The Complexity of Text-Preserving XML Transformations*

Timos Antonopoulos
Hasselt University and
Transnational University of
Limburg
timos.antonopoulos@uhasselt.be

Wim Martens†
TU Dortmund
wim.martens@udo.edu

Frank Neven
Hasselt University and
Transnational University of
Limburg
frank.neven@uhasselt.be

ABSTRACT

While XML is nowadays adopted as the de facto standard for data exchange, historically, its predecessor SGML was invented for describing electronic documents, i.e., marked-up text. Actually, today there are still large volumes of such XML texts. We consider simple transformations which can change the internal structure of documents, that is, the mark-up, and can filter out parts of the text but do not disrupt the ordering of the words. Specifically, we focus on XML transformations where the transformed document is a subsequence of the input document when ignoring mark-up. We call the latter *text-preserving* XML transformations. We characterize such transformations as copy- and rearrange-free transductions. Furthermore, we study the problem of deciding whether a given XML transducer is text-preserving over a given tree language. We consider top-down transducers as well as the abstraction of XSLT called DTL. We show that deciding whether a transformation is text-preserving over an unranked regular tree language is in PTIME for top-down transducers, EXPTIME-complete for DTL with XPath, and decidable for DTL with MSO patterns. Finally, we obtain that for every transducer in one of the above mentioned classes, the maximal subset of the input schema can be computed on which the transformation is text-preserving.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Languages—*Data manipulation languages (DML)*; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages; D.2.4 [**Software Engineering**]: Software/Program Verification

*We acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599.

†Supported by a grant of the North-Rhine Westfalian Academy of Sciences and Arts, and the Stiftung Mercator Essen.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
PODS'11, June 13–15, 2011, Athens, Greece.
Copyright 2011 ACM 978-1-4503-0660-7/11/06 ...$10.00.

General Terms

Algorithms, Theory, Verification

1. INTRODUCTION

While XML is embraced by the industry as the de facto standard for data exchange on the Web, XML can also be used to describe marked-up text. Actually, XML's predecessor, SGML, was exactly intended for this purpose. We refer to such documents as text-centric. Examples are poems, books, legislative text, e-governments text, and so on. A characteristic of such documents is that data values are words and sentences, and that their ordering matters. In this paper, we are interested in simple transformations which can change the internal structure of the document, that is, the mark-up, and can filter out parts of the text but do not disrupt the ordering of the words. In this way, the transformations can preserve the meaning of the filtered text. Specifically, we focus on XML transformations where the transformed document is a subsequence of the input document when ignoring mark-up. We call such transformations *text-preserving*. The central problem we consider in this paper, is to decide whether a given transformation is text-preserving for a given class of XML documents. Classes of XML documents will be defined by unranked regular tree languages and transformations by various kinds of transducers. We will show that, modulo some technical restrictions on transductions, transductions are text-preserving if and only if they are not copying and not rearranging.

We will not consider arbitrary transductions but will focus on XSLT-like transformations and use the abstraction of XSLT called DTL, introduced in [11], extended for dealing with data values. DTL is a rule-based language and is parameterized by a pattern language which is used for navigating in the tree and for deciding which rules can be executed. We consider three kinds of settings. The first is a top-down setting where navigation is restricted to children and rule patterns are restricted to label tests. Actually, to ease presentation, we define the top-down fragment of DTL separately as it corresponds to the formalism of top-down uniform tree transducers. In the two other settings, we consider XPath and MSO for both navigational and rule patterns. We refer to these fragments as $\mathrm{DTL}^{\mathrm{XPath}}$ and $\mathrm{DTL}^{\mathrm{MSO}}$, respectively. We show that testing whether a transduction is text-preserving is in PTIME, in EXPTIME and decidable, respectively, for top-down transducers, $\mathrm{DTL}^{\mathrm{XPath}}$ and $\mathrm{DTL}^{\mathrm{MSO}}$ over the class of regular tree languages represented by non-deterministic tree automata.

The high-level proof idea underlying all three results is

the same, but the details differ greatly, leading to the different complexities. We essentially show that the set of trees for which the given transduction is not text-preserving is a regular tree language. We refer to the latter as the language of counter-examples. The result then follows by testing emptiness of the intersection of that language with the tree automaton representing the input schema. Of course, the language of counter-examples depends on the transducers. We represent it in the three respective settings by non-deterministic top-down tree automata, alternating tree-walking automata, and non-deterministic tree-jumping automata with MSO-transitions. Of independent interest, we observe that non-deterministic tree-jumping automata with MSO-transitions define only regular tree languages. As regular languages are closed under complement, it readily follows that a regular tree language can be constructed which represents the largest subset of the input schema on which a given transducer is text-preserving.

Outline. In Section 2, we introduce the necessary definitions. In Section 3, we characterize text-preserving transductions. In Section 4 and Section 5, we consider top-down uniform tree transducers and DTL, respectively. In Section 6, we discuss related work and we conclude in Section 7.

2. DEFINITIONS

For any $n, m \in \mathbb{N}$ with $m \geq n$, $[n,m]$ denotes the set of integers $\{n, n+1, \dots, m-1, m\}$. In this paper we consider finite and infinite alphabets, but Σ always denotes a finite alphabet. We denote symbols from Σ by σ, σ_1, and so on. We assume that special symbols such as "(", ")", etc. are not in Σ.

Strings. A *symbol* is an element of a (finite or infinite) alphabet Δ and a *string* w is a finite sequence of symbols $\sigma_1 \cdots \sigma_n$ for some $n \in \mathbb{N}$. We define the *length* of a string $w = \sigma_1 \cdots \sigma_n$, denoted by $|w|$, to be n and we also refer to $|w|$ as the *size* of w. The empty string is denoted by ε. We assume that readers are familiar with standard operations on strings and sets of strings such as concatenation.

Automata. A *nondeterministic finite string automaton (NFA)* A over an alphabet Σ is a tuple $(Q, \Sigma, \delta, q_0, F)$, such that Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states and $\delta : Q \times \Sigma \to 2^Q$ is the transition function. A *run* of the automaton A on a string $w = a_1 \cdots a_n$ is a sequence of states $q_0 q_1 \cdots q_n$ such that, for each $i \in [1,n]$, $q_i \in \delta(q_{i-1}, a_i)$. A run on w is *accepting* if $q_n \in F$, and a string w is *accepted* by A, if there exists an accepting run of A on w. The *size* $|A|$ of an NFA A is its total number of states and transitions. By $\mathcal{L}(A)$ we denote the set of strings accepted by A. A string language L over Σ is *regular* if there is an NFA A such that $\mathcal{L}(A) = L$ We denote the set of regular languages over alphabet Σ by $\mathsf{REG}(\Sigma)$.

Hedges and trees. The set of *unranked trees* over alphabet Σ, denoted by Trees_Σ, is the smallest set S of strings over Σ and the parenthesis symbols "(" and ")" such that, $\varepsilon \in S$ and for each $\sigma \in \Sigma$ and $w \in S^*$, we have that $\sigma(w)$ is in S. For readability, we denote the tree $\sigma()$ by σ. A *tree language* over Σ is a subset of Trees_Σ. The set of *unranked hedges* over alphabet Σ, denoted by Hedges_Σ, is defined as $\mathsf{Hedges}_\Sigma = \mathsf{Trees}_\Sigma^*$. In particular, each tree is also a hedge. When we write a tree t as $t = \sigma(t_1 \cdots t_n)$ or a hedge h as $h = t_1 \cdots t_n$, we implicitly assume that all t_i are trees.

The *set of nodes* of a tree t and of a hedge h, denoted by Nodes^t and Nodes^h, respectively, are sets of strings in $\mathbb{N}^*$ and are inductively defined as follows. If $h = \sigma_1 \cdots \sigma_n$, then $\mathsf{Nodes}^h = \{1, \dots, n\}$. Here, for each $i \in [1,n]$, the node i is labelled by σ_i, which we denote by $\mathsf{lab}^h(i) = \sigma_i$. If $t = \sigma(h)$, then $\mathsf{Nodes}^t = \{1\} \cup \{1u \mid u \in \mathsf{Nodes}^h\}$. Here, the root node is 1 and the other nodes are nodes of the subhedge h, prefixed by a 1. The root is labeled σ, i.e., $\mathsf{lab}^t(1) = \sigma$ and, for every node $1u \in \mathsf{Nodes}^t$ we define $\mathsf{lab}^t(1u) = \mathsf{lab}^h(u)$. Finally, if $h = t_1 \cdots t_n$, then $\mathsf{Nodes}^h = \cup_{i=1}^n \{iu \mid 1u \in \mathsf{Nodes}^{t_i}\}$. Here, the labels from the t_i carry over to h, that is, for each $i \in [1,n]$ and each node $iu \in \mathsf{Nodes}^h$, we have $\mathsf{lab}^h(iu) = \mathsf{lab}^{t_i}(1u)$. The *depth* of a node $u \in \mathbb{N}^*$ is $|u|$. Hence, the depth of the root of a tree is one.

From the definition of nodes we can see that the lexicographic order $<_{\text{lex}}$ on Nodes^h corresponds to the order generated by the depth-first (left-to-right) traversal of the hedge h. More specifically, for two nodes iu and $jv \in \mathsf{Nodes}^h$ with $i, j \in \mathbb{N}$, we have $iu <_{\text{lex}} jv$ if $i < j$ or if $i = j$ and $u <_{\text{lex}} v$. The *children* of a node v in h are all nodes $v' \in \mathsf{Nodes}^h$ such that $v' = v \cdot i$ for $i \in \mathbb{N}$. A node $v \in \mathsf{Nodes}^h$ is a *leaf* if it has no children.

The *size* of a hedge h, denoted by $|h|$, is its number of nodes. For a tree t and a node $u \in \mathsf{Nodes}^t$, we denote by $\mathsf{anc\text{-}str}^t(u)$ the *ancestor string of u in t*, i.e., the string formed by the labels on the path in the tree t from the root to u, including the label of u. The *lowest common ancestor* of two nodes v_1 and v_2 in Nodes^t is the node corresponding to the longest common prefix of v_1 and v_2.

