WPE II Report

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

Actions Louder than Words? MSO-definable Transductions

J. Nathan Foster

17 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 [B60, 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 [B62], 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, focusing on the cases where the transducers are deterministic and the structures are strings [EH01], trees [BE00, EM99, EM03] 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 concatented 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, certain transformations that are difficult to specify in one may be easily described in the other. In these situations, having two ways to skin the same cat is quite useful. In particular, transformations where data is moved around the structure are often simpler to specify as MSO trandsuctions where it is as easy to specify an edge between far-apart vertices as it is to specify one between adjacent vertices. By comparison, in a finite-state transducer, the individual transitions are all local; to connect distant vertices the machine must save its current position in the state space, traverse the structure until it reaches the far-off data, generate the output, and return to its original location to continue processing the rest of the structure. 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 approaches.

Second, 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. This has obvious applications in the area of automatic verification because one can read off a detailed specification of a transduction directly from its implementation.

Third, 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. In the case of trees, one of the standard kinds of transducers—the macro tree transducers—is not closed under composition. The main result described in Section 4 is that by bounding the amount of copying they can perform, these restricted MTTs can be shown equivalent to MSO-GTs. This result seems to hit a "sweet spot" that optimizes several tradeoffs between expressiveness and tractability.

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. In most cases, one can let the copy number k be equal to the number of states and then 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 more 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 difference 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. Lastly, one can introduce conditions that force 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 areas for further investigation in the future.

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 for graphs. 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 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 a predicate that picks out certain elements of the universe. Accordingly, every second-order variable in MSO denotes a set. Following this intuition, I will call second-order variables *set variables* and, as usual, distinguish them from term variables by using upper-case letters for set variables and lower-case letters for set variables. Also, instead of $X(x)$ I will write $x \in X$.

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 quantifers $\exists x.\, \phi$, $\exists! x.\phi$, 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 interpretations of appropriate arity for every symbol in the vocabulary. To each graph G, we associate the γ-structure $\mathcal{G}_G$ obtained by letting the universe be the set of vertices V_G, and defining the interpretation of the relational symbols as:

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

A valuation of variables is a finite map from variables to elements of the universe or sets of such elements. I 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—$\rho(x) = \rho(y)$ 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 is 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, and write $u, v \models \phi(x, y)$.

As examples, we can write an MSO formula that is satisfied by a set of vertices iff it closed under the edge relation:

$$\mathit{closed}(X) \triangleq \forall x.\, \forall y.\ (x \in X \wedge \mathit{edge}(x, y)) \Rightarrow y \in X,$$

and one that is satisfied iff there exists a directed path from x to y:

$$\mathit{path}(x, y) \triangleq \forall X.\, (\mathit{closed}(X) \wedge x \in X) \Rightarrow y \in X.$$

Finally, we can write a formula that is satisfied iff the graph is connected:

$$\mathit{connected} \triangleq \forall x.\, \forall y.\, \mathit{path}(x, y).$$

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 their indices as follows: $\phi^i_\sigma(x)$ and $\psi^{i,j}_\delta(x, y)$. When either alphabet is a singleton, I will omit the subscript and write $\phi^i(x)$ and $\psi^{i,j}(x, y)$.

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 (Courcelle [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 I will survey connects the class of transductions computed by two-way determinstic generalized sequential machines (2DGSMs) and MSO-GTs over strings [EH01]:

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

Although strings and these machines are both quite simple, the techniques required to show this result exhibits much of the complexity of similar results over more complicated structures. I will present the results in this section in some detail since they serve as a good warmup for the material to come in the later sections.

In outline, the structure of this section is as follows. First I review definitions of 2DGSMs and of 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 first describe an extension of 2DGSMs where the tests on the input and moves of the read head are described using formulas in MSO (2DGSM-MSO). I then sketch the proof that 2DGSM-MSO and 2DGSMs have the same expressive power. Two results—equivalence of regularity and MSO-definability for 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 on the input tape that can move forwards and backwards, and a write head on the output tape that only moves to the right. Formally it is a sextuple $T = (Q, \Sigma, \Gamma, q_i, q_f, \delta)$ where Q is a finite set of states, Σ and Σ 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: $(q, \sigma) \rightsquigarrow (\gamma, \mu, q')$. 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 is the contents to the left of the read head, σ is the symbol currently under the read head, and α_r is the contents to the right—and β contains the contents of the output tape. The effect of a move on the read head is a partial function defined as:

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

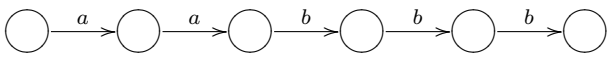
The single-step transition is a partial function $\Rightarrow$ on configuations is given by $(q, (\alpha_1, \sigma, \alpha_2), \beta) \Rightarrow (q', (\alpha'_1, \sigma', \alpha'_2), \beta\gamma)$ iff $(q, \sigma) \rightsquigarrow (q', \gamma, \mu) \in \delta$ and $\mathit{move}((\alpha_l, \sigma, \alpha_r), \mu) = (\alpha'_l, \sigma', \alpha'_r)$. The transduction realized by T is the partial function $T : \Sigma^* \rightarrow \Gamma^*$ defined by $T(w) = w'$ iff there exists $(\alpha_l, \sigma, \alpha_r) \in (\Sigma'^* \times \Sigma' \times \Sigma'^*)$ such that $(q_i, (\epsilon, \vdash, w \dashv), \epsilon) \Rightarrow^* (q_f, (\alpha_l, \sigma, \alpha_r, w'))$.

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

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

3.2 String Graphs

Let Σ be a finite alphabet. A string over Σ can be represented as a graph where the nodes are left blank and the edges are labeled with the elements of Σ appearing in the string.[1] 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 nodes with no incoming and outgoing edges respectively; an arbitrary graph satisfies the formula

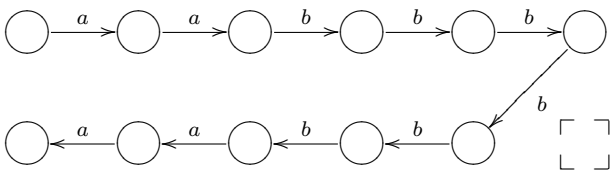
$$\mathit{string} \triangleq \exists l.\, \exists r.\, \mathit{left}(l) \wedge \mathit{right}(r) \wedge \forall x.\, [(x \neq l) \wedge (x \neq r) \Rightarrow \exists! w., \exists! y.\, \mathit{edge}(w,x) \wedge \mathit{edge}(x,y)]$$

iff it represents a string.

The transduction from the previous section can be written 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 drawn using a dashed box. The vertices are connected by edges as specified by the formulas in Ψ.

[1] One could also represent strings as graphs where the nodes carry the labels, but then the empty string is represented as the empty graph.

3.3 From 2DGSMs to MSO Graph Transductions

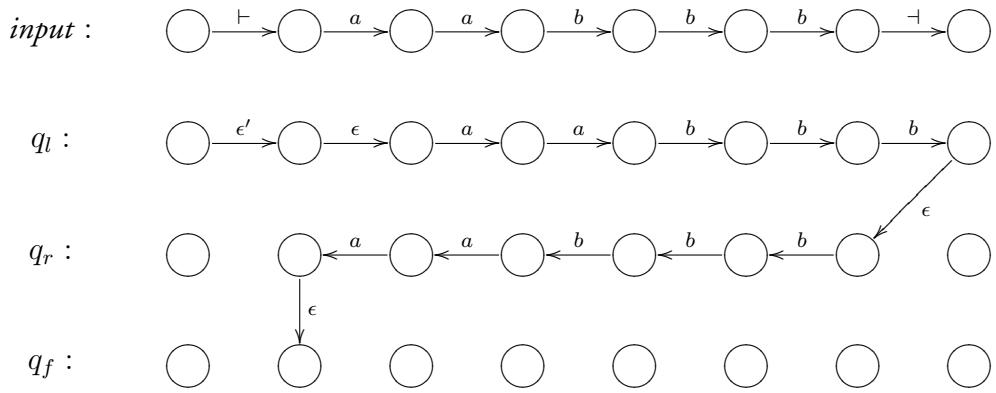
The translation from a 2DGSM $T = (Q, \Sigma, \Sigma', q_i, q_f, \delta)$ to an equivalent MSO graph transduction M follows the general techinque sketched in the introduction. The translation goes in two steps.

