Streaming Tree Transducers*

Rajeev Alur and Loris D'Antoni

University of Pennsylvania

Abstract. Theory of tree transducers provides a foundation for understanding expressiveness and complexity of analysis problems for specification languages for transforming hierarchically structured data such as XML documents. We introduce *streaming tree transducers* as an analyzable, executable, and expressive model for transforming unranked ordered trees (and hedges) in a single pass. Given a linear encoding of the input tree, the transducer makes a single left-to-right pass through the input, and computes the output using a finite-state control, a visibly pushdown stack, and a finite number of variables that store output chunks that can be combined using the operations of string-concatenation and tree-insertion. We prove that the expressiveness of the model coincides with transductions definable using monadic second-order logic (MSO). We establish complexity upper bounds of ExpTime for *type-checking* and NExpTime for checking *functional equivalence* for our model. We consider variations of the basic model when inputs/outputs are restricted to strings and ranked trees, and in particular, present the model of *bottom-up ranked-tree transducers*, which is the first known MSO-equivalent transducer model that processes trees in a bottom-up manner.

1 Introduction

Finite-state machines and logics for specifying tree transformations offer a suitable theoretical foundation for studying expressiveness and complexity of analysis problems for languages for processing and transforming XML documents. Representative formalisms for specifying tree transductions include finite-state top-down and bottom-up tree transducers, Macro tree transducers (MTT), attribute grammars, MSO (monadic second-order logic) definable graph transductions, and specialized programming languages such as XSLT and XDuce [1–8].

In this paper, we propose the model of *streaming tree transducers* (STT) which has the following three properties: (1) *Single-pass processing:* an STT is a deterministic machine that computes the output using a single left-to-right pass through the linear encoding of the input tree; (2) *Expressiveness:* STTs specify exactly the class of MSO-definable transductions; and (3) *Analyzability:* decision problems such as type checking and checking functional equivalence of two STTs, are decidable. The last two features indicate that our model has the commonly accepted trade-off between analyzability and expressiveness in formal

* A more detailed version of this paper is available at `http://www.cis.upenn.edu/~alur/stt12.pdf`.

language theory. The motivation for designing streaming algorithms that can process a document in a single pass has led to streaming models for checking membership in a regular tree language and for querying [9, 10, 5, 11], but the problem of computing all MSO-definable transformations in a single pass has not been studied previously.

The transducer model integrates features of *visibly pushdown automata*, equivalently *nested word automata* [12], and *streaming string transducers* [13, 14]. In our model, the input tree is encoded as a *nested word*, which is a string over alphabet symbols, tagged with open/close brackets (or equivalently, call/return types) to indicate the hierarchical structure [9, 12]. The streaming tree transducer reads the input nested word left-to-right in a single pass. It uses finitely many states, together with a stack, but the type of operation applied to the stack at each step is determined by the hierarchical structure of the tags in the input. The output is computed using a finite set of variables that range over output nested words, possibly with *holes* that are used as place-holders for inserting subtrees. At each step, the transducer reads the next symbol of the input. If the symbol is an internal symbol, then the transducer updates its state and the output variables. If the symbol is a call symbol, then the transducer pushes a stack symbol, along with updated values of variables, updates the state, and reinitializes the variables. While processing a return symbol, the stack is popped, and the new state and new values for the variables are determined using the current state, current variables, popped symbol, and popped values from the stack. In each type of transition, the variables are updated using expressions that allow adding new symbols, *string concatenation*, and *tree insertion* (simulated by replacing the hole with another expression). A key restriction is that variables are updated in a manner that ensures that each value can contribute at most once to the eventual output, without duplication. This *single-use-restriction* is enforced via a binary *conflict* relation over variables: no output term combines conflicting variables, and variable occurrences in right-hand sides during each update are consistent with the conflict relation. The transformation computed by the model can be implemented as a single-pass linear-time algorithm.

We show that the model can be simplified in natural ways if we want to restrict either the input or the output, to either strings or ranked trees. For example, to compute transformations that output strings it suffices to consider variable updates that allow only concatenation, and to compute transformations that output ranked trees it suffices to consider variable updates that allow only tree insertion. The restriction to the case of ranked trees as inputs gives the model of *bottom-up ranked-tree transducers*. As far as we know, this is the only transducer model that processes trees in a bottom-up manner, and can compute all MSO-definable transformations.

The main technical result in the paper is that the class of transductions definable using streaming tree transducers is exactly the class of MSO-definable transductions. The starting point for our result is the known equivalence of MSO-definable transductions and Macro Tree Transducers with regular look-ahead and single-use restriction, over ranked trees [3]. Our proof proceeds by

establishing two key properties of STTs: the model is closed under *regular look ahead* and under *functional composition*. These proofs are challenging due to the requirement that a transducer can use only a fixed number of variables that can be updated by assignments that obey the single-use-restriction rules.