The *subtree of hedge h at a node u* is the tree induced by the set of nodes with prefix u and is denoted by $\mathsf{subtree}^h(u)$. For any hedge h, node u, and for any hedge h', $h[u \leftarrow h']$, denotes the hedge obtained from h by replacing $\mathsf{subtree}^h(u)$ with h' and by redefining the set of nodes and the label function accordingly. For any two alphabets Σ and Γ, $\mathsf{Trees}_\Sigma(\Gamma)$ is the set of trees over the alphabet $\Sigma \cup \Gamma$, where only leaves are allowed to be labelled with symbols from Γ. Similarly, $\mathsf{Hedges}_\Sigma(\Gamma)$ is $(\mathsf{Trees}_\Sigma(\Gamma))^*$. Finally, the *frontier*[1] of a hedge h, denoted by $\mathsf{frontier}(h)$, is the largest sequence $\mathsf{lab}^h(v_1) \cdots \mathsf{lab}^h(v_n)$, where, for every $i \in [1,n]$, v_i is a leaf of h, and for every $i \in [1, n-1]$, $v_i <_{\text{lex}} v_{i+1}$.

In the following, we will often write Nodes, lab, etc. without index whenever the tree or hedge is clear from the context.

Text trees. Let Text be an infinite set, disjoint from any finite alphabet we consider here, such as Σ. A *text tree* t over Σ is a tree in $\mathsf{Trees}_\Sigma(\mathsf{Text})$ or, in other words, a tree over the alphabet $\Sigma \cup \mathsf{Text}$ where symbols from Text can only appear at the leaves. We refer to these nodes as *text nodes* and denote the set of text nodes of a tree t by $\mathsf{text\text{-}nodes}^t$. A text tree language is a subset of $\mathsf{Trees}_\Sigma(\mathsf{Text})$. In the following, unless otherwise stated, whenever we say *tree* we always mean *text tree*. In particular, we also simply say *tree language* instead of *text tree language*.

A tree language L is *closed under* Text*-substitutions* if, for any tree $t \in L$ and any text node u of t, the tree obtained from t by changing u's label to another value in Text is also in the language L. Formally, a Text*-substitution* ρ is a function from $\mathsf{Trees}_\Sigma(\mathsf{Text})$ to $\mathsf{Trees}_\Sigma(\mathsf{Text})$, such that

[1] A frontier is sometimes also called a *yield*.

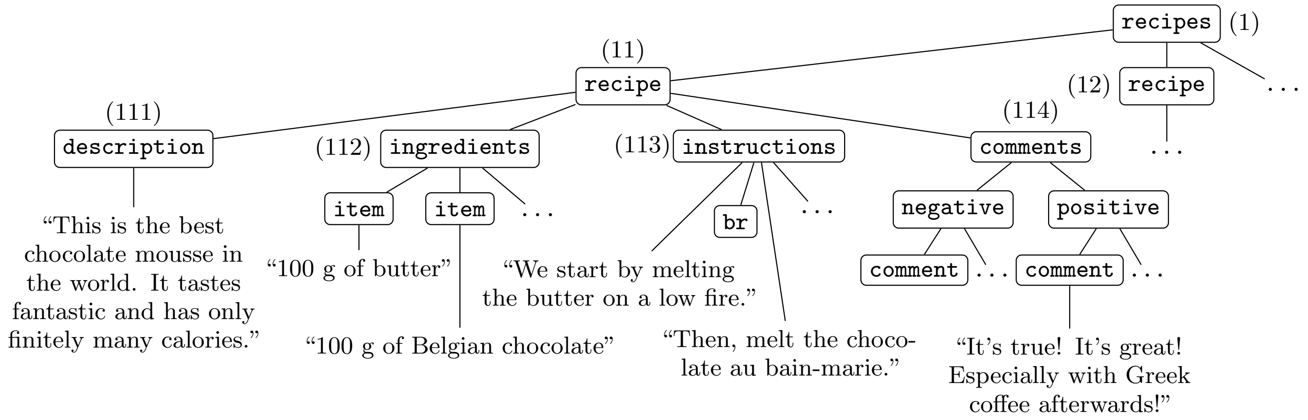

Figure 1: A text tree representing an XML document for recipes.

for each $t \in \mathsf{Trees}_\Sigma(\mathsf{Text})$, $\mathsf{Nodes}^t = \mathsf{Nodes}^{\rho(t)}$, for each node $v \notin \mathsf{text\text{-}nodes}^t$, $\mathsf{lab}^t(v) = \mathsf{lab}^{\rho(t)}(v)$ and for each node $v \in \mathsf{text\text{-}nodes}^t$, $\mathsf{lab}^{\rho(t)}(v) \in \mathsf{Text}$. In other words, a Text-substitution relabels zero or more leaf nodes labelled with a label in Text, to some other label in Text.

In the sequel we only consider tree languages that are closed under Text-substitutions primarily because we want to consider Text as an abstract set. Furthermore, when ignoring integrity constraints, XML schema languages are rather restricted in enforcing constraints on actual text values. Therefore, it makes sense to simply consider one infinite set of possible values so that replacing a certain text value with another does not change validity of the document with respect to the schema.

The *text-content* of a tree t, denoted by $\mathsf{text\text{-}content}(t)$, is the string over alphabet Text obtained by concatenating the text values of all text nodes in document order, which is the lexicographic order induced by $\mathbb{N}^*$. Notice that, for any tree t, $\mathsf{text\text{-}content}(t)$ is the substring of $\mathsf{frontier}(t)$ containing exactly the labels from Text.

Example 2.1. Figure 1 depicts a tree representation of an XML document underlying a web site with recipes. Each recipe has a description, a list of ingredients, instructions, and a list of comments by users. The text nodes of the tree are the ones that contain text in quotation marks. The ancestor path of the node labeled `positive` is `recipes recipe comments positive`. The text-content of the tree is the concatenation of all the text between the quotation marks, from left to right. For clarity, we annotate some nodes with their name in braces.

A *tree transduction* T is a mapping $T : L_1 \to L_2$ for tree languages L_1 and L_2. We say that a string $s_1 = \sigma_1 \cdots \sigma_n$ over alphabet Δ is a *subsequence* of s_2, denoted $s_1 \prec s_2$, if s_2 is of the form $w_0\sigma_1 w_1 \cdots w_{n-1}\sigma_n w_n$ for some $w_0, \ldots, w_n \in \Delta^*$. We are now ready to define the notion central to this paper.

Definition 2.2. A tree transduction T is *text preserving* over a tree language L if, for all trees $t \in L$,

$$\mathsf{text\text{-}content}(T(t)) \prec \mathsf{text\text{-}content}(t).$$

Schema languages. We abstract the schema languages *Document Type Definition (DTD)* and Relax NG by extended context-free grammars and by unranked tree automata, respectively.

A *Document Type Definition (DTD)* over some finite alphabet Σ is a tuple $D = (\Sigma \uplus \{\mathtt{text}\}, C, d, S_d)$ where C is a set of regular string languages over $\Sigma \uplus \{\mathtt{text}\}$, d is a function that maps symbols in Σ to languages in C, and $S_d \subseteq \Sigma$ is a set of start symbols. We refer to the languages in C as the *content models* of the DTD. A tree t is *valid with respect to* a DTD D or *satisfies* D, if its root is labelled by an element of S_d and, for every node labelled with some $\sigma \in \Sigma$, the sequence $\sigma_1 \cdots \sigma_n$ of labels of its children, where any element in Text is replaced by the symbol `text`, is in the language $d(\sigma)$. So, `text` serves as a placeholder for text nodes. In the remainder of this paper, we assume that the regular languages in C are represented by regular expressions or NFAs.

The set of trees that are valid with respect to a DTD D is denoted by $\mathcal{L}(D)$. A DTD D is *reduced* if, for every $\sigma \in \Sigma$ for which $d(\sigma)$ is defined, there exists a tree $t \in \mathcal{L}(D)$ such that the label σ occurs somewhere in t. Any DTD can be transformed to an equivalent reduced DTD in polynomial time [1, 16]. However, reducing a DTD is Ptime-complete.[2] In the following, we assume that all DTDs are reduced.

Example 2.3. The tree in Figure 1 is valid w.r.t. the DTD $(\Sigma \uplus \{\mathtt{text}\}, C, d, S_d)$, where Σ is the set of labels used in the boxed nodes of the tree, i.e., `recipes`, `recipe`, ... We represent the regular languages in C by regular expressions. Then, $S_d = \{\mathtt{recipes}\}$ and d is defined as:

$$
\begin{array}{lcl}
\mathtt{recipes} & \mapsto & \mathtt{recipe}^* \\
\mathtt{recipe} & \mapsto & \mathtt{description} \cdot \mathtt{ingredients} \\
 & & \qquad \cdot \mathtt{instructions} \cdot \mathtt{comments} \\
\mathtt{ingredients} & \mapsto & \mathtt{item}^* \\
\mathtt{instructions} & \mapsto & (\mathtt{br} + \mathtt{text})^* \\
\mathtt{br} & \mapsto & \varepsilon \\
\mathtt{comments} & \mapsto & \mathtt{negative} \cdot \mathtt{positive} \\
\mathtt{positive} & \mapsto & \mathtt{comment}^* \\
\mathtt{negative} & \mapsto & \mathtt{comment}^* \\
\mathtt{description} & \mapsto & \mathtt{text} \\
\mathtt{item} & \mapsto & \mathtt{text} \\
\ldots & &
\end{array}
$$

Notice that the children of `instructions` can come from Σ

[2] More accurately, *deciding* whether a DTD is reduced is Ptime-complete.

and from Text. For every Σ-symbol σ for which we did not yet define d, we define $d(\sigma) = \texttt{text}$.

