Characterization of Properties and Relations defined in Monadic Second Order Logic on the Nodes of Trees

Roderick Bloem* and Joost Engelfriet**

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

Abstract. A formula from monadic second order (MSO) logic with one free variable can be used to define a property of the nodes of a tree. Similarly, an MSO formula with two free variables can be used to define a binary relation between the nodes of a tree. It is proved that a node relation is MSO definable iff it can be computed by a finite-state tree-walking automaton, provided the automaton can test MSO definable properties of the nodes of the tree; if the relation is a function, the automaton is deterministic. It is also proved that a node property is MSO definable iff it can be computed by an attribute grammar of which all attributes have finitely many values. MSO definable node properties are computable in linear time, MSO definable node relations in quadratic time, and MSO definable node functions in linear time.

1 Introduction

It is shown in [Büc, Elg] that a set of strings can be defined in monadic second order logic if and only if it can be recognized by a finite-state automaton. This result can be viewed as an equivalence between specifications (in the specification language of monadic second order logic) and implementations (on the machine model of the finite-state automaton). It was generalized in [Don, ThaWri] to sets of node-labeled ordered trees, with an appropriate generalization of the finite-state automaton to the bottom-up finite-state tree automaton. The trees considered are the usual representations of terms over a finite set Σ of operators. In this paper we extend the result of [Don, ThaWri] to the specification and implementation of binary relations on the nodes of trees, and to the specification and implementation of properties of the nodes of trees.

A monadic second order (MSO) formula with k free object variables defines a k-ary relation on the nodes of each tree over Σ, and we are interested in

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

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

the implementation of these relations for $k = 2$ (binary relations) and $k = 1$ (properties). First we are looking for a simple machine model that computes the node relations defined by MSO formulas with two free variables. Here we say 'computes' rather than 'recognizes', because we are mainly interested in functions on nodes of trees and view relations as a nondeterministic variant of functions. Thus, for a given tree t and a given node u of t, we wish the machine to find the (or a) node v such that the pair (u, v) is in the relation specified by the MSO formula on t.

Note that there is an easy and well-known answer to the recognition problem: it is not difficult to see, using the result of [Don, ThaWri], that there is a finite-state tree automaton that, for given t, u, and v, finds out whether u and v satisfy the MSO formula in t, provided the nodes u and v are indicated by special labels. Since the finite-state tree automaton is basically a parallel recognition device, it does not seem to be useful for the purpose of sequential computation. Instead, we propose a variation of the tree-walking automaton introduced in [AhoUll]. A tree-walking automaton A is a finite-state automaton that walks on the tree from node to node, following the edges of the tree. It can read the label of the current node x. It can also test whether x is the root of the tree, and if not, whether it is the first, second, ..., or last child of its parent node. Depending on the outcome of these tests, A can decide to move up to the parent of x or down to a specific child of x. The automaton A computes the relation that consists of all pairs (u, v) such that when A is dropped on node u in its initial state, it makes a walk on the tree and reaches node v in a final state. We will show that this automaton model is not powerful enough to compute all MSO definable node relations. Therefore we strengthen its computation power by allowing it to test any MSO definable property of the current node rather than just the ones mentioned above, using MSO formulas with one free variable. Our first result is that the nondeterministic (deterministic) "tree-walking automaton with MSO tests" computes exactly the MSO definable binary node relations (functions, respectively).

Second we are looking for an implementation model that computes the node properties defined by MSO formulas with one free variable. One reason for doing so is that, through the tree-walking automaton with MSO tests, we have reduced the implementation of MSO definable node relations to that of MSO definable node properties.

Properties are boolean attributes, and attribute grammars are a well-known formalism for the computation of attributes of nodes of trees, introduced in [Knu]. Thus, an attribute grammar with a designated boolean attribute computes a node property. An attribute grammar is "finite-valued" if all its attributes have finitely many values, or equivalently, all its attributes are boolean. Our second result is that the finite-valued attribute grammar computes exactly the MSO definable node properties. Note that the finite-valued restriction of the attribute grammars corresponds to the finite-state restriction of the tree automata; without such a restriction attribute grammars can compute arbitrary node properties. Note also that properly speaking, attribute grammars are still a specification language, but they are certainly closer to implementation than

MSO logic, and their implementation has been studied extensively (see, e.g., [DerJouLor, Eng2]). Since the attributes of a tree can be evaluated in linear time (in the size of the tree), our second result implies that MSO definable node properties can be computed in linear time and that MSO definable binary node relations can be computed in quadratic time, and, in general, MSO definable k-ary node relations in time $O(n^k)$. Moreover, using additionally an algorithm from [KlaSch], MSO definable node functions can be computed in linear time.

There are two reasons for our interest in MSO definable functions, relations, and properties of nodes of trees. The first concerns graphs that consist of a tree with additional pointers. Trees can be naturally defined as a recursive datatype, such that each node of the tree is a record with a field containing relevant information, and fields with pointers to the children of the node. However, in many applications one needs additional pointer fields in the record. For example, to visit the nodes of the tree in pre-order, it is convenient to have a pointer from each node to the next one, in this order. Clearly, such a new pointer field defines a (partial) function on the nodes of the tree. The general idea of defining "graph types", i.e., recursive data types consisting of trees with additional pointer fields, is investigated in [KlaSch], where a pointer field is defined by a tree-walking automaton, or rather by its corresponding regular expression, called routing expression. By our results, such a pointer field may as well be specified by an MSO formula with two free variables.

The second reason is related to the first, but is of a more theoretical nature. In the theory of context-free graph grammars, i.e., grammars that generate sets of graphs, a notion of MSO definable graph transduction has been developed (see, e.g., [Eng3, EngOos2, Cou2, Cou3]) closely related to the well-known notion of interpretation of one logical structure in another (see [ArnLagSee] for the history of this notion). The output graph g' of such a transduction is defined in terms of the input graph g by means of MSO formulas, to be interpreted in g, as follows. The nodes of g' form a subset of the nodes of g, viz. all nodes of g that satisfy a given MSO formula with one free object variable. The edges of g' are defined by a given MSO formula with two free object variables: this formula defines a binary relation on the nodes of g, which, restricted to the nodes of g', is taken as the set of edges of g'. In the case that the nodes and edges of g' are labeled, there are such MSO formulas for each node label and each edge label. It is shown in [EngOos1, EngOos2] (for one type of context-free graph grammar) and in [CouEng] (for another type) that a set of graphs can be generated by a context-free graph grammar if and only if it is the image of an MSO definable tree language under an MSO definable graph transduction. Thus, using our results, the nodes of these graphs can be computed by a finite-valued attribute grammar, and their edges by (nondeterministic) tree-walking automata. As a special case of MSO graph transductions, one may consider the case that both the input and output graphs are trees: MSO tree transductions. Since the edges of the output tree can be viewed as pointers (to the i-th child, for each i), they correspond to functions on the nodes of the input tree, as explained above. Thus, they can be computed by deterministic tree-walking automata. This is used in [Blo, BloEng2]

to show that the MSO tree transductions can be computed by two-stage attribute grammars: in the first stage the MSO tests on the nodes of the input tree are evaluated, by attributes of type boolean, and in the second stage the output tree is computed, by attributes of type tree.

After recalling some notions from automata theory and monadic second order logic in Section 2, we define in Section 3 the MSO definable k-ary node relations and discuss some of their elementary properties, including some relationships to finite tree automata. In Section 4 tree-walking automata with MSO tests are defined, and some of their basic properties proved. Section 5 contains our first result: the equivalence between MSO formulas with two free variables and tree-walking automata with MSO tests, including the functional/deterministic case. It is also shown in Section 5 that the MSO tests are really needed. Our second result is proved in Section 6: the equivalence between MSO formulas with one free variable and finite-valued attribute grammars. Moreover, the first and second result are combined in a straightforward way to give an "MSO-free" characterization of the MSO definable binary node relations: they are computed by a finite-valued attribute grammar followed by an ordinary tree-walking automaton (without MSO tests). Section 7 contains the consequences of these results for the time complexity of computing MSO definable k-ary relations.

Readers who are not familiar with attribute grammars, and do not want to be bothered by them, can read Sections 2 to 5. They can also read Section 7, except for the case $k = 1$ in the proof of Theorem 20 which is based on the previous section and shows that MSO definable node properties can be computed in linear time. Readers who are interested in our second result only, can read Sections 2-4 (without Lemmas 4 and 8) and Section 6 (without the third subsection).

The results of this paper were proved as part of [Blo], the Master's Thesis of the first author. A preliminary version of this paper appeared in [BloEng1]. Recently, the second main result was shown independently in [NevBus].

2 Preliminaries

In this section we recall some well-known concepts concerning finite (tree) automata and monadic second order logic on trees.

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

Trees

We view trees as finite, directed graphs with labeled nodes and edges, in the usual way. Let Σ and Γ be alphabets of node labels and edge labels, respectively. A *graph* over (Σ, Γ) is a triple (V, E, lab), with V a finite set of nodes, $E \subseteq V \times \Gamma \times V$

the set of labeled edges, and $\mathrm{lab} : V \to \Sigma$ the node-labeling function. For a given graph g, its nodes, edges, and node-labeling function are denoted V_g, E_g, and lab_g, respectively.

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

The nodes of a tree over Σ are labeled by operators. To indicate the order of the arguments of an operator, we label the edges by natural numbers. A *tree* over Σ is an acyclic connected graph g over $(\Sigma, \mathrm{rki}(\Sigma))$ such that (1) no node of g has more than one incoming edge, and (2) for every node u of g and every $i \in [1, \mathrm{rk}(\mathrm{lab}_g(u))]$, u has exactly one outgoing edge with label i, and u has only outgoing edges with labels in $[1, \mathrm{rk}(\mathrm{lab}_g(u))]$. The set of all trees over Σ is denoted T_Σ. A subset of T_Σ is also called a *tree language*.

The root of a tree t is denoted $\mathrm{root}(t)$. Note that the direction of the edges of t is from $\mathrm{root}(t)$ to the leaves of t. For nodes u and v of t, if $(u, i, v) \in E_t$, then u is called the parent of v, and v is called the i-th child of u, denoted by $u \cdot i$. For technical convenience, we also define $u \cdot 0 = u$. An ancestor of a node u is either u itself, or an ancestor of its parent. The least common ancestor of nodes u and v is denoted $\mathrm{lca}(u, v)$. If u is an ancestor of v, then v is a descendant of u. For a node u of t, t_u denotes the subtree of t with root u: the subgraph of t induced by the set of all descendants of u.

Finite automata

We consider (nondeterministic) finite automata on strings and (deterministic and nondeterministic) finite tree automata (see, e.g., [HopUll] and [GécSte], respectively).

Let Σ be an (ordinary) alphabet. A *finite automaton* over Σ is a quintuple $A = (Q, \Sigma, \delta, I, F)$, where Q is a finite set of states, Σ is the input alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, $I \subseteq Q$ is the set of initial states, and $F \subseteq Q$ is the set of final states. The elements of δ are called transitions. For every string $w \in \Sigma^*$, A induces a state transition relation $R_A(w) \subseteq Q \times Q$, as follows. For $\sigma \in \Sigma$, $R_A(\sigma) = \{(q, q') \mid (q, \sigma, q') \in \delta\}$. For the empty string ε, $R_A(\varepsilon)$ is the identity on Q. For $\sigma_1, \ldots, \sigma_n \in \Sigma$, $R_A(\sigma_1 \cdots \sigma_n) = R_A(\sigma_1) \circ \cdots \circ R_A(\sigma_n)$. The language recognized by A is $L(A) = \{w \in \Sigma^* \mid R_A(w) \cap (I \times F) \neq \emptyset\}$. $L(A)$ is called a *regular language*.

The finite tree automata that we usually consider are the well-known total deterministic bottom-up finite-state tree automata. Let Σ be an operator alphabet. A *finite tree automaton* over Σ is a quadruple $M = (Q, \Sigma, \delta, F)$, where Q is a finite set of states, Σ is the input alphabet, $\delta = \{\delta_\sigma\}_{\sigma \in \Sigma}$ where, for $\sigma \in \Sigma_k$, $\delta_\sigma : Q^k \to Q$ is the transition function for σ, and $F \subseteq Q$ is the set of final states. For every tree $t \in T_\Sigma$ and node $u \in V_t$, the *state in which M reaches u*, denoted $\mathrm{state}_{M,t}(u)$, is defined by bottom-up induction as follows: if $\mathrm{lab}_t(u) = \sigma \in \Sigma_k$,

then $\text{state}_{M,t}(u) = \delta_\sigma(\text{state}_{M,t}(u \cdot 1), \ldots, \text{state}_{M,t}(u \cdot k))$. The language recognized by M is $L(M) = \{t \in T_\Sigma \mid \text{state}_{M,t}(\text{root}(t)) \in F\}$. $L(M)$ is called a *regular tree language*.

For a tree $t \in T_\Sigma$ and a node $u \in V_t$, the set of *successful states* of M at u, denoted $\text{succ}_{M,t}(u)$, is defined by top-down induction as follows: $\text{succ}_{M,t}(\text{root}(t)) = F$, and if $\text{lab}_t(u) = \sigma \in \Sigma_k$ and $1 \leq i \leq k$, then $\text{succ}_{M,t}(u \cdot i)$ is the set of all states $q \in Q$ such that $\delta_\sigma(q_1, \ldots, q_{i-1}, q, q_{i+1}, \ldots, q_k) \in \text{succ}_{M,t}(u)$, where $q_j = \text{state}_{M,t}(u \cdot j)$ for $1 \leq j \leq k$, $j \neq i$. Intuitively, q is in $\text{succ}_{M,t}(u)$ if the automaton, assuming that it reaches u in state q (rather than in $\text{state}_{M,t}(u)$), will reach the root of t in a final state. In the following lemma it is shown that for every node u of a tree t, t is recognized by M if and only if the state in which M reaches u is successful at u.