The first step constructs an MSO graph transducer M_1 that computes what Engelfriet and Hoogeboom call the *computation space* of T. When supplied with a string w, $M_1(w)$ creates a graph that represents the transition behavior of T on w in every state. Accordingly, M_1 is constructed by taking a copy of the vertices for every state in Q and connecting up the edges iff the transducer has a corresponding transition. More specifically, $M_1 = (|Q|, \mathit{string}, \Phi, \Psi$ where $\phi^i = \mathit{true}$ for every i, and the edge formulas are such that $u, v \models \psi^{j,k}_\alpha(x, y)$ iff v is the jth copy of a vertex with an incoming edge σ and u is the ith copy of the vertex to the left of v and the rule $(q_j, \sigma) \rightsquigarrow (q_j, \alpha, \leftarrow) \in \delta$ or similarly for the other move directives. Formally we define sets of binary formulas:

$$
\begin{array}{ll}
R(j,k,\alpha) = \bigcup_\sigma \{\exists z.\, \mathit{edge}_\sigma(z,x) \wedge \mathit{edge}(x,y)\} & \text{such that } (q_j,\sigma) \rightsquigarrow (q_k,\alpha,\rightarrow) \in \delta \\
L(j,k,\alpha) = \bigcup_\sigma \{\mathit{edge}_\sigma(y,x)\} & \text{such that } (q_j,\sigma) \rightsquigarrow (q_k,\alpha,\leftarrow) \in \delta \\
S(j,k,\alpha) = \bigcup_\sigma \{\exists z.\, \mathit{edge}_\sigma(x,z) \wedge x = y\} & \text{such that } (q_j,\sigma) \rightsquigarrow (q_k,\alpha,\downarrow) \in \delta
\end{array}
$$

and let $\phi^{j,k}_\alpha = \bigvee(R(j,k,\alpha) \cup L(j,k,\alpha) \cup S(j,k,\alpha))$. To finish the construction, we have to add one more edge to get the machine going. We set $\psi^{i,i}_{\epsilon'}(x,y) = \mathit{left}(x) \wedge \mathit{edge}_\vdash(x,y)$. Note that ϵ' is a special symbol different than ϵ that marks the entrance point into the computation space.

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



The second step takes the computation space 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 in the copy of the vertices for q_i labeled with ϵ' to a final state, which can easily be specified in MSO. There is one additional complication: because the computation space has edges labeled with ϵ and ϵ', this transduction also needs to tidy the output by supressing these edges. It is straightforward to construct a general MSO graph transducer M_1 that performs both of these tasks.

To finish the proof, we use Theorem 2.1 which states that MSO-GTs are closed under composition. Thus, $M_1 \circ M_0$ can be expressed as an MSO-GT.

3.4 2DGSMs with MSO Instructions

A key difference between a 2DGSM and a MSO graph transducers is that 2DGSMs operate locally—their instructions test the single symbol under the read head, and the directives move the head by at most one position. An MSO graph transducer, by comparison, can specify edges between copies of vertices that are far away in the graph. Before proving that every MSO-GT can be simulated by an equivalent 2DGSM I will show how to ameliorate these apparent restrictions to local tests and moves by extending the notion of 2DGSM to 2DGSM-MSO—a formalism that bridges the gap between 2DGSMs and MSO-GTs.

The operation of a 2DGSM-MSO with MSO instructions is like an ordinary 2DGSM, but its instructions are specified using MSO formulas. More specifically, a 2DGSM-MSO is the generalization of a 2DGSM where the test and move components of each instruction are formulas in MSO:

$$(q, \phi(x)) \rightsquigarrow (q', \gamma, \psi(x, y))$$

The test formula $\phi(x)$ allows the machine to evaluate regular properties of the input tape instead of just examining the value of the single symbol beneath the read head. Similarly, the move formula $\psi(x, y)$ allows the machine to jump along the input tape from its current position u to a position v such that $u, v \models \psi(x, y)$. As 2DGSM-MSOs are still deterministic, the tests must 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, as the following theorem shows:

Theorem 3.2 (Engelfriet and Hoogeboom [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 $\downarrow$ can be written as $x = y$, left moves $\leftarrow$ 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, I show that given a single regular look-around test—the a generalization of the standard notion of regular look-ahead to our setting, where the read head can move in two directions—one can construct a 2DGSM that pre-processes the input by copying the input tape and adding an annotation indicating the result of the look-around test to every symbol. I then show how to test and move formulas can be translated into a finite number of regular look-around tests. Putting these two pieces together, yields a sequence of 2DGSMs that pre-computes the result of every test and move in the given set of MSO instructions and annotates the input with this information. In the final step, I show how to use a standard 2DGSM to compute the actual transduction, making use of the annotations pre-computed in the previous step. The final result, the inclusion of 2DGSM-MSO in 2DGSM follows from the closure of 2DGSMs under composition:

Theorem 3.3 (Chytil and Jákl [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: $(R_l \cdot \sigma \cdot R_r) \subseteq (\vdash \cdot \Sigma^* \cdot \sigma \cdot \Sigma^* \cdot \dashv)$. First let us see how to construct a 2DGSM that copies the input tape to the output tape, adding an annotation—encoded in the output using an extended alphabet—to each symbol indicating if that position is described by the regular look-around expression. The 2DGSM is produced in three steps. The first 2DGSM 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. Similarly, 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 with three components—the original input, the annotation for $R_l\sigma$, and the annotation for R_r^r. This is almost what we want, but the second traversal, which processed the tape right to left, reversed 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, yielding the desired annotated tape.

Now I 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 $\phi(x)$ add the constraints that the second component of x is 1, and that the second component of every other vertex is 0. Then close this formula up using an existential quantifier, and use the equivalence of regular languages and MSO formulas to obtain a regular automaton recognizing the same language.

Theorem 3.4 (Büchi,Elgot, and Trakhtenbrot[B$\ddot{\text{o}}$60, 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 atomic symbols are each tagged with 0 or 1. In general, the regular expression will be finite union of where each disjunct has exactly one symbol tagged with 1; to finish the construction, use the tags to identify the unique symbol tagged with a 1 from each.

In a similar way the binary move formulas can be converted to finite sets of regular look-around tests. First observe that that the direction of each move can be determined using regular look-around. From a move formula $\psi(x, y)$ construct the unary formulas:

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

that describe the direction of the move. Note that since moves are deterministic, at most one will apply

during a given transduction. As these are unary formulas, by the technique just presented, they can be translated to (finite unions of) regular look-around expressions.

The last part of the construction shows how to determine the target of a move formula, once its direction has been determined. If the move formula is a stay directive, then simulating the behavior is trivial. Otherwise, assume that the move is to the left (the right case is symmetric). 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 produce a finite union of regular look-around expressions $(R_l \cdot \tau \cdot R_m \cdot \sigma \cdot R_r)$. By determinism, at most one of these regular look-around expressions describes the input tape; the specific expression can be determined using regular look-around. The actual move can then be executed by simulating the automaton for the reversal of R_m using the state space of the final 2DGSM. When it reaches a position accepted by R_m^r with symbol τ, it uses regular look-around with $(\vdash \cdot R_l \cdot \tau \cdot \Sigma^* \cdot \dashv)$ to check that the preceeding segment of the tape is also correct.

As the total number of tests and moves are finite, and as 2DGSMs are closed under composition, all of the 2DGSMs, whose composition fully annotates the input string with the results of the regular look-around expressions generated by tests and move formulas, can be combined into a single 2DGSM. The final step to finish the proof is simple: using the annotations, an ordinary 2DGSM can use its local tests and moves to simulate the behavior of the 2DGSM-MSO.

3.5 From MSO Graph Transductions to 2DGSMs

The final result given in this section shows how to translate an arbitrary MSO-GT $M = (n, \phi_{dom}, \Phi, \Psi)$ to an equivalent 2DGSM. The heavy lifting required was already done by the proof that 2DGSMs can be extended with tests and moves specified as MSO formulas. The construction is basically the inverse of the construction used to show the opposite direction. Construct a 2DGSM-MSO whose states correspond to the copy index n of M, and add transitions according to the elements in Ψ. Some care is needed, however, to ensure that the constructed 2DGSM-MSO is deterministic.

To every $\psi_\sigma^{j,k}(x,y) \in \Psi$ we associate the instruction $\theta_\sigma^{j,k}(x,y) = \psi_\sigma^{j,k}(x,y) \wedge \phi_{dom} \wedge \phi^j(x) \wedge \phi^k(y)$. These extra conditions ensure that the transition in T are functional. We define a 2DGSM-MSO with state set $[1,n] \cup \{q_i, q_f\}$, initial state q_i, final state q_f and instructions of the form

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

The transitions from and to the initial and final states are special. For the intial state q_i, we add a transition to state that produces the first symbol in the output. This condition can be identified by enumerating all the vertices and selecting the unique one that does not have an incoming edge. Finally, we add a single transition to the final state q_f by taking the negation of all the tests $\exists y.\, \theta_\sigma^{j,k}(x,y)$.