Relax NG schemas are abstracted by *nondeterministic unranked tree automata (*NTA*)*. An NTA N over some finite alphabet Σ, is a tuple $(Q, \Sigma \uplus \{\texttt{text}\}, \delta, q_0, F)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, F is the set of final states, and $\delta : Q \times (\Sigma \uplus \{\texttt{text}\}) \to \mathsf{REG}(Q)$ is the transition function. A *run* of N over some tree t is a function $\rho : \mathsf{Nodes}^t \to Q$ such that $\rho(1) = q_0$, i.e., the root of t is assigned q_0 and, for any node v labeled with $\sigma \in \Sigma$ with n children, it holds that $\rho(v1) \cdots \rho(vn)$ is in the language $\delta(\rho(v), \sigma)$. A run ρ of N on some tree t is *accepting* if $\varepsilon \in \delta(\rho(v), \mathsf{lab}(v))$ for every leaf v of t if $\mathsf{lab}(v) \in \Sigma$, and $\delta(\rho(v), \texttt{text}) = \{\varepsilon\}$ if $\mathsf{lab}(v) \in \mathsf{Text}$. So, also here text serves as a placeholder for Text-values. We define F to be the set of states q such that $\varepsilon \in \delta(q, a)$ for some $a \in \Sigma \uplus \{\texttt{text}\}$. A tree t is *accepted* by N if there exists an accepting run of N on t. By $\mathcal{L}(N)$ we denote the set of trees accepted by N. A tree language L is *regular*, if there is an NTA N such that $L = \mathcal{L}(N)$. It is well-known that regular tree languages are closed under union, intersection, and complementation.

Unless otherwise mentioned, we assume that the regular languages $\delta(q, \sigma)$ in NTAs are represented by NFAs. The *size* $|N|$ of an NTA $N = (Q, \Sigma, \delta, q_0, F)$ is equal to $|Q| + |\delta|$, where $|\delta| = \sum_{q \in Q, a \in \Sigma} |\delta(q, a)|$.

Central problem. In the rest of this paper, we consider the following decision problem. Let $\mathcal{T}$ be a class of tree transductions and let $\mathcal{L}$ be a class of tree languages. We then study the question:

> Given a language $L \in \mathcal{L}$ and a transduction $T \in \mathcal{T}$, is T text-preserving over L?

Recall that, in this paper, the languages $\mathcal{L}$ are always closed under Text-substitutions. In the conclusions we discuss how our technique also allows to solve more complex decision problems.

3. A CHARACTERIZATION OF TEXT-PRESERVING TRANSDUCTIONS

We provide a characterization of when a transduction is text-preserving in terms of its copying and rearranging behavior. First, we need some terminology. We say that a tree is *value-unique* when all its Text-values occurring at leaves are different. Notice that, since we only consider tree languages that are closed under Text-substitutions, all the tree languages we consider contain at least one tree which is value-unique.

Definition 3.1. A transduction T is *copying over a tree language* L if there is a value-unique $t \in L$ such that $T(t)$ contains multiple occurrences of the same Text-value. A transduction T is *rearranging over a tree language* L if there is a value-unique $t \in L$ such that, for some γ_1 and $\gamma_2 \in \mathsf{Text}$, $\gamma_1\gamma_2 \prec \mathsf{text\text{-}content}(t)$ and $\gamma_2\gamma_1 \prec \mathsf{text\text{-}content}(T(t))$.

Up to now, transductions are general mappings from trees to trees. We will next put some restrictions on them. Let $\maltese \notin \mathsf{Text}$. For $\gamma \in \mathsf{Text} \cup \{\maltese\}$, let ρ_γ be the Text-substitution that for every tree t, relabels every text node with γ. A transduction T is Text-*independent* if for every tree t and every Text-substitution ρ on t,

$$\rho_\maltese(T(\rho(t))) = \rho_\maltese(T(t)).$$

Informally, this ensures that the structure of the transduced tree $T(t)$ does not depend on the Text-values occurring in t. Only the concrete Text-values in $T(t)$ and $T(\rho(t))$ can differ. The final notion we need is that of a Text-functional transduction. A transduction T is Text-*functional* if, for every tree t, there exists a function f from the set of text nodes of $T(t)$ to the set of text nodes of t, such that for every Text-substitution ρ on t, $\mathsf{lab}^{T(\rho(t))}(v) = \mathsf{lab}^{\rho(t)}(f(v))$ for every node $v \in \mathsf{text\text{-}nodes}^{T(\rho(t))}$. Intuitively, this means that Text-values in the output tree are determined by the Text-value at the corresponding node, determined by f, in the input tree. Note that a Text-functional transduction can never introduce Text-values that do not appear in the input tree.

Definition 3.2. A transduction is *admissible* if it is Text-independent and Text-functional.

We will show in the sequel that all transductions we consider in this paper are admissible. We are now ready to prove the characterization of text-preserving transductions.

Theorem 3.3. *An admissible transduction T is text-preserving over a language L if and only if it is not copying and not rearranging over L.*

Proof. When T is copying or rearranging over L then T is obviously not text-preserving over L.

When T is not text-preserving, there exists a tree $t \in L$ such that $\mathsf{text\text{-}content}(T(t)) \not\prec \mathsf{text\text{-}content}(t)$. Let ρ be a Text-substitution such that $\rho(t)$ is value-unique. First, we argue that $\mathsf{text\text{-}content}(T(\rho(t))) \not\prec \mathsf{text\text{-}content}(\rho(t))$. Assume for the sake of contradiction that $\mathsf{text\text{-}content}(T(\rho(t))) \prec \mathsf{text\text{-}content}(\rho(t))$ and let g be a function from $\mathsf{text\text{-}nodes}^{T(\rho(t))}$ to $\mathsf{text\text{-}nodes}^{\rho(t)}$ that witnesses this subsequence relation. In other words, since $\rho(t)$ is value-unique, g maps every text node in $T(\rho(t))$ to the unique text node in $\rho(t)$ with the same text value. In particular, g is unique.

Since T is Text-functional by assumption, there exists a function $f : \mathsf{text\text{-}nodes}^{T(t)} \to \mathsf{text\text{-}nodes}^{t}$ such that, for every Text-substitution ρ' and every node $v \in \mathsf{text\text{-}nodes}^{T(\rho'(t))}$, $\mathsf{lab}^{T(\rho'(t))}(v) = \mathsf{lab}^{\rho'(t)}(f(v))$. Since this condition holds for every substitution ρ', it holds for ρ in particular. Therefore it is also the case that, for every node $v \in \mathsf{text\text{-}nodes}^{T(\rho(t))}$, $\mathsf{lab}^{T(\rho(t))}(v) = \mathsf{lab}^{\rho(t)}(f(v))$. Since $\rho(t)$ is value unique and T is Text-independent, it follows that f maps every text node in $T(\rho(t))$ to the unique text node in $\rho(t)$ with the same text value.

Since $\mathsf{text\text{-}nodes}^{t} = \mathsf{text\text{-}nodes}^{\rho(t)}$ and $\mathsf{text\text{-}nodes}^{T(t)} = \mathsf{text\text{-}nodes}^{T(\rho(t))}$ because T is Text-independent, it follows that f and g are the same function. By definition of f, for every node $v \in \mathsf{text\text{-}nodes}^{T(t)}$, $\mathsf{lab}^{T(t)}(v) = \mathsf{lab}^{t}(f(v))$ and therefore, f is a witness function to $\mathsf{text\text{-}content}(T(t)) \prec \mathsf{text\text{-}content}(t)$, which contradicts our assumption that T is not text-preserving on t.

It remains to show that T is either copying or rearranging on t. Consider the function f, and notice that since $\mathsf{text\text{-}content}(T(t)) \not\prec \mathsf{text\text{-}content}(t)$, f is either not injective or does not preserve the order $<_{\mathrm{lex}}$ in trees. In the first case, T is copying and in the second case T is rearranging. $\square$

4. TOP-DOWN TRANSDUCERS

We start with the simple uniform top-down tree transducers considered in [13, 14, 15] in the context of type checking.

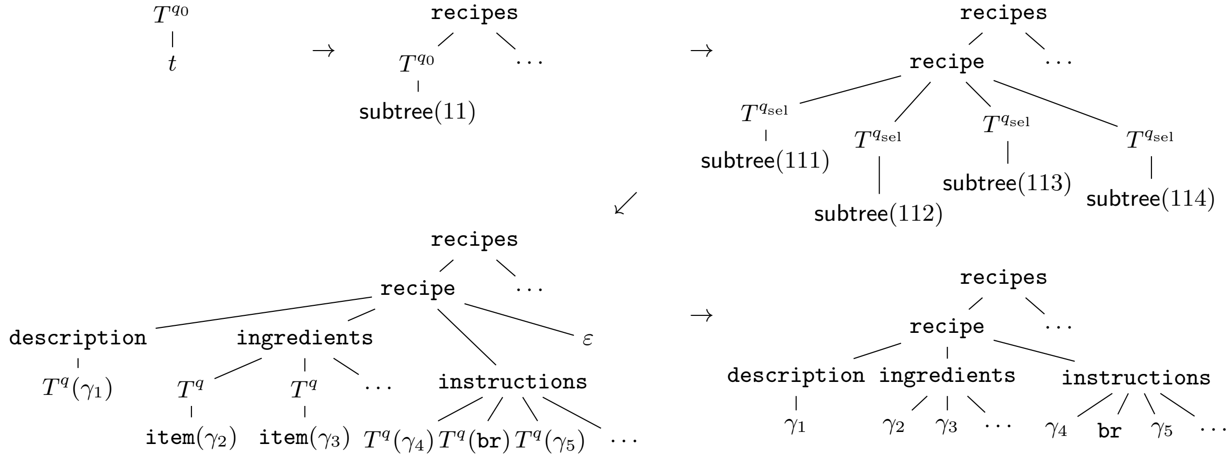

Figure 2: A uniform transducer selecting all descriptions, ingredients, and instructions from the tree of Fig. 1.

These correspond to a simple top-down fragment of XSLT and are equivalent to DTL, defined in the next section, instantiated with only downward navigation.