Lemma 1. *Let M be a finite tree automaton over Σ, and let u be a node of a tree $t \in T_\Sigma$. Then $t \in L(M)$ iff $\text{state}_{M,t}(u) \in \text{succ}_{M,t}(u)$.*

Proof. It is straightforward to prove this statement by top-down induction on u. By definition of $L(M)$ it holds for $u = \text{root}(t)$. Now let $i \in [1, \text{rk}(\text{lab}_t(u))]$. From the definition of 'succ' and 'state' it easily follows that $\text{state}_{M,t}(u \cdot i) \in \text{succ}_{M,t}(u \cdot i)$ iff $\text{state}_{M,t}(u) \in \text{succ}_{M,t}(u)$. This implies that the statement holds for $u \cdot i$ whenever it holds for u. □

A *nondeterministic finite tree automaton* over Σ is a quadruple $M = (Q, \Sigma, \delta, F)$, where Q, Σ, δ, and F are as above, except that, for every $\sigma \in \Sigma_k$, $\delta_\sigma \subseteq Q^k \times Q$ is a transition relation. As a shortcut, we take the usual subset construction as definition of $L(M)$: the language recognized by M is $L(M) = L(\mathcal{P}(M))$ where $\mathcal{P}(M)$ is the (deterministic) finite tree automaton $(Q', \Sigma, \delta', F')$ with $Q' = \mathcal{P}(Q)$, F' is the set of all $P \subseteq Q$ such that $P \cap F \neq \emptyset$, and, for every $\sigma \in \Sigma_k$, $\delta'_\sigma(P_1, \ldots, P_k) = \{q \in Q \mid \exists q_1, \ldots, q_k : ((q_1, \ldots, q_k), q) \in \delta_\sigma \text{ with } q_i \in P_i \text{ for all } i \in [1, k]\}$.

Monadic second order logic

Monadic second order logic can be used to describe properties of graphs (see, e.g., [Cou1, Cou4, Eng3, Eng5, EngOos2]), and hence in particular to describe properties of trees. For an operator alphabet Σ, we use the language $\text{MSOL}(\Sigma)$ of monadic second order (MSO) formulas over Σ. Formulas in $\text{MSOL}(\Sigma)$ describe properties of trees over Σ. This logical language has node variables $x, y, \ldots$, and node-set variables $X, Y, \ldots$. For a given tree t over Σ, node variables range over the elements of V_t, and node-set variables range over the subsets of V_t.

There are three types of atomic formulas in $\text{MSOL}(\Sigma)$: $\text{lab}_\sigma(x)$, for every $\sigma \in \Sigma$, denoting that x has label σ; $\text{edg}_i(x, y)$, for every $i \in \text{rki}(\Sigma)$, denoting that there is an edge labeled i from x to y, i.e., that y is the i-th child of x; and $x \in X$, denoting that x is an element of X. The formulas are built from the atomic formulas using the connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$, as usual. Both node variables and node-set variables can be quantified with $\exists$ and $\forall$. We will use

$\mathrm{edg}(x,y)$ to abbreviate the disjunction of all $\mathrm{edg}_i(x,y)$, $i \in \mathrm{rki}(\Sigma)$; it denotes that x is the parent of y. Moreover, we will use $\mathrm{root}(x)$ for $\neg\exists y(\mathrm{edg}(y,x))$, which denotes that x is the root, and $\mathrm{leaf}(x)$ for $\neg\exists y(\mathrm{edg}(x,y))$, which denotes that x is a leaf. Finally, we will use $x = y$ for $\forall X(x \in X \leftrightarrow y \in X)$, denoting that x equals y.

For every $k \in \mathbb{N}$, the set of MSO formulas over Σ with k free node variables and no free node-set variables is denoted $\mathrm{MSOL}_k(\Sigma)$.

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

The tree language defined by a formula $\phi \in \mathrm{MSOL}_0(\Sigma)$ is $L(\phi) = \{t \in T_\Sigma \mid t \models \phi\}$. $L(\phi)$ is called an MSO *definable tree language*.

The following classical result from [Don, ThaWri] shows the equivalence between monadic second order logic and finite tree automata, as a means of defining tree languages by specification and computation, respectively. It was first shown for the special case of string languages in [Büc, Elg]. See Sections 3 and 11 of [Tho], and [Eng4], for a discussion of such results.

Proposition 2. *A tree language is* MSO *definable if and only if it is regular.*

Since emptiness of regular tree languages is decidable (see, e.g., [GécSte]), Proposition 2 implies that MSO logic on trees is decidable.

Proposition 3. *It is decidable for a formula $\phi \in \mathrm{MSOL}_0(\Sigma)$ whether or not $L(\phi) = \emptyset$.*

3 MSO Node Properties and Relations

The aim of this paper is to investigate ways of defining properties of the nodes of trees, and relations between the nodes of trees. One way of doing this is by MSO formulas with free node variables. In this section we define the MSO definable node properties and node relations, and we discuss the well-known technique of coding the values of the free variables by marking the nodes of the input tree.

Let Σ be an operator alphabet. A *node property* over Σ is a subset of $\{(t,u) \mid t \in T_\Sigma \text{ and } u \in V_t\}$. For $k \in \mathbb{N}$, a k-ary *node relation* over Σ is a subset of $\{(t,u_1,\ldots,u_k) \mid t \in T_\Sigma \text{ and } u_i \in V_t \text{ for all } i \in [1,k]\}$. Thus, a node property associates with each tree t a property of the nodes of t, and a k-ary node relation associates with each tree t a k-ary relation on the nodes of t. Note that a unary node relation is the same as a node property, and that a nullary node relation is the same as a tree language. We will be interested in particular in binary node relations.

Let $\phi(x_1, \ldots, x_k) \in \mathrm{MSOL}_k(\Sigma)$ be an MSO formula with k free node variables. For each tree $t \in T_\Sigma$, $\phi(x_1, \ldots, x_k)$ defines the relation

$$R_t(\phi) = \{(u_1, \ldots, u_k) \in V_t^k \mid (t, u_1, \ldots, u_k) \models \phi(x_1, \ldots, x_k)\}.$$

In this way, $\phi(x_1, \ldots, x_k)$ defines the node relation

$$R(\phi) = \{(t, u_1, \ldots, u_k) \mid t \in T_\Sigma, (u_1, \ldots, u_k) \in R_t(\phi)\}.$$

$R(\phi)$ is called an MSO *definable k-ary node relation*.

Note that for $k = 0$, i.e., for a closed formula ϕ, $R(\phi)$ is the tree language $L(\phi)$ as defined in Section 2. For $k = 1$ we also write $P(\phi)$ for $R(\phi)$, to stress that it is a node property.

The set of all MSO definable node properties is denoted MSO-P, and the set of all MSO definable binary node relations is denoted MSO-R.

We will need the following basic, well-known fact: if a binary node relation is MSO definable, then so is its transitive reflexive closure (see, e.g., [Cou1]). For a formula $\phi(x, y) \in \mathrm{MSOL}_2(\Sigma)$, we define the formula $\phi^*(x, y) = \forall X((\mathrm{closed}(X) \wedge x \in X) \rightarrow y \in X)$, where $\mathrm{closed}(X) = \forall x, y((x \in X \wedge \phi(x, y)) \rightarrow y \in X)$.

Proposition 4. *Let Σ be an operator alphabet. For every tree $t \in T_\Sigma$ and every formula $\phi(x, y) \in \mathrm{MSOL}_2(\Sigma)$, $R_t(\phi^*) = R_t(\phi)^*$.*

In order to compute the node relation of an MSO formula $\phi(x_1, \ldots, x_k)$ by a tree-walking automaton (for $k = 2$) or by an attribute grammar (for $k = 1$), we will first use the classical result of Proposition 2 to construct a finite tree automaton that recognizes all trees $\mathrm{mark}(t, u_1, \ldots, u_k)$ such that $(t, u_1, \ldots, u_k) \models \phi(x_1, \ldots, x_k)$. Here $\mathrm{mark}(t, u_1, \ldots, u_k)$ is the tree t in which the nodes $u_1, \ldots, u_k$ are marked by special labels. This is a slight variation of a well-known technique, which is in fact used in the proof of Proposition 2.

Let Σ be an operator alphabet, and $k \geq 1$. Define $B_k = \{0, 1\}^k \setminus \{0\}^k$. Let $\Sigma \cup (\Sigma \times B_k)$ be the operator alphabet such that $\mathrm{rk}(\sigma) = \mathrm{rk}_\Sigma(\sigma)$ for every $\sigma \in \Sigma$, and $\mathrm{rk}(\langle \sigma, b_1, \ldots, b_k \rangle) = \mathrm{rk}_\Sigma(\sigma)$ for all $\langle \sigma, b_1, \ldots, b_k \rangle \in \Sigma \times B_k$. Note that the symbol $\langle \sigma, 0, \ldots, 0 \rangle$ is excluded; its role is played by the symbol σ. We use this alphabet to attach k different marks to the labels of the nodes of a tree. Let t be a tree over Σ, and let $u_1, \ldots, u_k \in V_t$. The *marked tree* $\mathrm{mark}(t, u_1, \ldots, u_k)$ over $\Sigma \cup (\Sigma \times B_k)$ is defined as (V_t, E_t, lab) where, for every $v \in V_t$, $\mathrm{lab}(v) = \mathrm{lab}_t(v)$ if $v \neq u_i$ for all i, and $\mathrm{lab}(v) = \langle \mathrm{lab}_t(v), (v = u_1), \ldots, (v = u_k) \rangle$ otherwise (where $(v = u_i) = 1$ iff v equals u_i).

The following lemma shows that it is possible to move arbitrarily between free variables in the formula and marks in the tree.

Lemma 5. *Let $k \geq 1$ and $j \in [1, k]$. For every $\phi(x_1, \ldots, x_k) \in \mathrm{MSOL}_k(\Sigma)$ there is a formula $\psi(x_{j+1}, \ldots, x_k) \in \mathrm{MSOL}_{k-j}(\Sigma \cup (\Sigma \times B_j))$ such that, for all $t \in T_\Sigma$ and $u_1, \ldots, u_k \in V_t$,*

$$(t, u_1, \ldots, u_k) \models \phi(x_1, \ldots, x_k) \text{ iff } (\mathrm{mark}(t, u_1, \ldots, u_j), u_{j+1}, \ldots, u_k) \models \psi(x_{j+1}, \ldots, x_k),$$

and vice versa, i.e., for every formula $\psi(x_{j+1},\ldots,x_k) \in \mathrm{MSOL}_{k-j}(\Sigma \cup (\Sigma \times B_j))$ *there exists a formula* $\phi(x_1,\ldots,x_k) \in \mathrm{MSOL}_k(\Sigma)$ *such that the above equivalence holds for all* $t \in T_\Sigma$ *and* $u_1,\ldots,u_k \in V_t$.

Proof. The proof is straightforward. From an MSO formula $\phi(x_1,\ldots,x_k)$ we construct $\psi(x_{j+1},\ldots,x_k)$ as follows:

$$\psi(x_{j+1},\ldots,x_k) = \\ \forall x_1,\ldots,x_j((\mathrm{marked}_1(x_1) \wedge \cdots \wedge \mathrm{marked}_j(x_j)) \rightarrow \phi'(x_1,\ldots,x_k)),$$

where $\mathrm{marked}_i(x) = \exists\langle\sigma,b_1,\ldots,b_j\rangle \in \Sigma \times B_j : b_i = 1 \wedge \mathrm{lab}_{\langle\sigma,b_1,\ldots,b_j\rangle}(x)$, and $\phi'(x_1,\ldots,x_k)$ is obtained from $\phi(x_1,\ldots,x_k)$ by replacing all occurrences of all $\mathrm{lab}_\sigma(y)$ by $(\mathrm{lab}_\sigma(y) \vee \exists(b_1,\ldots,b_j) \in B_j : \mathrm{lab}_{\langle\sigma,b_1,\ldots,b_j\rangle}(y))$.

The other way around is similar. Let us assume that the variables $x_1,\ldots,x_j$ do not occur in $\psi(x_{j+1},\ldots,x_k)$. We construct $\phi(x_1,\ldots,x_k)$ from $\psi(x_{j+1},\ldots,x_k)$ by replacing all occurrences of all $\mathrm{lab}_\sigma(y)$ by $(\mathrm{lab}_\sigma(y) \wedge \forall i \in [1,j] : y \neq x_i)$, and all occurrences of all $\mathrm{lab}_{\langle\sigma,b_1,\ldots,b_j\rangle}(y)$ by $(\mathrm{lab}_\sigma(y) \wedge \forall i \in [1,j] : (b_i = 1 \rightarrow y = x_i))$. □

Note that in the above proof we have made use of formulas that strictly speaking are not yet MSO formulas. As an example, the formula $\mathrm{marked}_i(x)$ really is the disjunction of all $\mathrm{lab}_{\langle\sigma,b_1,\ldots,b_j\rangle}(x)$ such that $\langle\sigma,b_1,\ldots,b_j\rangle \in \Sigma \times B_j$ and $b_i = 1$. In the remainder of the paper we will keep using such informal ways of writing MSO formulas.

The next corollary, which is the case $j = k$ of Lemma 5, allows us to apply Proposition 2 to formulas with free node variables.