While decidability for a variety of analysis questions for our transducer model follows from corresponding results for MSO-definable transformations [15], we are interested in understanding the complexity bounds for a variety of analysis questions. Given a regular language L_1 of input trees and a regular language L_2 of output trees, the *type checking* problem is to determine if the output of the transducer on an input in L_1 is guaranteed to be in L_2. We establish an ExpTime upper bound on type checking. For checking functional equivalence of two streaming tree transducers, we show that if the two transducers are inequivalent, then we can construct a pushdown automaton A over the alphabet $\{0, 1\}$ such that A accepts a word with equal number of 0's and 1's exactly when there is an input on which the two transducers compute different outputs. Using known techniques for computing the Parikh images of context-free languages [15–17], this leads to a NExpTime upper bound for checking functional inequivalence of two STTs. Assuming a bounded number of variables, the upper bound on the parametric complexity becomes NP. Improving the NExpTime bound remains a challenging open problem.

2 Transducer Model

Nested Words: Data with both linear and hierarchical structure can be encoded using nested words [12]. Given a set Σ of symbols, the *tagged alphabet* $\hat{\Sigma}$ consists of the symbols a, $\langle a$, and $a\rangle$, for each $a \in \Sigma$. A *nested word* over Σ is a finite sequence over $\hat{\Sigma}$. For a nested word $a_1 \cdots a_k$, a position j, for $1 \leq j \leq k$, is said to be a *call* position if the symbol a_j is of the form $\langle a$, a *return* position if the symbol a_j is of the form $a\rangle$, and an *internal* position otherwise. The tags induce a natural matching relation between call and return positions, and in this paper, we are interested only in *well-matched* nested words in which all calls/returns have matching returns/calls. A string over Σ is a nested word with only internal positions. Nested words naturally encode ordered trees. The empty tree is encoded by the empty string ε. The tree with a-labeled root with subtrees $t_1, \ldots t_k$ as children, in that order, is encoded by the nested word $\langle a \langle\!\langle t_1 \rangle\!\rangle \cdots \langle\!\langle t_k \rangle\!\rangle a\rangle$, where $\langle\!\langle t_i \rangle\!\rangle$ is the encoding of the subtree t_i. This transformation can be viewed as an inorder traversal of the tree. The encoding extends to *hedges* also.

Nested Words with Holes: A key operation that our transducer model relies on is *insertion* of one nested word within another. In order to define this, we consider nested words with holes, where a hole is represented by the special symbol ?. We require that a nested word can contain at most one hole, and we use a binary type to keep track of whether a nested word contains a hole or not. A type-0 nested word does not contain any holes, while a type-1 nested word contains one hole. We can view a type-1 nested word as a unary function

from nested words to nested words. The set $W_0(\Sigma)$ of type-0 nested words over the alphabet Σ is defined by the grammar $W_0 := \varepsilon \mid a \mid \langle a\, W_0\, b\rangle \mid W_0\, W_0$, for $a, b \in \Sigma$. The set $W_1(\Sigma)$ of type-1 nested words over the alphabet Σ is defined by the grammar $W_1 := ? \mid \langle a\, W_1\, b\rangle \mid W_1\, W_0 \mid W_0\, W_1$, for $a, b \in \Sigma$. A *nested-word language* over Σ is a subset L of $W_0(\Sigma)$, and a *nested-word transduction* from an input alphabet Σ to an output alphabet Γ is a *partial* function f from $W_0(\Sigma)$ to $W_0(\Gamma)$.

Nested Word Expressions: In our transducer model, the machine maintains a set of variables that range over output nested words with holes. Each variable has an associated binary type: a type-k variable has type-k nested words as values, for $k = 0, 1$. The variables are updated using typed expressions, where variables can appear on the right-hand side, and we also allow substitution of the hole symbol by another expression. Formally, a set X of typed variables is a set that is partitioned into two sets X_0 and X_1 corresponding to the type-0 and type-1 variables. Given an alphabet Σ and a set X of typed variables, a *valuation* α is a function that maps X_0 to $W_0(\Sigma)$ and X_1 to $W_1(\Sigma)$. Given an alphabet Σ and a set X of typed variables, we define the sets $E_k(X, \Sigma)$, for $k = 0, 1$, of type-k expressions by the grammars: $E_0 := \varepsilon \mid a \mid x_0 \mid \langle a\, E_0\, b\rangle \mid E_0\, E_0 \mid E_1[E_0]$ and $E_1 :=? \mid x_1 \mid \langle a\, E_1\, b\rangle \mid E_0\, E_1 \mid E_1\, E_0 \mid E_1[E_1]$, where $a, b \in \Sigma$, $x_0 \in X_0$ and $x_1 \in X_1$. The clause $e[e']$ corresponds to substitution of the hole in a type-1 expression e by another expression e'. A valuation α for the variables X naturally extends to a type-consistent function that maps the expressions $E_k(X, \Sigma)$ to values in $W_k(\Sigma)$, for $k = 0, 1$.

