, t^i) \wedge ord_{<, fun}(z^a, y^a, R^{ai}))))
\end{aligned}
$$

$\square$

Summary of described properties

Altogether, we described 12 properties of attributed tree transducers by logical formulas. They are summarized in Figure 12, together with the fragments of second–order logic that were necessary to do so.

single–use	**FO+TC+EQ**
nonnested	**FO+TC+EQ**
one–visit	**MSO**
absolutely noncircular	**SO**
simple κ–visit	**MSO**
simple κ–sweep	**MSO**
simple κ–alt pass	**MSO**
simple κ–pass	**MSO**
simple multi–visit	**SO**
simple multi–sweep	**SO**
simple multi–alt pass	**SO**
simple multi–pass	**SO**

Figure 12: Summary of described properties

Note that we did not show that these fragments of second–order logic are really the minimal ones, with which the properties can be described. For example, we have the strong conjecture that the property of an attributed tree transducer to be absolutely noncircular is **not** expressible in monadic second–order logic, but we were not able to prove this conjecture. Such a proof might be done by reducing to the known inability of monadic second–order logic to express that two finite sets are of the same size. Therefore, one could try to define a family of attributed tree transducers dependent on two natural numbers n and m (for example, with the sets of inherited and synthesized attributes having n and m elements, respectively) such that an attributed tree transducer of this family is absolutely noncircular iff $n = m$ (or alternatively, iff $n \neq m$). If this family was suitably chosen, one might be able to show that then from a monadic second–order formula describing the property absolutely noncircular one would be able to construct a monadic second–order formula expressing that two sets have the same number of elements, which would lead to contradiction. This technique is used in [17] to show that the property of a graph to have a Hamilton cycle can not be expressed in monadic second–order logic. However, it is not clear, whether this approach can also be used to solve our problem.

Another unsolved question is, whether the fragments **FO+TC+EQ** and **MSO** are really different with respect to expressive power. As was mentioned in the preliminaries, we obviously have **FO+TC+EQ** $\subseteq$ **MSO**, but it is not known, whether in general monadic second–order logic is more expressive than first order–logic plus the construction of transitive closures of relations (this is unknown even for the special case of trees over a ranked alphabet). However, in the next section we will see that for us the distinction between **FO+TC+EQ** and **MSO** is useful, because for formulas of the first fragment we can optimize the decision algorithm.

5 Implementational aspects

In this section we want to discuss some aspects that become important if one wants to practically use the described formulas to test properties of attributed tree transducers. There are several points of interest: the construction of formulas, the encoding of attributed tree transducers into relational structures and the decidability of the question, whether a structure models a given formula.

With construction of formulas we mean the expansion of macros and abbreviations that were used in the paper to gain readability. Given a suitable representation of formulas by an algebraic data type, the expansion of macros such as the notation for the transitive closure can be implemented by following the definition of substituting variables in logical formulas from page 7. It should also be clear that the instantiation of formula schemes like the $\phi_{s\kappa v}$, $\phi_{s\kappa s}$ formulas, etc., for specific values of κ (as was shown in Example 4.38 for $ord_{<,3}$), can be implemented with a time polynomial complexity in the size of the formulas.

The encoding from attributed tree transducers (given by an algebraic data type representing the $(Att, \Sigma, \Delta, a_0, R, E)$–tuple) to relational structures over SIG_{ATT}, as defined in section 3 and shown by example on page 11, could also be implemented with a polynomial time complexity in the size (overall number of tree nodes) of the rule sets and the environment.

Now to the question of the decidability, whether a structure is a model for a given formula. On page 7, we have the definition of semantics of second–order formulas. This already gives a decision algorithm, because following this definition, one can write a program, for example in a functional programming language, that, given an arbitrary formula, state and relational structure (represented by suitable algebraic data types) decides by structural recursion over the input formula, whether the state satisfies the formula in the structure. Such a program would always terminate, because every formula is finite and all the domains of input structures are finite, so there are only finitely many possible branchings in the last four defining rules. Note that the usage of signatures with sorts and typed formulas of course complicates the implementation, but one can gain efficiency, as in evaluating formulas with quantifiers only the elements and relations of appropriate sort and type, respectively, have to be taken and tested. There are several other ways to gain efficiency, such as using lazy evaluation, e.g., in that one does not need to test for both sub–formulas of a disjunctive formula $\phi \vee \psi$, if already ϕ is evaluated to true.