4.1 Definition

Definition 4.1. A *top-down uniform tree transducer* is a tuple $T = (Q, \Sigma \cup \{\texttt{text}\}, q^0, R)$, where Q is a finite set of states, Σ is a finite alphabet, $q^0 \in Q$ is the initial state, and R is a finite set of rules of the form $(q, a) \to h$, where $q \in Q$ and
(a) if $a \in \Sigma$ then $h \in \mathsf{Hedges}_\Sigma(Q)$ and
(b) if $a = \texttt{text}$ then h is $\texttt{text}$.
When $q = q^0$, then h is restricted to $\mathsf{Trees}_\Sigma(Q)$ and should contain at least one Σ-label. This restriction of h ensures that the output of T is always a tree. For each state $q \in Q$ and symbol $a \in \Sigma$ there is at most one hedge $h \in \mathsf{Hedges}_\Sigma(Q)$ such that $(q, a) \to h$ is a rule in R.

The translation defined by T on a tree t in state q, denoted by $T^q(t)$, is defined inductively as follows:

(i) $T^q(\varepsilon) = \varepsilon$;

(ii) for each $\gamma \in \mathsf{Text}$, if there is a rule $(q, \texttt{text}) \to \texttt{text}$ then $T^q(\gamma) = \gamma$, if there is no such rule, then $T^q(\gamma) = \varepsilon$;

(iii) if $t = a(t_1 \cdots t_n)$ for some $a \in \Sigma$ and there is a rule $(q, a) \to h$, then $T^q(t)$ is obtained from h by replacing every node u in h labelled with $p \in Q$, by the hedge $T^p(t_1) \cdots T^p(t_n)$. If there is no such rule, then $T^q(t) = \varepsilon$.

The transformation of t by T, denoted by $T(t)$, is defined as $T^{q^0}(t)$. To make use of our characterization in Theorem 3.3, we need to be able to reason about the function that is defined by T. To this end, we say that the *transduction defined by* T or *transduction of* T is the function mapping every tree t onto $T(t)$. For any rule $(q, a) \to h$ in R, we denote the hedge h on its right hand side by $\mathsf{rhs}(q, a)$. The *size* of T, denoted by $|T|$, is equal to $|Q| + |R|$, where $|R| = \sum_{q \in Q, a \in \Sigma} |\mathsf{rhs}(q, a)|$. In the remainder of this section, we simply say transducer rather than top-down uniform tree transducer.

Example 4.2. The uniform transducer defined by the rules

$$
\begin{array}{lll}
(q_0, \texttt{recipes}) & \to & \texttt{recipes}(q_0) \\
(q_0, \texttt{recipe}) & \to & \texttt{recipe}(q_{\mathrm{sel}}) \\
(q_{\mathrm{sel}}, \sigma) & \to & \sigma(q) \quad (\sigma \in \{\texttt{description}, \texttt{ingredients}, \texttt{instructions}\}) \\
(q, \texttt{item}) & \to & q \\
(q, \texttt{br}) & \to & \texttt{br}(q) \\
(q, \texttt{text}) & \to & \texttt{text}
\end{array}
$$

Selects, from the tree of Figure 1, all recipes, their descriptions, ingredients lists, and instructions; and deletes the comments. Furthermore, it keeps the markup (i.e., the `br` nodes) in the instructions, but deletes the `item` nodes. The transformation on the tree in Figure 1 is illustrated in Figure 2. For readability in the figure, we sometimes write a function call of the form $T^q(t)$ as a (sub)tree where t is a child of a node labelled T^q.

We call a state q of T *reachable* if there is a sequence of pairs $(q_1, a_1) \cdots (q_n, a_n)$ for some $n \in \mathbb{N}$, such that $q_1 = q^0$, $q_n = q$ and, for all $i \in [1, n-1]$, we have that q_{i+1} occurs as the label of a leaf in the hedge $\mathsf{rhs}(q_i, a_i)$. Notice that, if a transducer has a rule of the form $(q, a) \to \varepsilon$, we can remove this rule and the resulting transducer is equivalent (due to rule (iii) in the definition of $T^q(t)$). We therefore say that a rule of the form $(q, a) \to \varepsilon$ is *useless*. A transducer is *reduced*, if all its states are reachable and it contains no useless rules. In the following, we assume that transducers are always reduced.

Notice that rules in transducers do not contain any values from Text. This ensures that transducers can not introduce a Text-value not present in the input tree. We note that the transducers in this section are admissible, in the sense of Definition 3.2.

Lemma 4.3. *Top-down uniform tree transducers are admissible.*

4.2 Copying and rearranging transducers

In this section, we provide equivalent formalizations for the notions of copying and rearranging introduced in Section 3. In Section 4.3, we then show that it is decidable in Ptime whether a transducer is copying or rearranging thereby obtaining Ptime-decidability for testing whether a transducer is text-preserving.

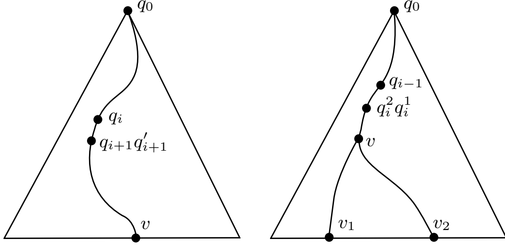

Figure 3: Illustration of copying (left) and rearranging (right) for top-down uniform tree transducers.

First we formally define copying and rearranging for a transducer.

DEFINITION 4.4. Let L be a tree language. A transducer T is *copying over* L if the transduction defined by T is copying over L. It is *rearranging over* L if the transduction defined by T is rearranging over L.

In the following lemma, we provide an operational condition that is equivalent to the general notion of copying defined above. First, we introduce the following definitions. A *text path* is a sequence in $\Sigma^* \cdot \mathsf{Text}$. A *path run* of T on a text path $a_1 \cdots a_n\gamma$ is a sequence $q_1, \ldots, q_n, q_{n+1}$ such that $q_1 = q^0$, and for all $i \in [1, n]$, q_{i+1} occurs at a leaf of $\mathsf{rhs}(q_i, a_i)$ and $(q_{n+1}, \mathtt{text}) \to \mathtt{text}$. Intuitively, a path run is a set of states that T assumes when processing the text path. Note that there can be several path runs for the same text path as the transducer can use rules containing multiple states in a right-hand side.

LEMMA 4.5. *A transducer $T = (Q, \Sigma \cup \{\mathtt{text}\}, q^0, R)$ is copying over a tree language L if and only if there exists a tree $t \in L$ and a leaf node v in t such that* $\mathsf{anc\text{-}str}(v) = a_1 \cdots a_n \cdot \gamma$ *is a text path on which either*
(1) T has two different path runs, or
(2) T has a path run $q_1, \ldots, q_n, q_{n+1}$, where for some $i \in [1, n]$, $q_{i+1} \cdot q_{i+1} \prec \mathit{frontier}(\mathsf{rhs}(q_i, a_i))$.

Recall that $\prec$ denotes the subsequence relation. Condition (2) therefore states that q_{i+1} occurs on two different leaves of $\mathsf{rhs}(q_i, a_i)$. The condition in Lemma 4.5 is illustrated in Figure 3. In the figure, if $q_{i+1} \neq q'_{i+1}$, there are two path runs and if $q_{i+1} = q'_{i+1}$, then $q_{i+1} \cdot q_{i+1} \prec \mathsf{frontier}(\mathsf{rhs}(q_i, a_i))$.

Next, we characterize when a transducer rearranges two Text-values in a tree t. Similarly as for copying, we provide an operational condition for rearranging. Intuitively, a transducer rearranges when there are two leaves v_1 and v_2 in t and $T(t)$ swaps the Text-values corresponding to v_1 and v_2. In terms of a top-down uniform tree transducer, this means that, on the path from the root to the lowest common ancestor of v_1 and v_2, a swap operation takes place.

LEMMA 4.6. *A transducer $T = (Q, \Sigma \cup \{\mathtt{text}\}, q^0, R)$ is rearranging over a tree language L if and only if there exists a tree $t \in L$, two leaf nodes v_1, v_2 of t with $v_1 <_{lex} v_2$ with text paths $\mathsf{anc\text{-}str}(v_1) = a_1^1 \cdots a_n^1 \cdot \gamma^1$ and $\mathsf{anc\text{-}str}(v_2) = a_1^2 \cdots a_m^2 \cdot \gamma^2$ on which which T has path runs $q_1^1, \ldots, q_n^1, q_{n+1}^1$ and $q_1^2, \ldots, q_m^2, q_{m+1}^2$, such that the following holds. If the lowest common ancestor v of v_1 and v_2 has depth k, then there exists an $i \leq k$ such that*
(1) for all $j < i$, we have $(q_j^1, a_j^1) = (q_j^2, a_j^2)$ and
(2) $q_i^2 \cdot q_i^1 \prec \mathit{frontier}(\mathsf{rhs}(q_{i-1}^1, a_{i-1}^1))$.

The condition in Lemma 4.6 for rearranging is illustrated in Figure 3. From Lemmas 4.5 and 4.6 and from Theorem 3.3 we now obtain the following characterization for text-preserving top-down uniform transducers:

THEOREM 4.7. *A uniform top-down transducer T is not text-preserving over a tree language L if and only if the condition in Lemma 4.5 or the condition in Lemma 4.6 holds.*

4.3 PTIME Result

This section is devoted to the proof of Theorem 4.11 which states that it is decidable in PTIME whether a top-down tree transducer is text-preserving over an unranked regular tree language represented by an NTA.

