WPE II Report

Computer & Information Science
http://www.cis.upenn.edu

Actions Louder than Words? MSO-definable Transductions

J. Nathan Foster

18 November 2006

Contents

Words and deeds are quite indifferent modes of the divine energy. Words are also actions, and actions are a kind of words.

—Ralph Waldo Emerson, *The Poet*

1 Introduction

A wormhole is probably not the first image that comes to mind when someone mentions formal language theory. But it is an apt metaphor for the deep connection between logic and machines that was independently discovered by Büchi, Elgot, and Trakhtenbrot [BÖ60, Elg61, Tra62] in the early 1960s. Like a wormhole, their result connects two apparently distinct points in the space of formal languages—the languages characterized by formulas in monadic second-order logic (MSO) and those recognized by finite-state automata. Unlike physical wormholes, however, the connection is not merely a theoretical conjecture or the stuff of science fiction—the translations between a formula and automaton can both be effectively computed. Therefore, it is therefore possible to pass back and forth between the logical description of a language and its automata representation.

Since their results were first published, Büchi, Elgot, and Trakhtenbrot's idea has been extended in several directions. One line of work established the corresponding connections between MSO formulas and automata operating on richer structures including: ranked trees [TW68, Don70], graphs [Cou90], texts [HtP97], infinite strings [BÖ62], and infinite trees [Rab69]. The comprehensive handbook article by Thomas [Tho97] gives an excellent survey of these results. Another body of work has investigated the connections between so-called *MSO graph transductions* (MSO-GTs) and *finite-state transducers*—automata with output. The idea of using MSO formulas to transform graphs was originally developed by Engelfriet and van Oostrom [EvO97] and later generalized by Courcelle [Cou91, Cou94]. This paper surveys the results in the latter area and techniques used to establish those results, focusing on the cases where the transducers are deterministic and the structures are strings [EH01], trees [BE00, EM99, EM03b] and infinite trees [CL04].

A finite-state transducer and an MSO-GT transform a structure in fundamentally different ways. The operation of a transducer is like that of an automaton: it traverses the input making transitions from state to state and recognizing local bits of the input structure while incrementally building up the output. An MSO-GT transforms a graph by interpreting formulas that specify the structure of the output graph over the input graph. More specifically, an MSO-GT is specified by an integer k, and finite sets of unary and binary formulas; it transforms the graph in three steps. First it creates k copies of the vertices of the input graph. It then selects a subset of the these vertices by retaining the ones that satisfy a unary vertex formula and discarding the rest. Finally, it adds edges between every pair of vertices that satisfy a binary edge formula. Notice that whereas a transducer operates locally, the formulas in an MSO-GT are interpreted the entire graph.

As an example, let Σ be an finite alphabet and consider the string transduction where the input is concatenated with its reversal: $\{(w, ww^r) \mid w \in \Sigma^*\}$. It is simple to construct a two-way finite-state string transducer realizing this transduction: it makes two passes over the input string, one forwards and one backwards, and copies the symbols from input to output on both passes. An equivalent

MSO-GT can be obtained by taking two copies of the vertices and adding edges between adjacent vertices in each copy. In the first copy the edges go in the same direction as in the input, and in the second copy they go in the opposite direction. One final edge is needed to connect the last vertex in the first copy to the last vertex of the second copy—i.e., to connect the string with its reversal. See Section 3 for precise definitions of both transductions.

The benefits of relating a finite-state transducer formalism with one based on logic are significant. First, because the two transducer formalisms are so different, transformations that are difficult to specify in one are sometimes easy to describe in the other. In these situations, having "more than one way to skin a cat" is quite useful. The syntax of transducers makes them especially well-suited to expressing transformations whose behavior is obtained by iterating a local transformation. Conversely, transformations where data is shuffled around the structure are often simpler to specify as an MSO-GT because it is as easy to specify an edge between far-apart vertices as it is to specify one between vertices in the same neighborhood. As we will see, one of the main technical challenges in showing that every MSO-GT can be simulated by a finite-state transducer is overcoming this gap between the global and local modes of operation.

Second, because MSO-GTs have many desirable properties—they are closed under regular look-ahead and serial composition, and can be computed in linear time—a proof that finite-state transducer is equivalent to MSO-GTs gives good evidence that the transducer formalism is natural, and that it balances the trade-offs between expressiveness and tractability. In the case of trees, one of the standard transducer formalisms—macro tree transducers—is not closed under composition. By the main result described in Section 4, which equates MSO-GTs and MTTs that only increase the size their input linearly, we obtain a subclass that is closed under composition. Hence, in this example, pursuing the connection to MSO resulted in a class of tree transductions that has some better properties than the formalism we started with.

Third, every finite-state transducer directly corresponds to an implementation, while an MSO-GT is essentially a declarative specification. Connecting the two relates every specification to an implementation and vice versa. Thus, studying MSO-GTs and transducers has applications to the areas of program synthesis, since one can automatically generate an implementation from a high-level specification, and to automatic verification, since one can extract a specification directly from an implementation.

Of course, the correspondences described in this survey each depend on precise arguments that are highly-tailored to the specific structures and transducer formalisms in each case. Nevertheless, some common themes emerge from these studies and are worth highlighting before we dive into the details.

Showing that a finite-state transducer can be simulated by an MSO-GT is usually fairly simple. The general idea is to let the copy number k be equal to the number of states and to construct edges that simulate to the transitions of the transducer. The output structure can be read off from this graph by identifying a path from an initial state to a final state. However, this construction depends on the size of the output structure being bounded by the product of the number of states and the size of the input structure. For strings this condition holds, but on trees a restriction is needed to ensure that the finite-state transducer does not perform unbounded copying.

The other direction—from MSO-GT to finite-state transducer—is typically more difficult, because of the differences highlighted above between a locally-operating finite-state transducer and a globally-

operating MSO-GT. There are several approaches one can take to bridge this gap. First, one can try to show that although the finite-state formalism is described using local operations, it can use its state to examine wide enough regions of the structure. This is used in the case of strings where, because finite-state transducers are closed under composition and are two-way, one can show that they are also closed under regular look-ahead. Alternatively, one can try to make the connection in the other direction and demonstrate that every formula appearing in an MSO graph transduction can normalized so that it only expresses only local properties. This technique is put to use in trees where every binary formula can be translated to a shortest-path walk between the vertices that satisfy it. A completely different idea is to introduce conditions so that MSO-GTs to also operate locally. This approach is used with infinite trees, where the condition of bisimulation-preserving is added to MSO-GTs as a way to control the shape of transductions.

The rest of this survey is organized as follows. Section 2 establishes some notation for graphs, and reviews the syntax and semantics of MSO and of MSO-GTs. The main body of the paper, given in Sections 3, 4, and 5 describes the main technical results surveyed here, which each connect logic and transducers over a particular class of structures. Over strings, I describe the result of Engelfriet and Hoogeboom [EH01] that demonstrates the equivalence of MSO-GTs and two-way deterministic finite-state string transducers. Over ranked trees, I describe a series of results by Engelfriet and his co-authors [BE97, BE00] whose culmination is a proof that MSO-GTs and macro tree transducers of linear size increase are equivalent [EM99]. Over infinite trees, generated as the unfoldings of finite term graphs, I describe a recent result due to Thomas Colcombet and Christof Löding [CL04] that deterministic top-down tree transducers are equivalent to bisimulation-preserving MSO-GTs. Section 6 describes an extension of MSO-GTs with non-determinism and the corresponding extension to strings: 2-ary compositions of two-way non-deterministic string transducer are equivalent to non-deterministic MSO-GTs iff there is a fixed bound on the number of times that the transducer visits each character of the input. Finally, Section 7 suggests some topics for future study.

2 Preliminaries

Each of the structures studied in this survey—strings, trees, and infinite trees—can be represented as graphs and described by MSO formulas over an appropriate vocabulary. In this section, we define notation for graphs, review the syntax and semantics of MSO, and show how MSO formulas can be used to define graph transductions.

2.1 Graphs

Let Σ and Δ be finite alphabets. A graph over Σ and Δ is a triple $G = \langle V_G, E_G, \mathit{lab}_G \rangle$ where V_G is the set of vertices, $E_G \subseteq V_G \times \Delta \times V_G$ is the set of edges, and $\mathit{lab}_G : V_G \to \Sigma$ is the labeling function that assigns a label in Σ to every vertex. Note that both vertices and edges are labeled, with symbols drawn from Σ and Δ respectively, that edges are directed, and that graphs may be infinite although the sets of labels are finite.

2.2 Monadic Second-Order Logic

I will assume familiarity with the basic concepts associated with first-order logic (FO) including vocabularies, structures, terms, formulas, valuations, satisfaction, etc.; standard definitions can be found in Enderton's textbook [End77].

MSO is the extension of first-order logic where variables may range over unary (i.e., monadic) first-order formulas as well as terms. Every unary formula is interpreted as a predicate that picks out the elements of the universe that satisfy it. Accordingly, every second-order variable in MSO corresponds to a set. Following this interpretation, I will call second-order variables *set variables* and, as usual, distinguish them typographically by using upper-case letters for set variables and lower-case letters for term variables. Also, I will write $x \in X$ instead of the formula $X(x)$.

To describe graphs, a simple relational vocabulary γ suffices. It contains two sorts of symbols: a unary symbol node_σ for each $\sigma \in \Sigma$ and a binary symbol edge_δ for each $\delta \in \Delta$ respectively. The syntax of MSO formulas is as follows:

$$\phi, \psi ::= x = y \mid x \in X \mid \mathit{node}_\sigma(x) \mid \mathit{edge}_\delta(x, y) \mid \forall x.\, \phi \mid \forall X.\, \phi \mid \phi \wedge \psi \mid \neg\phi$$

The existential quantifiers $\exists x.\, \phi$, $\exists! x.\phi$, and $\exists X.\, \phi$ and the boolean connectives $(\phi \vee \psi)$ and $(\phi \Rightarrow \psi)$ can all be expressed as derived forms. To save space, I will also write $\mathit{edge}(x, y)$ as an abbreviation for $\bigvee_{\delta \in \Delta} \mathit{edge}_\delta(x, y)$.