4 Finite Trees

The next result in this survey, due to Engelfriet and Maneth, is the capstone in a series of papers by Engelfriet and his co-authors. It establishes the equivalence between MSO-GTs over ranked trees and

macro tree transducers (MTTs) of linear size increase [EM03]. 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 they can copy subtrees many times over, 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 limits 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. Moreover, the finite copying property is decidable. This may sound like the end of the story, but it awkward to apply in practice because any particular MTT of linear size increase may or may not be finite copying. The result described here bridges this gap by demonstrating that every MTT transduction that is semantically of linear size increase can be normalized to an equivalent MTT that is finite copying. Thus, they obtain an equivalence between MSO-GTs and a class of MTTs that is defined by a purely semantic condition.

In outline, the section is structured as follows. I begin by reviewing notation for ranked trees and MTTs, and by summarizing the previous results connecting MSO tree transductions 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, the algorithm that normalizes a linear size increase MTT and yields a finite copying one.

4.1 Ranked Trees

A ranked set comprises a finite set of symbols Σ and a function *rank* : $\Sigma \to \mathbb{N}$ 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 an arbitrary set we write $\langle \Sigma, A \rangle$ for the ranked set 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.

It is straightforward to represent a ranked tree as a graph, modulo some bookeeping to ensure that the ranks of symbols are respected. However, as none of the results in this section require explicit notation for graphs, I will omit the formal definition. See Section 5.1 for the representation of ranked infinte trees.

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 compared to ordinary top-down transducers. MTTs are closed under regular look-ahead [EV85], but it will be simpler to work with a formal definition that has explicit regular look-ahead.

Formally, a MTT with regular-lookahead 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 bare parameters y_i; ranked trees $\langle q_i, x_j\rangle(\zeta_1, \ldots, \zeta_l)$ processing 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 $\zeta_1, \ldots \zeta_l \in T_{\langle Q, X_k\rangle \cup \Delta}(Y_m)$.

Formally specifying the of the transition relation that is induced by an MTT formally is a little complicated, although the basic idea is simple; our presentation will remain somewhat informal; see Fülop and Vogler's textbook for a standard 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 correspnding rule in R. A MTT with no parameters—i.e., where the rank of each state is 0—is a top-down tree transducer with regular look-ahead (TT^R). If the lookup automaton has only one state then it and the states on the right-hand sides of rules are both trivial and can be omitted. We also assume, without loss of generality, that M is *non-deleting*—i.e., each state uses each of its parameter at least once.

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

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

Given the linear tree $t = \sigma(\sigma(\alpha))$, $M(t)$ computes the complete binary tree whose height is one greater than the input tree: $\delta(\delta(\beta, \beta), \delta(\beta, \beta))$. This example also demonstrates that the size (i.e., number of nodes) of the output tree computed by an MTT 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 [EM03] does not directly establish the equivalence of MSO-GTs with MTTs of linear size increase. In this section, I wil place this result in context, by briefly describing the previous results that establish the connection.

The starting point is the analog of Büchi, Elgot, and Trakhtenbrot's theorem over ranked trees due to Thatcher and Wright and Doner. It equates the languages of ranked trees accepted by bottom-up tree transducers and those definable by closed MSOformulas [TW68, Don70].

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 autotomaton 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. The 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 use the regular expression syntax along with the 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 decription 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 an article reporting the results from his Master's thesis [BE00], Bloem and Engelfriet proved the equivalent of MSO-GTs over trees and restricted ATTs with regular look-ahead. Attribute grammars are a generalization of context-free grammars invented by Knuth for the purpose of assigning semantics to a formal language [Knu68]. In an attribute grammar, there are two kinds of attributes, inherited and synthesized. The grammar contains a set of rules that specify how the values of inherited attributes percolate down to subtrees and the values of synthesized attributes flow up to ancestors. 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. An MSO transduction walks over the computation space and picks out this tree. To show

the other direction, Bloem and Engelfriet first put the MSO graph transduction into a local normal form where every edge formula connects adjacent tree nodes. The key part of this proof uses their result connecting binary MSO formulas and tree-walking automata: from an arbitrary edge formula they obtain a tree-walking automaton that gives a shortest walk on trees between vertices satisfying the formuula. These local transductions can then be translated to single-use restricted ATTs; regular look-ahead is needed to pre-process the input tree.