Single Use Restriction: The transducer updates variables X using type-consistent assignments. To achieve the desired expressiveness, we need to restrict the reuse of variables in right-hand sides. In particular, we want to disallow the assignment $x := xx$ (which would double the length of x), but allow the assignment $(x, y) := (x, x)$, provided the variables x and y are guaranteed not to be combined later. For this purpose, we assume that the set X of variables is equipped with a binary relation η: if $\eta(x, y)$, then x and y cannot be combined. This "conflict" relation is required to be reflexive and symmetric (but need not be transitive). Two conflicting variables cannot occur in the same expression used in the right-hand side of an update or as an output. During an update, two conflicting variables can occur in multiple right-hand sides for updating conflicting variables. Thus, the assignment $(x, y) := (\langle a\, xa\rangle[y], a?)$ is allowed, provided $\eta(x, y)$ does not hold; the assignment $(x, y) := (ax[y], y)$ is not allowed; and the assignment $(x, y) := (ax, x[b])$ is allowed, provided $\eta(x, y)$ holds. Formally, given a set X of typed variables with a reflexive symmetric binary conflict relation η, and an alphabet Σ, an expression e in $E(X, \Sigma)$ is said to be *consistent* with η, if (1) each variable x occurs at most once in e, and (2) if $\eta(x, y)$ holds, then e does not contain both x and y. Given sets X and Y of typed variables, a conflict relation η, and an alphabet Σ, a *single-use-restricted assignment* is a function ρ that maps each type-k variable x in X to a right-hand side expression in $E_k(Y, \Sigma)$, for $k = 0, 1$, such that (1) each expression $\rho(x)$ is consistent with η, and (2)

if $\eta(x, y)$ holds, and $\rho(x')$ contains x, and $\rho(y')$ contains y, then $\eta(x', y')$ must hold. The set of such single-use-restricted assignments is denoted $\mathcal{A}(X, Y, \eta, \Sigma)$.

At a return, the transducer assigns the values to its variables X using the values popped from the stack as well as the values returned. For each variable x, we will use x_p to refer to the popped value of x. Thus, each variable x is updated using an expression over the variables $X \cup X_p$. The conflict relation η extends naturally to variables in X_p: $\eta(x_p, y_p)$ holds exactly when $\eta(x, y)$ holds. Then, the update at a return is specified by assignments in $\mathcal{A}(X, X \cup X_p, \eta, \Sigma)$.

When the conflict relation η is the purely reflexive relation $\{(x, x) \mid x \in X\}$, the single-use-restriction means that a variable x can appear at most once in at most one right-hand side. We refer to this special case as "copyless".

STT syntax: A *deterministic streaming tree transducer* (STT) S from input alphabet Σ to output alphabet Γ consists of a finite set of states Q; a finite set of stack symbols P; an initial state $q_0 \in Q$; a finite set of typed variables X with a reflexive symmetric binary conflict relation η; a partial output function $F : Q \mapsto E_0(X, \Gamma)$ such that each expression $F(q)$ is consistent with η; an internal state-transition function $\delta_i : Q \times \Sigma \mapsto Q$; a call state-transition function $\delta_c : Q \times \Sigma \mapsto Q \times P$; a return state-transition function $\delta_r : Q \times P \times \Sigma \mapsto Q$; an internal variable-update function $\rho_i : Q \times \Sigma \mapsto \mathcal{A}(X, X, \eta, \Gamma)$; a call variable-update function $\rho_c : Q \times \Sigma \mapsto \mathcal{A}(X, X, \eta, \Gamma)$; and a return variable-update function $\rho_r : Q \times P \times \Sigma \mapsto \mathcal{A}(X, X \cup X_p, \eta, \Gamma)$. An STT S with variables X is called *copyless* if the conflict relation η equals $\{(x, x) \mid x \in X\}$.

STT semantics: To define the semantics of a streaming tree transducer, we consider configurations of the form (q, Λ, α), where $q \in Q$ is a state, α is a type-consistent valuation from variables X to typed nested words over Γ, and Λ is a sequence of pairs (p, β) such that $p \in P$ is a stack symbol and β is a type-consistent valuation from variables in X to typed nested words over Γ. The initial configuration is $(q_0, \varepsilon, \alpha_0)$ where α_0 maps each type-0 variable to ε and each type-1 variable to ?. The transition function δ over configurations is defined by:

1. **Internal transitions:** $\delta((q, \Lambda, \alpha), a) = (\delta_i(q, a), \Lambda, \alpha \cdot \rho_i(q, a))$.
2. **Call transitions:** $\delta((q, \Lambda, \alpha), \langle a) = (q', (p, \alpha \cdot \rho_c(q, a))\Lambda, \alpha_0)$, where $\delta_c(q, a) = (q', p)$.
3. **Return transitions:** $\delta((q, (p, \beta)\Lambda, \alpha), a\rangle) = (\delta_r(q, p, a), \Lambda, \alpha \cdot \beta_p \cdot \rho_r(q, p, a))$, where β_p is the valuation for variables X_p defined by $\beta_p(x_p) = \beta(x)$ for $x \in X$.