The satisfaction relation relates γ-structures, valuations, and formulas. Recall that a structure comprises a universe of objects, and relations of appropriate arity for every symbol in the vocabulary. To each graph G, we associate the structure $\mathcal{G}_G$ whose universe is the set of vertices V_G, and whose interpretations of the relational symbols are:

$$\mathit{node}_\sigma^{\mathcal{G}} \triangleq \{v \in V_G \mid \mathit{lab}_G(v) = \sigma\} \qquad \mathit{edge}_\delta^{\mathcal{G}} \triangleq \{(u, v) \in V_G \times V_g \mid (u, \delta, v) \in E_G\}$$

A valuation of variables is a finite map from variables to elements of the universe or sets of such elements. I will write $\emptyset$ for the empty valuation and $\rho[x \mapsto v]$ for the extension of ρ that maps x to v and every other variable to whatever it is mapped to by ρ. I take application to be strict—i.e., $\rho(x) = \rho(y)$ holds iff $\rho(x)$ and $\rho(y)$ are both defined and map to the same element of the universe. With these pieces in place, the satisfaction relation can be defined as follows:

$$\begin{array}{ll}
\mathcal{G}, \rho \models x = y & \text{if } \rho(x) = \phi(y); \\
\mathcal{G}, \rho \models \mathit{node}_\sigma(x) & \text{if } \rho(x) \in \mathit{node}_\sigma^{\mathcal{G}}; \\
\mathcal{G}, \rho \models \mathit{edge}_\delta(x, y) & \text{if } (\rho(x), \rho(y)) \in \mathit{edge}_\delta^{\mathcal{G}}; \\
\mathcal{G}, \rho \models x \in X & \text{if } \rho(x) \in \rho(X); \\
\mathcal{G}, \rho \models \forall x.\, \phi & \text{if } \mathcal{G}, \rho[x \mapsto v] \models \phi \text{ for every } v \in V_{\mathcal{G}}; \\
\mathcal{G}, \rho \models \forall X.\, \phi & \text{if } \mathcal{G}, \rho[X \mapsto S] \models \phi \text{ for every } S \in \mathcal{P}(V); \\
\mathcal{G}, \rho \models \phi \wedge \psi & \text{if } \mathcal{G}, \rho \models \phi \text{ and } \mathcal{G}, \rho \models \psi; \\
\mathcal{G}, \rho \models \neg\phi & \text{if } \mathcal{G}, \rho \not\models \phi.
\end{array}$$

If an order of the free variables is specified I will leave the valuation implicit and write $\mathcal{G}, u, v \models \phi(x, y)$ instead of $\mathcal{G}, \emptyset[x \mapsto u, y \mapsto v] \models \phi$. I will also omit $\mathcal{G}$ when it is clear from the context.

As examples, we can write an MSO formulas that are satisfied by (1) sets of vertices closed under the edge relation; (2) pairs of connected vertices; and (3) connected graphs as follows:

$$\begin{array}{rcl} \mathit{closed}(X) & \triangleq & \forall x.\, \forall y.\, (x \in X \wedge \mathit{edge}(x,y)) \Rightarrow y \in X \\ \mathit{path}(x,y) & \triangleq & \forall X.\, (\mathit{closed}(X) \wedge x \in X) \Rightarrow y \in X \\ \mathit{connected} & \triangleq & \forall x.\, \forall y.\, \mathit{path}(x,y) \end{array}$$

2.3 MSO-definable Graph Transductions

An MSO-GT is a quadruple $M = (k, \phi_{\mathit{dom}}, \Phi, \Psi)$ where k is an integer called the copy index of the transduction, ϕ_{dom} is a closed MSO formula, Φ is a finite set of $k|\Sigma|$ MSO unary formulas indexed by $i \in [1,k]$ and $\sigma \in \Sigma$, and Ψ is a finite set of $k^2|\Delta|$ unary formulas indexed by $(i,j) \in [1,k] \times [1,k]$ and $\delta \in \Delta$. I will identify particular elements of Φ and Ψ by index, writing $\phi^i_\sigma(x)$ and $\psi^{i,j}_\delta(x,y)$. When the vertex or edge alphabets are trivial, I will omit the subscript and write $\phi^i(x)$ and $\psi^{i,j}(x,y)$. Occasionally we will use a finite set of symbols instead of the naturals $[1,k]$.

For a given input graph $G = \langle V_G, E_G, \mathit{lab}_G \rangle$ the transduction specified by M is evaluated by interpreting the formulas in M over G in the following way. The formula ϕ_{dom} specifies the domain of the transduction; if $\not\models \phi_{\mathit{dom}}$ then $M(G)$ is undefined. Otherwise, the vertices of $M(G)$ are all the elements $(v,i) \in (V_G \times [1,k])$ such that there exists a unique $\sigma \in \Sigma$ with $\models \phi^i_\delta(x)$. Similarly, the edges are all the elements $((u,i),\delta,(v,i)) \in (V_{M(G)} \times \Delta \times V_{M(G)})$ such that $u, v \models \psi^{i,j}_\delta(x,y)$. The labeling function is implied by the definition of $V_{M(G)}$: $\mathit{lab}_{M(G)}(v,i) = \sigma$ where $\sigma \in \Sigma$ is the unique $\sigma \in \Sigma$ such that $v \models \phi^i_\sigma(x)$.

Theorem 2.1 ([Cou94])**.** *MSO-GTs are closed under composition.*

The proof of this fact is straightforward: take the copy index to be the product of the copy indices of the given transductions, and rewrite the formulas of the second transduction over the original structure using the formulas in the first transduction.

3 Strings

The first technical result described in this survey connects the class of transductions computed by two-way deterministic finite-state string transducers, which are also known as two-way deterministic generalized sequential machines (2DGSMs), and MSO-GTs over strings [EH01]:

Theorem 3.1 ([EH01])**.** *A transduction is definable by a 2DGSM iff it is definable by an MSO-GT over strings.*

Although strings and 2DGSMs are both quite simple, the techniques required to show this result exhibit much of the complexity needed to establish the corresponding results over more complicated structures. I will present the results in this section rather more detail since they serve as a good warm up for the material to come.

In outline, the structure of this section is as follows. First I review definitions of 2DGSMs and describe a representation of strings as graphs. Next, I describe the translation a 2DGSM to an equivalent MSO-GT, which proves one direction of Theorem 3.1. Before giving the other direction, I describe an extension of 2DGSMs where the tests on the input and moves of the read head are specified as formulas in MSO (2DGSM-MSO). I then sketch the proof that 2DGSM-MSO and 2DGSMs have the same expressive power. Two results—equivalence of regular and MSO-definable string languages, due to Büchi, Elgot, and Trakhtenbrot [Bö60, Elg61, Tra62], and closure of 2DGSMs under composition, due to Chytil and Jákl [CJ77]—play critical roles in that proof. Finally, I show that every MSO-GT on strings can be simulated by a 2DGSM-MSO, which proves the other direction of Theorem 3.1.

3.1 Two-way Finite-State String Transducers

A 2DGSM has two tapes—a read head that operates on the input tape and can move forwards and backwards, and a write head that operates on the output tape and only moves forward. Formally a 2DGSM is a sextuple $T = (Q, \Sigma, \Gamma, q_i, q_f, \delta)$ where Q is a finite set of states, Σ and Γ are the input and output alphabets, q_i and q_f are the initial and final states and δ is a finite set of instructions, whose syntax is defined below. To allow the machine to detect when its read head is at either end of the input tape, we adjoin two new symbols $\vdash$ and $\dashv$ to Σ and write Σ' for $\Sigma \cup \{\vdash, \dashv\}$. The moves of the read head are specified using the directives $\{\leftarrow, \rightarrow, \downarrow\}$, and the transition function δ is a finite subset of $(Q \times \Sigma' \times \Gamma^* \times \{\leftarrow, \rightarrow, \downarrow\} \times Q)$. I will call the instruction with a q and σ in its first two components a (q, σ)-instruction and write it in a manner that suggests its transition behavior: $(q, \sigma) \rightsquigarrow (\gamma, \mu, q')$. We will assume without loss of generality that $|\gamma| \leq 1$. As the machine is deterministic, there may be at most one (q, σ)-instruction in δ for every $q \in Q$ and $\sigma \in \Sigma$.

A configuration of a 2DGSM is a triple $(q, (\alpha_l, \sigma, \alpha_r), \beta)$ where $q \in Q$ is the current state, $(\alpha_l, \sigma, \alpha_r\alpha_2) \in (\Sigma'^* \times \Sigma' \times \Sigma'^*)$ represents the input tape and current position of the read head—α_l, σ, and α_r represents the symbols to the left, under, and to the right of the read head respectively—and β contains the contents of the output tape. The effect of a move on the read head is a partial function (since we do not allow the read head to move off the end of the tape) defined as:

$$\mathit{move}((\alpha_l, \sigma, \alpha_r), \mu) = \begin{cases} (\alpha_l, \sigma, \alpha_r) & \text{if } \mu = \downarrow \\ (\alpha_l\sigma, \sigma', \alpha_r') & \text{if } \mu = \rightarrow \text{ and } \alpha_r = \sigma'\alpha_r' \\ (\alpha_l', \sigma', \sigma\alpha_r') & \text{if } \mu = \leftarrow \text{ and } \alpha_l = \alpha_l'\sigma' \\ \text{undefined} & \text{otherwise} \end{cases}$$

The single-step transition is a partial function $\Rightarrow$ on configurations is given by $(q, (\alpha_1, \sigma, \alpha_2), \beta) \Rightarrow (q', (\alpha_1', \sigma', \alpha_2'), \beta\gamma)$ iff $(q, \sigma) \rightsquigarrow (\gamma, \mu, q') \in \delta$ and $\mathit{move}((\alpha_l, \sigma, \alpha_r), \mu) = (\alpha_l', \sigma', \alpha_r')$. The transduction realized by T is the partial function $T : \Sigma^* \rightarrow \Gamma^*$ defined by $T(w) = w'$ iff $(q_i, (\epsilon, \vdash, w \dashv), \epsilon) \Rightarrow^* (q_f, (\alpha_l, \sigma, \alpha_r, w'))$ for some $(\alpha_l, \sigma, \alpha_r) \in (\Sigma'^* \times \Sigma' \times \Sigma'^*)$.