Corollary 6. *Let* $k \geq 1$. *For every* $\phi(x_1,\ldots,x_k) \in \mathrm{MSOL}_k(\Sigma)$ *there is a closed* MSO *formula* $\psi \in \mathrm{MSOL}_0(\Sigma \cup (\Sigma \times B_k))$ *such that, for all* $t \in T_\Sigma$ *and* $u_1,\ldots,u_k \in V_t$,

$$(t,u_1,\ldots,u_k) \models \phi(x_1,\ldots,x_k) \text{ iff } \mathrm{mark}(t,u_1,\ldots,u_k) \models \psi,$$

and vice versa, i.e., for every formula $\psi \in \mathrm{MSOL}_0(\Sigma \cup (\Sigma \times B_k))$ *there exists a formula* $\phi(x_1,\ldots,x_k) \in \mathrm{MSOL}_k(\Sigma)$ *such that the above equivalence holds for all* $t \in T_\Sigma$ *and* $u_1,\ldots,u_k \in V_t$.

Using Corollary 6 and Proposition 2 one can construct, for every MSO formula $\phi(x_1,\ldots,x_k)$ over Σ, a finite tree automaton M over $\Sigma \cup (\Sigma \times B_k)$ such that

$$(t,u_1,\ldots,u_k) \models \phi(x_1,\ldots,x_k) \text{ iff } \mathrm{mark}(t,u_1,\ldots,u_k) \in L(M).$$

We will have to compare the behavior of M on $\mathrm{mark}(t,u_1,\ldots,u_k)$ with its behavior on t. Note that every tree t over Σ is also a tree over the alphabet of M, which is the reason for using σ instead of $\langle\sigma,0,\ldots,0\rangle$. Recall from Section 2 that, for a tree t and a node w of t, $\mathrm{state}_{M,t}(w)$ is the state in which M reaches w, and $\mathrm{succ}_{M,t}(w)$ is the set of successful states at w, i.e., the set of all states q such that M, assuming that it reaches node w in state q, will reach the root in a final state.

Lemma 7. *Let M be a finite tree automaton over $\Sigma \cup (\Sigma \times B_k)$. Let $t \in T_\Sigma$, $u_1, \ldots, u_k \in V_t$, and $w \in V_t$.*

1. *If w is not an ancestor of any u_i, $i \in [1,k]$, then* $\mathrm{state}_{M,\mathrm{mark}(t,u_1,\ldots,u_k)}(w) = \mathrm{state}_{M,t}(w)$.
2. *If w is an ancestor of every u_i, $i \in [1,k]$, then* $\mathrm{succ}_{M,\mathrm{mark}(t,u_1,\ldots,u_k)}(w) = \mathrm{succ}_{M,t}(w)$.

Proof. The proof of (1) is by an easy bottom-up induction on w. The proof of (2) is by an easy top-down induction on w, using statement (1). The details are left to the reader. □

Moreover, we will have to express the behavior of M on $t \in T_\Sigma$ as MSO definable properties of the nodes of t. For this purpose, we now show that, for each $q \in Q$, $\{(t,w) \mid \mathrm{state}_{M,t}(w) = q\}$ and $\{(t,w) \mid q \in \mathrm{succ}_{M,t}(w)\}$ are MSO definable node properties.

Lemma 8. *Let Σ be an operator alphabet, and let $M = (Q, \Sigma', \delta, F)$ be a finite tree automaton with $\Sigma \subseteq \Sigma'$. For every $q \in Q$ there are* MSO *formulas* $\mathrm{state}_q(x)$ *and* $\mathrm{succ}_q(x)$ *in* $\mathrm{MSOL}_1(\Sigma)$ *such that for all $t \in T_\Sigma$ and $w \in V_t$:*
$(t,w) \models \mathrm{state}_q(x)$ *iff* $\mathrm{state}_{M,t}(w) = q$, *and* $(t,w) \models \mathrm{succ}_q(x)$ *iff* $q \in \mathrm{succ}_{M,t}(w)$.

Proof. To obtain the formula $\mathrm{state}_q(x)$ we will construct a finite tree automaton M_q over $\Sigma \cup (\Sigma \times B_1)$ such that $\mathrm{mark}(t,w) \in L(M_q)$ iff $\mathrm{state}_{M,t}(w) = q$. Then, by Proposition 2 there is a closed formula ψ such that $\mathrm{mark}(t,w) \models \psi$ iff $\mathrm{mark}(t,w) \in L(M_q)$, and by Corollary 6 (for $k = 1$) there is a formula $\phi(x)$ such that $(t,w) \models \phi(x)$ iff $\mathrm{mark}(t,w) \models \psi$; thus, $\phi(x)$ is the required formula $\mathrm{state}_q(x)$. It is not difficult to construct the tree automaton M_q. It simulates M, but if it reaches the marked node, i.e., the node with some label $\langle \sigma, 1 \rangle$ (recall that $B_1 = \{1\}$), it verifies whether or not M is in state q, and continues to the root in an accepting or rejecting state, respectively. Formally, $M_q = (Q \cup \{a,r\}, \Sigma \cup (\Sigma \times B_1), \delta', \{a\})$ and δ' is defined as follows. Let $\sigma \in \Sigma_k$ and $q_1, \ldots, q_k \in Q \cup \{a,r\}$. If $q_i = a$ ($q_i = r$) for at least one i, then $\delta'_{\langle \sigma,1 \rangle}(q_1, \ldots, q_k) = \delta'_\sigma(q_1, \ldots, q_k) = a$ ($= r$, respectively). Now assume that all q_i are in Q. Then $\delta'_\sigma(q_1, \ldots, q_k) = \delta_\sigma(q_1, \ldots, q_k)$ and $\delta'_{\langle \sigma,1 \rangle}(q_1, \ldots, q_k) = a$ if $\delta_\sigma(q_1, \ldots, q_k) = q$ and r otherwise.

Similarly, to obtain the formula $\mathrm{succ}_q(x)$ it suffices to construct a finite tree automaton M'_q over $\Sigma \cup (\Sigma \times B_1)$ such that $\mathrm{mark}(t,w) \in L(M'_q)$ iff $q \in \mathrm{succ}_{M,t}(w)$. M'_q simulates M, but if it reaches the marked node, it switches to state q. Formally, $M'_q = (Q, \Sigma \cup (\Sigma \times B_1), \delta'', F)$ and for $\sigma \in \Sigma_k$ and $q_1, \ldots, q_k \in Q$, $\delta''_\sigma(q_1, \ldots, q_k) = \delta_\sigma(q_1, \ldots, q_k)$ and $\delta''_{\langle \sigma,1 \rangle}(q_1, \ldots, q_k) = q$. □

4 Tree-Walking Automata

A tree-walking automaton (see, e.g., [AhoUll, EngRozSlu, KamSlu]) is a finite-state automaton that walks on a tree from node to node, following the edges

of the tree (downwards or upwards). The automaton can test the label of the current node, and it can test the labels of the incident edges. To enhance the power of this basic model, we will allow the tree-walking automaton to test any MSO definable property of the current node, using MSO formulas with one free node variable. For a given tree t, two nodes u and v are in the node relation computed by the automaton if the automaton can walk from u to v in t, starting in an initial state and ending in a final state. In this section we define the tree-walking automaton, and we show that its behavior is determined by a regular "walking" language, consisting of all strings of instructions it can execute.

We start by defining the (infinite) set of instructions of our tree-walking automata; they will be called directives (as in [KlaSch]). Let Σ be an operator alphabet. The set of *directives* over Σ is

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

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

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

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

Let Σ be an operator alphabet. A *tree-walking automaton* (*with* MSO *tests*) over Σ is a finite automaton A over a finite subset Δ of D_Σ. We will call it a *basic* tree-walking automaton if Δ contains no other MSO formulas than $\mathrm{lab}_\sigma(x)$, for all $\sigma \in \Sigma$, $\mathrm{root}(x)$, and $\exists y(\mathrm{edg}_i(y,x))$, for all $i \in \mathrm{rki}(\Sigma)$.

We note here that the directives $\uparrow_i$, $i \in \mathrm{rki}(\Sigma)$, can be replaced by the single directive $\uparrow$, with $R_t(\uparrow) = \{(u,v) \mid (v,i,u) \in E_t \text{ for some } i \in \mathrm{rki}(\Sigma)\}$, for all $t \in T_\Sigma$. They are, however, convenient in examples and proofs.

For a tree-walking automaton $A = (Q, \Delta, \delta, I, F)$ and a tree t, an element (q,u) of $Q \times V_t$ is a *configuration* of the automaton. It signifies that A is in state q at node u. A configuration (q,u) is *initial* if $q \in I$, and *final* if $q \in F$. A pair of nodes (u,v) is in the relation defined by A if A can move from an initial configuration (q,u) to a final configuration (q',v). This is now formalized, in the obvious way. Recall that, for a directive d, the state transition relation $R_A(d)$ is defined in Section 2, and the node relation $R_t(d)$ is defined above.

Let $t \in T_\Sigma$. One step of $A = (Q, \Delta, \delta, I, F)$ on t is defined by the binary relation $\twoheadrightarrow_{A,t}$ on the set of configurations, as follows. For every $q, q' \in Q$ and

$u, u' \in V_t$,

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

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

$$R_t(A) = \{(u, v) \in V_t \times V_t \mid (q, u) \twoheadrightarrow^*_{A,t} (q', v) \text{ for some } q \in I \text{ and } q' \in F\}.$$

Thus, A computes the node relation

$$R(A) = \{(t, u, v) \mid t \in T_\Sigma, (u, v) \in R_t(A)\}.$$

$R(A)$ is called a *regular node relation*. The set of all regular node relations is denoted REG-R.

Example 1. Let Σ be the operator alphabet $\Sigma_0 \cup \Sigma_2$, with $\Sigma_0 = \{\text{white}, \text{red}\}$ and $\Sigma_2 = \{\sigma\}$. We consider a tree-walking automaton A over Σ, i.e., an automaton that walks on binary trees with white and red leaves. For a tree t over Σ, the automaton A connects certain leaves of t, i.e., $R_t(A)$ consists of pairs (u, v) where both u and v are leaves of t. If t has exactly one red leaf, say v_{red}, then all leaves have a pointer to that red leaf, i.e., $R_t(A)$ consists of all (u, v_{red}) where u is a leaf. Otherwise, i.e., if there is no red leaf or if there is more than one, all leaves are linked in left-to-right circular order, i.e., $R_t(A)$ consists of all (u, v) such that v is the next leaf after u, in that order. Note that $R_t(A)$ is a partial function for every $t \in T_\Sigma$.

Let 'orl' be a closed MSO formula that is true iff there is exactly one red leaf, i.e., $\text{orl} = \exists y(\text{lab}_{\text{red}}(y) \wedge \forall z(\text{lab}_{\text{red}}(z) \to z = y))$. We define $A = (Q, \Delta, \delta, I, F)$ with $Q = \{q_{\text{in}}, r, u, p, d, q_{\text{fin}}\}$, $I = \{q_{\text{in}}\}$, $F = \{q_{\text{fin}}\}$, $\Delta = \{\downarrow_1, \downarrow_2, \uparrow_1, \uparrow_2\} \cup \{\text{leaf}(x) \wedge \text{orl}, \text{leaf}(x) \wedge \neg\text{orl}, \text{lab}_{\text{red}}(x), \text{root}(x), \text{leaf}(x)\}$, and δ consists of the following transitions:

$(q_{\text{in}}, \text{leaf}(x) \wedge \text{orl}, r)$, $(q_{\text{in}}, \text{leaf}(x) \wedge \neg\text{orl}, u)$,

(r, d, r) for all $d \in \{\downarrow_1, \downarrow_2, \uparrow_1, \uparrow_2\}$, $(r, \text{lab}_{\text{red}}(x), q_{\text{fin}})$,

$(u, \uparrow_2, u)$, $(u, \text{root}(x), d)$, $(u, \uparrow_1, p)$, $(p, \downarrow_2, d)$, and

$(d, \downarrow_1, d)$, $(d, \text{leaf}(x), q_{\text{fin}})$.

On a given tree, the automaton A first checks that it is at a leaf, and tests whether or not there is exactly one red leaf. If so, it walks nondeterministically on the tree (in state r) until it happens to find that red leaf. If not, it walks to the "next" leaf along the shortest path: it first moves upwards over 2-labeled edges as far as possible (in state u); then it either is at the root or it moves to the other child of its parent (visiting its parent in state p); and finally it moves downwards along 1-labeled edges as far as possible (in state d). □

The node relation computed by a tree-walking automaton depends only on the language (of strings of directives) it recognizes. In other words, for tree-walking automata A_1 and A_2, if $L(A_1) = L(A_2)$ then $R(A_1) = R(A_2)$. We prove this by associating a node relation with every language of strings of directives, in a natural way (as is well known from program scheme theory, cf. [Eng1]).

Let Σ be an operator alphabet. A *walking language* over Σ is a (string) language over a finite subset of D_Σ. A *walking string*, i.e., a string from a walking language, can be viewed as a sequence of instructions to be executed as a walk on a tree; the language itself can be viewed as a nondeterministic choice between such instruction sequences. This is formalized next. Let $t \in T_\Sigma$. A walking string $w \in D_\Sigma^*$ computes the relation $R_t(w) \subseteq V_t \times V_t$, defined as follows: $R_t(\varepsilon)$ is the identity on V_t, and, for $d_1, \ldots, d_n \in D_\Sigma$, $R_t(d_1 \cdots d_n) = R_t(d_1) \circ \cdots \circ R_t(d_n)$. A walking language $W \subseteq D_\Sigma^*$ computes the relation $R_t(W) = \bigcup\{R_t(w) \mid w \in W\}$. Now, a walking language W computes the node relation $R(W) = \{(t, u, v) \mid t \in T_\Sigma \text{ and } (u, v) \in R_t(W)\}$.