For a tree language L, we denote by $L_{\mathtt{text}}$ the language over alphabet $\Sigma \uplus \{\mathtt{text}\}$ obtained from L by replacing, in each tree, each label $\gamma \in \mathsf{Text}$ by the label $\mathtt{text}$. The *path language* of a regular tree language L is the language of all root-to-leaf paths in trees of $L_{\mathtt{text}}$ that end with $\mathtt{text}$. More formally, the text path language of L is $\{w \mid t \in L_{\mathtt{text}}, \mathsf{lab}^t(u) = \mathtt{text}, \text{ and } w = \mathsf{anc\text{-}str}^t(u)\}$. We say that an NFA A_L is a *path automaton for regular tree language* L if A_L accepts the path language of L. For a tree transducer T, we say that an NFA A_T is a *transducer path automaton for* T if it accepts exactly the text paths on which T has a path run. Notice that A_L and A_T accept only strings that end with $\mathtt{text}$. We show that they can be constructed in PTIME.

LEMMA 4.8. *(1) Given an* NTA *N, we can construct a path automaton A_N for $\mathcal{L}(N)$ in polynomial time.*
(2) Given a uniform tree transducer T, we can construct a transducer path automaton A_T for T in polynomial time.

From now on, we refer to A_T as *the* transducer path automaton of T and to A_N as *the* path automaton of N. As our next step, we use the two types of path automata to test for copying or rearranging. First, we treat copying.

LEMMA 4.9. *Given a uniform tree transducer T and an* NTA *N, it is decidable in* PTIME *whether T is copying over $\mathcal{L}(N)$.*

PROOF SKETCH. We define an NFA M that accepts text paths satisfying condition (1) or condition (2) of Lemma 4.5. So, M is non-empty iff T is copying over $\mathcal{L}(N)$. As M can be constructed in PTIME and testing non-emptiness of NFAs is in PTIME as well, the result follows. Basically, M simulates both A_N (as the text path should come from a tree accepted by N) and two copies of A_T (as there should be a path run of T) while checking for the existence of two different path runs or while trying to reach a rule in T which copies. □

LEMMA 4.10. *Given a uniform tree transducer T and an* NTA *N, it is decidable in* PTIME *whether T is rearranging over $\mathcal{L}(N)$.*

PROOF SKETCH. We will construct in PTIME an NTA M accepting the set of trees on which T is rearranging. Therefore, the intersection of M and N is non-empty if and only if T is rearranging over $\mathcal{L}(N)$. As the latter can be done in PTIME, the result follows. Intuitively, M directly searches for nodes satisfying the proporties mentioned in Lemma 4.6. We describe M as if operating in a top-down fashion. Starting from the root of an input tree t, M guesses a path, simulating A_T. At a certain non-deterministically chosen

point, M decides it has found state $q^1_{i-1} = q^2_{i-1}$ and non-deterministically picks q^1_i and q^2_i (cf. Lemma 4.6). From then on, M continues until it (non-deterministically) arrives at the lowest common ancestor of v_1 and v_2. On this path, M will simulate two copies of A_T; one copy continuing from state q^1_i and one copy continuing from state q^2_i. At the lowest common ancestor of v_1 and v_2, the simulation of these two copies of M will split again. Towards v_1, M simulates the copy that started from q^1_i, and towards v_2, M simulates the copy that started from q^2_i. Finally, M accepts when the copy for q^1_i leads to acceptance of A_T at node v_1 and the copy of q^2_i leads to acceptance of A_T at node v_2. Recall that the acceptance condition of A_T means that the text value of the last node is copied to the output of T. □

By the previous lemmas the main complexity result of this section readily follows:

THEOREM 4.11. *Given a uniform tree transducer T and an* NTA N, *it is decidable in* PTIME *whether T is* Text-*preserving over $\mathcal{L}(N)$.*

5. DTL

5.1 General DTL Transducers

In this section, we consider the abstraction of XSLT called DTL, which was introduced in [11]. DTL programs can navigate in a tree through the use of *binary patterns.* While XSLT employs XPath, we leave the concrete pattern language implicit for now and introduce the following terminology. A *unary pattern* over Σ is a subset of $\bigcup_{t\in \mathsf{Trees}_\Sigma}(\{t\}\times \mathsf{Nodes}^t)$ and a *binary pattern* over Σ is a subset of $\bigcup_{t\in \mathsf{Trees}_\Sigma}(\{t\}\times \mathsf{Nodes}^t \times \mathsf{Nodes}^t)$. We refer to the set of unary and binary patterns over Σ as $\mathcal{UP}(\Sigma)$ and $\mathcal{BP}(\Sigma)$, respectively. We will denote unary patterns by φ, ψ and binary patterns by α, β. To emphasize that unary and binary patterns will be specified by pattern languages, we will also write $(t, u) \in \varphi$ as $t \models \varphi(u)$ and $(t, u, v) \in \alpha$ as $t \models \alpha(u, v)$.

Our definition of DTL differs slightly from the one in [11]. To simplify presentation, we disregard construction functions (cf. Section 6 for a discussion). On the other hand, we do extend DTL to deal with text values.

DEFINITION 5.1. A DTL *transducer* is a tuple $T = (\Sigma, \Delta, Q, q_0, R_\Sigma, R_{\mathsf{Text}})$, where

- Σ is the finite alphabet of input symbols,
- Δ is the finite alphabet of output symbols,
- Q is the finite set of states,
- $q_0 \in Q$ is the initial state,
- R_Σ is a finite set of rules of the form $(q, \varphi) \to h$, where $q \in Q$, $\varphi \in \mathcal{UP}(\Sigma)$, and h is in $\mathsf{Hedges}_\Delta(Q \times \mathcal{BP}(\Sigma))$. If $q = q_0$, h is required to be a tree such that $\mathsf{lab}^h(1) \notin Q \times \mathcal{BP}(\Sigma)$.[3]
- R_{Text} is a finite set of rules of the form $(q, \mathtt{text}) \to \mathtt{text}$, with $q \in Q$.

To ensure determinism, we require that when $(q, \varphi) \to h$ and $(q, \varphi') \to h'$ are rules in R_Σ, then there exists no tree t such that $t \models \varphi(v)$ and $t \models \varphi'(v)$ for any node v in t.[4] Notice in the above definition that there are no restrictions on the allowed patterns. Later, we will use concrete pattern languages like monadic second-order logic and XPath to define patterns.

We are now ready to define the transformation induced by T on a tree t. Intuitively, T starts in state q_0 at the root of t (note that the root of a tree is always 1). The latter is encoded by the initial configuration $(q_0, 1)$. Formally, a *configuration* is a pair in $Q \times \mathsf{Nodes}^t$. During computation T will rewrite the initial configuration into a partial output tree $\xi \in \mathsf{Trees}_\Delta(Q \times \mathsf{Nodes}^t)$. That is, a Δ-tree where some leaves will be labeled with configurations. The transducer then proceeds by rewriting these configurations extending ξ until all configurations are gone and the transduction stops. Basically, rewriting of a configuration (q, v) can be done by a rule $(q, \varphi) \to h$ when $t \models \varphi(v)$. The output tree is then appended with the hedge h where the leaves carrying binary patterns result in new configurations.

We next formally define the transduction relation. Thereto, given a tree t, the transformation relation induced by T on t, denoted by $\Rightarrow_{T,t}$, is the binary relation on $\mathsf{Trees}_\Delta(Q \times \mathsf{Nodes}^t)$ defined as follows. For $\xi, \xi' \in \mathsf{Trees}_\Delta(Q \times \mathsf{Nodes}^t)$, $\xi \Rightarrow_{T,t} \xi'$, if there is a leaf node $u \in \mathsf{Nodes}^\xi$ such that

1. $\mathsf{lab}^\xi(u) = (q, v)$ with $q \in Q$ and $v \in \mathsf{Nodes}^t$,
2. if $\mathsf{lab}^t(v) \in \mathsf{Text}$ then $\xi' = \xi[u \leftarrow \mathsf{lab}^t(v)]$ if there is a rule $(q, \mathtt{text}) \to \mathtt{text}$ and $\xi' = \xi[u \leftarrow \varepsilon]$ otherwise.[5]
3. if $\mathsf{lab}^t(v) \in \Sigma$,
   - if there is a rule $r = ((q, \varphi) \to h) \in R_\Sigma$ such that $t \models \varphi(v)$, then $\xi' = \xi[u \leftarrow h\Theta]$, where Θ denotes the substitution of replacing every pair (q', α) by the hedge $(q', v_1) \cdots (q', v_m)$, where
     - $\{v_1, \ldots, v_m\} = \{u \mid t \models \alpha(v, u)\}$, and
     - $v_1 <_{\mathrm{lex}} \cdots <_{\mathrm{lex}} v_m$,
   - if there is no rule $r = ((q, \varphi) \to h) \in R_\Sigma$ s.t. $t \models \varphi(v)$, then $\xi' = \xi[u \leftarrow \varepsilon]$.

For any tree t and DTL transducer T, $T(t)$ is defined to be the tree $t' \in \mathsf{Trees}_\Delta$, such that $(q_0, 1) \Rightarrow^*_{T,t} t'$, when such a tree exists and is undefined otherwise.[6] As configurations for which no rule applies are rewritten into the empty hedge, $T(t)$ can only be undefined when T does not stop. Furthermore, when $T(t)$ is defined, it is unique because of the determinism restriction on unary patterns.

Example 5.15 contains a DTL transducer which selects all recipes with their description, ingredients, and instructions if they have at least three positive comments. The unary and binary patterns in the example are given in an XPath syntax, formally defined in Section 5.4.

Notice that each top-down tree transducer T as introduced in the previous section can be defined by a DTL program T'. Indeed, for each rule $(q, a) \to h$ of T, T' has a rule $(q, \varphi) \to h'$ where $t \models \varphi(v) \Leftrightarrow \mathsf{lab}^t(v) = a$; and h' is obtained from h by replacing each q that appears as a leaf with the pair $(q, \mathtt{children})$ where $\mathtt{children}$ is the pattern selecting all children of the node at hand.

[3]This is just a technical restriction which forces the transducer to output trees.

[4]Note that for the tree patterns we consider, testing determinism is decidable.

[5]Recall that ε is the empty hedge.

[6]Here, R^* denotes the reflexive transitive closure of the binary relation R.