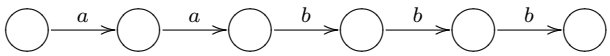
As an example, the transduction from the introduction that maps each string $w \in \{a, b\}^*$ to ww^r is given by the 2DGSM $T = (\{q_r, q_l, q_f\}, \{a, b\}, \{a, b\}, q_l, q_f, \delta)$ (to shorten the description of the rules, we use c as a schematic variable ranging over $\{a, b\}$):

$$\delta = \left\{ \begin{array}{lll} (q_l, \vdash) \rightsquigarrow (\epsilon, \rightarrow, q_l), & (q_l, c) \rightsquigarrow (c, \rightarrow, q_l), & (q_l, \dashv) \rightsquigarrow (\epsilon, \leftarrow, q_r), \\ (q_r, \vdash) \rightsquigarrow (\epsilon, \downarrow, q_f), (q_r, c) \rightsquigarrow (c, \leftarrow, q_r), & (q_r, \dashv) \rightarrow (\epsilon, \downarrow, q_r) & \end{array} \right\}.$$

The rule at the lower right of the display is never encountered and does not play a role in transductions; it is included only to illustrate a point below.

3.2 String Graphs

Let Σ be a finite alphabet. A string over Σ can be represented as a graph where the vertices are blank[1] and the edges are labeled with the elements of Σ appearing in the string.[2] For example, the string $aabbb$ is represented by the graph:



In a graph representing a string, the formulas

$$\mathit{left}(x) \triangleq \forall y.\, \neg \mathit{edge}(y,x) \wedge \exists z.\, \mathit{edge}(x,z) \qquad \mathit{right}(x) \triangleq \forall y.\, \neg \mathit{edge}(x,y) \wedge \exists z.\, \mathit{edge}(z,x)$$

describe the unique vertices at the start and end of the string; an arbitrary graph satisfies the formula

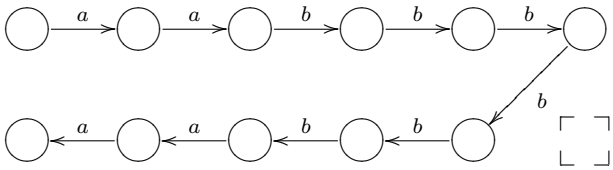
$$\mathit{string} \triangleq \exists l.\, \exists r.\, \mathit{left}(l) \wedge \mathit{right}(r) \wedge \forall x.\, [(x \neq l) \wedge (x \neq r) \Rightarrow \exists! w., \exists! y.\, \mathit{edge}(w,x) \wedge \mathit{edge}(x,y)]$$

iff it represents a string.

The transduction that concatenates the input string with its reversal can be expressed as an MSO graph transduction $M = (2, \mathit{string}, \Phi, \Psi)$ where Φ and Ψ are the following sets of formulas (again, I use c as a schematic variable ranging over $\{a, b\}$):

$$\Phi = \left\{ \begin{array}{l} \phi^1(x) \triangleq \mathit{true} \\ \phi^2(x) \triangleq \neg \mathit{right}(x) \end{array} \right\} \qquad \Psi = \left\{ \begin{array}{l} \psi_c^{1,1}(x,y) \triangleq \mathit{edge}_c(x,y) \\ \psi_c^{2,2}(x,y) \triangleq \mathit{edge}_c(y,x) \\ \psi_c^{1,2}(x,y) \triangleq \mathit{right}(x) \wedge \exists z.\, \mathit{right}(z) \wedge \mathit{edge}_c(y,z) \\ \psi_a^{2,1}(x,y) \triangleq \psi_b^{2,1} = \mathit{false} \end{array} \right\}$$

On the graph representing $aabbb$, the operation of the MSO transduction is depicted by the following diagram:



Each copy of the vertices is depicted as a row of the input graph. Only one copied vertex is discarded: the right-most vertex of the second copy where $\phi^2(x) = \neg \mathit{right}(x)$ is false. Its position is shown as a dashed box. The vertices are connected by edges as specified by the formulas in Ψ.

[1] That is, the vertex alphabet consists of a single symbol.

[2] One could also represent strings as graphs where the nodes carry the labels, but then the empty string is represented as the empty graph, whose associated structure has an empty universe; typically the universe of a relational structure is assumed to be non-empty.

3.3 From 2DGSMs to MSO Graph Transductions

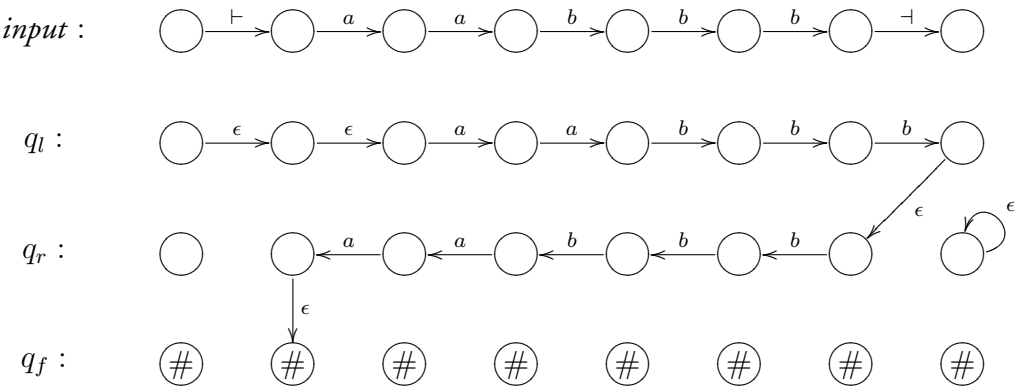
The translation from a 2DGSM $T = (Q, \Sigma, \Sigma', q_i, q_f, \delta)$ to an equivalent MSO-GT M follows the general technique sketched in the introduction, and is constructed as the composition of three MSO-GTs.

First, since the instructions of the 2DGSM make use of the special end markers $\vdash$ and $\dashv$, we need to wrap w with those symbols. This is accomplished using a fixed MSO-GT M_1. The second MSO-GT computes what Engelfriet and Hoogeboom call the *computation space* of T. When it is supplied with an input string $(\vdash\ w\ \dashv)$, it constructs the graph that represents the transition behavior of T on w from every state. The MSO-GT M_2 is constructed by taking a copy of the vertices for every state in Q, and edges corresponding to rules in T. Specifically, $M_1 = (Q, \mathit{string}, \Phi, \Psi)$ where the edge formulas are such that $u, v \models \psi_\alpha^{j,k}(x, y)$ iff v is the q_j copy of a vertex with an incoming edge σ and u is the q_i copy of the vertex to the left of v and the (q_j, σ)-rule writes α, moves to the left, and enters state q_j, and similarly for right and stay moves. Formally, for each $\alpha \in \{\epsilon \cup \Delta\}$ we let $\psi_\alpha^{q_j,q_k}(x, y)$ be the disjunction of the set of formulas

$$\begin{array}{lll} \{\exists z.\, \mathit{edge}_\sigma(z, x) \wedge \mathit{edge}(x, y) & | & (q_j, \sigma) \rightsquigarrow (\alpha, \rightarrow, q_k) \in \delta\} \\ \cup\, \{\mathit{edge}_\sigma(y, x) & | & (q_j, \sigma) \rightsquigarrow (\alpha, \leftarrow, q_k) \in \delta\} \\ \cup\, \{\exists z.\, \mathit{edge}_\sigma(x, z) \wedge x = y & | & (q_j, \sigma) \rightsquigarrow (\alpha, \downarrow, q_k) \in \delta\} \end{array}$$

We also have to add one more edge to get the machine going. We add the formula $\psi_{\epsilon^\star}^{q_i,q_i}(x, y) = (\mathit{left}(x) \wedge \mathit{edge}_\vdash(x, y))$ and set $\psi_{\epsilon^\star}^{q_j,q_0}(x, y)$ to *false* for every other pair of states. This special edge ϵ^* is used to mark the entry point of the transducer. To identify the final states, we label the vertices in the q_f copy with $\#$. All of the other vertices have a blank label.

As an example, the computation space of the reversal 2DGSM from above computes the following graph on $aabbb$ (for reference, the graph representing the input is also shown on the first line):



Because 2DGSMs are deterministic, every vertex in the computation space has at most one outgoing edge. But as the above example shows, it also contains edges labeled with ϵ, and spurious structure generated by unreachable rules (such as the loop in q_r). The third MSO-GT M_3 takes the computation space graph as input and picks out the unique path that corresponds to a run from the initial state to a final state if it exists (and otherwise produces the empty graph). Concretely this entails finding a path from the node that has an outgoing edge labeled with ϵ^* to a final state marked with $\#$. This transduction also tidies the output by suppressing ϵ.

To finish the proof, we use Theorem 2.1, which states that MSO-GTs are closed under composition; hence an MSO-GT equivalent to $(M_3 \circ M_2 \circ M_1)$ can be constructed.

3.4 2DGSMs with MSO Instructions

A key difference between 2DGSMs and MSO-GTs is that 2DGSMs operate locally—their instructions test the single symbol under the read head, and the move directives shift the head by at most one position. By contrast, a single formula in an MSO-GT can specify edges between copies of vertices that are far apart in the graph. As a step towards showing that every MSO-GT can be simulated by an equivalent 2DGSM, I will show how to ameliorate these seeming restrictions to local tests and moves by extending 2DGSMs with test and move instructions specified in MSO.