Example 2. We define a walking language W that computes the same node relation as the tree-walking automaton A from Example 1. Let Δ be defined as in Example 1. Then W is the regular language $W_1 \cup W_2$ over Δ, where

$$W_1 = (\mathrm{leaf}(x) \wedge \mathrm{orl}) \cdot \{\downarrow_1, \downarrow_2, \uparrow_1, \uparrow_2\}^* \cdot \mathrm{lab}_{\mathrm{red}}(x), \text{ and}$$
$$W_2 = (\mathrm{leaf}(x) \wedge \neg\mathrm{orl}) \cdot \uparrow_2^* \cdot \{\uparrow_1 \cdot \downarrow_2, \mathrm{root}(x)\} \cdot \downarrow_1^* \cdot \mathrm{leaf}(x).$$

Note that $W = L(A)$. According to the next lemma this implies that $R(W) = R(A)$, which should also intuitively be clear. □

We now show that the node relation computed by a tree-walking automaton depends only on the language it recognizes.

Lemma 9. *For every tree-walking automaton A, $R(A) = R(L(A))$.*

Proof. Let $A = (Q, \Delta, \delta, I, F)$. We have to show that $R_t(A) = R_t(L(A))$ for every tree t. It is straightforward to prove that, for all $q, q' \in Q$, $u, u' \in V_t$, and $w = d_1 \cdots d_n$ with $d_i \in \Delta$, the following equivalence holds: there is a walk $(q_0, u_0) \overset{d_1}{\twoheadrightarrow}_{A,t} (q_1, u_1) \overset{d_2}{\twoheadrightarrow}_{A,t} \cdots \overset{d_n}{\twoheadrightarrow}_{A,t} (q_n, u_n)$ of A on t with $(q_0, u_0) = (q, u)$ and $(q_n, u_n) = (q', u')$ if and only if $(q, q') \in R_A(w)$ and $(u, u') \in R_t(w)$. This equivalence implies that $R_t(A) = \{(u, u') \mid \exists q \in I, q' \in F, w \in \Delta^* : (q, q') \in R_A(w) \text{ and } (u, u') \in R_t(w)\} = \{(u, u') \mid \exists w \in L(A) : (u, u') \in R_t(w)\} = R_t(L(A))$. □

As a corollary we obtain the fact that a node relation can be computed by a tree-walking automaton iff it can be computed by a regular walking language (i.e., a walking language that can be recognized by a finite automaton).

Corollary 10. REG-R *is the set of all $R(W)$ where W is a regular walking language.*

It is not difficult to see how the well-known language operations of union ($W_1 \cup W_2$), concatenation ($W_1 \cdot W_2$), and Kleene star (W^*), when applied to walking languages, correspond to operations on the relations they compute. In fact, union of languages corresponds to union of relations, concatenation of languages to composition of relations, and Kleene star of a language to the transitive reflexive closure of a relation. This is expressed in the next lemma (cf. Lemma 2.9 of [Eng1]).

Lemma 11. *Let t be a tree over Σ, and let W, W_1, W_2 be walking languages over Σ. Then $R_t(W_1 \cup W_2) = R_t(W_1) \cup R_t(W_2)$, $R_t(W_1 \cdot W_2) = R_t(W_1) \circ R_t(W_2)$, and $R_t(W^*) = R_t(W)^*$.*

Proof. The first equality is immediate from the definition of $R_t(W)$. It also holds for arbitrary unions, i.e., if W_i is a walking language for every $i \in \mathbb{N}$, then $R_t(\bigcup_{i\in\mathbb{N}} W_i) = \bigcup_{i\in\mathbb{N}} R_t(W_i)$. The second equality is proved as follows:

$$\begin{aligned} R_t(W_1 \cdot W_2) &= R_t(\{w_1 \cdot w_2 \mid w_1 \in W_1, w_2 \in W_2\}) \\ &= \bigcup\{R_t(w_1 \cdot w_2) \mid w_1 \in W_1, w_2 \in W_2\} \\ &= \bigcup\{R_t(w_1) \circ R_t(w_2) \mid w_1 \in W_1, w_2 \in W_2\} \\ &= \bigcup\{R_t(w_1) \mid w_1 \in W_1\} \circ \bigcup\{R_t(w_2) \mid w_2 \in W_2\} \\ &= R_t(W_1) \circ R_t(W_2). \end{aligned}$$

This implies that $R_t(W^i) = R_t(W)^i$ for every $i \in \mathbb{N}$, and hence that $R_t(W^*) = R_t(\bigcup_{i\in\mathbb{N}} W^i) = \bigcup_{i\in\mathbb{N}} R_t(W^i) = \bigcup_{i\in\mathbb{N}} R_t(W)^i = R_t(W)^*$. □

5 Characterizing MSO Node Relations

In this section we show one of our main results: a binary node relation can be defined by an MSO formula iff it can be computed by a tree-walking automaton. Moreover, if the formula defines a functional node relation on every tree, then the automaton is deterministic. We also show that the MSO tests of the automaton are really needed; there is a (functional) binary formula that cannot be computed by any basic tree-walking automaton.

We first prove that the node relation of a tree-walking automaton is MSO definable. Corollary 10 allows us to show in a straightforward way, using Kleene's theorem, that all regular node relations are MSO definable.

Lemma 12. REG-R $\subseteq$ MSO-R.

Proof. Let Δ be a finite subset of D_Σ. By Kleene's theorem, the class of regular (string) languages over Δ is the smallest class of languages that is closed under union, concatenation, and Kleene star, and contains the empty language and every language $\{d\}$, with $d \in \Delta$. By induction on this characterization we define for every regular walking language W a formula $\phi_W(x, y) \in \mathrm{MSOL}_2(\Sigma)$, such

that $R(\phi_W) = R(W)$. This proves the inclusion by Corollary 10. We use 'false' to stand for any formula that never holds (such as $\exists x : \neg(x = x)$). Recall also the definition of the transitive reflexive closure $\phi^*(x, y)$ of a formula $\phi(x, y)$ from Section 3 (just before Proposition 4). Let $i \in \mathrm{rki}(\Sigma)$, $\psi(x) \in \mathrm{MSOL}_1(\Sigma)$, and let W, W_1, W_2 be walking languages over Σ. Then we define

$$\begin{aligned}
\phi_{\emptyset}(x, y) &= \text{false} \\
\phi_{\{\downarrow_i\}}(x, y) &= \mathrm{edg}_i(x, y) \\
\phi_{\{\uparrow_i\}}(x, y) &= \mathrm{edg}_i(y, x) \\
\phi_{\{\psi(x)\}}(x, y) &= \psi(x) \wedge (x = y) \\
\phi_{W_1 \cup W_2}(x, y) &= \phi_{W_1}(x, y) \vee \phi_{W_2}(x, y) \\
\phi_{W_1 . W_2}(x, y) &= \exists z(\phi_{W_1}(x, z) \wedge \phi_{W_2}(z, y)) \\
\phi_{W^*}(x, y) &= \phi^*_W(x, y)
\end{aligned}$$

The correctness of the first four cases should be clear. The remaining three cases follow from Lemma 11, together with the following equations that hold for every tree $t \in T_\Sigma$: $R_t(\phi_{W_1 \cup W_2}) = R_t(\phi_{W_1}) \cup R_t(\phi_{W_1})$, $R_t(\phi_{W_1 . W_2}) = R_t(\phi_{W_1}) \circ R_t(\phi_{W_2})$, and $R_t(\phi_{W^*}) = R_t(\phi_W)^*$, of which the last one follows from Proposition 4. □

We note here that Lemma 12 also holds for graphs in general, with the generalization of all relevant concepts from trees to graphs (in particular that of a tree-walking automaton to a graph-walking automaton), see Theorem 5 of [BloEng1]. However, in that case the inclusion is of course proper. For instance, the relation that holds between two nodes u and v when they are in different connected components of the graph, is MSO definable by the formula $\neg(\phi^*(x, y))$ where $\phi(x, y) = \mathrm{edg}(x, y) \vee \mathrm{edg}(y, x)$. But a graph-walking automaton cannot compute this relation, simply because it cannot walk from u to v. However, as shown in Theorem 6 of [BloEng1], even for connected graphs there are node relations that can be defined by an MSO formula, but cannot be computed by a graph-walking automaton.

We now prove the first main result of this paper. The idea of the proof is similar to the one of Lemma 12 of [EngOos2], which is a key lemma in the proof of the main result of [EngOos1, EngOos2], viz. the equivalence between context-free graph grammars and MSO transductions of trees. In that lemma, there is no tree-walking automaton, but an appropriate relabeling of the nodes of the tree (instead of the MSO tests) and a regular language of strings of node labels (similar to the walking language of the automaton, see Lemma 9).

Theorem 13. MSO-R = REG-R.

Proof. By Lemma 12 it suffices to show MSO-R $\subseteq$ REG-R. Let Σ be an operator alphabet and let $\phi(x, y) \in \mathrm{MSOL}_2(\Sigma)$. By Corollary 6 (for $k = 2$) and Proposition 2 (applied to the resulting ψ) there is a finite tree automaton M over $\Sigma \cup (\Sigma \times B_2)$ such that for every tree $t \in T_\Sigma$ and nodes u and v of t, $(t, u, v) \models \phi(x, y)$ if and only if $\mathrm{mark}(t, u, v) \in L(M)$.

Let $M = (Q, \Sigma \cup (\Sigma \times B_2), \delta, F)$. We will prove that there exists a tree-walking automaton A over Σ with $R(A) = R(\phi)$, i.e., for every tree $t \in T_\Sigma$ and nodes u and v of t, $(u, v) \in R_t(A)$ iff $\mathrm{mark}(t, u, v) \in L(M)$. This tree-walking automaton A behaves in a special way: it walks along paths in the tree t in which no edge occurs more than once, that is, for every pair of nodes $(u, v) \in R_t(A)$ it always takes the shortest path from u to v. In fact, A simulates the finite tree automaton M on the path from u to v, using MSO definable properties of the nodes on that path to get information on the behavior of the automaton M on the rest of the tree. More precisely, A simulates the computation of M on the path from u to the least common ancestor $\mathrm{lca}(u, v)$ of u and v, and then simulates the reverse computation of M on the path from $\mathrm{lca}(u, v)$ to v, nondeterministically.

The behavior of M on the parts of the tree outside the shortest path from u to v, can be expressed through the formulas $\mathrm{state}_q(x)$ and $\mathrm{succ}_q(x)$ in $\mathrm{MSOL}_2(\Sigma)$, defined in Lemma 8. The formulas $\mathrm{state}_q(x)$ are used to compute the state of M at all those nodes that are children of nodes on the shortest path, but do not lie on that path themselves. The formulas $\mathrm{succ}_q(x)$ are used to check whether a given state q is successful at the node $\mathrm{lca}(u, v)$. The reader should realize that M works on $\mathrm{mark}(t, u, v)$ whereas A walks on t. Thus, the formulas $\mathrm{state}_q(x)$ and $\mathrm{succ}_q(x)$ were defined for t, and not for $\mathrm{mark}(t, u, v)$. However, since the parts of t outside the shortest path from u to v are labeled by symbols from Σ, Lemma 7 guarantees that A will test the correct information on the behavior of M.

We now define the tree-walking automaton $A = (Q_A, \Delta_A, \delta_A, I_A, F_A)$ over Σ. To simplify the description of A we allow its transition relation δ_A to contain transitions of the form $(q, d_1 \cdots d_n, q')$ with $q, q' \in Q_A$ and $d_1, \ldots, d_n \in \Delta_A$; such a transition stands for the n transitions

$$(q, d_1, q_1), (q_1, d_2, q_2), \ldots, (q_{n-2}, d_{n-1}, q_{n-1}), (q_{n-1}, d_n, q')$$

where $q_1, \ldots, q_{n-1}$ are (unique) new states.

The set of states of A is $Q_A = \{q_{\mathrm{in}}, q_{\mathrm{fin}}\} \cup \{\mathrm{up}_q \mid q \in Q\} \cup \{\mathrm{down}_q \mid q \in Q\}$, with $I_A = \{q_{\mathrm{in}}\}$ and $F_A = \{q_{\mathrm{fin}}\}$. The states up_q are used by A when walking from a node u to $\mathrm{lca}(u, v)$ and simulating M, whereas it uses the states down_q when walking from $\mathrm{lca}(u, v)$ to the node v and simulating M reversely. We define Δ_A to be the set of all directives that occur in δ_A.

It remains to define δ_A. Let $k \in \mathbb{N}$, $\sigma \in \Sigma_k$, and $q_1, \ldots, q_k, q \in Q$. For each such choice, δ_A contains the transitions as specified below in points (1-8). Apart from the formula $\mathrm{succ}_q(x)$, we will use the unary formula

$$\mathrm{test}(x) = \mathrm{lab}_\sigma(x) \wedge \forall m \in [1, k] : \forall y(\mathrm{edg}_m(x, y) \rightarrow \mathrm{state}_{q_m}(y))$$

which expresses that the current node has label σ and $q_1, \ldots, q_k$ are the states in which M reaches its children. Similarly, for $i, j \in [1, k]$ we will use the formulas $\mathrm{test}_i(x)$ and $\mathrm{test}_{i,j}(x)$ which are the same as $\mathrm{test}(x)$ except that the quantification of m is restricted to $[1, k] \setminus \{i\}$ and $[1, k] \setminus \{i, j\}$, respectively. Recall that $B_2 = \{(1, 0), (0, 1), (1, 1)\}$ where $(1, 0)$ indicates u and $(0, 1)$ indicates v (if $u \neq v$) and $(1, 1)$ indicates both u and v (if $u = v$).