For an input word $w \in W_0(\Sigma)$, if $\delta^*((q_0, \varepsilon, \alpha_0), w) = (q, \varepsilon, \alpha)$ then if $F(q)$ is undefined then so is $[\![S]\!](w)$, otherwise $[\![S]\!](w) = \alpha(F(q))$. We say that a nested word transduction f from input alphabet Σ to output alphabet Γ is *STT-definable* if there exists an STT S such that $[\![S]\!] = f$.

Example: tree swap: Streaming tree transducers can easily implement standard tree-edit operations such as insertion, deletion, and relabeling. We show

how swapping can be implemented as a copyless STT that mirrors the corresponding natural single-pass algorithm. Figure **??** shows the transduction that transforms the input tree by swapping the first (in inorder traversal) b-rooted subtree t_1 with the next (in inorder traversal) b-rooted subtree t_2, not contained in t_1, For clarity of presentation, let us assume that the input word encodes a tree: it does not contain any internal symbols and if a call position is labeled $\langle a$ then its matching return is labeled $a\rangle$.

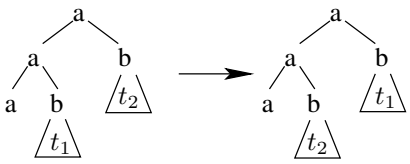

Fig. 1. Tree Swap

The initial state is q_0 which means that the transducer has not yet encountered a b-label. In state q_0, the STT records the tree traversed so far using a type-0 variable x: upon an a-labeled call, x is stored on the stack, and is reset to ε; and upon an a-labeled return, x is updated to $x_p\langle a\, x\, a\rangle$. In state q_0, upon a b-labeled call, the STT pushes q_0 along with current x on the stack, resets x to ε, and updates its state to q'. In state q', the STT constructs the first b-labeled subtree t_1 in variable x: as long as it does not pop stack symbol q_0, at a call it pushes q' and x, and at a return, updates x to $x_p\langle a\, x\, a\rangle$ or $x_p\langle b\, x\, b\rangle$, depending on whether the current return symbol is a or b. When it pops q_0, it updates x to $\langle b\, x\, b\rangle$ (at this point, x contains the tree t_1, and its value will be propagated), sets another type-1 variable x' to $x_p\,?$, and changes its state to q_1. In state q_1, the STT is searching for the next b-labeled call, and processes a-labeled calls and returns exactly as in state q_0, but now using the type-1 variable x'. At a b-labeled call, it pushes q_1 along with x' on the stack, resets x to ε, and updates the state to q'. Now in state q', the STT constructs the second b-labeled subtree t_2 in variable x as before. When it pops q_1, the subtree t_2 corresponds to $\langle b\, x\, b\rangle$. The transducer updates x to $x'_p[\langle b\, x\, b\rangle]x_p$ capturing the desired swapping of the two subtrees t_1 and t_2 (the variable x' is no longer needed and is reset to ε to ensure copyless restriction), and switches to state q_2. In state q_2, the remainder of the tree is traversed adding it to x. The output function is defined only for the state q_2 and maps q_2 to x.

3 Properties and Variants

In this section, we note some properties and variants of streaming tree transducers aimed at understanding their expressiveness. First, STTs compute *linearly-bounded* outputs, that is, the length of the output word is within at most a constant factor of the length of the input word. The single-use-restriction ensures that at every step of the execution of the transducer on an input word, the sum of the sizes of all the variables that contribute to the output term at the end of the execution, can increase only by an additive constant.

Regular Nested-Word Languages: A streaming tree transducer with empty sets of string variables can be viewed as an *acceptor* of nested words: the input is accepted if the output function is defined in the terminal state, and rejected

otherwise. In this case, the definition coincides with (deterministic) nested word automata (NWA). A language $L \subseteq W_0(\Sigma)$ of nested words is *regular* if it is accepted by such an automaton. This class includes all regular word languages, regular tree languages, and is a subset of deterministic context-free languages [12]. Given a nested-word transduction f from input alphabet Σ to output alphabet Γ, the *domain* of f is the set $Dom(f) \subseteq W_0(\Sigma)$ of input nested words w for which $f(w)$ is defined, and the *image* of f is the set $Img(f) \subseteq W_0(\Gamma)$ of output nested words w' such that $w' = f(w)$ for some w. It is easy to establish that: for an STT-definable transduction f from Σ to Γ, $Dom(f)$ is a regular language of nested words over Σ, and there exists an STT-definable transduction f from Σ to Γ, such that $Img(f)$ is not a regular language of nested words over Γ.