A 2DGSM with MSO instructions (2DGSM-MSO) is the generalization of a 2DGSM where the test and move components of each rule are formulas in MSO: $(q, \phi(x)) \rightsquigarrow (\gamma, \psi(x, y), q')$. The test formula $\phi(x)$ allows the machine to inspect more of the input tape than the single symbol currently under the read head. Similarly, the move formula $\psi(x, y)$ allows the machine to jump along the input tape. As 2DGSM-MSOs are deterministic, the tests for a given state must still be pairwise disjoint, and the moves must uniquely determine a new head position from every configuration.

Somewhat surprisingly, allowing 2DGSMs to be described using MSO instructions does not increase their expressive power:

Theorem 3.2 ([EH01]). *2DGSM= 2DGSM-MSO*

One direction, the inclusion 2DGSM $\subseteq$ 2DGSM-MSO, is trivial to show. Every single-symbol test σ can be equivalently written as an MSO formula $(\exists y.\, \mathit{edge}_\sigma(y, x))$. Stay moves can be written as $(x = y)$, left moves as $\mathit{edge}(y, x)$, and right moves as $\mathit{edge}(x, y)$.

For the other inclusion, 2DGSM-MSO $\subseteq$ 2DGSM we must show how to simulate the test and move formulas using the state space and local transitions of a 2DGSM. The proof goes in three parts. First we construct a 2DGSM that evaluates the result of a *regular look-around test*—the generalization of the regular look-ahead tests to machines with two-way read heads—at every position on the input tape. Next we show how to translate each test and move formula into regular look-around tests that provide enough data to simulate their behavior using strictly local moves. Putting these first two pieces together, we can obtain a sequence of 2DGSMs that pre-computes the result of every test and move needed to simulate every MSO instructions appearing in the 2DGSM-MSO. In the final step, we use a standard 2DGSM to compute the actual transduction, using the results of regular look-around tests pre-computed in the previous steps. The final result, the inclusion of 2DGSM-MSO in 2DGSM follows from the closure of 2DGSMs under composition:

Theorem 3.3 ([CJ77]). *Let R and S be 2DGSMs. There is an effective 2DGSM T such that $T(w) = S(R(w))$.*

A regular look-around test is a regular expression of the form $(R_l \cdot \sigma \cdot R_r)$ where σ is a single symbol and R_l and R_r are regular languages describing the strings to the left and right σ. Without

loss of generality, in our setting, it suffices to consider regular look-around tests that match the entire input tape: $(R_l \cdot \sigma \cdot R_r) \subseteq (\vdash \cdot \Sigma^* \cdot \sigma \cdot \Sigma^* \cdot \dashv)$. Our goal is to construct a 2DGSM that copies the input tape to the output tape and adds an annotation to each symbol (encoded along with the output in some extended alphabet) indicating if the regular look-around expression describes that position on the tape. The desired 2DGSM can be obtained as the composition of three auxiliary 2DGSMs. The first processes the input tape from left to right and uses its state space to run the finite automaton for $R_l \cdot \sigma$. On every input symbol if the finite automaton is in an accepting state then it copies the symbol to the output tape and annotates the test as true; otherwise it copies the symbol and annotates the test as false. The second 2DGSM processes the input in the opposite direction—from right to left—and uses its state space to simulate the finite automaton for the reversed language of R_r (the reversal of a regular language is also regular and can be effectively computed). The output after this second stage is a string of symbols each representing three pieces of data—the original symbol, the annotation for $R_l\sigma$, and the annotation for R_r^r. This is almost what we want, but the second traversal, which copied the tape going from right to left, reverses the order of symbols on the tape. A third 2DGSM reverses the input one last time and merges the annotation for the components of the regular look-around into a single annotation.

Next we will show how regular look-around can be used to simulate the MSO tests and moves of a 2DGSM-MSO. The idea is to use the equivalence of regular languages and closed MSO formulas to translate each MSO formulas into regular look-around expressions. However, the test and move formulas have free variables; picking out the symbol σ that appears in the middle of the look-around expression, and which corresponds to the free variable in the original formula, is a little subtle. Given a test formula $\phi(x)$ we work over an extended alphabet $\Sigma \times \{0, 1\}$. To $\phi(x)$ we add the constraints that the second component of x is 1, and that the second component of every other vertex is 0. We then close this formula up using an existential quantifier, and use the following theorem to obtain a regular automaton recognizing the same language:

Theorem 3.4 ([Bö60, Elg61, Tra62])**.** *A language L is accepted by a finite automaton iff there exists a closed MSO formula ϕ such that L is the set of strings that satisfy ϕ.*

This regular automaton can then be converted into a regular expression, whose characters are symbols in the extended alphabet. We can then extract a finite set (as the alphabet is finite) of regular look-around expressions by picking out the symbols tagged with 1.

The binary move formulas can also be converted to regular look-around tests that together provide enough information to simulate the move using a 2DGSM. First we construct regular look-around expressions that can be used to determine the direction of each move. From a move formula $\psi(x, y)$ construct the unary formulas:

$$\begin{aligned} \mathit{move\_left}(x) &\triangleq \exists y.\, y \neq x \wedge \mathit{path}(y, x) \wedge \psi(x, y) \\ \mathit{stay}(x) &\triangleq \psi(x, x) \\ \mathit{move\_right}(x) &\triangleq \exists y.\, y \neq x \wedge \mathit{path}(y, x) \wedge \psi(x, y) \end{aligned}$$

that describe the direction of the move. Note that since moves are deterministic, at most one will apply. By the construction described above, these unary formulas can be translated to finite sets of regular

look-around expressions. Second we construct regular look-around expressions that can be used to determine when the target of a move has been reached. Using the extension of the above technique to two variables, tag the occurrences of x and y in ψ and use Theorem 3.4 to calculate a finite set of regular look-around expressions, each of the form $(R_l \cdot \tau \cdot R_m \cdot \sigma \cdot R_r)$. For each of these, we compute regular look-around expressions needed to determine the target of a left or right move; I will show the left case, the other is symmetric. For the final 2DGSM to execute a left move described from $(R_l \cdot \tau \cdot R_m \cdot \sigma \cdot R_r)$ it will need to be able to detect when it has reached a position whose label is τ where the segment of the input tape between the starting position and that one belongs to R_m, and where the segment of the tape to the left of τ belongs to R_l. The 2DGSM can simulate the automaton for the reversal of R_m using its state space; the only additional regular look-around expression required is $(\vdash \cdot R_l \cdot \tau \cdot \Sigma^* \cdot \dashv)$, which describes the initial segment of the tape.

Putting all these pieces together, we produce a 2DGSM that annotates the input with the result of each regular look-around expression described above. The final 2DGSM simulates the tests of the 2DGSM-MSO using the corresponding annotation on the input tape. It simulates the tests of the 2DGSM-MSO by determining the direction of the move and the particular regular look-ahead expression that applies at the current position of the set of expressions calculated for the given formula (by determinism, there is at most one). It then starts moving in the appropriate direction and simulates the middle language until it reaches a position that is labeled with the target symbol and described by the expression for the initial segment.

3.5 From MSO Graph Transductions to 2DGSMs

The final result in this section shows how to translate an arbitrary MSO-GT $M = (k, \phi_{dom}, \Phi, \Psi)$ to an equivalent 2DGSM. The heavy lifting has already been done in the proof that 2DGSMs can simulate 2DGSMs. The construction is essentially the inverse of the construction used to show the opposite translation. We construct a 2DGSM-MSO with a state for every $i \in [1, k]$, and add transitions according to the elements in Ψ. The only difficult parts of this construction are ensuring that the resulting 2DGSM-MSO is actually deterministic, and setting up the transitions for the initial and final states. For every $\psi_\sigma^{j,k}(x, y) \in \Psi$ we construct the formula $\theta_\sigma^{j,k}(x, y) = \psi_\sigma^{j,k}(x, y) \wedge \phi_{dom} \wedge \phi^j(x) \wedge \phi^k(y)$. The extra constraints ensure that the moves of the 2DGSM-MSO will be functional. The 2DGSM-MSO has state set $[1, n] \cup \{q_i, q_f\}$, initial state q_i, final state q_f and instructions of the form

$$(j, \exists y.\, \theta_\sigma^{j,k}(x, y)) \rightsquigarrow (k, \sigma, \theta_\sigma^{j,k}(x, y)).$$

Finally, we add a transitions from the initial state q_i by describing the unique node with no incoming edges and transitions from every state to the final state q_f by negating the union of all of the move formulas. This completes the second half of the proof of Theorem 3.1.

4 Finite Trees

The next result in this survey is the capstone in a series of papers by Engelfriet and his co-authors. It establishes the equivalence between MSO-GTs over finite ranked trees and macro tree transducers

(MTTs) of linear size increase [EM03b]. For MSO graph transducers, the size of the output graph is at most linear in the size of the input—the copy index provides the bound. By contrast, because their rules allow the copying of, unrestricted MTTs can produce trees that are exponentially larger than the input. The main challenge in equating the two formalisms is identifying restrictions on MTTs that limit their power to transductions of linear size increase. The previous results in this line of work introduced a restriction on the dynamic transition behavior of MTTs called finite copying that achieves this goal and showed that every MSO-GT is equivalent to an MTT and vice versa. This may sound like the end of the story, but it was not known if ever MTT of linear size increase could be turned into a finite copying one. The result described here bridges this gap by demonstrating that every MTT transduction that is semantically of linear size increase can be normalized in exactly this way. The main result is as follows:

Theorem 4.1 ([EM03b]). *Let M be an MTT. The following are equivalent:*
(1) M is equivalent to an MSO-GT;
(2) M of linear size increase;
(3) M is equivalent to a finite copying MTT.

This gives a purely semantic condition that identifies a subclass of MTTs equivalent to MSO-GTs.

I begin this section by reviewing notation for ranked trees and MTTs, and by summarizing the previous results connecting MSO-GTs first to (restricted forms) of attributed tree transducers and then to finite copying MTTs. In the last part of the section, I describe the main result, an algorithm that normalizes a linear size increase MTT to a finite copying one, and its correctness proof.