(1) Start moving up:
if $\delta_{\langle\sigma,1,0\rangle}(q_1,\ldots,q_k) = q$, then

$$(q_{\text{in}}, \text{test}(x), \text{up}_q) \in \delta_A.$$

(2) Move one step up:
if $\delta_\sigma(q_1,\ldots,q_k) = q$, then, for all $i \in [1,k]$,

$$(\text{up}_{q_i}, \uparrow_i \cdot \text{test}_i(x), \text{up}_q) \in \delta_A.$$

(3) Turn around:
if $\delta_\sigma(q_1,\ldots,q_k) = q$, then, for all $i,j \in [1,k]$ with $i \neq j$,

$$(\text{up}_{q_i}, \uparrow_i \cdot(\text{test}_{i,j}(x) \wedge \text{succ}_q(x))\cdot \downarrow_j, \text{down}_{q_j}) \in \delta_A.$$

(4) Move one step down:
if $\delta_\sigma(q_1,\ldots,q_k) = q$, then, for all $i \in [1,k]$,

$$(\text{down}_q, \text{test}_i(x)\cdot \downarrow_i, \text{down}_{q_i}) \in \delta_A.$$

(5) Stop moving down:
if $\delta_{\langle\sigma,0,1\rangle}(q_1,\ldots,q_k) = q$, then

$$(\text{down}_q, \text{test}(x), q_{\text{fin}}) \in \delta_A.$$

Transition types (1-5) deal with the case that u and v are independent nodes, i.e., one is not an ancestor of the other: A walks from u up to some node z, turns around, and walks from z down to v. The fact that, when turning around, A moves to a different child of z (in (3): $i \neq j$) guarantees that $z = \text{lca}(u,v)$. In the next transition types we additionally deal with the cases that v is a proper descendant of u, that v is a proper ancestor of u, and that $v = u$, respectively.

(6) Start moving down:
if $\delta_{\langle\sigma,1,0\rangle}(q_1,\ldots,q_k) = q$, then, for all $i \in [1,k]$,

$$(q_{\text{in}}, (\text{test}_i(x) \wedge \text{succ}_q(x))\cdot \downarrow_i, \text{down}_{q_i}) \in \delta_A.$$

(7) Stop moving up:
if $\delta_{\langle\sigma,0,1\rangle}(q_1,\ldots,q_k) = q$, then, for all $i \in [1,k]$,

$$(\text{up}_{q_i}, \uparrow_i \cdot(\text{test}_i(x) \wedge \text{succ}_q(x)), q_{\text{fin}}) \in \delta_A.$$

(8) Start and stop:
if $\delta_{\langle\sigma,1,1\rangle}(q_1,\ldots,q_k) = q$, then

$$(q_{\text{in}}, \text{test}(x) \wedge \text{succ}_q(x), q_{\text{fin}}) \in \delta_A.$$

This ends the description of the tree-walking automaton A. To show the correctness of A the following three statements can be proved:

(Sa) For every $t \in T_\Sigma$ and all nodes u, v, w of t such that w is not an ancestor of v, $(q_{\text{in}}, u) \twoheadrightarrow^*_{A,t} (\text{up}_q, w)$ if and only if w is an ancestor of u and $q = \text{state}_{M,\text{mark}(t,u,v)}(w)$.

(Sb) For every $t \in T_\Sigma$ and all nodes u, v, w of t such that w is an ancestor of v, $(q_{\text{in}}, u) \twoheadrightarrow^*_{A,t} (\text{down}_q, w)$ if and only if w is not an ancestor of u and $q \in \text{succ}_{M,\text{mark}(t,u,v)}(w)$.

(Sc) For every $t \in T_\Sigma$ and all nodes u, v of t, $(q_{\text{in}}, u) \twoheadrightarrow^*_{A,t} (q_{\text{fin}}, v)$ if and only if $\text{state}_{M,\text{mark}(t,u,v)}(v) \in \text{succ}_{M,\text{mark}(t,u,v)}(v)$.

Statements (Sa) and (Sb) can easily be proved by induction on the length of the given walk, and on the distance between w and u. (Sa) is a consequence of transition types (1) and (2), and (Sb) follows from (Sa) and transition types (3), (6), and (4). Statement (Sc) can then be proved from (Sa) and (Sb), and transition types (5), (7), and (8), by considering the last step of the walk and the last node on the path from u to v. All the proofs need Lemmas 7 and 8. The details of the proofs are left to the reader. The correctness of A now follows from (Sc) and Lemma 1: $\text{state}_{M,\text{mark}(t,u,v)}(v) \in \text{succ}_{M,\text{mark}(t,u,v)}(v)$ iff $\text{mark}(t, u, v) \in L(M)$. □

We have proved that every MSO specification $\phi(x, y)$ of a binary relation on the nodes of a tree can be implemented by a tree-walking automaton A. However, in general the automaton is nondeterministic, i.e., chooses nondeterministically its way in the tree. This is of course unavoidable if, for some tree t, $R_t(\phi)$ is not a function: then there are distinct (u, v_1) and (u, v_2) in $R_t(\phi)$ and, after starting at node u, A has to walk either to v_1 or to v_2, nondeterministically. We now show that every functional MSO specification can be implemented on a deterministic tree-walking automaton.

A node relation R is *functional* if, for every tree t, the relation $\{(u, v) \mid (t, u, v) \in R\}$ is functional, i.e., is a partial function. We also say that an MSO formula, a tree-walking automaton, or a walking language is functional if it defines or computes a functional node relation. The tree-walking automaton of Example 1 and the walking language of Example 2 are functional. The walking languages in [KlaSch], which are described by so-called routing expressions (similar to those in Example 2), are required to be functional.

A tree-walking automaton $A = (Q, \Delta, \delta, I, F)$ over Σ is *deterministic* if the following three conditions hold: (1) I is a singleton, (2) if $(q, d, q') \in \delta$, then $q \notin F$, and (3) for all distinct transitions $(q, d_1, q_1), (q, d_2, q_2) \in \delta$, d_1 and d_2 are two mutually exclusive formulas in $\text{MSOL}_1(\Sigma)$. Here, two formulas $\phi_1(x), \phi_2(x) \in \text{MSOL}_1(\Sigma)$ are *mutually exclusive* if $t \models \neg\exists x(\phi_1(x) \wedge \phi_2(x))$ for every $t \in T_\Sigma$; note that this holds iff $L(\exists x(\phi_1(x) \wedge \phi_2(x))) = \emptyset$, and hence is decidable by Proposition 3. Obviously, every deterministic tree-walking automaton is functional.

Theorem 14. *For every functional tree-walking automaton A over Σ there is a deterministic tree-walking automaton A' over Σ with $R(A') = R(A)$.*

Proof. Let $A = (Q, \Delta, \delta, I, F)$. We may assume that I is a singleton. If not, add all transitions $(q_{\text{in}}, \text{lab}_\sigma(x), q)$, with $\sigma \in \Sigma$ and $q \in I$, to δ where q_{in} is the new initial state.

The automaton A' simulates A, but whenever A can choose between the execution of several different transitions, A' executes just one of these transitions, after having tested that its execution can be continued with a successful walk of A on the tree. To see that this is an MSO definable node property, let, for every $q \in Q$, $\text{sw}_q(x)$ be the MSO formula such that $(t, u) \models \text{sw}_q(x)$ iff A has a successful walk on t that starts in configuration (q, u), i.e., $(q, u) \twoheadrightarrow^*_{A,t} (q', v)$ for some final configuration (q', v). Note that $\text{sw}_q(x) = \exists y(\phi_q(x, y))$, where $\phi_q(x, y)$ is the MSO formula corresponding to the tree-walking automaton $A_q = (Q, \Delta, \delta, \{q\}, F)$ according to Lemma 12. Now, for a transition $(p, d, q) \in \delta$, let $\text{sw}_{d,q}(x)$ be the MSO formula defined as follows: if $d = \downarrow_i$ then $\text{sw}_{d,q}(x) = \exists y(\text{edg}_i(x, y) \wedge \text{sw}_q(y))$, if $d = \uparrow_i$ then $\text{sw}_{d,q}(x) = \exists y(\text{edg}_i(y, x) \wedge \text{sw}_q(y))$, and if $d = \psi(x)$ then $\text{sw}_{d,q}(x) = \psi(x) \wedge \text{sw}_q(x)$. Clearly, the formula $\text{sw}_{d,q}(x)$ expresses that the execution of transition (p, d, q) can be continued with a successful walk.

Let $(p, d_1, q_1), \ldots, (p, d_n, q_n)$ be all transitions in δ that start with a state $p \notin F$. Corresponding to these, the automaton A' has the following transitions, where $p_1, \ldots, p_n$ are new states:

$$(p, \text{sw}_{d_1,q_1}(x), p_1),\ (p, \neg\, \text{sw}_{d_1,q_1}(x) \wedge \text{sw}_{d_2,q_2}(x), p_2),\ \ldots,$$
$$(p, \neg\, \text{sw}_{d_1,q_1}(x) \wedge \cdots \wedge \neg\, \text{sw}_{d_{n-1},q_{n-1}}(x) \wedge \text{sw}_{d_n,q_n}(x), p_n),$$
$$\text{and } (p_1, d_1, q_1), \ldots, (p_n, d_n, q_n).$$

The automaton A' has no other transitions; it has the same initial state and final states as A. Obviously, A' is deterministic and computes the same node relation as A. □

Since every deterministic tree-walking automaton is functional, Theorems 13 and 14 together show that the class of node relations that can be computed by deterministic tree-walking automata is exactly the class of functional MSO definable node relations. In a formula: fMSO-R = dREG-R, where 'f' and 'd' indicate functionality and determinism, respectively.

We end this section by showing that the MSO tests of the tree-walking automaton are necessary: the usual tree-walking automaton from the literature (see, e.g., [AhoUll, EngRozSlu]) cannot compute all MSO definable node relations. Recall from Section 4 that a *basic* tree-walking automaton over Σ is one that uses only the MSO tests $\text{lab}_\sigma(x)$, $\sigma \in \Sigma$, that test the label of node x, and the MSO tests $\text{root}(x)$ and $\exists y(\text{edg}_i(y, x))$, $i \in \text{rki}(\Sigma)$, that test whether x is the root or the i-th child of its parent. A regular walking language that only makes use of these MSO tests, is called a *routing language* in [KlaSch]. Regular tree languages together with functional routing languages are proposed in [KlaSch] as a way of defining recursive data structures, built with pointers. Recursive data structures have an intrinsic tree structure, determined by a regular tree language. The routing languages are used to specify additional pointers in the tree structure, for instance to produce the data structure of circularly linked lists

or root-linked binary trees. If t is an instance of a data structure (i.e., a tree), a tuple (u, v) in the relation defined by the routing language signifies a pointer from node u to node v. Since the routing language is functional, the pointer at u has a unique destination v. It is shown in [KlaSch] that many useful data structures can be defined in this way. We show here that there are data structures that can be defined by functional regular walking languages, but not by routing languages. Note that by Lemma 9 a node relation can be computed by a routing language iff it can be computed by a basic tree-walking automaton.

Theorem 15. *There is a deterministic tree-walking automaton A, such that there is no* basic *tree-walking automaton B with $R(B) = R(A)$.*

Proof. Let Σ be the operator alphabet of Example 1, and let A_{red} be the tree-walking automaton of Example 1. Since A_{red} is functional, there is a deterministic tree-walking automaton A that computes the same node relation, by Theorem 14. This node relation cannot be defined by a basic tree-walking automaton. We prove so by contradiction.

Suppose there is a basic tree-walking automaton $B = (Q, \Delta, \delta, I, F)$ with $R(B) = R(A_{\mathrm{red}})$. We may assume that $I = \{q_0\}$, cf. the proof of Theorem 14. Intuitively, when B starts at a leaf u of a tree with white leaves only, it first would have to visit all other leaves in order to be sure that they are all white. However, in doing so, B cannot remember its starting point u, and therefore is not able to find the leaf that is next to u in the left-to-right circular order. Formally this is proved as follows.

Let t be any tree over Σ with $\#Q + 1$ leaves, which are all white. Let t' be the tree obtained from t by changing the label of one of the leaves, say, the first leaf, into red. Name the leaves in both trees u_0 through $u_{\#Q}$ in left-to-right order. Thus, u_0 is white in t and red in t', and $u_1, \ldots, u_{\#Q}$ are white in both trees. In the following, let $\mathrm{succ}(k) = (k + 1) \bmod (\#Q + 1)$, giving the successor of a node number in left-to-right circular order. Since t has no red leaves, for every $k \in [0, \#Q]$, there is a state $f_k \in F$, such that $(q_0, u_k) \twoheadrightarrow^*_{B,t} (f_k, u_{\mathrm{succ}(k)})$. On the other hand, since t' has exactly one red leaf, for all $k \in [0, \#Q - 1]$, there is *no* $f \in F$ such that $(q_0, u_k) \twoheadrightarrow^*_{B,t'} (f, u_{\mathrm{succ}(k)})$. This implies that for all $k \in [0, \#Q - 1]$ the walk of B on t from u_k to $u_{\mathrm{succ}(k)}$ must visit u_0. In fact, if it would not visit u_0 the same walk could also be done on t' because B cannot test the label of a node that it does not visit. Since the walk on t from $u_{\#Q}$ to u_0 also visits u_0, this means that for all $k \in [0, \#Q]$, there is a $q_k \in Q$, with

$$(q_0, u_k) \twoheadrightarrow^*_{B,t} (q_k, u_1) \twoheadrightarrow^*_{B,t} (f_k, u_{\mathrm{succ}(k)}).$$

But since there are $\#Q + 1$ possibilities for k, and only $\#Q$ states, there have to be distinct $k, k' \in [0, \#Q]$ such that $q_k = q_{k'}$. This implies

$$(q_0, u_k) \twoheadrightarrow^*_{B,t} (q_k, u_1) = (q_{k'}, u_1) \twoheadrightarrow^*_{B,t} (f_{k'}, u_{\mathrm{succ}(k')})$$

and thus $(u_k, u_{\mathrm{succ}(k')}) \in R_t(B)$ with $k \neq k'$, a contradiction. □

This shows that, when defining additional pointers in trees, MSO specifications are more powerful than routing languages. Moreover, MSO logic is of course a higher level specification language than regular expressions or finite automata. With respect to efficiency, it is shown in [KlaSch] that, in recursive data structures, the pointers that are defined by (functional) routing languages can be computed in linear time. It will be shown in Section 7 (Theorem 21) that functional MSO specifications can be evaluated just as efficiently as functional routing languages. Altogether, when defining additional pointers in trees, one may as well use MSO specifications instead of routing languages.