However, there is a general problem with the time complexity of such an algorithm. Where in a formula with first–order quantification such as $\forall x^s.(\phi(x^s))$ one has to test ϕ for all elements of sort s and thus will have a linear complexity in the size of the domain of sort s (assuming the test of $\phi(x^s)$ to take constant time), already the quantification over set variables such as in $\forall X^s.(\phi(X^s))$ will lead to an exponential complexity, because ϕ has to be tested for all subsets of the domain of sort s, and for every finite set S, we have $|\mathcal{P}(S)| = 2^{|S|}$. With full second–order formulas, the situation gets even worse.
In the next subsection we want to propose how some of the inefficiencies caused by monadic second–order quantifications can be avoided.

5.1 Optimizing computation of transitive closures

In the formulas, which we used to describe properties of attributed tree transducers, one of the main uses of quantification over set variables was to express transitive closures of certain relations. However, this can be done more efficiently, as we see in the following example.

Example 5.1 Consider a signature with one sort n of nodes and a binary relation $edg : nn$. Structures over this signatures can be seen as directed graphs. To express that two nodes of the graph are connected by a nonempty path, the following formula can be used:

$$path(x,y) = edg^{+}(x,y) = \forall X.(\forall u,v.(u \in X \wedge edg(u,v) \Rightarrow v \in X) \\ \wedge \forall u.(edg(x,u) \Rightarrow u \in X) \Rightarrow y \in X)$$

The intuition behind this construction is to check, whether y is included in the least fix–point, with respect to expansion of a set of nodes by edges, that contains the node x. Therefore, all potential fix–points (sets of nodes) have to be checked. Thus, by the observations above, testing the existence of a nonempty path between two nodes of a certain structure with help of this formula would have a time–complexity that is exponential in the number of nodes of the graph. From graph theory we know that there are algorithms that do better. □

Thus, a strategy to optimize the decision algorithm is the following: If the formula under question contains constructions like rel^{+}, do not expand the macro as given in section 2. When in the algorithm such a construction is met, compute the binary relation that is represented by the formula rel (by testing all pairs of elements of the domain of corresponding sort) and then use an efficient algorithm to compute the transitive closure of this relation. This strategy is immediately applicable to the describing formulas given in **FO+TC+EQ**, namely for the formulas describing the properties single–use and nonnested. But with a few refinements we could also use this strategy for the other formulas. For example, in

$$\phi_{one-visit} = \neg\exists x^{s}.(in(x^{s}) \wedge \exists y^{i}\forall X^{i}.(\forall u^{i},v^{i}.(u^{i} \in X^{i} \wedge brother(x^{s},u^{i},v^{i}) \Rightarrow v^{i} \in X^{i}) \\ \wedge \forall u^{i}.(brother(x^{s},y^{i},u^{i}) \Rightarrow u^{i} \in X^{i}) \Rightarrow y^{i} \in X^{i}))$$

quantification over set variables is also only used for expressing a transitive closure. We just could not use a construction like rel^{+}, because here we do not have a binary relation formula, but we want to compute the transitive closure of $brother$ with respect to a fixed x^{s}, without having a notation for such a situation.
With a bit more syntactic sugar, however, we would be able to handle this. For example, we could define a notation like

$$\phi_{one-visit} = \neg\exists x^{s}.(in(x^{s}) \wedge \exists y^{i}.(brother[x^{s},\_,\_]^{+}(y^{i},y^{i}))),$$

meaning the same as above but allowing an optimization of computing transitive closures also in all the other formulas.

5.2 Further remarks

In this subsection we shortly want to discuss results that connect logical descriptions of relational structures with an automata theoretic approach, as well as the implications these results can have for implementational aspects.
For structures over special signatures there are algorithms that can decide in linear time, whether a monadic second–order formula is satisfied. This comes from a connection between logic and automata that has been studied since the works of Büchi and Elgot in the sixties. From [21] and [22] it is known that for trees over a ranked alphabet, the expressive power of monadic second–order logic and finite automata is the same. That is, for every monadic second–order formula, one can construct a tree automaton, which accepts exactly those trees over the ranked alphabet that are models for the formula. Results like this one are attractive, because they allow to compute with linear time complexity (via simulation of the

corresponding automaton) what above we have said might take exponential time when using the generic decision algorithm for second–order logic. However, the constructed automaton might have a transition table that in the worst case has a size exponential in the quantifier–depth of the input formula. This phenomenon is also known as *exponential blow–up*. It implies that the exponential time complexity is transferred to the construction of the tree automaton. Of course, this construction only needs to be done once for every formula we want to check on various structures, but still there remains the problem of a now exponential memory space complexity.