4.1 Ranked Trees

A ranked set comprises a finite set of symbols Σ and a mapping *rank* that associates an integer to every symbol in Σ. We write $\sigma^{(k)}$ to indicate that *rank*$(sigma) = k$ and $\Sigma^{(k)}$ for the subset of Σ with rank k. Given a ranked alphabet the set of trees over Σ, written T_Σ is defined inductively as follows: every symbol $\sigma^{(0)} \in T_\Sigma$, and if $\sigma^{(0)} \in \Sigma$ and $t_1, \ldots, t_k \in T_\Sigma$ then $\sigma(t_1, \ldots, t_k) \in T_\Sigma$. We will write X and Y for the infinite set of variables $\{x_1, x_2, \ldots\}$ and parameters $\{y_1, y_2, \ldots\}$ and X_k for the finite set $\{x_1, \ldots, x_k\}$ and similarly for Y_k. When Σ is a ranked set and A is any set we write $\langle \Sigma, A \rangle$ for the ranked set of pairs with *rank*$(\langle \sigma, a \rangle =$ *rank*(σ), and $\Sigma \cup A$ for the ranked set with *rank*$(\sigma) = \sigma$ for $\sigma \in \Sigma$ and *rank*$(a) = 0$ for $a \in A$.

A bottom-up tree automaton is a triple $A = (P, \Sigma, h$ where P is a finite set of states, Σ is a ranked alphabet, and h is a set of rules, indexed by elements of Σ, and for each $\sigma \in \Sigma^{(k)}$ there is a rule h_σ that maps k-tuples of states to a state. The rules specify the bottom-up transition behavior of the automaton on a tree in the obvious way: $h(\sigma(t_1, \ldots t_k) = h_\sigma(h(t_1), \ldots, h(t_k))$. Instead of identifying final states, each state is associated with the set of trees in T_Σ that it accepts; this set, $h^{-1}(p)$, is denoted L_p. The analog of Büchi, Elgot, and Trakhtenbrot's theorem over ranked trees is due to Thatcher and Wright and Doner:

Theorem 4.2 ([TW68, Don70]). *A tree language L is accepted by a bottom-up tree automaton iff there exists a closed MSO formula ϕ such that L is the set of ranked trees that satisfy ϕ.*

4.2 Macro Tree Transducers

MTTs are powerful translation devices that operate on ranked trees. They traverse trees in a top-down fashion, but the states are equipped with parameters, which significantly increases their power. MTTs are closed under regular look-ahead [EV85], but we will work with a formalism with explicit regular look-ahead.

An MTT is a septuple $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ where Q is a ranked set of states, (P, Σ, h) forms a bottom-up tree automaton, Σ and Δ are the ranked input and output alphabets respectively, $q_0 \in Q^{(0)}$ is the initial state, and R is a finite set of rules, each of the form

$$\langle q, \sigma(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \rightsquigarrow \zeta\ \langle p_1, \ldots, p_k\rangle$$

with $q \in Q^m$, $\sigma \in \Sigma^{(k)}$, $\zeta \in T_{\langle Q, X_k\rangle \cup \Delta}(Y_m)$ and $p_1, \ldots, p_k \in P$. The notation for the right-hand sides of rules is a bit dense, so let us take a moment to unwind the definitions. Elements of $T_{\langle Q, X_k\rangle \cup \Delta}(Y_m)$ have one of three forms. They are either parameters y_i; ranked trees $\langle q_i, x_j\rangle(\zeta_1, \ldots, \zeta_l)$ that process a subtree, where $q_i \in Q^{(l)}$ and $\zeta_1, \ldots \zeta_l \in T_{\langle Q, X_k\rangle \cup \Delta}(Y_m)$; or ranked output trees $\delta(\zeta_1, \ldots, \zeta_k)$ where again $\zeta_1, \ldots \zeta_l \in T_{\langle Q, X_k\rangle \cup \Delta}(Y_m)$.

Specifying the transition relation induced by an MTT formally is a little complicated, although the basic idea is simple; see Fülop and Vogler's textbook for a detailed exposition [FV98]. The transition relation operates on configurations, which are trees in $T_{\langle Q, T_\Sigma\rangle \cup \Delta}(Y)$. The rules induce a singe-step transition relation $\Rightarrow$ as follows: if the current configuration u has a subtree $\langle q, \sigma(t_1, \ldots, t_k)\rangle(s_1, \ldots, s_m)$ at a path π, and there is a matching rule $\langle q, \sigma(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \rightsquigarrow \zeta\ \langle p_1, \ldots, p_k\rangle$ in R with each $t_i \in L_{p_i}$, then the transducer can take a step to the tree where the subtree at π is replaced with the result of substituting t_i for x_i and s_i for y_i in the right-hand side of the rule, ζ. The transition relation is confluent and terminating; we write $M(t) = s$ iff $\langle q_0, t\rangle \Rightarrow^* s$.

We consider only deterministic and total MTTs—i.e., for each q, σ and $\langle p_1, \ldots, p_k\rangle \in P^k$ there is exactly one corresponding rule in R. A MTT with no parameters is a top-down tree transducer with regular look-ahead (TT^R). If the look-ahead automaton has only one state then it and the states on the right-hand sides of rules are both trivial and can be omitted. A TT^R with a trivial look-ahead automaton is a top-down tree transducer (TT). We assume, without loss of generality, that M is *non-deleting*—i.e., each state uses each of its parameter at least once.

As an example, let $M = (\{q^{(0)}, r^{(2)}\}, \Sigma, \Delta, q, R)$ where $\Sigma = \{\sigma^{(1)}, \alpha^{(0)}\}$, $\Delta = \{\delta^{(2)}, \beta^{(0)}\}$, and R has rules

$$\begin{array}{rcl@{\qquad}rcl} q(\alpha) & \rightsquigarrow & \beta & r(\alpha)(y) & \rightsquigarrow & y \\ q(\sigma(x)) & \rightsquigarrow & r(x, \delta(\beta, \beta)) & r(\sigma(x)(y) & \rightsquigarrow & r(x, \delta(y, y)) \end{array}$$

Given the linear tree $t = \sigma(\sigma(\alpha))$, $M(t)$ computes the complete binary tree of the same height: $\delta(\delta(\beta, \beta), \delta(\beta, \beta))$. This example demonstrates that the size of the output tree computed by an MTT (measured as the number of nodes) can be exponentially larger than the input. The source of the exponential increase here is the copying of the y parameter in the (r, σ)-rule.

4.3 Previous Results

The technical material in Engelfriet and Maneth's paper [EM03b] does not directly establish the equivalence of MSO-GTs with MTTs of linear size increase. In this section I place their result in context, by summarizing the previous results needed to make the connection to MSO-GTs.

Bloem and Elgelfriet studied the connection between MSO-definable node relations and so-called tree-walking automata [BE97]. A node relation on a graph G is a subset of $(V_G \times V_G)$. The node relation defined by a binary MSO formula $\phi(x, y)$ is the set of vertices such that $u, v \models \phi$. A tree-walking automaton is a finite state machine that computes trips on trees. A pair of vertices (u, v) is in the relation computed by a tree-walking automaton if there is a run of the machine that starts at u in the initial state and ends at v in a final state. At each step the automaton can go up the tree to an ancestor, go down the tree to a child, or test a unary MSO formula at that node. Tree-walking automata can also be written using a regular-expression syntax. Their main theorem states that binary MSO formulas and tree-walking automata describe the same node relations. Translating a tree-walking automaton to an equivalent MSO formula is simple; they work from a regular expression syntax and use composition, union, and Kleene-star operators on binary MSO formulas. The other direction uses a tagging technique similar to the proof in the previous section. Using Thatcher, Wright, and Doner's theorem, they tag the free variables in the given binary formula, close it under existential quantifiers, and obtain a bottom-up tree automaton that recognizes a tree where two nodes are tagged iff the nodes satisfy the original formula. They then use the description of this tree automaton to construct a tree-walking automaton that finds the shortest path between any two such vertices. It uses unary MSO extracted from the tree automaton as regular look-ahead tests to determine the direction of the next step of the traversal.

In a second article [BE00], Bloem and Engelfriet proved the equivalent of MSO-GTs over trees and restricted forms of *attributed tree transducers* (ATTs) with regular look-ahead. ATTs are based on attribute grammars, which are a generalization of context-free grammars invented by Knuth for the purpose of assigning a semantics to a formal language [Knu68]. The rules of an attribute grammar specify how the values of of attributes—either inherited and synthesized—percolate down from ancestors or flow up from children. An ATT is a specialization of attribute grammar where all of the values are ranked trees. The main result in their article is that MSO graph transductions and ATTs satisfying the so-called single-use restriction are equivalent. The single-use restriction is a condition on the graph whose vertices are pairs of tree nodes and attributes, and whose edges represent the dependencies among attributes induced by the rules. The condition is satisfied if every vertex has at most one outgoing edge—i.e., that attribute is used at most once. The inclusion of single-use restricted ATTs in the MSO graph transductions is proved in a similar fashion as for 2DGSMs: they take one copy of the vertices for every attribute and construct the computation space by connecting the edges in a way that simulates the rules of the ATT. The single-use restriction is critical to ensuring that the output can be uniquely read off from this computation space graph. To show the other direction, Bloem and Engelfriet first put the MSO-GT into a localized normal form where every edge formula connects adjacent tree nodes. To obtain this normal form they use their result connecting binary MSO formulas and tree-walking automata to obtain a deterministic tree-walking automaton that computes the shortest walk trees between vertices for every edge formula in the MSO-GT. These local transductions can then

translated to single-use restricted ATTs.