LEMMA 5.2. DTL *transducers are admissible.*

5.2 Copying and Rearranging

Next, we will give an operational characterization of text-preserving DTL transducers in terms of copying and rearranging. Copying and rearranging for DTL transducers is defined analogously as before.

DEFINITION 5.3. For any DTL transducer T and tree language L, T is *copying over* L if the transduction defined by T is copying over L. It is *rearranging over* L, if the transduction defined by T is rearranging over L.

For characterizing these notions, we need some terminology. For the remainder of the discussion, fix a tree t. We next define when a configuration (q', v') can *follow* a configuration (q, v) in the transduction of t. Formally, we define $(q, v) \rightsquigarrow (q', v')$ to hold whenever there are $\xi, \xi' \in \mathsf{Trees}_\Delta(Q \times \mathsf{Nodes}^t)$ with $(q_0, 1) \Rightarrow^*_{T,t} \xi \Rightarrow_{T,t} \xi'$ such that $\mathsf{frontier}(\xi) = \theta_1(q, v)\theta_2$, $\mathsf{frontier}(\xi') = \theta_1\theta\theta_2$, and $(q', v') \prec \theta$ for $\theta_1, \theta_2, \theta \in (\Delta \cup (Q \times \mathsf{Nodes}^t))^*$. Basically, this says that (q, v) is rewritten into a tree containing the configuration (q', v') at its frontier.

A *path run of* T *over* t is a sequence $(q_0, v_0) \cdots (q_n, v_n)$, where $v_0 = 1$ (the root of t), and, for all $i \in [1, n]$, (q_i, v_i) follows (q_{i-1}, v_{i-1}). A *text path run* is a path run such that $\mathsf{lab}^t(v_n) \in \mathsf{Text}$ and $(q_n, \mathtt{text}) \to \mathtt{text} \in R_{\mathsf{Text}}$. We say that the path run ends in node v_n. Intuitively, a text path run is part of a run (which incidentally is a path) of T on t which outputs a text value. The next definition determines when a transducer copies, that is, outputs the Text-value carried by the same leaf node v, either using two different text path runs (ending in v), or by using one text path run (ending in v) which actually occurs twice in the transduction.

We need one additional notion. Let $(q, v) \rightsquigarrow (q', v')$. Then, we say that (q', v') *doubles* at (q, v) if there is a rule $(q, \varphi) \to h$ and binary patterns $\alpha_1, \alpha_2 \in \mathcal{BP}(\Sigma)$, such that $t \models \varphi(v)$, $t \models \alpha_1(v, v')$, $t \models \alpha_2(v, v')$ and $(q', \alpha_1) \cdot (q', \alpha_2) \prec \mathsf{frontier}(h)$. That is, rewriting (q, v) introduces two occurrences of the configuration (q', v').

The following lemma gives our operational characterization of copying for DTL transducers.

LEMMA 5.4. *A* DTL *transducer T is copying over a tree language L, if and only if there exists a tree $t \in L$, such that there are two different text path runs over t ending in the same node, or one text path run $(q_0, v_0) \cdots (q_n, v_n)$ such that (q_i, v_i) doubles at (q_{i-1}, v_{i-1}) for some $i \in [1, n]$.*

Next, we characterize when a transducer rearranges two Text-values in a tree. Recall that this happens when there are two leaves v and u in t both carrying a Text-value and where $v <_{\text{lex}} u$, but $T(t)$ swaps the order of the Text-values corresponding to v and u. We will show that the latter happens only when during the transduction a rule with rhs h is used such that

- $(q_u, \alpha_1) \cdot (q_v, \alpha_2) \prec \mathsf{frontier}(h)$, and the configurations (q_u, α_1) and (q_v, α_2) proceed to output the text value at u and v, respectively; or,
- when $(q', \alpha') \prec \mathsf{frontier}(h)$ and α' selects u_1, v_1 with $u_1 <_{\text{lex}} v_1$ where (q', u_1) and (q', v_1) proceed to output the Text-value at u and v, respectively.

We are now ready to prove a characterization for when a DTL transducer is rearranging.

LEMMA 5.5. *A* DTL *transducer T is rearranging over a tree language L, if and only if there exists a tree $t \in L$, two text path runs $\theta(q, v)(q_1, v_1) \cdots (q_n, v_n)$ and $\theta(q, v)(p_1, u_1) \cdots (p_m, u_m)$ with $v_n <_{lex} u_m$, a rule $(q, \varphi) \to h$, and binary patterns α_1, α_2, such that $t \models \varphi(v)$, $t \models \alpha_1(v, v_1)$, $t \models \alpha_2(v, u_1)$ and either*
(1) $(p_1, \alpha_2) \cdot (q_1, \alpha_1) \prec \mathsf{frontier}(h)$, or
(2) $(q_1, \alpha_1) = (p_1, \alpha_2)$, $(q_1, \alpha_1) \prec \mathsf{frontier}(h)$ and $u_1 <_{lex} v_1$.

From Lemmas 5.2, 5.4, and 5.5 and from Theorem 3.3 we now obtain a characterization of text-preserving DTL transducers that can be syntactically tested.

THEOREM 5.6. *A* DTL *transducer T is not text-preserving over a tree language L if and only if the condition in Lemma 5.4 or the condition in Lemma 5.5 holds.*

5.3 Transducers with MSO Transitions

In the present section, we consider the instantiation of DTL with MSO-definable patterns, called DTL^{MSO}. The main result of this section is a proof that testing whether a DTL^{MSO} transduction is text-preserving with respect to a regular tree language, is decidable.

We first define trees over the alphabet Σ as relational structures over the vocabulary E, $<$ and $(\mathsf{lab}_\sigma)_{\sigma \in \Sigma}$. The domain of a tree then is its set of nodes. Furthermore, for two nodes v_1, v_2 of a tree t, $E(v_1, v_2)$ if v_2 is a child of v_1, and $v_1 < v_2$ if v_1 and v_2 share the same parent v, and $v_1 <_{\text{lex}} v_2$. Finally, for each $\sigma \in \Sigma$, lab_σ is the set of nodes labelled with σ. *Monadic Second-order Logic* (MSO) then is the extension of First-order Logic (FO) that allows the use of set variables ranging over subsets of the domain, in addition to the individual variables ranging over elements of the domain, as provided by first-order logic. We assume standard practice. We denote by $\phi(x_1, \ldots, x_k)$ that ϕ is a k-ary formula with k free first-order variables. Furthermore, for a tree t, and nodes $v_1, \ldots, v_k$, we denote by $t \models \phi(v_1, \ldots, v_k)$ that ϕ holds when the free variables of ϕ are instantiated by $v_1, \ldots, v_k$. We refer the reader to, e.g., [8] for more background.

Unary and binary patterns will now be defined by unary and binary MSO formulas, respectively. For instance, the formula $\varphi(x)$ defines the pattern $\bigcup_{t \in \mathsf{Trees}_\Sigma} \{t\} \times \{u \mid t \models \varphi(u)\}$. Then, DTL^{MSO} is the instantiation of DTL with MSO-definable patterns. In the definition of DTL^{MSO} transducers, we will simply specify the formulas rather than the patterns they define.

To prove the main result of this section, we need a number of different automata which we define next: tree-jumping automata with MSO transitions, and tree-walking automata with and without MSO tests.

DEFINITION 5.7. A nondeterministic *tree-jumping automaton with* MSO *transitions* over an alphabet Σ, denoted by TJA^{MSO}, is a tuple $B = (Q, \Sigma, \delta, q_0, F, M_u, M_b)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a set of final states, M_u is a finite set of unary MSO formulas, M_b is a finite set of binary MSO formulas, and $\delta : Q \times M_u \times M_b \to 2^Q$ is the transition function. A *run* of B on a tree t that starts at a node v_0 of t and ends at a node v_n of t, is a sequence $(q_0, v_0, \varphi_0, \alpha_0) \cdots (q_n, v_n, \varphi_n, \alpha_n)$, where for each $i \in [0, n-1]$, $t \models \varphi_i(v_i)$, $t \models \alpha_i(v_i, v_{i+1})$, and $q_{i+1} \in \delta(q_i, \varphi_i, \alpha_i)$. A run is *accepting* if $q_n \in F$.

The tree language accepted by B, denoted $\mathcal{L}(B)$, consists of those trees t for which there exists an accepting run starting

at the root of t. If there exists a run of B that starts at a node v_1 in state q_1 and ends at node v_2 in state q_2, we sometimes also say that, when started in state q_1 in v_1, B ends in state q_2 in v_2.

We define two restrictions of tree-jumping automata. A nondeterministic *tree-walking automaton with* MSO *tests*, denoted by TWA^{MSO}, is a TJA^{MSO} where M_b contains only the predicates first-child(x, y), next-sibling(x, y), parent(x, y), previous-sibling(x, y) and equality. Here, the just mentioned predicates have the obvious semantics. For instance, first-child(x, y) holds when y is the first-child of x, and so on. Note that these are easily MSO-definable.

We show that TJA^{MSO} coincides with the class of unranked regular tree languages strongly bearing on a result by Bloem and Engelfriet [4].

LEMMA 5.8. *(1) For each* MSO *formula* $\alpha(x, y)$*, there exists a* TWA^{MSO} $A_\alpha = (Q, \Sigma, \delta, q_0, f, M_u)$*, such that for any tree* t *and nodes* $v, u \in \mathsf{Nodes}^t$*,* $t \models \alpha(v, u)$ *if and only if there exists a run of* A_α *on* t *starting at node* v *at state* q_0 *and ending at node* u *at state* f*. The converse statement also holds.*

(2) For each TJA^{MSO} $B = (Q, \Sigma, \delta, q_0, F, M_u, M_b)$ *there exists a* TWA^{MSO} A*, such that* $\mathcal{L}(B) = \mathcal{L}(A)$*.*

PROOF SKETCH. (1) Bloem and Engelfriet [4] have shown the exact same statement for ranked trees. As the first-child next-sibling encoding between ranked and unranked trees is MSO-definable, the result follows for unranked trees as well.