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. Every ATT can be simulated by a MTT: the basic idea is to use the parameters to store the inherited values and to create synthesized values at each tree node. The other inclusion, however, does not hold. The problem addressed in this paper was to find a restriction on MTTs to make them equivalent to single-use restricted ATTs with regular look-ahead so that the equivalence with MSO-GTs would also carry over. Two restrictions are proposed in the article. The first restriction is a rather heavy syntactic restriction on the syntax of rules, also called single-use restricted. The second restriction is a more flexible condition on the dynamic transition behavior of the transducer and is called finite copying. Intuitively, unrestricted MTTs can exhibite unbounded copying in two ways: by rules that process the same subtrees many times, and by rules that copy the parameters many times (as was shown 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.

Now we turn to the main focus of this section—Engelfriet and Maneth's result that finite copying and linear size increase MTTs are equivalent.

4.4 Finite Copying MTTs

I first give the two conditions that an MTT must obey to be finite copying, and show that these are decidable.

The MTT from above that translates a linear tree into the full binary tree demonstrates one of the ways that an MTT can fail to be finite copying. That transducer is not finite copying because the value $\delta(\beta, \beta)$ that the initial state places in the parmeter slot is used to form all of the trees near the leaves the output tree—the parameter is copied exponentially many times. The notion of finite copying in the parameters forbids this kind of unbounded copying of parameters. The formal definition is formulated as a syntactic condition on MTTs. Let $M = (Q, P, \Sigma, \Delta, q_0, R, h)$ be a MTT. Although the transduction associated to an MTT 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 $\langle q, t\rangle(y_1, \ldots, y_m)$. Note that although the parameters usually contain tree values, during evaluation runs starting from M_q the bare parameters will be copied and in general will appear in the normal form. An M is finite copying in the parameters if the number of occurences of each parameter is a priori bounded—i.e., 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 $M_q(t)$, viewed as a string, is less than or equal to k. The property is decidable because we can construct an MTTs M' and M'' where M' takes trees of the form $q(t)$ and, using the definition of M, simulates $M_q(t)$ for every $q \in Q$, and M' takes an arbitrary tree and produces one whose size is linear in the number of parameters that occurr 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 subtrees arbitrarily many times. One could try to define finite copying in the input analogously to finite copying in the parameters—M is finite copying in the input if there exists 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. But if we evaluate $M(t)$ to a normal form, then each copy of u will also be evaluated, and there is no way to count the number of copies that were dynamically created during execution. The solution used is to extend M to a machine that works on trees where the subtree in question is replaced with a state P of the look-ahead automaton. To make this work, the look-ahead automaton must be extended so that it can process 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 nodes that were originally occurrence of u will be stuck as the transducer does not have rules for processing trees of the form $\langle q_i, p\rangle(s_1, \ldots, s_k)$. 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 iff there exists an integer k such that for every $t \in T_\Sigma$ with u a subtree, the length of the state sequence of t at u is less than or equal to 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 inpiut 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 4.1 (Engelfriet and Maneth [EM99])**.** *If an MTT M is globally finite copying in the input and linear in the parameters then M is finite copying in the input. Moreover, given an MTT that is finite copying in the parameters, one can effectively compute an equivalent MTT that is linear in the parameters.*

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 of this fact 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 4.1 to compute an MTT that is also linear in the parameters. By the same theorem, 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 MTT (the example is due to Engelfriet and Maneth [EM03]): $M = (Q, \Sigma, \Delta, q, R)$ with $Q = \{q^{(0)}, r^{(0)}\}$, $\Sigma = \{\sigma^{(1)}, \alpha^{(0)}, \beta^{(0)}\}$, $\Delta = \{\delta^{(1)}, \alpha^{(0)}, \beta^{(0)}\}$, and R is defined as follows:

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

M translates a linear trees into binary trees—e.g., $M(\sigma(\sigma(\sigma(\alpha))) = \delta(\alpha, \delta(\alpha, \delta(\alpha, \alpha))$—but it uses make many unecessary copies of the input tree in doing so. In particular, each subtree of the input is processed from the state r, but the rules for r discard its structure and extract the leaf. Intuitively r is redundant because it only ever produces one of two trees—α or β. 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. $M' = (\{q^{(0)}\}, \Sigma, \Delta, q, R, h, \{p_a, p_b\})$ where $h_a = p_a$, $h_b = p_b$, $h_\sigma = \lambda p.\, p$ and R is defined as follows:

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

In general, an MTT is called *input proper* iff every state q of the transducer produces infinitely many output trees (to be completely precise: it produces infinitely many trees belonging to L_p for every state p of the look-ahead automaton, where $\langle q, p\rangle$ occurs in $M_{q_0}(s)$ such that s contains p at some path). 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 $\langle q, p\rangle$ one can decide, using finiteness of range of an MTT [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 its elements. The final MTT is obtained by repeatedly replacing uses of these states with rules that use regular look-ahead to pick correct tree in each case and build that tree immediately.