The third result in this line of work is due to Engelfriet and Maneth [EM99]. The paper does not directly address MSO-GTs, but rather establishes the equivalence between single-use restricted ATTs with regular look-ahead and two restricted forms of MTTs. It is well known that every ATT can be simulated by a MTT. The basic idea is to use the parameters to store the inherited values and to fold the rules for synthesizing values into the rules of the MTT. The opposite inclusion, however, does not hold. This paper identifies a restriction on MTTs such that the restricted class of MTTs and single-use restricted ATTs with regular look-ahead are equivalent. The paper proposes two restrictions. The first restriction is a rather heavy syntactic restriction on the syntax of rules. The second restriction is a condition on the dynamic transition behavior of the transducer and is called finite copying. Intuitively, unrestricted MTTs can exhibit unbounded copying in two ways: by rules that process many copies of the same subtrees, and by rules that copy parameters many times (as we saw in the example above). An MTT is called finite copying in the input if there is a fixed bound on the number of times that any subtree is processed and finite copying in the parameters if there is a fixed bound on the number of times that each parameter appears in a configuration during a run of the transducer; the MTT is finite copying if it obeys both conditions. Finite copying MTTs are equivalent to ATTs obeying the single-use restriction, and are therefore also equivalent to MSO-GTs.

4.4 Finite Copying MTTs

The MTT that translates a linear tree into the full binary tree is fails to be finite copying because the value $\delta(\beta, \beta)$ that the initial state places in the parameter slot is copied once for every node in the input tree. An MTT that is finite copying in the parameters does not exhibit this kind of unbounded copying. The formal definition is a syntactic condition on MTTs. Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be an MTT. Although the transduction associated to M begins from the initial state q_0, we can also execute the transducer from an arbitrary state $q^{(m)} \in Q$. If $t \in T_\Sigma$ we write $M_q(t)$ for the normal form of $\langle q, t\rangle(y_1, \ldots, y_m)$. Because M is non-deleting, the parameters occur in this normal form. M is finite copying in its parameters if there is a fixed integer k such that for every tree t, state q, and parameter y_i the number of occurrences of y_i in $M_q(t)$ is less than or equal to k. To show that this property is decidable, we construct MTTs M' and M'' where M' takes trees of the form $q(t)$ and simulates $M_q(t)$, and M' takes an arbitrary tree and produces a tree whose size is linear in the number of parameters that occur in it. Then M is finite copying in the input iff $M''(M'(q(t)))$ is finite for every t and q. The language $\{q(t) \mid q \in Q \wedge t \in T_\Sigma\}$ is regular; by a theorem of Drewes and Engelfriet [DE98], it is decidable whether a composition of MTTs has finite range when restricted to a regular language. Hence, it is decidable whether M is finite copying in the parameters.

The other way that an MTT can fail to be finite copying is if there are inputs for which it copies a subtree arbitrarily many times. Intuitively, M is finite copying in the input if there is a fixed integer k such that for every tree t and subtree u the number of copies of u that is processed by M is less than or equal to k. To count the number of times that M copies a subtree, we extend M to a machine that works on trees that may contain states of the look-ahead automaton. To make this work, the look-ahead automaton is extended to handle its own states, by adding the rule $h_p = p$ for every $p \in P$.

Now letting $p = h^{-1}(u)$ and t' be the tree obtained from t by replacing u with p and evaluate $M(t'$ to a normal form, each of the copies of u will occur as trees $\langle q_i, p\rangle(s_1, \ldots, s_k)$. The *state sequence* of t at u is defined as the list of states that appear in the normal form $M(t')$; M is finite copying in the input if the length of every such state sequence is bounded by k. This property is decidable by a similar construction as above. It is straightforward to construct an MTT M' that converts a tree in $T_{\langle Q,P\rangle\cup\Delta}$ to a linear tree containing the states Q encountered during a traversal. Then M is finite copying in the input iff $M'(M(t))$ is finite for every t containing exactly one occurrence of a $p \in P$. Since the set of $t \in T_{\Sigma\cup P}$ where t has exactly one occurrence of a $p \in P$ is regular, by the same theorem [DE98], it is decidable whether an MTT is finite copying in the input.

There is one additional definition related to finite copying in the input that is needed for the main result. From a MTT M we can produce a TT^R T by dropping the parameters from each rule. Then M is *globally finite copying in the input* iff T is finite copying in the input. The global condition is weaker than the finite copying in the input as stated above. However if in the right-hand side of every rule in M, every parameter appears at most once, then M is called *linear in the parameters* and the two coincide. The following theorem establishes this fact:

Theorem 4.3 ([EM99])**.** *Let M be an MTT.*

(1) If M is globally finite copying in the input and linear in the parameters then M is finite copying.

(2) If M is finite copying in the parameters then there exists an equivalent MTT that is linear in the parameters that can also be effectively constructed.

4.5 Linear Size Increase MTTs

The next theorem demonstrates that every linear size increase MTT is equivalent to a finite copying MTT. The proof goes in several steps. The first step shows that every MTT can be normalized to one that is finite copying in the parameters and globally finite copying in the input. These facts are proved using two pumping lemmas, which demonstrate that that if a normalized MTT is not finite copying in the parameters or globally finite copying in the input, then it is not of linear size increase, a contradiction. The final step uses Theorem 4.3(2) to compute an MTT that is also linear in the parameters. By Theorem 4.3(1), this final machine is finite copying.

The normalization of an MTT again goes in two steps, one for the input and one for the parameters. I will show the input normalization first. As an example of an MTT that is linear size increase but not globally finite copying in the input, consider the following (the example is due to Engelfriet and Maneth [EM03b]): $M = (Q, \Sigma, \Delta, q, R)$ with $Q = \{q^{(0)}, r^{(0)}\}$, $\Sigma = \{\sigma^{(1)}, \alpha^{(0)}, \beta^{(0)}\}$, $\Delta = \{\delta^{(1)}, \alpha^{(0)}, \beta^{(0)}\}$, and R defined as follows:

$$\begin{array}{rcl@{\qquad}rcl} \langle q, \alpha\rangle & \rightsquigarrow & \alpha & \langle r, \alpha\rangle & \rightsquigarrow & \alpha \\ \langle q, \beta\rangle & \rightsquigarrow & \beta & \langle r, \beta\rangle & \rightsquigarrow & \beta \\ \langle q, \sigma(x)\rangle & \rightsquigarrow & \delta(\langle r, x\rangle, \langle q, x\rangle) & \langle r, \sigma(x)\rangle & \rightsquigarrow & \langle r, x\rangle \end{array}$$

M translates a linear trees into binary trees—e.g., $M(\sigma(\sigma(\sigma(\alpha))) = \delta(\alpha, \delta(\alpha, \delta(\alpha, \alpha))$—but it makes many unnecessary copies of the input tree in doing so. In particular, each subtree of the input is

processed by the state r, but the rules for r throw away all of the structure of those trees except for the values at the leave. Because r only ever produces one of two trees—α or β—it is redundant. An equivalent MTT that is globally finite copying in the input can be obtained by eliminating r and using regular look-ahead to extract the leaf value instead. The MTT $M' = (\{q^{(0)}\}, \Sigma, \Delta, q, R, h, \{p_\alpha, p_\beta\})$ where h and R are defined as follows:

$$\begin{array}{rcl} h_\alpha() & = & p_\alpha \\ h_\beta() & = & p_\beta \\ h_\sigma(p) & = & p \end{array} \qquad \begin{array}{rcl} \langle q, \alpha\rangle & \rightsquigarrow & \alpha \\ \langle q, \beta\rangle & \rightsquigarrow & \beta \\ \langle q, \sigma(x)\rangle & \rightsquigarrow & \delta(\alpha, \langle q, x\rangle)\ \langle p_a\rangle \\ \langle q, \sigma(x)\rangle & \rightsquigarrow & \delta(\beta, \langle q, x\rangle)\ \langle p_b\rangle \end{array}$$

In general, an MTT is called *input proper* if every state q of the transducer produces infinitely many output trees. The notion of input proper was invented by Aho and Ullman and was called "reduced" [AU71]. An MTT can be made input proper using the straightforward generalization of the state elimination procedure illustrated above in going from M to M'. For every pair $\langle q, p\rangle$ it is decidable, using finiteness of ranges [DE98], whether the set of trees $\{M_q(s) \mid s \in L_p\}$ is finite. If it is, then it is also possible to enumerate the elements. To make an MTT input proper, we eliminate every state that generates only finitely many trees and adjust the remaining rules to replace uses of that state with ones that build the output immediately, using regular look-ahead to pick the correct output.

The next lemma characterizes the input copying behavior of input proper MTTs that are also of linear size increase:

Lemma 4.4 ([EM03b])**.** *If M is an input proper MTT of linear size increase, then M is globally finite copying in the input.*

The proof goes by contradiction. Assume that M is input proper and of linear size increase, but not globally finite copying in the input. By the following pumping lemma:

Lemma 4.5 ([EM03b])**.** *If M is not globally finite copying in the input then there is a tree $t \in T_\Sigma$ and a state q such that $M(t)$ processes two distinct subtrees of t in state q.*

there is a state q with the stated properties. Moreover, since M is also input proper, the state q produces infinitely many subtrees. Then it is possible to pick a tree s produced by q such that when we pump the tree along the subtrees given by the lemma, the state sequence becomes arbitrarily large. As M is non-deleting, the size of the output is at least as large as the product of the number of symbols in the output alphabet appearing $M_q(s)$ and the number of times that the tree is pumped. It can then be shown that these choices exceed any a priori linear bound on the sizes of trees produced by M.