Tree-walking automata can also be used to recognize tree languages: for a tree-walking automaton A over Σ, the tree language recognized by A is $\{t \in T_\Sigma \mid (\mathrm{root}(t), \mathrm{root}(t)) \in R_t(A)\}$. It is not difficult to prove, using Proposition 2 and Theorem 13, that tree-walking automata with MSO tests recognize exactly the regular tree languages. In analogy to the case of node relations (Theorem 15), we conjecture that there is a regular tree language that cannot be recognized by a basic tree-walking automaton. A similar result has been proved in Theorem 5.5 of [KamSlu]. However, the tree-walking automaton in [KamSlu] has directive $\uparrow$ instead of all directives $\uparrow_i$ with $i \in \mathrm{rki}(\Sigma)$, and does not have the tests $\exists y(\mathrm{edg}_i(y, x))$, $i \in \mathrm{rki}(\Sigma)$. This means that it cannot test the "child number" of a node, and hence it cannot even perform a systematic search of the input tree. Such an automaton seems to be too weak to be interesting. The tree-walking automata of [AhoUll, EngRozSlu, KamSlu] are also of this type, but they are used to compute string transductions, and hence the child number of a node of the input tree can always be added to its label.

6 Characterizing MSO Node Properties

In this section we show our second main result: a node property can be defined by an MSO formula iff it can be computed by an attribute grammar of which all attributes have finitely many values. We start by recalling some terminology concerning attribute grammars (see, e.g., [Knu, DerJouLor, Eng2]). Then we define the so-called node-selecting attribute grammar, which computes a property of the nodes of the input tree, and show that these attribute grammars compute exactly the MSO definable node properties. Finally, we define the concept of an attributed relabeling of the nodes of a tree, and characterize the MSO definable node relations by basic tree-walking automata that walk on relabeled trees.

Attribute Grammars

In order to allow the attribute grammar to work on arbitrary trees over an operator alphabet, rather than on derivation trees of an underlying context-free grammar, we consider a slight variation of the attribute grammar that was introduced in [Fül]. The semantic rules of the attribute grammar are grouped by operator rather than by grammar production, and there are special semantic rules for the inherited attributes of the root. All operators have the same attributes.

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

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

    $$\langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_r, i_r \rangle)$$

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

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

    for every $\alpha_0 \in I$, where $r \geq 0$, $\alpha_1, \ldots, \alpha_r \in S \cup I$, and f is a function from $W(\alpha_1) \times \cdots \times W(\alpha_r)$ to $W(\alpha_0)$.

Usually, for an (internal or root) rule $\langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_r, i_r \rangle)$ the function f is given as $f = \lambda x_1, \ldots, x_r.e$ for some expression e with variables in $\{x_1, \ldots, x_r\}$. We will then informally denote the rule by $\langle \alpha_0, i_0 \rangle = e'$ where e' is obtained from e by substituting $\langle \alpha_j, i_j \rangle$ for x_j, for all $j \in [1, r]$.

Noncircularity is defined for attribute grammars in the usual way, and can be tested in the usual way (see [Knu]). All attribute grammars are assumed to be noncircular.

If all sets in Ω are finite, then G is said to be *finite-valued*. This means that each attribute has finitely many values. For our characterization of MSO definable properties we will consider finite-valued attribute grammars only.

Let t be a tree over Σ. The set of attributes of t is $A(t) = (S \cup I) \times V_t$. We now define how to give the correct values to the attributes of the tree. Let dec be a function from $A(t)$ to $\cup\Omega$, such that $\mathrm{dec}(\langle \alpha, u \rangle) \in W(\alpha)$ for all $\langle \alpha, u \rangle \in A(t)$. Recall from Section 2 that, for $u \in V_t$, $u \cdot 0 = u$. The function dec is a *rootless decoration* of t if all internal rules are satisfied, i.e., for every node $u \in V_t$ and every rule

$$\langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_r, i_r \rangle)$$

in $R(\mathrm{lab}_t(u))$,

$$\mathrm{dec}(\langle \alpha_0, u \cdot i_0 \rangle) = f(\mathrm{dec}(\langle \alpha_1, u \cdot i_1 \rangle), \ldots, \mathrm{dec}(\langle \alpha_r, u \cdot i_r \rangle)).$$

The function dec is a *decoration* of t if it is a rootless decoration of t and, moreover, all root rules are satisfied, i.e., for every rule

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

in $R(\text{root})$,

$$\text{dec}(\langle \alpha_0, \text{root}(t) \rangle) = f(\text{dec}(\langle \alpha_1, \text{root}(t) \rangle), \ldots, \text{dec}(\langle \alpha_r, \text{root}(t) \rangle)).$$

Since G is noncircular, every tree t has a unique decoration, which will be denoted by $\text{dec}_{G,t}$. This decoration can be computed bottom-up, in a nondeterministic way. That is of course not the way it is usually done, but we need it to show that a finite tree automaton can simulate an attribute grammar with finite semantic domains. The idea is to compute a rootless decoration of every subtree of the input tree, and check the root rules at the root. The following easy lemma shows how to make a bottom-up move in the tree. Recall from Section 2 that t_u denotes the subtree of t rooted at node u.

Lemma 16. *Let t be a tree over Σ, and let u be a node of t. A function* dec *is a rootless decoration of t_u iff the following conditions hold:*

- *the restriction of* dec *to $A(t_{u \cdot i})$ is a rootless decoration of $t_{u \cdot i}$, for every $i \in [1, \text{rk}(\text{lab}_t(u))]$, and*
- *for every rule*

  $$\langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_r, i_r \rangle)$$

  in $R(\text{lab}_t(u))$,

  $$\text{dec}(\langle \alpha_0, u \cdot i_0 \rangle) = f\big(\text{dec}(\langle \alpha_1, u \cdot i_1 \rangle), \ldots, \text{dec}(\langle \alpha_r, u \cdot i_r \rangle)\big).$$

The Characterization

The purpose of attribute grammars is to define attributes of the nodes of a tree, and thus, in particular, properties of these nodes. MSO formulas with one free variable can also be used to define properties of the nodes of a tree. These two methods turn out to be equivalent when we restrict the attribute grammar to be finite-valued, as will be shown in this subsection.

In order to define one particular node property with attribute grammars, we use an attribute grammar G with a designated boolean attribute ρ. Given a tree t, the attribute grammar "selects" all nodes u of t with $\text{dec}_{G,t}(\langle \rho, u \rangle) = \text{true}$.

A *node-selecting attribute grammar* over Σ is a pair (G, ρ) where G is a finite-valued attribute grammar over Σ, and ρ is a boolean attribute of G, i.e., $W(\rho) = \{\text{true}, \text{false}\}$. The node property computed by (G, ρ) is

$$P(G, \rho) = \{(t, u) \mid t \in T_\Sigma, u \in V_t \text{ and } \text{dec}_{G,t}(\langle \rho, u \rangle) = \text{true}\}.$$

$P(G, \rho)$ is called an ATT-*computable node property*, and the set of all ATT-computable node properties is denoted ATT-P. We will prove that MSO-P = ATT-P.

Lemma 17. ATT-P $\subseteq$ MSO-P.

Proof. Let (G, ρ) be a node-selecting attribute grammar over Σ, with $G = (\Sigma, S, I, \Omega, W, R)$. To show that the node property $P(G, \rho)$ is MSO definable, it suffices, by Proposition 2 and Corollary 6 (for $k = 1$), to construct a nondeterministic finite tree automaton $M = (Q, \Sigma \cup (\Sigma \times B_1), \delta, F)$ such that, for every $t \in T_\Sigma$ and $u \in V_t$, $(t, u) \in P(G, \rho)$ iff $\mathrm{mark}(t, u) \in L(M)$. In other words, M should recognize an input tree $\mathrm{mark}(t, u)$ iff $\mathrm{dec}_{G,t}(\langle \rho, u \rangle) = \mathrm{true}$. To do this, M simulates the attribute grammar G on t, by calculating a rootless decoration of each subtree of t, in a nondeterministic way. Moving up in the tree, it makes sure that all internal rules of G are satisfied. At the marked node of $\mathrm{mark}(t, u)$ it makes sure that the rootless decoration assigns the value true to attribute ρ. M accepts $\mathrm{mark}(t, u)$ if the final rootless decoration satisfies the root rules. To compute the rootless decorations, it suffices by Lemma 16 that M keeps in its finite control the values of the attributes of the current node according to the (guessed) rootless decoration of the subtree rooted at that node.

The set of states and the set of final states of M are

$$Q = \{d : S \cup I \to \cup\Omega \mid \forall \alpha \in S \cup I : d(\alpha) \in W(\alpha)\},$$
$$F = \{d \in Q \mid \forall \langle \alpha_0, 0 \rangle = f(\langle \alpha_1, 0 \rangle, \ldots, \langle \alpha_r, 0 \rangle) \in R(\mathrm{root}) : d(\alpha_0) = f(d(\alpha_1), \ldots, d(\alpha_r))\}.$$

Please note that Q is indeed finite, because G is finite-valued. The automaton M has the following transition relations. For $\sigma \in \Sigma_k$, it makes sure that all internal rules at σ are satisfied:

$$\delta_\sigma = \{((d_1, \ldots, d_k), d_0) \mid \forall \langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_r, i_r \rangle) \in R(\sigma) : d_{i_0}(\alpha_0) = f(d_{i_1}(\alpha_1), \ldots, d_{i_r}(\alpha_r))\},$$

and, for $\langle \sigma, 1 \rangle \in \Sigma_k \times B_1$, it does so too, and at the same time makes sure that the designated boolean attribute is set:

$$\delta_{\langle \sigma, 1 \rangle} = \{((d_1, \ldots, d_k), d_0) \in \delta_\sigma \mid d_0(\rho) = \mathrm{true}\}.$$

This ends the construction of M. To show its correctness, let t' be a tree over $\Sigma \cup (\Sigma \times B_1)$, and let t be the tree over Σ obtained from t' by changing every label $\langle \sigma, 1 \rangle$ into σ. It can easily be proved by bottom-up induction on v, using Lemma 16, that for every node v (of both t' and t), $d \in \mathrm{state}_{\mathcal{P}(M),t'}(v)$ iff there is a rootless decoration dec of t_v such that (1) $d(\alpha) = \mathrm{dec}(\langle \alpha, v \rangle)$ for every $\alpha \in I \cup S$, and (2) $\mathrm{dec}(\langle \rho, u \rangle) = \mathrm{true}$ for every marked node $u \in V_{t'_v}$ (i.e., for every descendant u of v with label $\langle \sigma, 1 \rangle$ in t', for some $\sigma \in \Sigma$). Now, $\mathrm{state}_{\mathcal{P}(M),t'}(\mathrm{root}(t')) \cap F \neq \emptyset$ iff for every marked node $u \in V_{t'}$, the unique decoration $\mathrm{dec}_{G,t}$ of t by G has $\mathrm{dec}_{G,t}(\langle \rho, u \rangle) = \mathrm{true}$. This shows that, for every $t \in T_\Sigma$ and $u \in V_t$, $\mathrm{mark}(t, u) \in L(M)$ iff $\mathrm{dec}_{G,t}(\langle \rho, u \rangle) = \mathrm{true}$ iff $(t, u) \in P(G, \rho)$. □

Theorem 18. MSO-P = ATT-P.

Proof. By Lemma 17 it now remains to show that MSO-P $\subseteq$ ATT-P. Let $\phi(x)$ be a formula in $\mathrm{MSOL}_1(\Sigma)$. By Corollary 6 (for $k = 1$) and Proposition 2, there is a finite tree automaton $M = (Q, \Sigma \cup (\Sigma \times B_1), \delta, F)$ such that, for every $t \in T_\Sigma$ and $u \in V_t$, $(t, u) \in P(\phi)$ iff $\mathrm{mark}(t, u) \in L(M)$.

We will construct a node-selecting attribute grammar $G = (\Sigma, S, I, \Omega, W, R)$ over Σ with designated attribute ρ, such that $P(G, \rho) = P(\phi)$. In other words, G should be constructed in such a way that for every $t \in T_\Sigma$ and $u \in V_t$, $\mathrm{dec}_{G,t}(\langle \rho, u \rangle) = \mathrm{true}$ iff $\mathrm{mark}(t, u) \in L(M)$.

The attribute grammar G has inherited attribute β, with semantic domain $\mathcal{P}(Q)$, and synthesized attributes α, μ, and ρ, with semantic domains Q, Q, and $\{\mathrm{true}, \mathrm{false}\}$, respectively. Formally we define $S = \{\alpha, \mu, \rho\}$, $I = \{\beta\}$, and $\Omega = \{Q, \mathcal{P}(Q), \{\mathrm{true}, \mathrm{false}\}\}$, and W is defined as $W(\alpha) = W(\mu) = Q$, $W(\beta) = \mathcal{P}(Q)$, and $W(\rho) = \{\mathrm{true}, \mathrm{false}\}$.

Intuitively, for a tree t over T_Σ and a node u of t, the value of α at u is $\mathrm{state}_{M,t}(u)$ and the value of β at u is $\mathrm{succ}_{M,t}(u)$, i.e., α and β simulate the behavior of the tree automaton M on the unmarked parts of $\mathrm{mark}(t, u)$. Thus, if u has label σ, then $\mathrm{mark}(t, u)$ is recognized by M if $\delta_{\langle \sigma, 1 \rangle}$ applied to the α-values of its children yields a successful value, i.e., a value in β. Attribute μ computes this value, and attribute ρ of u is true if and only if it is in β.