Bottom-up Transducers: A nested-word automaton is called *bottom-up* if it resets its state along the call transition: if $\delta_c(q, a) = (q', p)$ then $q' = q_0$. The well-matched nested word sandwiched between a call and its matching return is processed by a bottom-up NWA independent of the outer context. It is known that bottom-up NWAs are as expressive as NWAs over well-matched words [12]. We show that a similar result holds for transducers also: there is no loss of expressiveness if the STT is disallowed to propagate information at a call to the linear successor. Note than every STT reinitializes all its variables at a call. An STT S is said to be a *bottom-up STT* if for every state $q \in Q$ and symbol $a \in \Sigma$, if $\delta_c(q, a) = (q', p)$ then $q' = q_0$. We can prove that: every STT-definable transduction is definable by a bottom-up STT.

Regular Look Ahead: Now we consider an extension of the STT model in which the transducer can make its decisions based on whether the remaining (well-matched) suffix of the input word belongs to a regular language of nested words. Such a test is called *regular look ahead*. Given a nested-word $w = a_1 a_2 \dots a_k$, for each position $1 \leq i \leq k$, let $\textsc{wms}(w, i)$ be the (well-matched) nested word $a_i, \dots a_j$, where j is the maximal index l such that $a_i, \dots a_l$ is well-matched. Then, a look-ahead test at step i can test a regular property of the word $\textsc{wms}(w, i)$. Let L be a regular language of nested words, and let A be a (deterministic) bottom-up NWA for $reverse(L)$. Then, while processing a nested word, testing whether the word $\textsc{wms}(w, i)$ belongs to L corresponds to testing whether the state of A after processing $reverse(\textsc{wms}(w, i))$ is an accepting state of A. Since regular languages of nested words are closed under intersection, the state of a single bottom-up NWA A reading the input word in reverse can be used to test membership of the well-matched suffix at each step in different languages. This motivates the following formalization. Let $w = a_1 \dots a_k$ be a nested word over Σ, and let A be a bottom-up NWA with states R processing nested words over Σ. Given a state $r \in R$, we define the r-look-ahead labeling of w to be the nested word $w_r = r_1 r_2 \dots r_k$ over the alphabet R such that for each position $1 \leq j \leq k$, the call/return/internal type of r_j is the same as the type of a_j, and the corresponding symbol is the state of the NWA A after reading $reverse(a_j \dots a_k)$ starting in state r. Then the *A-look-ahead labeling of w*, is the nested word $w_A = w_{r_0}$. An *STT-with-regular-look-ahead* consists of a bottom-up

NWA A over Σ with states R, and an STT S from R to Γ. Such a transducer defines a streaming tree transduction from Σ to Γ: for an input word $w \in W(\Sigma)$, the output $[\![S, A]\!](w)$ is defined to be $[\![S]\!](w_A)$. The critical closure property for STTs is captured by the following result: the transductions definable by STTs with regular look-ahead are STT-definable. For the proof, given an NWA A with states R, and a bottom-up STT S over R, we construct an equivalent STT S'. Intuitively, at every step, for every possible look-ahead type $r \in R$ of the suffix, S' needs to maintain a copy of the state and variables of S. Updating these copies consistently while obeying single-use-restriction is the challenging part of the construction. To simplify the proof, we define an intermediate equivalent model of STTs, *multi-parameter STTs*, in which values of variables can contain multiple parameters.

Copyless STTs with RLA: Recall that an STT is said to be copyless if η only contains the reflexive relation. In an STT, an assignment of the form $(x, y) := (z, z)$ is allowed if x and y are guaranteed not to be combined, and thus, if only one of x and y contributes to the final output. In presence of regular-look-ahead test, the STT can check which variables contribute to the final output, and avoid redundant updates, and can thus be copyless: a nested-word transduction f is STT-definable iff it is definable by a copyless STT with regular-look-ahead.

Closure Under Composition: Many of our results rely on the crucial property that STTs are closed under sequential composition: given two STT-definable transductions, f_1 from Σ_1 to Σ_2 and f_2 from Σ_2 to Σ_3, the composite transduction $f_2 \cdot f_1$ from Σ_1 to Σ_3 is STT-definable. In the constructive proof, given a copyless STT S_1 with regular-look-ahead, and a bottom-up STT S_2, we construct a multi-parameter STT S_3 with regular-look-ahead. The core idea of the construction is that S_3 maintains all possible "summaries" of executions of S_2 on each nested word stored in a variable of S_1.