(2) Essentially, the TWA^{MSO} A simulates the TJA^{MSO} B by invoking the automaton A_α for every transition α of B. $\square$

We note that the constructions in Lemma 5.8 are effective. Furthermore, note that Lemma 5.8(1) together with the fact that MSO-definable unranked tree languages are regular, implies that TWA^{MSO} define the unranked regular tree languages. Therefore, we have the following corollary:

COROLLARY 5.9. TJA^{MSO} *define precisely the unranked regular tree languages.*

The above implies that emptiness of TJA^{MSO} is decidable. In the remainder of this section we will reduce testing whether a DTL^{MSO} transducer is text-preserving to the emptiness test for TJA^{MSO}. More specifically, for a transducer T, we will create a TJA^{MSO} A_T^{copy} and $A_T^{\text{rearrange}}$ such that A_T^{copy} and $A_T^{\text{rearrange}}$ are empty iff T is not copying and not rearranging, respectively. By Theorem 3.3, T then is text-preserving.

We need to introduce a number of automata. We fix a transducer $T = (Q, \Sigma, \Delta, q_0, R_\Sigma, R_{\mathsf{Text}})$. For states $q, q' \in Q$, let $A_T^{q,q'}$ be the TJA^{MSO} which when started in state q on node v of a tree t ends in state q' at node v' iff $(q, v) \rightsquigarrow^* (q', v')$. That is, when T reaches configuration (q, v) when processing t, T can reach configuration (q', v'). By $A_{T,\sigma,\sigma'}^{q,q'}$, we denote the automaton $A_T^{q,q'}$ which additionally checks that the start position is labelled with σ and the end position is labelled with σ'.

We start with $A_T^{\text{rearrange}}$. The automaton will accept a tree t when T is rearranging over $\{t\}$. Intuitively, by Lemma 5.5, the automaton needs to look for the following kind of configurations (ignore the bullets for now):

$$\begin{array}{ccccccc} & & & & (q_1, v_1) & \rightsquigarrow^* & (q_n, v_n) \\ & & & \nearrow^* & \bullet_1 & & \circ_1 \\ (q_0, 1) & \rightsquigarrow^* & (\overset{\bullet}{q}, v) & & & & & (\dagger) \\ & & & \searrow^* & \bullet_2 & & \circ_2 \\ & & & & (p_1, u_1) & \rightsquigarrow^* & (p_m, u_m) \end{array}$$

which satisfy some additional criteria. In particular, we will make sure later that v_n and u_m are text nodes and that $(q, v) \rightsquigarrow (q_1, v_1)$ and $(q, v) \rightsquigarrow (p_1, u_1)$, i.e., we can make these transitions in one step.

The automaton makes use of an extended alphabet. Let $\Sigma_{\text{mark}} = (\Sigma \cup \{\texttt{text}\}) \times 2^{\{\circ, \circ_1, \circ_2, \bullet, \bullet_1, \bullet_2\}}$. Intuitively, the alphabet allows us to mark a node with a set of markers. Then, for a marker $c \in \{\circ, \circ_1, \circ_2, \bullet, \bullet_1, \bullet_2\}$, we slightly abuse notation and say simply that a node is labelled with c, when actually there is a $C \subseteq \{\circ, \circ_1, \circ_2, \bullet, \bullet_1, \bullet_2\}$ with $c \in C$ and a $\sigma \in \Sigma$ such that it is labelled with (σ, C). To check $(\dagger)$, we assume v is labelled with $\bullet$, v_1 is labelled with $\bullet_1$, u_1 is labelled with $\bullet_2$, v_n is labelled with $\circ_1$, u_m is labelled with $\circ_2$. Then, the configurations in $(\dagger)$ occur in a tree t if and only if there is a marking t' of t such that

$$t' \in A_{T,\text{root},\bullet}^{q_0,q} \cap A_{T,\bullet,\bullet_1}^{q,q_1} \cap A_{T,\bullet,\bullet_2}^{q,p_1} \cap A_{T,\bullet_1,\circ_1}^{q_1,q_n} \cap A_{T,\bullet_2,\circ_2}^{p_1,p_m}.$$

Here, the automaton $A_{T,\text{root},\bullet}^{q_0,q}$ tests whether the state q is reachable from q_0 in T. By $A_{(\dagger)}^{q,q_1,p_1}$ we refer to this automaton that tests whether there are states q_n, p_m such that condition $(\dagger)$ holds for given states q, q_1, p_1.

We need some additional regular languages over Σ_{mark}. They can for instance easily be defined in MSO. We assume that they are represented by NTAs:

(i) A_{mark} : for all $c \in \{\bullet, \bullet_1, \bullet_2, \circ_1, \circ_2\}$, there is exactly one node labelled with c. Furthermore, the node labelled with $\circ_1$ (resp., $\circ_2$), is a leaf labelled with $(\texttt{text}, C)$ and $\circ_1 \in C$(resp., $\circ_2 \in C$). We will abuse notation and refer to the nodes labelled with c simply as c.
(ii) $A_{<,\bullet}$: $\bullet_2 <_{\text{lex}} \bullet_1$;
(iii) $A_{<,\circ}$: $\circ_1 <_{\text{lex}} \circ_2$;
(iv) $A_\bullet^\varphi$: $t \models \varphi(\bullet)$;
(v) $A_{\bullet_1}^\alpha$: $t \models \alpha(\bullet, \bullet_1)$; and,
(vi) $A_{\bullet_2}^\beta$: $t \models \beta(\bullet, \bullet_2)$.

We define two automata $A_1^{\text{rearrange}}$ and $A_2^{\text{rearrange}}$ corresponding to the two conditions of Lemma 5.5. To state the conditions, we assume as before that the nodes are marked as in condition $(\dagger)$.

In the first condition, we need to test, in addition to $(\dagger)$, whether $\circ_1 <_{\text{lex}} \circ_2$ and there is a rule $(q, \varphi) \to h$ in T with $(p_1, \beta) \cdot (q_1, \alpha) \prec \mathsf{frontier}(h)$ such that $t \models \varphi(\bullet)$, $t \models \alpha(\bullet, \bullet_1)$, and $t \models \beta(\bullet, \bullet_2)$. In order to formally capture this condition, we define a relation G that contains precisely the tuples $(q, \varphi, p_1, \beta, q_1, \alpha)$ such that there exists a rule $(q, \varphi) \to h$ in T with $(p_1, \beta) \cdot (q_1, \alpha) \prec \mathsf{frontier}(h)$. We are now ready to define $A_1^{\text{rearrange}}$:

$$A_1^{\text{rearrange}} := A_{\text{mark}} \cap A_{<,\circ} \cap \Big(\bigcup_{(q,\varphi,p_1,\beta,q_1,\alpha) \in G} (A_{(\dagger)}^{q,q_1,p_1} \cap A_\bullet^\varphi \cap A_{\bullet_1}^\alpha \cap A_{\bullet_2}^\beta) \Big)$$

In the second condition of Lemma 5.5, we need to test, in addition to $(\dagger)$, whether $\circ_1 <_{\text{lex}} \circ_2$ and there is a rule $(q, \varphi) \to h$ in T with $(q_1, \alpha) \prec \mathsf{frontier}(h)$ such that $(q_1, \alpha) = (p_1, \beta)$, $\bullet_2 <_{\text{lex}} \bullet_1$, $t \models \varphi(\bullet)$, $t \models \alpha(\bullet, \bullet_1)$, and $t \models \alpha(\bullet, \bullet_2)$.

In order to formally capture this condition, we define a relation H that contains precisely the tuples $(q, \varphi, q_1, \alpha)$ such that there exists a rule $(q, \varphi) \to h$ in T with $(q_1, \alpha) \prec \mathsf{frontier}(h)$. We are now ready to define $A_2^{\text{rearrange}}$:

$$A_2^{\text{rearrange}} := A_{\text{mark}} \cap A_{<,\circ} \cap A_{<,\bullet} \left(\bigcup_{(q,\varphi,q_1,\alpha)\in H} (A^{q,q_1,q_1}_{(\dagger)} \cap A^{\varphi}_{\bullet} \cap A^{\alpha}_{\bullet_1} \cap A^{\alpha}_{\bullet_2}) \right)$$

LEMMA 5.10. *Given a* DTL^{MSO} *transducer* T*, the language defined by the* TJA^{MSO} $A_1^{\textit{rearrange}} \cup A_2^{\textit{rearrange}}$ *is nonempty if and only if* T *is rearranging over* Trees_Σ*.*

The overall procedure for A_T^{copy} is similar. The automaton will accept a tree t when T is copying over $\{t\}$. Intuitively, by Lemma 5.4, the automaton needs to look for the following kind of configurations

$$\begin{array}{ccccccc} & & & \nearrow^* & \overset{(q_1,u_1)}{\bullet_1} & \rightsquigarrow^* & \overset{(q_n,v)}{\circ} \\ (q_0,1) & \rightsquigarrow^* & (q,\overset{\bullet}{u}) & & & & & (\ddagger) \\ & & & \searrow^* & \underset{(p_1,u_2)}{\bullet_2} & \rightsquigarrow^* & \underset{(p_m,v)}{\circ} \end{array}$$

which satisfy some additional criteria. The node v should be a text node and the transitions from (q, u) to (q_1, u_1) and to (p_1, u_2) should be possible in one step. Note that both paths in the picture lead to the same node v.

As before the automaton makes use of the extended alphabet Σ_{mark}. With the same conventions as before, to check ($\ddagger$), we assume u is labelled with $\bullet$, u_1 is labelled with $\bullet_1$, u_2 is labelled with $\bullet_2$, and v is labelled with $\circ$. Then, the configurations in ($\ddagger$) occur in a tree t if and only if there is a marking t' of t such that

$$t' \in A^{q_0,q}_{T,\text{root},\bullet} \cap A^{q,q_1}_{T,\bullet,\bullet_1} \cap A^{q,p_1}_{T,\bullet,\bullet_2} \cap A^{q_1,q_n}_{T,\bullet_1,\circ} \cap A^{p_1,p_m}_{T,\bullet_2,\circ}.$$

Analogously as before, the automaton $A^{q_0,q}_{T,\text{root},\bullet}$ tests whether the state q is reachable from q_0 in T. By $A^{q,q_1,p_1}_{(\ddagger)}$ we refer to this automaton that tests whether there are states q_n, p_m such that condition ($\ddagger$) holds for given states q, q_1, p_1.