Next I will show how to normalize the use of the parameters of an MTT. Again, let us start with an example of an MTT that is linear size increase but not finite copying in the parameters (the example is also due to Engelfriet and Maneth [EM03b]). Let $M = (Q, \Sigma, \Delta, q, R)$ with $Q = \{q^{(0)}, r^{(1)}\}$, $\Sigma = \{\sigma^{(2)}, \gamma^{(2)}, \alpha^{(0)}, \beta^{(0)}\}$, $\Delta = \{\sigma^{(2)}, \gamma^{(2)}, \alpha^{(1)}, \beta^{(1)}, \overline{\alpha}^{(0)}, \overline{\beta}^{(0)}, \overline{\sigma}^{(0)}, \overline{\gamma}^{(0)}\}$, and R is defined as follows:

$$\begin{array}{rcl} \langle q, \alpha\rangle & \rightsquigarrow & \alpha(\overline{\alpha}) \\ \langle q, \beta\rangle & \rightsquigarrow & \alpha(\overline{\beta}) \\ \langle q, \sigma(x_1, x_2)\rangle & \rightsquigarrow & \sigma(\langle r, x_1\rangle(\overline{\sigma}), \langle r, x_2\rangle(\overline{\sigma})) \\ \langle q, \gamma(x_1, x_2)\rangle & \rightsquigarrow & \sigma(\langle r, x_1\rangle(\overline{\gamma}), \langle r, x_2\rangle(\overline{\gamma})) \end{array} \qquad \begin{array}{rcl} \langle r, \alpha\rangle(y) & \rightsquigarrow & \alpha(y) \\ \langle r, \beta\rangle(y) & \rightsquigarrow & \alpha(y) \\ \langle r, \sigma(x_1, x_2)(y)\rangle & \rightsquigarrow & \sigma(\langle r, x_1\rangle(y), \langle r, x_2\rangle(y)) \\ \langle r, \gamma(x_1, x_2)(y)\rangle & \rightsquigarrow & \sigma(\langle r, x_1\rangle(y), \langle r, x_2\rangle(y)) \end{array}$$

The function computed by M copies the input tree, but adds the overbarred version of the symbol at the root to every leaf. Although M is of linear size increase, it is not finite copying in the parameters because the number of copies of y in $M_r(s)$ equals the number of leaves of s. Intuitively, the parameter y is redundant because it is only ever used to carry a finite set of trees: $\{\overline{\alpha}^{(0)}, \overline{\beta}^{(0)}, \overline{\sigma}^{(0)}, \overline{\gamma}^{(0)}\}$. An equivalent MTT that is finite copying in the parameters can be obtained by eliminating y and instead using the state space of M to keep track of the correct value for the leaves. Let $M' = (\{q^{(0)}, q_\sigma^{(0)}, q_\gamma^{(0)}\}, \Sigma, \Delta, q, R)$ where R is defined as follows:

$$\begin{array}{rcl}
\langle q, \alpha\rangle & \rightsquigarrow & \alpha(\overline{\alpha}) \\
\langle q, \beta\rangle & \rightsquigarrow & \alpha(\overline{\beta}) \\
\langle q, \sigma(x_1, x_2)\rangle & \rightsquigarrow & \sigma(\langle q_\sigma, x_1\rangle, \langle q_\sigma, x_2\rangle) \\
\langle q, \gamma(x_1, x_2)\rangle & \rightsquigarrow & \sigma(\langle q_\gamma, x_1\rangle, \langle q_\gamma, x_2\rangle)
\end{array}$$

$$\begin{array}{rcl}
\langle q_\sigma, \alpha\rangle & \rightsquigarrow & \alpha(\overline{\sigma}) \\
\langle q_\sigma, \beta\rangle & \rightsquigarrow & \alpha(\overline{\sigma}) \\
\langle q_\sigma, \sigma(x_1, x_2)\rangle & \rightsquigarrow & \sigma(\langle q_\sigma, x_1\rangle, \langle q_\sigma, x_2\rangle) \\
\langle q_\sigma, \gamma(x_1, x_2)\rangle & \rightsquigarrow & \sigma(\langle q_\sigma, x_1\rangle, \langle q_\sigma, x_2\rangle)
\end{array}
\qquad
\begin{array}{rcl}
\langle q_\gamma, \alpha\rangle & \rightsquigarrow & \alpha(\overline{\gamma}) \\
\langle q_\gamma, \beta\rangle & \rightsquigarrow & \alpha(\overline{\gamma}) \\
\langle q_\gamma, \sigma(x_1, x_2)\rangle & \rightsquigarrow & \sigma(\langle q_\gamma, x_1\rangle, \langle q_\gamma, x_2\rangle) \\
\langle q_\gamma, \gamma(x_1, x_2)\rangle & \rightsquigarrow & \sigma(\langle q_\gamma, x_1\rangle, \langle q_\gamma, x_2\rangle)
\end{array}$$

An MTT is called *parameter proper* iff every parameter carries infinitely many values during runs of M. For every MTT there is an equivalent parameter proper MTT that is obtained using the generalization of the parameter elimination procedure illustrated above in going from M to M'.

The parameter copying behavior of parameter-proper MTTs of linear size increase is characterized by the following lemma:

Lemma 4.6 ([EM03b])**.** *If M is an parameter proper MTT that is also of linear size increase, then M is finite copying in the parameters.*

This proof, like the one above for input properness, goes by contradiction. Assume that M is parameter proper and of linear size increase, but not finite copying in the parameters. By the following pumping lemma:

Lemma 4.7 ([EM03b])**.** *If M is not finite copying in the parameters then there is a tree $t \in T_\Sigma$ and a state q such that $M(t)$ generates at least two copies of a parameter y_i and processes a subtree of t in state q using y_i.*

there is a state q and parameter y_i with the stated properties. Moreover, since M is also parameter proper, y_i carries infinitely many values. It is possible to pick a value carried by y_i such that when we pump the tree along the subtrees given by the lemma, the number of occurrences of y_i becomes arbitrarily large. As M is non-deleting, it can then be shown that these choices exceed any any a priori linear bound on the sizes of trees produced by M.

In the proof of Theorem 4.1, the implication $(1) \Rightarrow (2)$ holds since, by the definition, every MSO-GT is of linear size increase; $(2) \Rightarrow (3)$ holds since every MTT can be normalized to an input proper and parameter proper MTT that is finite copying by Lemmas 4.4 and 4.6; $(3) \Rightarrow (1)$ follows by the results discussed in Section 4.3.

5 Infinite Trees

In each the previous sections, we took the MSO-GTs as fixed and identified an equivalent finite-state transducer formalism for the given structure. The approach used by Colcombet and Löding over infinite trees is different [CL04]. They instead start from a transducer formalism that is strictly less powerful than MSO-GTs, and give an equivalence between transducers and a subclass of MSO-GTs. Specifically, they start from the extension of TT^{R}s to infinite trees generated as the unfoldings of graphs, and work with bisimulation-preserving MSO-GTs. Their main result, stated as two separate theorems below, is that every TT^R is equivalent to a bisimulation-preserving MSO-GT in the sense that for every graph, running the TT^R on the unfolding of a graph and unfolding the result of computing the MSO-GT on the graph gives the same answer. The core of this result depends on an argument demonstrating that every bisimulation MSO-GT can be normalized so that its edges go "down" in the tree generated by the graph.

5.1 Terms and Unfoldings

A *term* over a ranked set is like a ranked tree, except that it may additionally contain back edges. Every term is finite but the *unfolding* of a term t, obtained by replacing each back edge with the unfolding of the term at the target of that edge and written $\mathit{unfold}(t)$, is a tree and may be infinite. Two terms t and t' are *equivalent*, written $t \sim t'$ iff $\mathit{unfold}(t) = \mathit{unfold}(t')$; this equivalence coincides with the standard notion of bisimulation equivalence on their term representations. The operation of a TT^R is extended to a term t by applying its rules infinitely deep on $\mathit{unfold}(t)$. Although TT^{R}s are applied to the infinite unfoldings of terms, MSO-GTs are applied to their finite representations. An MSO-GT M is *bisimulation-preserving* iff for every term t the graph $M(t)$ is a term (i.e., it respects the ranks of symbols) and if $t \sim t'$ then $M(t) \sim M(t')$.

Over infinite trees the generalization of Büchi, Elgot, and Trakhtenbrot's theorem was shown by Rabin, who also invented automata for recognizing sets of infinite trees:

Theorem 5.1 ([Rab69])**.** *A language of infinite trees L is accepted by a Rabin automaton iff there exists a closed MSO formula ϕ such that L is the set of infinite trees that satisfy ϕ.*

5.2 From TT^{R}s to MSO Graph Transductions

The translation from a TT^R operating on the unfoldings of terms to MSO-GTs operating on the terms themselves uses the same technique we have seen before: an MSO-GT constructs the computation space which is then supplied as the input to a second MSO that extracts the output term. One copy of the tree is taken for each set of states and the edges are specified by the rules in the TT^R. Some TT^R rules consume input but do not produce any output:

$$\langle q(x_1, \ldots, x_k) \rightsquigarrow \langle q, x_i \rangle.$$

As with 2DGSMs, the edges in the computation space corresponding to these rules are labeled with a special symbol ϵ that is suppressed by the second MSO-GT. To implement regular look-ahead, we use

the fact that TT^{R}s preserve regular tree languages under inverse along with Rabin's theorem to obtain an equivalent MSO formula. The first theorem connecting TT^{R}s with MSO-GTs is as follows:

Theorem 5.2 ([CL04])**.** *Let t be a term and M be a TT^R. There exists an MSO graph transduction N such that $M($unfold$(t) =$ unfold$(N(t))$.*

5.3 From MSO Graph Transductions to TT^{R}s

A TT^R processes a tree in a strictly top-down fashion, and is finite state device. Because TT^{R}s are limited in these ways, there are some MSO-GTs that cannot be simulated using a TT^R. The transductions that cause problems are ones with edges that go upwards in the input term. Simulating such transductions using a TT^R can entail creating structure near to the root of the output tree from conditions that depend on nodes arbitrarily nodes in the input tree. Although a TT^R can use its regular look-ahead and state space to delay creating output for a while, it can only build up a finite amount of output before its memory is exhausted.

However, every bisimulation-preserving MSO-GT can be put into a normal form where if a formulas specifies the edge (u, v) and another formula specifies the edge (v, w) then w is lower than u (measured from the root) in the term—i.e., all of the edges go down in the term. The normal form is obtained by reinterpreting formulas about t over a *tree-like structure* t^*. The tree-like structure t^* is a graph whose vertices are sequences of nodes in t, and whose edges connect vertices where the last elements in each sequence have an t. It can be shown that every edge in t^* specified by a bisimulation-preserving MSO graph transduction goes down. Then, by a theorem of Walukiewicz [Wal02], each edge formula can be translated to an equivalent formula over the original structure that also goes down. This yields a MSO-GT in normal form.