Mapping Strings: A nested word captures both linear and hierarchical structure. There are two natural classes of nested words: strings are nested words with only linear structure, and ranked trees are nested words with only hierarchical structure. A transducer without a stack cannot use the hierarchical structure of the input nested word, and processes the input as a string of symbols. Since the transducer does not interpret the tags in a special manner, we will assume that the input is a string over Σ. This restricted transducer can still map strings to nested words (or trees) over Γ with interesting hierarchical structure, and hence, is called a *string-to-tree* transducer. This leads to the following definition: a *streaming string-to-tree transducer* (SSTT) S from input alphabet Σ to output alphabet Γ consists of a finite set of states Q; an initial state $q_0 \in Q$; a finite set of typed variables X; a partial output function $F : Q \mapsto E_0(X, \Gamma)$ such that for each state q, a variable x appears at most once in $F(q)$; a state-transition function $\delta_i : Q \times \Sigma \mapsto Q$; and a variable-update function $\rho_i : Q \times \Sigma \mapsto \mathcal{A}(X, X, \Gamma)$. Configurations of such a transducer are of the form (q, α), where $q \in Q$ is a state, and α is a type-consistent valuation for the variables X. The semantics

$[\![S]\!]$ of such a transducer is a partial function from Σ^* to $W_0(\Gamma)$. In this case, the copyless restriction is enough to capture MSO completeness due to the following closure property: the transductions definable by SSTTs with regular look-ahead are SSTT-definable.

Mapping Ranked Trees: We consider how the definition can be simplified for the case of inputs restricted to ranked trees. The set $B(\Sigma)$ of *binary trees* over the alphabet Σ is then the subset of nested words defined by the grammar $T := \mathbf{0} \mid \langle a\, T\, T\, a\rangle$, for $a \in \Sigma$. The definition of an STT can be simplified in the following way if we know that the input is a binary tree. First, we do not need to worry about processing of internal symbols. Second, we restrict to bottom-up STTs due to their similarity to bottom-up tree transducers, where the transducer returns, along with the state, values for variables ranging over output nested words, as a result of processing a subtree. Finally, at a call, we know that there are exactly two subtrees, and hence, the propagation of information across matching calls and returns using a stack can be combined into a unified combinator: the transition function computes the result corresponding to a tree $a\langle t_l\, t_r\rangle$ based on the symbol a, and the results of processing subtrees t_l and t_r.

A *bottom-up ranked-tree transducer* (BRTT) S from binary trees over Σ to nested words over Γ consists of a finite set of states Q; an initial state $q_0 \in Q$; a finite set of typed variables X equipped with a conflict relation η; a partial output function $F : Q \mapsto E_0(X, \Gamma)$ such that for each state q, the expression $F(q)$ is consistent with η; a state-combinator function $\delta : Q \times Q \times \Sigma \mapsto Q$; and a variable-combinator function $\rho : Q \times Q \times \Sigma \mapsto \mathcal{A}(X_l \cup X_r, X, \eta, \Gamma)$, where X_l denotes the set of variables $\{x_l \mid x \in X\}$, X_r denotes the set of variables $\{x_r \mid x \in X\}$, and conflict relation η extends to these sets naturally. The state-combinator extends to trees in $B(\Sigma)$: $\delta^*(\mathbf{0}) = q_0$ and $\delta^*(a\langle t_l\, t_r\rangle) = \delta(\delta^*(t_l), \delta^*(t_r), a)$. The variable-combinator is used to map trees to valuations for X: $\alpha^*(\mathbf{0}) = \alpha_0$, where α_0 maps each type-0 variable to ε and each type-1 variable to ?, and $\alpha^*(a\langle t_l\, t_r\rangle) = \rho(\delta^*(t_l), \delta^*(t_r), a)[X_l \mapsto \alpha^*(t_l)][X_r \mapsto \alpha^*(t_r)]$. Given a tree $t \in B(\Sigma)$, let $\delta^*(t)$ be q and let $\alpha^*(t)$ be α. Then, if $F(q)$ is undefined then $[\![S]\!](t)$ is undefined, else $[\![S]\!](t)$ equals $\alpha(F(q))$ obtained by evaluating the expression $F(q)$ according to valuation α. We then prove that: a partial function from $B(\Sigma)$ to $W_0(\Gamma)$ is STT-definable iff it is BRTT-definable.

4 Expressiveness

The goal of this section is to prove that the class of nested-word transductions definable by STTs coincides with the class of transductions definable using Monadic Second Order logic (MSO). Our proof relies on the known equivalence between MSO and Macro Tree Transducers over *ranked trees* [3].

MSO for Nested Word Transductions: Formulas in monadic second-order logic (MSO) can be used to define functions from (labeled) graphs to graphs [2]. We adapt this general definition for our purpose of defining transductions over nested words. By adapting the simulation of string transducers by MSO [18, 13],

we show that the computation of an STT can be encoded by MSO, and thus, every transduction computable by an STT is MSO definable.