The set $R(\sigma)$, for $\sigma \in \Sigma_k$, consists of the internal rules

$$\begin{aligned}
&\langle \alpha, 0 \rangle = \delta_\sigma(\langle \alpha, 1 \rangle, \langle \alpha, 2 \rangle, \ldots, \langle \alpha, k \rangle) \\
&\langle \mu, 0 \rangle = \delta_{\langle \sigma, 1 \rangle}(\langle \alpha, 1 \rangle, \langle \alpha, 2 \rangle, \ldots, \langle \alpha, k \rangle) \\
&\langle \beta, i \rangle = \{q \in Q \mid \delta_\sigma(\langle \alpha, 1 \rangle, \ldots, \underset{i}{q}, \ldots, \langle \alpha, k \rangle) \in \langle \beta, 0 \rangle\} \quad \text{for } 1 \leq i \leq k \\
&\langle \rho, 0 \rangle = (\langle \mu, 0 \rangle \in \langle \beta, 0 \rangle)
\end{aligned}$$

where $\delta_\sigma(\langle \alpha, 1 \rangle, \ldots, \underset{i}{q}, \ldots, \langle \alpha, k \rangle)$ stands for $\delta_\sigma(e_1, \ldots, e_k)$ with $e_j = \langle \alpha, j \rangle$ for all $j \in [1, k] \setminus \{i\}$ and $e_i = q$.

The set $R(\mathrm{root})$ of root rules consists of one rule:

$$\langle \beta, 0 \rangle = F.$$

This ends the construction of G. Clearly, G is noncircular. In fact, it is a two-pass attribute grammar (cf. [Boc, Eng2]): in the first pass over the input tree all values of α and μ can be computed bottom-up, and in the second pass all values of β and ρ can be computed top-down.

To show the correctness of G, let t be a tree over Σ. In what follows we write 'dec' instead of '$\mathrm{dec}_{G,t}$'. Since the rules for α and β mirror the inductive definitions of $\mathrm{state}_{M,t}$ and $\mathrm{succ}_{M,t}$, respectively, it immediately follows that for every node u of t, $\mathrm{dec}(\langle \alpha, u \rangle) = \mathrm{state}_{M,t}(u)$ and $\mathrm{dec}(\langle \beta, u \rangle) = \mathrm{succ}_{M,t}(u)$. We now claim that for every node u of t, $\mathrm{dec}(\langle \mu, u \rangle) = \mathrm{state}_{M,\mathrm{mark}(t,u)}(u)$. In fact, by the semantic rule for μ, $\mathrm{dec}(\langle \mu, u \rangle) = \delta_{\langle \sigma, 1 \rangle}(\mathrm{dec}(\langle \alpha, u \cdot 1 \rangle), \ldots, \mathrm{dec}(\langle \alpha, u \cdot k \rangle))$

where $\sigma \in \Sigma_k$ is the label of u. Since for every $i \in [1,k]$, $\mathrm{dec}(\langle \alpha, u \cdot i \rangle) = \mathrm{state}_{M,t}(u \cdot i) = \mathrm{state}_{M,\mathrm{mark}(t,u)}(u \cdot i)$ by Lemma 7(1), we obtain

$$\begin{aligned}\mathrm{dec}(\langle \mu, u \rangle) &= \delta_{\langle \sigma,1 \rangle}(\mathrm{state}_{M,\mathrm{mark}(t,u)}(u \cdot 1), \ldots, \mathrm{state}_{M,\mathrm{mark}(t,u)}(u \cdot k)) \\ &= \mathrm{state}_{M,\mathrm{mark}(t,u)}(u)\end{aligned}$$

which proves the claim. Since, for every node u of t, $\mathrm{dec}(\langle \beta, u \rangle) = \mathrm{succ}_{M,t}(u) = \mathrm{succ}_{M,\mathrm{mark}(t,u)}(u)$ by Lemma 7(2), it follows from Lemma 1 that $\mathrm{dec}(\langle \rho, u \rangle) = (\mathrm{state}_{M,\mathrm{mark}(t,u)}(u) \in \mathrm{succ}_{M,\mathrm{mark}(t,u)}(u)) = (\mathrm{mark}(t,u) \in L(M))$. This shows the correctness of G. □

Attributed relabelings

In Section 5 we have characterized the MSO definable binary node relations to be those that are computed by tree-walking automata. However, these automata still make use of MSO tests, i.e., they test MSO definable node properties. The results of this section now allow us to give an "MSO-free" characterization of the MSO definable node relations. To this aim we define the notion of an "attributed relabeling": it is a relabeling of the nodes of a tree by the value of a designated attribute. Formally this is defined as follows.

Let Σ and Δ be operator alphabets (the input and output alphabet, respectively). A *relabeling attribute grammar* from Σ to Δ is a pair (G, ρ) where G is a finite-valued attribute grammar over Σ, and ρ is an attribute of G with semantic domain Δ (i.e., $W(\rho) = \Delta$), such that for every $t \in T_\Sigma$ and $u \in V_t$, $\mathrm{dec}_{G,t}(\langle \rho, u \rangle)$ has the same rank as $\mathrm{lab}_t(u)$. The relabeling computed by (G, ρ) is $r(G,\rho) = \{(t, t') \in T_\Sigma \times T_\Delta \mid V_{t'} = V_t, E_{t'} = E_t, \text{ and } \mathrm{lab}_{t'}(u) = \mathrm{dec}_{G,t}(\langle \rho, u \rangle) \text{ for all } u \in V_t\}$. $r(G,\rho)$ is called an *attributed relabeling*. Note that it is a total function $T_\Sigma \to T_\Delta$. The set of all attributed relabelings is denoted ATT-REL.

If h is an attributed relabeling from Σ to Δ, and R is a binary node relation over Δ, then $h^{-1}(R)$ is the binary node relation over Σ defined by $h^{-1}(R) = \{(t,u,v) \mid (h(t),u,v) \in R\}$. Thus, $h^{-1}(R)$ defines the same node relation on a tree $t \in T_\Sigma$ as R does on the relabeled tree $h(t) \in T_\Delta$.

By BREG-R we denote the set of node relations that are computed by basic tree-walking automata, and by ATT-REL^{-1}(BREG-R) we denote the set of all node relations $h^{-1}(R)$ with $h \in$ ATT-REL and $R \in$ BREG-R. Intuitively, a node relation R' in ATT-REL^{-1}(BREG-R) is computed in two stages: for a tree t, an attribute grammar is used to relabel the nodes of t and then a basic tree-walking automaton is used to walk from u to v on the relabeled tree, for every $(t,u,v) \in R'$. We now show that ATT-REL^{-1}(BREG-R) is exactly the class of MSO-definable node relations.

Theorem 19. MSO-R = ATT-REL^{-1}(BREG-R).

Proof. By Theorem 13 it suffices to show REG-R = ATT-REL^{-1}(BREG-R). We first prove ATT-REL^{-1}(BREG-R) $\subseteq$ REG-R. Let $h = r(G,\rho)$ be an attributed relabeling from Σ to Δ, and let A be a basic tree-walking automaton over Δ.

We claim that for every $\delta \in \Delta$ there is a formula $\phi_\delta(x)$ in $\mathrm{MSOL}_1(\Sigma)$ such that for every $t \in T_\Sigma$ and $u \in V_t$, $(t,u) \models \phi_\delta(x)$ iff $\mathrm{lab}_{h(t)}(u) = \delta$. In fact, let (G', ρ') be the node-selecting attribute grammar that is obtained from G by adding a synthesized boolean attribute ρ' which has, for every $\sigma \in \Sigma$, the internal rule $\langle \rho', 0\rangle = (\langle \rho, 0\rangle = \delta)$. Clearly, $P(G', \rho') = \{(t,u) \mid \mathrm{lab}_{h(t)}(u) = \delta\}$ and hence, by Theorem 18, this node property is MSO-definable, i.e., there is a formula $\phi_\delta(x)$ that satifies the above requirement. Now define A' to be the tree-walking automaton (with MSO tests) over Σ that is obtained from A by changing every transition $(q, \mathrm{lab}_\delta(x), q')$ into $(q, \phi_\delta(x), q')$. It should be clear that $R(A') = h^{-1}(R(A))$.

It remains to show that REG-R $\subseteq$ ATT-REL^{-1}(BREG-R). Let A be a tree-walking automaton (with MSO tests) over Σ, and let $\phi_1(x), \ldots, \phi_n(x)$ be the unary MSO formulas that appear in the transitions of A. By Theorem 18 there are node-selecting attribute grammars (G_i, ρ_i), for $i \in [1,n]$, such that $P(G_i, \rho_i) = P(\phi_i)$. Define the operator alphabet $\Delta = \Sigma \times \{\mathrm{true}, \mathrm{false}\}^n$ with $\mathrm{rk}(\sigma, b_1, \ldots, b_n) = \mathrm{rk}(\sigma)$ for all $(\sigma, b_1, \ldots, b_n) \in \Delta$. We now construct the relabeling attribute grammar (G, ρ) by taking the (disjoint) union of all G_i, in the obvious way, and adding the synthesized attribute ρ with semantic domain Δ which has, for $\sigma \in \Sigma$, the internal rule $\langle \rho, 0\rangle = (\sigma, \langle \rho_1, 0\rangle, \ldots, \langle \rho_n, 0\rangle)$. We define A' to be the basic tree-walking automaton over Δ that is obtained from A by changing every transition $(q, \phi_i(x), q')$ into all transitions $(q, \mathrm{lab}_{(\sigma, b_1, \ldots, b_n)}(x), q')$ for all $\sigma \in \Sigma$ and all $b_1, \ldots, b_n \in \{\mathrm{true}, \mathrm{false}\}$ with $b_i = \mathrm{true}$. It should be clear that $h^{-1}(R(A')) = R(A)$. Note that A' does not even need the tests $\mathrm{root}(x)$ and $\exists y(\mathrm{edg}_i(y,x))$. Thus, this theorem also holds for the tree-walking automata of [AhoUll, EngRozSlu, KamSlu], cf. the end of Section 5. □

By a slight adaptation of the proof, it can be shown that the construction of the tree-walking automata preserves determinism. Hence, in a formula, fMSO-R = ATT-REL^{-1}(dBREG-R), cf. Theorem 14 and the remark following it.

7 Time Complexity

Let Σ be an operator alphabet, and let $\phi(x_1, \ldots, x_k) \in \mathrm{MSOL}_k(\Sigma)$ be a fixed MSO formula with $k \geq 0$ free node variables. In this section we investigate the time complexity of computing the node relation $R_t(\phi)$, for given $t \in T_\Sigma$.

It is well known that it can be checked in linear time, for a tree t and nodes $u_1, \ldots, u_k$ of t, whether or not $(u_1, \ldots, u_k) \in R_t(\phi)$. In fact, by Corollary 6 and Proposition 2 there is a (deterministic) finite tree automaton M such that $(t, u_1, \ldots, u_k) \models \phi(x_1, \ldots, x_k)$ iff $\mathrm{mark}(t, u_1, \ldots, u_k) \in L(M)$. Converting t into $\mathrm{mark}(t, u_1, \ldots, u_k)$ and running M on $\mathrm{mark}(t, u_1, \ldots, u_k)$ takes linear time.

Here we wish to consider the complexity of computing $R_t(\phi)$, for a given tree t; in other words, on input t, we want to compute all node sequences that satisfy the formula ϕ. A naive way of doing this is to use the above linear time algorithm for all possible node sequences $(u_1, \ldots, u_k)$. That takes time $O(n^{k+1})$, where n is the size of the tree t. But we can do better than that.

Theorem 20. *Let Σ be an operator alphabet and let $\phi(x_1, \ldots, x_k)$ be a fixed* MSO *formula in* $\mathrm{MSOL}_k(\Sigma)$ *with $k \geq 1$. For $t \in T_\Sigma$, $R_t(\phi)$ can be computed in time $O(n^k)$, where n is the size of t.*

Proof. First we consider the problem for formulas with one free node variable, i.e., $k = 1$. With the help of Theorem 18 we can transform $\phi(x_1)$ to a node-selecting attribute grammar (G, ρ), with $R(\phi) = P(\phi) = P(G, \rho)$, or, in other words, $(t, u) \models \phi(x_1)$ iff $\mathrm{dec}_{G,t}(\langle \rho, u \rangle) = \mathrm{true}$. It is well known that attribute evaluation takes linear time, counting the computation of a semantic rule as one unit of time (see, e.g., [DerJouLor, Eng2]). Since G is finite-valued, the computation of a semantic rule takes constant time, and so it takes linear time to find all u such that $\mathrm{dec}_{G,t}(\langle \rho, u \rangle) = \mathrm{true}$.

Now suppose $k > 1$. Using Lemma 5 (with $j = k - 1$), $\phi(x_1, \ldots, x_k)$ can be transformed into an MSO formula $\psi(x)$ over $\Sigma \cup (\Sigma \times B_{k-1})$ with one free node variable, such that

$$(t, u_1, \ldots, u_k) \models \phi(x_1, \ldots, x_k) \text{ iff } (\mathrm{mark}(t, u_1, \ldots, u_{k-1}), u_k) \models \psi(x).$$

For any $u_1, \ldots, u_{k-1} \in V_t$, we can construct $\mathrm{mark}(t, u_1, \ldots, u_{k-1})$ and then find all u_k such that $(\mathrm{mark}(t, u_1, \ldots, u_{k-1}), u_k) \models \psi(x)$ in $O(n)$ time, by using an attribute grammar, as shown above. There are $O(n^{k-1})$ possible combinations for $u_1, \ldots, u_{k-1}$, so the time needed to find all $u_1, \ldots, u_k$ such that $(\mathrm{mark}(t, u_1, \ldots, u_{k-1}), u_k) \models \psi(x)$ is $O(n^k)$. □

For binary formulas ϕ an important special case is that ϕ is functional, i.e., specifies a partial function on the nodes of every tree (see Theorem 14 where we have shown that such MSO formulas are computed by deterministic tree-walking automata). By Theorem 20, $R_t(\phi)$ can be computed in quadratic time. We now show that it can in fact be computed in linear time. The proof is a straightforward generalization of the one in [KlaSch] (Appendix A3), where this result is shown for functional routing languages, i.e., for *basic* deterministic tree-walking automata.