On the other hand, these results are not available for structures over arbitrary signatures, but only for very specific ones such as trees over ranked alphabets as mentioned above. We could not easily represent attributed tree transducers by trees over a fixed ranked alphabet, but what we can do, is to translate our structures over SIG_{ATT} into term graphs, that is, trees with sharing (nodes can have more than one predecessor), over a fixed ranked alphabet. By expressing the relations of SIG_{ATT} as formulas over these term graphs, we could also transform our describing monadic second–order formulas, such that the problem of deciding, whether a SIG_{ATT}–structure satisfies a certain formula, reduces to solving the same problem over term graphs. However, while there are several generalizations of the above results from trees to other classes of graphs and traces (see [19]), there is no such result for term graphs. These generalizations depend on a somehow canonical finite automata model for the classes of graphs under question, together with a way to reduce nondeterministic automata to deterministic automata. Such an automata model is not available for term graphs.

References

[1] W.C. Rounds. Context–free grammars on trees. *Proc. ACM Symp. on Theory of Comput.*, pages 143–148. ACM Press, 1969.

[2] W.C. Rounds. Mappings and grammars on trees. *Math. Syst. Theory,* 4:257–287, 1970.

[3] J.W. Thatcher. Generalized sequential machine maps. *J. Comput. Syst. Sci.,* 4:339–367, 1970.

[4] J. Engelfriet. Bottom–up and top–down tree transformations – a comparison. *Math. Syst. Theory,* 9:198–231, 1975.

[5] Z. Fülöp. On attributed tree transducers. *Acta Cybernetica,* 5:261–279, 1981.

[6] J. Engelfriet. Some open questions and recent results on tree transducers and tree languages. In R.V. Book (ed.), *Formal language theory; perspectives and open problems*, pages 241–286. Academic Press, New York, 1980.

[7] B. Courcelle and P. Franchi–Zannettacci. Attribute grammars and recursive program schemes I and II. *Theoret. Comput. Sci.,* 17:163–191 and 235–257, 1982.

[8] J. Engelfriet and H. Vogler. Macro tree transducers. *J. Comput. Syst. Sci.,* 31:71–146, 1985.

[9] A. Kühnemann. Benefits of Tree Transducers for Optimizing Functional Programs. *Foundations of Software Technology & Theoretical Computer Science – FST & TCS'98, 18th Conference, Chennai, India, Proceedings*, pages 146–157. Springer–Verlag, 1998.

[10] Z. Fülöp and H. Vogler. *Syntax–directed semantics — Formal models based on tree transducers.* EATCS Monographs on Theoretical Computer Science, Springer Verlag, 1998.

[11] D.E. Knuth. Semantics of context–free languages. *Math. Syst. Theory,* 2:127–145, 1968. Corrections in *Math. Syst. Theory,* 5:95–96, 1971.

[12] P. Franchi–Zannettacci. *Attributs semantiques et schemas de programmes.* PhD thesis, Universite de Bordeaux I, 1982. These d'Etat.

[13] Z. Fülöp and H. Vogler. A Characterization of Attributed Tree Transformations by a Subclass of Macro Tree Transducers. *Theory Comput. Systems,* 32:649–676, 1999.

[14] Z. Fülöp. *Characterizing Important Subclasses of Attributed Tree Transformations by Macro Tree Transducers.* manuscript, 2000.

[15] R. Giegerich. Composition and evaluation of attribute coupled grammars. *Acta Informatica,* 25:355–423, 1988.

[16] J. Engelfriet and G. File. Passes, sweeps and visits in attribute grammars. *J. ACM,* 36:841–869, 1989.

[17] 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.), pages 313–400. World Scientific, 1997.

[18] R. Bloem and 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.

[19] W. Thomas. Languages, Automata and Logic, Chapter 7 of *Handbook of formal languages*, Vol. 3: *Beyond Words* (G. Rozenberg, A. Salomaa, eds.), pages 389–455. Springer–Verlag, 1997.

[20] S. Hölldobler. *Computational Logic.* working material, Technical University Dresden, 1999.

[21] J. Doner. Tree acceptors and some of their applications. *J. of Comp. Syst. Sci.,* 4:406–451, 1970.

[22] J.W. Thatcher and J.B. Wright. Generalized finite automata theory with an application to a decision problem of second–order logic. *Math. Syst. Theory,* 2:57–81, 1968.