The following theorem characterizes the input copying behavior of input proper MTTs of linear size increase:

Theorem 4.2 (Engelfriet and Maneth [EM03])**.** *If M is an input proper MTT that is also 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 4.3 (Engelfriet and Maneth [EM03])**.** *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 [EM03]): $M = (Q, \Sigma, \Delta, q, R)$ 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:

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

M copies the structure of 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 with a finite set of trees—the four nullary symbols in Δ. An equivalent MTT that is finite copying in the parameters can be obtained by eliminating y and using the state space of M to keep track of the correct value for the leaves. $M' = (\{q^{(0)}, q_\sigma^{(0)}, q_\gamma^{(0)}\}, \Sigma, \Delta, q, R)$ where R is defined as follows:

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

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

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 theorem:

Theorem 4.4 (Engelfriet and Maneth [EM03]). *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 4.5 (Engelfriet and Maneth [EM03]). *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 that uses parameter 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 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.

The main theorem states the equivalence of finite copying and MTTs of linear size increase.

Theorem 4.6 (Engelfriet and Maneth [EM03]). *Let f be the function on trees computed by an MTT M. The following are equivalent:*
(1) f is equivalent to an MSO-GT;
(2) f of linear size increase;
(3) f is equivalent to a finite copying MTT.

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; $(3) \Rightarrow (1)$ follows by the results discussed in Section 4.3.

5 Infinite Trees

The final result described in this survey, due to Colcombet and Löding, establishes the equivalence between bisimulation-preserving MSO-GTs over infinite trees and top-down determinstic tree transducers with regular look-ahead [CL04]. Many of the same techniques presented in previous sections are used to connect these two formalisms. For this reason, and to keep the presentation moving, the results in this section will be presented at a somewhat higher level than previous sections, highlighting the relevant similarities and differences. Also, to avoid defining even more notation, I will reuse the notation for ranked trees and top-down tree transducers that were defined in the previous section.

In each the previous cases studied in this survey, the class of MSO graph transductions was assumed fixed and the challenge was to find an equivalent finite-state transducer formalism for the given structure. The approach of Colcombet and Löding is somewhat different. They start with a transducer formalism—TT^{R}s operating on infinite terms—that is strictly less powerful than the class of MSO-GTs, and cut down the space of the latter by only considering ones that preserve bisimulation equivalence. With this restriction, every MSO-GT can be put into a normal form where each of the edges go "down", which makes it possible to simulate the transduction using a TT^R.

5.1 Terms and Unfoldings

A *term* over a ranked set is like a ranked tree, except that it may contain back edges. Every term is finite but the *unfolding* of a term t, written *unfold*(t), is a tree and may be infinite. The unfolding of a term is obtained by replacing each back edge with the unfolding of the term at the target of that edge. Two terms t and t' are *equivalent*, written $t \sim t'$ iff *unfold*$(t) =$ *unfold*(t'); equivalence also 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 *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 and if $t \sim t'$ then $M(t) \sim M(t')$.

For infinite trees the generalization of Büchi, Elgot, and Trakhtenbrot's theorem is due to Rabin. It equates the languages of ranked trees accepted by so-called Rabin automata and those definable by closed MSOformulas [Rab69].

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

The translation from a TT^R M operating on the unfoldings of terms to MSO-GTs operating on the terms themselves uses the same approach as in previous cases: a first MSO graph transduction constructs the computation space of M; a second transduction picks out the output tree from this graph. The copy index is the cardinality of the set of states, and the edges are specified by the rules of M. As with some rules in a 2DGSM, there are TT^R rules, such as the following, that consume part of the input but do not produce any output:

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

In the graph of the computation space, these edges are labeled with a special symbol ϵ that is surpressed by the second MSO graph transduction. Finally, as TT^{R}s preserve regular tree languages under inverse, and each regular tree language can be translated to an equivalent MSO formula, the conditions checked by uses of regular look-ahead in the rules of a TT^R can be converted into equivalent MSO formulas. The first theorem is as follows:

Theorem 5.1 (Colcombet and Löding [CL04])**.** *Let t be a term and M be a TT^R. There exists an MSO graph transduction N such that unfold$(N(t)) = M($unfold(t); moreover, N can be effectively computed from the definition of M.*

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

In each of the previous results equating a finite-state transducer formalism with MSO-GTs, the transducer has had way to examine the input structure multiple times: the read head on a 2DGSM can move in both directions on the input tape, a tree-walking transducer can perform an arbitrary walks on trees, values in an attributed tree transducers flow up and down, an MTT can delay the construction of the output tree by saving intermediate values in its parameters. These abilities are necessary for simulating the behavior of MSO-GTs, because a binary edge formulas may connect arbitrary vertices. By contrast,

a TT^R processes its input tree in a top-down fashion, and is a strictly finite state device. Because TT^{R}s are limited in this way, a restriction on MSO transductions is needed to ensure that a TT^R can construct the same edges as are specified in a MSO transduction.

The key result is that bisimulation-preserving MSO graph transductions 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 over a *tree-like structure* [Wal02] t^* from 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 had an edge in t. In this structure, it can be shown that every edge constructed by a bisimulation-preserving MSO graph transduction goes down. Then, using a theorem due to Walukiewicz, each edge formula can be translated to an equivalent formula over the original structure that also goes down. In this way, they obtain the desired normal form for the MSO-GT.

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 each variable appearing in the edge formulas, and then obtain a Rabin automaton A such that A accepts a tree marked with those nodes iff the original formula is satisfied at the same nodes. At each step, the TT^R uses its regular look-ahead to select the Rabin automaton corresponding to the next edge, and then simulates the behavior of that automaton on the input tree until it has constructed the edge. The second theorem is as follows:

Theorem 5.2 (Colcombet and Löding [CL04])**.** *Let t be a term and N be a MSO-GT. There exists a TT^R M such that unfold$(N(t)) = M($unfold(t); moreover, M can be effectively computed from the definition of N.*

6 Non-Deterministic Transductions

As was first shown by Courcelle, the the basic framework of MSO graph transductions can be extended with non-determinism. In a non-deterministic MSO-GT each of the vertex formulas in Φ and edge formulas in Ψ are extended with a fixed set of set variables $X_1, \ldots, X_K$. Two graphs are in 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. In this final technical section, I will briefly describe the extension of the results from Section 3 to non-deterministic MSO graph transductions. The material is drawn from the last section of Engelfriet and Hoogeboom's paper [EH01].

One might hope that the equivalence of 2DGSMs and MSO graph transductions would carry over to the non-deterministic case as well, but the two formalisms are actually incomparable. For example, the transduction mapping a^n to a^{nm} cannot be computed by a non-deterministic MSO-GT because its size is not of linear size increase. Conversely, non-deterministic two-way generalized sequential machines (2NGSMs) are not closed under composition but non-deterministic MSO-GTs are. If, however, we restrict 2NGSMs by placing an upper bound on the number of times that they can visit each position on the input tape, then we can obtain the equivalence of non-determinstic MSO graph transductions

with 2-ary compositions of 2NGSMs. Both directions of the inclusion can be shown by decomposing each transduction into a non-deterministic relabeling followed by a determinstic transduction.

7 Discussion

In this survey, we have examined several cases where finite-state transducer formalisms and MSO-GTs can be shown equivalent. During the course of this study, we have seen several techniques for bridging the gap between the local scope of a finite-state device and the global scope of interpretation of a formula on a structure, and for limiting the size of the output of a transducer. Both of these observations suggest possible directions for future work. On the logic side, it would be interesting to study the effects of limiting the scope of formulas in some way. For example, transductions described using finite-variable logics would be limited to local investigations of structures, which might match up better with less powerful transducer formalisms. Are their natural classes of transducers corresponding to first-order graph transductions? Conversely, one could try to enhance MSO definable transductions to obtain a formalisms where the size of the output graph is not linearly bounded by the size of the input graphs—e.g., is there a way to specify transductions that are of polynomial-size increase? Are there are regular-expression syntax for transductions? Also new transducer formalisms—e.g., with pebbles, so-called query automata—new structures—unranked trees, nested words, etc.—and non-determinism all deserve further study.

References

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

[B̈60] J. Richard Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für Mathematik, Logik und Grund lagen der Mathematik*, 6:66–92, 1960.

[B̈62] 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.

[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.

[EM03] 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.

[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.