Nested Words as Binary Trees: Nested words can be encoded as binary trees. This encoding is analogous to encoding of *unranked* trees as binary trees. Such an encoding increases the depth of the tree by imposing unnecessary hierarchical structure, and thus, is not suitable for processing of inputs. However, it is useful to simplify proofs of subsequent results about expressiveness. The desired transduction nw_bt from $W_0(\Sigma)$ to $B(\Sigma)$ is defined by $nw\_bt(\varepsilon) = \mathbf{0}$; $nw\_bt(aw) = a\langle\, nw\_bt(w)\, \mathbf{0}\,\rangle$; and $nw\_bt(\langle a\, w_1\, b\rangle\, w_2) = a\langle\, nw\_bt(w_1)\, b\langle\, nw\_bt(w_2)\, \mathbf{0}\,\rangle\,\rangle$. Observe that nw_bt is a one-to-one function. We can define the inverse partial function bt_nw from binary trees to nested words as follows: given $t \in B(\Sigma)$, if t equals $nw\_bt(w)$, for some $w \in W_0(\Sigma)$ (and if so, the choice of w is unique), then $bt\_nw(t) = w$, and otherwise $bt\_nw(t)$ is undefined. The next proposition shows that both these mappings can be implemented as STTs: the transductions $nw\_bt : W_0(\Sigma) \mapsto B(\Sigma)$ and $bt\_nw : B(\Sigma) \mapsto W_0(\Sigma)$ are STT-definable.

For a nested-word transduction f from $W_0(\Sigma)$ to $W_0(\Gamma)$, we can define another transduction $\tilde{f}$ that maps binary trees over Σ to binary trees over Γ: given a binary tree $t \in B(\Sigma)$, if t equals $nw\_bt(w)$, then $\tilde{f}(t) = nw\_bt(f(w))$, and otherwise $\tilde{f}(t)$ is undefined. The following proposition can be proved easily from the definitions of the encodings: if f is an MSO-definable transduction from $W_0(\Sigma)$ to $W_0(\Gamma)$, then the transduction $\tilde{f} : B(\Sigma) \mapsto B(\Gamma)$ is an MSO-definable binary-tree transduction and $f = bt\_nw \cdot \tilde{f} \cdot nw\_bt$. Since STT-definable transductions are closed under composition, to establish that every MSO-definable transduction is STT-definable, it suffices to consider MSO-definable transductions from binary trees to binary trees.

Macro Tree Transducers: A Macro Tree Transducer (MTT) [4, 3] is a tree transducer in which the translation of a tree may not only depend on its subtrees but also on its context. While the subtrees are represented by input variables, the context information is handled by parameters. We only need to consider deterministic MTTs with regular look ahead (MTTR) that map binary trees to binary trees. In general, MTTs are more expressive than MSO. The restrictions needed to limit the expressiveness rely on the *single-use-restriction* for using the parameters (SURP) and *finite copying* restriction in processing of the input (FCI). The following theorem is proved in [3]: a ranked-tree transduction f is MSO-definable iff there exists an MTTR M with SURP/FCI such that $f = [\![M]\!]$.

MSO Equivalence: We first show that bottom-up ranked-tree transducers are as expressive as MTTs with regular-look-ahead, with single-use restriction for the parameters, and finite-copying restriction for the input: if a ranked-tree transduction $f : B(\Sigma) \mapsto B(\Gamma)$ is definable by an MTTR with SURP/FCI, then it is BRTT-definable. Now, we can put together all the results to obtain the main technical result of the paper:

Theorem 1 (MSO Equivalence). *A nested-word transduction $f : W_0(\Sigma) \mapsto W_0(\Gamma)$ is STT-definable iff it is MSO-definable.*

5 Decision Problems

In this section, we show that a number of analysis problems for our model are decidable.

Output Analysis: Given an input nested word w over Σ, and an STT S from Σ to Γ, consider the problem of computing the output $[\![S]\!](w)$. To implement the operations of the STT efficiently, we can store the nested words corresponding to variables in linked lists with reference variables pointing to positions that correspond to holes. To process each symbol in w, the copyless update of variables can be executed by changing only a constant number of pointers. This leads to: given an input nested word w and an STT S with k variables, the output word $[\![S]\!](w)$ can be computed in time $O(k|w|)$.

The second problem we consider corresponds to *type checking*: given regular languages L_{pre} and L_{post} of nested words over Σ, and an STT S from Σ to Γ, the type checking problem is to determine if $[\![S]\!](L_{pre}) \subseteq L_{post}$ (that is, if for every $w \in L_{pre}$, $[\![S]\!](w) \in L_{post}$). This form of type checking is useful in checking consistency of XML schemas. For STTs, type checking can be solved in ExpTime: given an STT S from Σ to Γ, an NWA A accepting nested words over Σ, and an NWA B accepting nested words over Γ, checking $[\![S]\!](L(A)) \subseteq L(B)$ is solvable in time $O(|A|^3 \cdot |S|^3 \cdot n^{kn^2})$ where n is the number of states of B, and k is the number of variables in S.