From a normalized MSO-GT, the construction of an equivalent TT^R uses techniques that we have already seen. Working over an extended alphabet, we can tag the variables appearing in the edge formulas and obtain a Rabin automaton A such that A accepts trees with marked nodes iff the original formula is satisfied at the same nodes. For a given Rabin automaton A, a TT^R can construct the edge corresponding to A by simulating the automaton using regular look-ahead. A TT^R realizing the given transduction is obtained by iterating this technique. The second theorem is as follows:

Theorem 5.3 ([CL04])**.** *Let t be a term and N be a MSO-GT. There exists a TT^R M such that $\mathit{unfold}(N(t)) = M(\mathit{unfold}(t)$.*

6 Non-Deterministic Transductions

In this final section, I will briefly describe some results on non-deterministic MSO-GTs, drawing on the material from the last part of Engelfriet and Hoogeboom's article [EH01]. MSO-GTs can be easily extended to non-deterministic versions. The idea is to add free set variables $(X_1, \ldots, X_K)$ to each vertex and edge formula. Two graphs belong to the relation computed by a non-deterministic MSO

transduction if there is some valuation of the X_is such that interpreting the formulas over the input graph yields the output graph in the usual way.

One might hope that the equivalences between finite-state transductions and MSO-GTs would carry over from the deterministic case, but this is not the case. Even over strings, the non-deterministic versions of MSO-GTs and 2DGSMs are incomparable! As examples, the transduction mapping a^n to a^{nm} cannot be computed by a non-deterministic MSO-GT because its size is arbitrarily large. Conversely, non-deterministic two-way generalized sequential machines are not closed under composition but non-deterministic MSO-GTs are. If, however, we place a bound on the number of times that they can visit each position on the input tape, then we obtain the equivalence of non-deterministic MSO graph transductions with 2-ary compositions of non-deterministic two-way generalized sequential machines. Both directions of the inclusion can be shown by decomposing each transduction into a non-deterministic relabeling followed by a deterministic transduction.

7 Discussion

The MSO-definable graph transductions describe a large and natural class of functions on graphs. As the results described in this survey show, over many structures the MSO-GTs have the same expressive power as standard finite-state transducers operating on those structures. So far, we have focused rather narrowly on the specific strategies used to overcome the superficial differences between MSO-GTs and the various finite-state formalisms and especially the limitation of MSO-GTs to linearly increasing transductions, and the local operation of finite-state transducers. With the rest of this section, let us instead turn our focus forward, and explore some possible directions for future work.

Although the structures studied here—strings and ranked trees—are pervasive and canonical structures, they are not exhaustive. There are many other interesting data models and transducer formalisms that also deserve study. An obvious next step is to attempt to establish connections between MSO-GTs and new transducers as they arise. Unranked trees [Lib05], unordered trees [DZLM04], and nested words [AM06] are all good candidates. The case of unordered trees is of particular interesting because their automata are often formulated using counting constraints, which were first proposed as an extension of MSO over graphs [Cou90].

Even over strings and ranked trees, it would be interesting to explore the relationships between MSO-GTs and different transducers. For example, pebble transducers have emerged as a popular formalism for specifying transformations on trees [EM02]. Although pebble transducers have been related to MTTs [EM03a], it would be useful to study their relationship to MSO-GTs directly.

Although we have focused on transformations of entire structures, one can also use MSO-GTs to express queries that extract parts of structures. The hope would be that this study would inform the development of new query formalisms [NS02].

On the logic side, it would be interesting to study the effects of limiting the scope of formulas in various ways way. For example, transductions described using first-order or finite-variable logics would be limited to more local investigations of structures, which might match up better with some of the less powerful transducer formalisms.

Conversely, one could try to enhance the power of MSO-GTs by developing formalisms where the size of the output graph is not linearly bounded by the size of the input graphs. One possibility is to use a fixed operator, such as the unfolding operator defined for infinite trees, to grow the graph either before or after it is processed by an ordinary MSO-GT.

References

[AM06] Rajeev Alur and P. Madhusudan. Adding nesting structure to words. In Oscar H. Ibarra and Zhe Dang, editors, *Developments in Language Theory (DLT), Santa Barbara, CA*, volume 4036 of *Lecture Notes in Computer Science*, pages 1–13. Springer Verlag, 2006.

[AU71] Alfred V. Aho and Jeffrey D. Ullman. Translations on a context-free grammar. *Information and Control*, 19(5):439–475, 1971.

[B$\ddot{\text{o}}$60] J. Richard Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematik, Logik und Grund lagen der Mathematik*, 6:66–92, 1960.

[B$\ddot{\text{o}}$62] J. Richard Büchi. On a decision method in restricted second order arithmetic. In *International Congress on Logic, Method, and Philosophy of Science*, pages 1–12. Stanford University Press, 1962.

[BE97] Roderick Bloem and Joost Engelfriet. Monadic second order logic and node relations on graphs and trees. In Jan Mycielski, Grzegorz Rozenberg, and Arto Salomaa, editors, *Structures in Logic and Computer Science, A Selection of Essays in Honor of Andrzej Ehrenfeucht*, volume 1261 of *Lecture Notes in Computer Science*, pages 144–161. Springer-Verlag, 1997.

[BE00] Roderick Bloem and Joost Engelfriet. A comparison of tree transductions defined by monadic second order logic and by attribute grammars. *Journal of Computer and System Sciences*, 61(1):1–50, 2000.

[CJ77] Michal Chytil and Vojtech Jákl. Serial composition of 2-way finite-state transducers and simple programs on strings. In Arto Salomaa and Magnus Steinby, editors, *Automata, Logic, and Programming (ICALP), Turku, Finland*, volume 52 of *Lecture Notes in Computer Science*, pages 135–147. Springer-Verlag, 1977.

[CL04] Thomas Colcombet and Christof Löding. On the expressiveness of deterministic transducers over infinite trees. In Volker Diekert and Michel Habib, editors, *Symposium on Theoretical Aspects of Computer Science (STACS), Montpellier, France*, volume 2996 of *Lecture Notes in Computer Science*, pages 428–439. Springer-Verlag, 2004.

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs: I. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

[Cou91] Bruno Courcelle. The monadic second-order logic of graphs v: On closing the gap between definability and recognizability. *Theoretical Computer Science*, 80(2):153–202, 1991.

[Cou94] Bruno Courcelle. Monadic second-order definable graph transductions: A survey. *Theoretical Computer Science*, 126(1):53–75, 1994.

[DE98] Frank Drewes and Joost Engelfriet. Decidability of the finiteness of ranges of tree transductions. *Information and Computation*, 145(1):1–50, 1998.

[Don70] John Doner. Tree accepteors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451, 1970.

[DZLM04] Silvano Dal-Zilio, Denis Lugiez, and Charles Meyssonnier. A logic you can count on. In Neil D. Jones and Xavier Leroy, editors, *Symposium on Principles of Programming Languages (POPL), Venice, Italy*, pages 135–146. ACM, 2004.

[EH01] Joost Engelfriet and Hendrik Jan Hoogeboom. Mso definable string transductions and two-way finite-state transducers. *ACM Transactions on Computational Logic*, 2(2):216–254, 2001.

[Elg61] Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Transactions of the American Mathematical Society*, 98:21–52, 1961.

[EM99] Joost Engelfriet and Sebastian Maneth. Macro tree transducers, attribute grammars, and mso definable tree translations. *Information and Computation*, 154(1):34–91, 1999.

[EM02] Joost Engelfriet and Sebastian Maneth. Two-way finite state transducers with nested pebbles. In Krzysztof Diks and Wojciech Rytter, editors, *Mathematical Foundations of Computer Science (MFCS), Warsaw, Poland*, volume 2420 of *Lecture Notes in Computer Science*, pages 234–244. Springer Verlag, 2002.

[EM03a] Joost Engelfriet and Sebastian Maneth. A comparison of pebble tree transducers with macro tree transducers. *Acta Informatica*, 39(9):613–698, 2003.

[EM03b] Joost Engelfriet and Sebastian Maneth. Macro tree translations of linear size increase are mso definable. *SIAM Journal of Copmuting*, 32(4):950–1006, 2003. Preliminary version in *Symposium on Theoretical Aspects of Computer Science (STACS)*, 2000.

[End77] Herbert Enderton. *A Mathematical Introduction to Logic*. Academic Press, New York, NY, 1977.

[EV85] Joost Engelfriet and Heiko Vogler. Macro tree transducers. *Journal of Computer and System Sciences*, 31(1):71–146, 1985.

[EvO97] Joost Engelfriet and Vincent van Oostrom. Logical description of context-free graph languages. *Journal of Computer and System Sciences*, 55:489–503, 1997.

[FV98] Zoltan Fülöp and Heiko Vogler. *Syntax-Directed Semantics: Formal Methods Based on Tree Transducers*. Springer-Verlag, 1998.

[HtP97] Hendrik Jan Hoogeboom and Paulien ten Pas. Monadic second-order definable text languages. *Theory of Computing Systems*, 30:335–354, 1997.

[Knu68] Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968.

[Lib05] Leonid Libkin. Logics for unranked trees: An overview. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *International Colloquium on Automata, Languages and Programming (ICALP), Lisbon, Portugal*, volume 3580 of *Lecture Notes in Computer Science*, pages 35–50. Springer Verlag, 2005.

[NS02] Frank Neven and Thomas Schwentick. Query automata over finite trees. *Theoretical Compututer Science*, 275(1–2):633–674, 2002.

[Rab69] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

[Tho97] Wolfgang Thomas. *Handbook of Formal Languages, vol. 3: Beyond Words*, chapter Languages, Automata, and Logic, pages 389–455. Springer-Verlag, New York, NY, 1997.

[Tra62] Boris A. Trakhtenbrot. Finite automata and monadic second order logic. *Siberian Mathematical Journal*, 3:101–131, 1962.

[TW68] James W. Thatcher and Jesse B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.

[Wal02] Igor Walukiewicz. Monadic second-order logic on tree-like structure. *Theoretical Computer Science*, 275(1–2):311–346, 2002.