Theorem 21. *Let Σ be an operator alphabet and let $\phi(x, y)$ be a fixed functional* MSO *formula in* $\mathrm{MSOL}_2(\Sigma)$. *For $t \in T_\Sigma$, $R_t(\phi)$ can be computed in linear time.*

Proof. This follows immediately from Theorem 19 and the result of [KlaSch] mentioned above, using the fact that attribute evaluation takes linear time as in the proof of Theorem 20. However, for completeness sake, we give a direct proof that uses Theorem 20 instead of Theorem 19, and is just a slight generalization of the proof in [KlaSch].

Let t be a tree over Σ. To be able to compute $R_t(\phi)$ in linear time, we first transform $\phi(x, y)$ into an equivalent deterministic tree-walking automaton $A = (Q, \Delta, \delta, \{q_0\}, F)$ over Σ, as was shown in Theorems 13 and 14. Thus, for every tree $t \in T_\Sigma$ and nodes u and v of t, $(u, v) \in R_t(\phi)$ iff $(u, v) \in R_t(A)$ iff $(q_0, u) \twoheadrightarrow^*_{A,t} (q_f, v)$, for some $q_f \in F$.

The algorithm that computes $R_t(\phi)$ is in two stages. In the first stage it computes and stores the sets $\{u \mid (t,u) \models \psi(x)\}$, for every formula $\psi(x) \in \mathrm{MSOL}_1(\Sigma)$ that occurs in Δ. By Theorem 20 this takes linear time.

The second stage of the algorithm, shown below, is taken over (almost) literally from [KlaSch]. In the second stage a table T is computed that is indexed with the configurations of the automaton A, walking on t. For every configuration $(q,u) \in Q \times V_t$, $T[q,u]$ contains the unique node v with $(q,u) \twoheadrightarrow^*_{A,t} (q_f, v)$, for some $q_f \in F$, if such a node exists. If it does not, $T[q,u] = \textsc{nil}$. Then $R_t(\phi) = \{(u,v) \in V_t \times V_t \mid T[q_0,u] = v\}$. The algorithm employs a queue L of configurations that is initially empty.

```
type
    Config = Q × V_t;
var
    T : array [Config] of V_t ∪ {NIL};
    (q,u), (q',u') : Config;
    L : queue of Config;
begin
    Init(L);
    for all (q,u) ∈ Q × V_t do
        T[q,u] := NIL
    od;
    for all (q,u) ∈ F × V_t do
        T[q,u] := u;
        L ⇐ (q,u)
    od;
    while ¬ IsEmpty(L) do
        (q',u') ⇐ L;
        for all (q,u) ∈ Q × V_t with T[q,u] = NIL and (q,u) ↠_{A,t} (q',u') do
            T[q,u] := T[q',u'];
            L ⇐ (q,u)
        od
    od
end
```

In this algorithm, Init initializes the queue, $L \Leftarrow (q,u)$ adds (q,u) to the end of queue L, $(q',u') \Leftarrow L$ removes the first element from the queue and assigns it to (q',u'), and IsEmpty(L) returns true if L is empty.

The correctness of the algorithm should be clear: in the second for-loop, the table is filled in correctly (with respect to the intended contents of the table) for all final configurations. From there on, the automaton is followed back on its walk, and every possible previous configuration is filled in correctly. Every configuration (q,u) that has a walk leading to a final configuration (q_f, v) is eventually considered, as can easily be shown by induction on the length of the walk $(q,u) \twoheadrightarrow^*_{A,t} (q_f, v)$.

The algorithm runs in time $O(n)$, where n is the size of the tree or, equivalently, the number of configurations. In fact, since each configuration is placed

in the queue at most once, the last for-loop (within the while-statement) is executed $O(n)$ times. Thus, it remains to show that each execution of that for-loop takes constant time. Let (q', u') be the configuration considered in such an execution. Because the automaton A is fixed, there is a fixed number of transitions $(q, d, q') \in \delta$. For each such transition, there is at most one u such that $(q, u) \overset{d}{\twoheadrightarrow}_{A,t} (q', u')$. It should be noted that every such u is either u' itself or one of its immediate neighbors in t, and that it can be computed in constant time. For the directives $d = \uparrow_i$, u exists iff $i \in [1, \mathrm{rk}(\mathrm{lab}_t(u'))]$, and $u = u' \cdot i$. For the directives $d = \downarrow_i$, u exists iff u' has a parent v with $u' = v \cdot i$, in which case $u = v$. Finally, for the directives $d = \psi(x)$, u exists iff $(t, u') \models \psi(x)$, in which case $u = u'$. Note that the latter test takes constant time because the set $\{u' \mid (t, u') \models \psi(x)\}$ has been precomputed in the first stage of the algorithm. Thus, only a constant number of configurations (q, u) have to be considered and they can be computed in constant time. □

Theorem 20 states that we can find $R_t(\phi)$ in $O(n^k)$ time if ϕ has k free node variables. Using Theorem 21 we can speed up the calculation of $R_t(\phi)$ for formulas with more than one free node variable, if one of the variables depends on (some of) the others.

First we define dependency. We speak of a dependency in a relation when the value of one of the elements of a tuple in the relation is fully determined by the value of some of the others. Formally, let R be a k-ary relation, $i \in [1, k]$, and $D \subseteq [1, k]$. We say that i (*functionally*) *depends on* D (*in* R) if for all $(a_1, \ldots, a_k), (a'_1, \ldots, a'_k) \in R$ with $a_d = a'_d$ for all $d \in D$, a_i is equal to a'_i.

Theorem 22. *Let Σ be an operator alphabet and let $\phi(x_1, \ldots, x_k)$ be a fixed* MSO *formula in* $\mathrm{MSOL}_k(\Sigma)$ *with $k \geq 2$. Let there be an $i \in [1, k]$, and a $D \subseteq [1, k]$ with $i \notin D$, such that i depends on D in $R_t(\phi)$ for every $t \in T_\Sigma$. Then, for $t \in T_\Sigma$, $R_t(\phi)$ can be computed in time $O(n^{k-1})$.*

Proof. The case $k = 2$ is proven in Theorem 21: for $i = 2$, $R_t(\phi)$ is functional, and for $i = 1$ the inverse of $R_t(\phi)$ is functional.

For the other cases, without loss of generality, we can assume $i = k$ and $D = [1, k-1]$. What we will do is the following. For every possibility for the first $k-2$ arguments, we build a tree with $k-2$ marks at the appropriate places, and we have a formula checking the last two arguments, in which the last argument depends on the previous one.

Using Lemma 5 (with $j = k - 2$) we transform $\phi(x_1, \ldots, x_k)$ into an MSO formula $\psi(x, y)$ over $\Sigma \cup (\Sigma \times B_{k-2})$ with two free node variables, such that

$$(t, u_1, \ldots, u_k) \models \phi(x_1, \ldots, x_k) \Leftrightarrow (\mathrm{mark}(t, u_1, \ldots, u_{k-2}), u_{k-1}, u_k) \models \psi(x, y).$$

Moreover, without loss of generality, we may assume that $(t', u_{k-1}, u_k) \models \psi(x, y)$ does not hold whenever t' is *not* of the form $\mathrm{mark}(t, u_1, \ldots, u_{k-2})$. In fact, it should be clear that $\{\mathrm{mark}(t, u_1, \ldots, u_{k-2}) \mid t \in T_\Sigma, u_1, \ldots, u_{k-2} \in V_t\}$ is a regular tree language, and hence can be expressed with a closed MSO formula by Proposition 2. For all tuples $(u_1, \ldots, u_{k-1}, u_k)$ and $(u_1, \ldots, u_{k-1}, u'_k)$ in the

relation $R_t(\phi)$, by the given dependency of k on $[1, k-1]$, u_k is equal to u'_k. Hence, if

$$(\mathrm{mark}(t, u_1, \ldots, u_{k-2}), u_{k-1}, u_k) \models \psi(x, y),$$

and

$$(\mathrm{mark}(t, u_1, \ldots, u_{k-2}), u_{k-1}, u'_k) \models \psi(x, y),$$

then $u_k = u'_k$. So, $R_{\mathrm{mark}(t,u_1,\ldots,u_{k-2})}(\psi)$ is functional. Hence, by the above assumption, $\psi(x, y)$ is functional.

For any sequence of nodes $u_1, \ldots, u_{k-2} \in V_t$, we can find the corresponding marked tree $\mathrm{mark}(t, u_1, \ldots, u_{k-2})$ in $O(n)$ time. Because $\psi(x, y)$ is functional, we can then find all pairs (u_{k-1}, u_k) such that $(\mathrm{mark}(t, u_1, \ldots, u_{k-2}), u_{k-1}, u_k) \models \psi(x, y)$ in $O(n)$ time, as shown in Theorem 21. There are $O(n^{k-2})$ possible combinations for $u_1, \ldots, u_{k-2}$, so the time needed to find all $u_1, \ldots, u_k$ such that $(\mathrm{mark}(t, u_1, \ldots, u_{k-2}), u_{k-1}, u_k) \models \psi(x, y)$ is $O(n^{k-1})$. □

The assumption in Theorem 22 is decidable, i.e., it is decidable for a formula $\phi(x_1, \ldots, x_k) \in \mathrm{MSOL}_k(\Sigma)$, $i \in [1, k]$, and $D \subseteq [1, k]$, whether or not i functionally depends on D in $R_t(\phi)$ for all $t \in T_\Sigma$. In fact, this holds if and only if $L(\neg\, \mathrm{func}_{\phi,i,D}) = \emptyset$, for the closed formula

$$\mathrm{func}_{\phi,i,D} = \forall x_1, \ldots, x_k, y_1, \ldots, y_k \\ (\phi(x_1, \ldots, x_k) \wedge \phi(y_1, \ldots, y_k) \wedge \forall d \in D(x_d = y_d)) \rightarrow x_i = y_i,$$

which is decidable by Proposition 3. In particular it is decidable whether or not an MSO formula is functional, cf. [KlaSch] (Appendix A4).

We finally note that it follows from Theorem 20 that every MSO definable graph transduction of which the input graph is a tree (cf. the Introduction) can be computed in quadratic time. Moreover, if all outgoing edges of each node of the output graph have distinct labels (as, e.g., for term graphs), then, by Theorem 21, it can be computed in linear time. In particular, MSO definable tree transductions can be computed in linear time. As mentioned in the Introduction, it is shown in [Blo, BloEng2] that they can in fact be computed by two-stage attribute grammars.

References

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

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

[Blo] R. Bloem; *Attribute Grammars and Monadic Second Order Logic*, Master's Thesis, Leiden University, June 1996

[BloEng1] R. Bloem, J. Engelfriet; Monadic second order logic and node relations on graphs and trees, to appear in Lecture Notes in Computer Science, 1997

[BloEng2] R. Bloem, J. Engelfriet; A comparison of tree transductions defined by monadic second order logic and by attribute grammars, in preparation

[Boc] G. V. Bochmann; Semantic evaluation from left to right, Comm. of the ACM 19 (1976), 55–62

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

[Cou1] B. Courcelle; Graph rewriting: an algebraic and logic approach, in *Handbook of Theoretical Computer Science*, Vol. B (J. van Leeuwen, ed.), Elsevier, 1990, 193–242

[Cou2] B. Courcelle; The monadic second-order logic of graphs V: On closing the gap between definability and recognizability, Theor. Comput. Sci. 80 (1991), 153–202

[Cou3] B. Courcelle; Monadic second-order definable graph transductions: a survey, Theor. Comput. Sci. 126 (1994), 53–75

[Cou4] B. Courcelle; The expression of graph properties and graph transformations in monadic second-order logic, Chapter 5 of *Handbook of Graph Grammars and Computing by Graph Transformation*, Vol. 1: *Foundations* (G. Rozenberg, ed.), World Scientific, 1997

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

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

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

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

[Eng1] J. Engelfriet; *Simple Program Schemes and Formal Languages*, Lecture Notes in Computer Science 20, Springer-Verlag, Berlin, 1974

[Eng2] J. Engelfriet; Attribute grammars: attribute evaluation methods, in *Methods and Tools for Compiler Construction* (ed. B. Lorho), Cambridge University Press, 1984, 103–138

[Eng3] J. Engelfriet; A characterization of context-free NCE graph languages by monadic second-order logic on trees, in *Graph Grammars and their Application to Computer Science* (H. Ehrig, H.-J. Kreowski, G. Rozenberg, eds.), Lecture Notes in Computer Science 532, Springer-Verlag, Berlin, 1991, 311–327

[Eng4] J. Engelfriet; A regular characterization of graph languages definable in monadic second-order logic, Theor. Comput. Sci. 88 (1991), 139–150.

[Eng5] J. Engelfriet; Context-free graph grammars, Chapter 3 of *Handbook of Formal Languages*, Vol. 3: *Beyond Words* (G. Rozenberg, A. Salomaa, eds.), Springer-Verlag, 1997

[EngOos1] J. Engelfriet, V. van Oostrom; Regular description of context-free graph languages, J. of Comp. Syst. Sci. 53 (1996), 556–574

[EngOos2] J. Engelfriet, V. van Oostrom; Logical description of context-free graph languages, Tech. Report 96–22, Leiden University, August 1996, to appear in J. of Comp. Syst. Sci.

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

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

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

[HopUll] J. E. Hopcroft, J. D. Ullman; *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, Mass., 1979

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

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

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

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

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

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

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