As noted earlier, the image of an STT is not necessarily regular. However, the pre-image of a given regular language is regular, and can be computed. Given an STT S from input alphabet Σ to output alphabet Γ, and a language $L \subseteq W_0(\Gamma)$ of output words, the set $PreImg(L, S)$ consists of input nested words w such that $[\![S]\!](w) \in L$. We show that: given an STT S from Σ to Γ, and an NWA B over Γ, there is an algorithm to compute an NWA A over Σ such that $L(A) = PreImg(L(B), S)$. It follows that given an STT S and a regular language L of output nested words, there is an ExpTime algorithm to test whether $Img(S) \cap L$ is non-empty.

Functional Equivalence: Finally, we consider the problem of checking *functional equivalence* of two STTs: given two streaming tree transducers S and S', we want to check if they define the same transduction. Given two *streaming string transducers* S and S', [13, 14] shows how to construct an NFA A over the alphabet $\{0, 1\}$ such that the two transducers are *inequivalent* exactly when A accepts some word w such that w has equal number of 0's and 1's. The idea can be adopted for the case of STTs, but A now will be a nondeterministic pushdown automaton. The size of A is polynomial in the number of states of the input STTs, but exponential in the number of variables of the STTs. Results in [17, 16] can be adopted to check whether this pushdown automaton accepts a word with the same number of 0's and 1's.

Theorem 2 (Checking Equivalence). *Given two STTs S and S', the problem of checking whether $[\![S]\!] \neq [\![S']\!]$ is solvable in* NExpTime.

If the number of variables is bounded, then the size of A is polynomial, and this gives an upper bound of NP. For the transducers that map strings to nested words, that is, for streaming string-to-tree transducers (SSTT), the above construction yields a PSPACE bound.

Acknowledgements: We thank Joost Engelfriet for his valuable feedback.

References

1. Comon, H., Dauchet, M., Gilleron, R., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications. Draft, Available at `http://www.grappa.univ-lille3.fr/tata/` (2002)
2. Courcelle, B.: Monadic second-order definable graph transductions: A survey. Theor. Comput. Sci. **126**(1) (1994) 53–75
3. Engelfriet, J., Maneth, S.: Macro tree transducers, attribute grammars, and MSO definable tree translations. Information and Computation **154** (1999) 34–91
4. Engelfriet, J., Vogler, H.: Macro tree transducers. J. Comput. System Sci. **31** (1985) 71–146
5. Milo, T., Suciu, D., Vianu, V.: Typechecking for XML transformers. In: Proceedings of the 19th ACM Symposium on PODS. (2000) 11–22
6. Hosoya, H., Pierce, B.C.: XDuce: A statically typed XML processing language. ACM Trans. Internet Techn. **3**(2) (2003) 117–148
7. Martens, W., Neven, F.: On the complexity of typechecking top-down XML transformations. Theor. Comput. Sci. **336**(1) (2005) 153–180
8. Hosoya, H.: Foundations of XML Processing: The Tree-Automata Approach. Cambridge University Press (2011)
9. Segoufin, L., Vianu, V.: Validating streaming XML documents. In: Proceedings of the 21st ACM Symposium on PODS. (2002) 53–64
10. Neven, F., Schwentick, T.: Query automata over finite trees. Theor. Comput. Sci. **275**(1-2) (2002) 633–674
11. Madhusudan, P., Viswanathan, M.: Query automata for nested words. In: Mathematical Foundations of Computer Science 2009, 34th International Symposium. LNCS 5734 (2009) 561–573
12. Alur, R., Madhusudan, P.: Adding nesting structure to words. Journal of the ACM **56**(3) (2009)
13. Alur, R., Cerný, P.: Expressiveness of streaming string transducers. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science. LIPIcs 8 (2010) 1–12
14. Alur, R., Cerný, P.: Streaming transducers for algorithmic verification of single-pass list-processing programs. In: Proceedings of 38th ACM Symposium on POPL. (2011) 599–610
15. Engelfriet, J., Maneth, S.: The equivalence problem for deterministic MSO tree transducers is decidable. Inf. Process. Lett. **100**(5) (2006) 206–212
16. Seidl, H., Schwentick, T., Muscholl, A., Habermehl, P.: Counting in trees for free. In: Automata, Languages and Programming: 31st International Colloquium. LNCS 3142 (2004) 1136–1149
17. Esparza, J.: Petri nets, commutative context-free grammars, and basic parallel processes. Fundam. Inform. **31**(1) (1997) 13–25
18. Engelfriet, J., Hoogeboom, H.: MSO definable string transductions and two-way finite-state transducers. ACM Trans. Comput. Log. **2**(2) (2001) 216–254