In addition to the NTA*s* for the regular languages *(i)–(vi)* above, we need

(vii) A'_{mark}: for any $c \in \{\bullet, \bullet_1, \bullet_2, \circ\}$, there is exactly one node labelled with c and the node labelled $\circ$ is a text node in t,

(viii) $A_{\bullet_1\neq\bullet_2}$: $\bullet_1 \neq \bullet_2$.

(ix) $A_{\bullet_1=\bullet_2}$: $\bullet_1 = \bullet_2$.

We define two automata A_1^{copy} and A_2^{copy} corresponding to the two conditions of Lemma 5.4. To state the conditions, we assume as before that the nodes are marked as in condition ($\ddagger$). We also assume that the nodes labelled with $\bullet$, $\bullet_1$, $\bullet_2$, $\circ$ are unique.

In the first condition, we need to test, in addition to ($\ddagger$), if there are two different text path runs over t ending in the same node v. Since the path from $(q_0, 1)$ to (q, u) can have arbitrary length, we can assume w.l.o.g. that the last configuration in which these two text path runs are the same is (q, u). This means that there exists a rule $(q, \varphi) \to h$ in T with $(q_1, \alpha) \prec \mathsf{frontier}(h)$ and $(p_1, \beta) \prec \mathsf{frontier}(h)$ such that $t \models \varphi(\bullet)$, $t \models \alpha(\bullet, \bullet_1)$, and $t \models \beta(\bullet, \bullet_2)$. Furthermore, since (q, u) is the last configuration for which the text path runs are the same, we either have that (a) $q_1 \neq p_1$ or (b) $\bullet_1 \neq \bullet_2$. In order to formalize this condition, let I be the set that contains all tuples $(q, \varphi, q_1, \alpha, p_1, \beta)$ such that $(q, \varphi) \to h$ is a rule in T with $(q_1, \alpha) \prec \mathsf{frontier}(h)$ and $(p_1, \beta) \prec \mathsf{frontier}(h)$. Then, we can define A_{1a}^{copy} and A_{1b}^{copy}:

$$A_{1a}^{\text{copy}} := A'_{\text{mark}} \cap \left(\bigcup_{(q,\varphi,p_1,\beta,q_1,\alpha)\in I \wedge p_1\neq q_1} (A^{q,q_1,p_1}_{(\ddagger)} \cap A^{\varphi}_{\bullet} \cap A^{\alpha}_{\bullet_1} \cap A^{\beta}_{\bullet_2}) \right)$$

$$A_{1b}^{\text{copy}} := A'_{\text{mark}} \cap A_{\bullet_1\neq\bullet_2} \left(\bigcup_{(q,\varphi,p_1,\beta,q_1,\alpha)\in I} (A^{q,q_1,p_1}_{(\ddagger)} \cap A^{\varphi}_{\bullet} \cap A^{\alpha}_{\bullet_1} \cap A^{\beta}_{\bullet_2}) \right)$$

In the second condition of Lemma 5.4 we need to test whether there exists a text path run with two consecutive configurations such that the second one doubles at the first one. W.l.o.g., we can assume that (q_1, u_1) doubles at (q, u). This means that there exists a rule $(q, \varphi) \to h$ in T with $(q_1, \alpha)\cdot(q_1, \beta) \prec \mathsf{frontier}(h)$, such that $t \models \varphi(\bullet)$, $t \models \alpha(\bullet, \bullet_1)$, $t \models \beta(\bullet, \bullet_2)$, and $u_1 = u_2$, i.e., the node u_1 is labelled by $\bullet_1$ and $\bullet_2$. In order to formalize this condition, let J be the set that contains all tuples $(q, \varphi, q_1, \alpha, \beta)$ such that $(q, \varphi) \to h$ is a rule in T with $(q_1, \alpha) \cdot (q_1, \beta) \prec \mathsf{frontier}(h)$. Then, we can define A_2^{copy}:

$$A_2^{\text{copy}} := A'_{\text{mark}} \cap A_{\bullet_1=\bullet_2} \cap \left(\bigcup_{(q,\varphi,q_1,\alpha,\beta)\in J} (A^{q,q_1,q_1}_{(\ddagger)} \cap A^{\varphi}_{\bullet} \cap A^{\alpha}_{\bullet_1} \cap A^{\beta}_{\bullet_2}) \right)$$

LEMMA 5.11. *Given a* DTL^{MSO} *transducer* T*, the language defined by the* TJA^{MSO} $A_{1a}^{\textit{copy}} \cup A_{1b}^{\textit{copy}} \cup A_2^{\textit{copy}}$ *is nonempty if and only if* T *is copying over* Trees_Σ*.*

To test whether a DTL^{MSO} transducer T is text-preserving over a regular tree language $\mathcal{L}$, it follows from Theorem 3.3 that it is sufficient to test the conditions in Lemmas 5.10 and 5.11. We therefore obtain the following theorem.

THEOREM 5.12. *Given a* DTL^{MSO} *transducer* T *and a regular language* $\mathcal{L}$*, it is decidable whether* T *is text-preserving.*

As satisfiability of MSO is non-elementary over trees, testing whether a DTL^{MSO} transducer is text-preserving is non-elementary as well. Indeed, consider a transducer which outputs two copies of the input tree when an MSO-sentence is satisfied at the root. Then this transduction is text-preserving if and only if the formula is not satisfiable.

5.4 Transducers with XPath Transitions

We recall the definition of Core XPath. Since we do not use other fragments of XPath, we will use the term XPath to refer to Core XPath. We use the definition from [24]. We denote XPath node expressions by $\varphi, \psi, \ldots$ and path expressions by $\alpha, \beta, \ldots$

DEFINITION 5.13. XPath node expressions and path expressions are defined by simultaneous induction, as follows:
Path expressions: $\alpha ::= R \mid R^* \mid \cdot \mid \alpha/\beta \mid \alpha \cup \beta \mid \alpha[\varphi]$
Node expressions: $\varphi ::= \sigma \mid \langle\alpha\rangle \mid \top \mid \neg\varphi \mid \varphi \wedge \psi$

Here, $\sigma \in \Sigma$ is a label test and R is one of the relations child, parent, next-sibling, or previous-sibling. We denote these relations by $\downarrow$, $\uparrow$, $\rightarrow$, and $\leftarrow$, respectively.

$$
\begin{array}{lcl}
[\![R]\!]_{\mathsf{PExpr}} & = & \text{the pairs in relation } R \\
[\![R^*]\!]_{\mathsf{PExpr}} & = & \text{reflexive and transitive closure of } [\![R]\!]_{\mathsf{PExpr}} \\
[\![\cdot]\!]_{\mathsf{PExpr}} & = & \{(u,u) \mid u \in \mathsf{Nodes}^t\} \\
[\![\alpha/\beta]\!]_{\mathsf{PExpr}} & = & \{(u,w) \mid \exists v.(u,v) \in [\![\alpha]\!]_{\mathsf{PExpr}} \text{ and } \\
 & & \qquad (v,w) \in [\![\beta]\!]_{\mathsf{PExpr}}\} \\
[\![\alpha \cup \beta]\!]_{\mathsf{PExpr}} & = & [\![\alpha]\!]_{\mathsf{PExpr}} \cup [\![\beta]\!]_{\mathsf{PExpr}} \\
[\![\alpha[\varphi]]\!]_{\mathsf{PExpr}} & = & \{(u,v) \in [\![\alpha]\!]_{\mathsf{PExpr}} \mid v \in [\![\varphi]\!]_{\mathsf{NExpr}}\} \\
\\
[\![\sigma]\!]_{\mathsf{NExpr}} & = & \{u \in \mathsf{Nodes}^t \mid \mathsf{lab}^t(u) = \sigma\} \\
[\![\langle\alpha\rangle]\!]_{\mathsf{NExpr}} & = & \{u \in \mathsf{Nodes}^t \mid \exists v \in \mathsf{Nodes}^t.(u,v) \in [\![\alpha]\!]_{\mathsf{PExpr}}\} \\
[\![\top]\!]_{\mathsf{NExpr}} & = & \mathsf{Nodes}^t \\
[\![\neg\varphi]\!]_{\mathsf{NExpr}} & = & \mathsf{Nodes}^t \setminus [\![\varphi]\!]_{\mathsf{NExpr}} \\
[\![\varphi \wedge \psi]\!]_{\mathsf{NExpr}} & = & [\![\varphi]\!]_{\mathsf{NExpr}} \cap [\![\psi]\!]_{\mathsf{NExpr}}
\end{array}
$$

Table 1: Semantics of XPath.

The semantics of XPath expressions relative to a tree t is given by the functions $[\![\ ]\!]^t_{\mathsf{PExpr}}$ and $[\![\ ]\!]^t_{\mathsf{NExpr}}$. These functions are defined in Table 1. We omit the tree t if it is clear from the context. In the remainder of this section, we denote by $\varphi(u)$ and $\alpha(u,v)$ that $u \in [\![\varphi]\!]_{\mathsf{NExpr}}$ and $(u,v) \in [\![\alpha]\!]_{\mathsf{PExpr}}$, respectively.

DEFINITION 5.14. A $\text{DTL}^{\text{XPath}}$ transducer is a DTL transducer in which all unary and binary patterns are XPath node and XPath path expressions, respectively.

EXAMPLE 5.15. The following DTL transducer selects the descriptions, ingredients, and instructions from all recipes that have at least three positive comments in our running example.

$$
\begin{array}{lcl}
(q_0, \texttt{recipes}) & \to & \texttt{recipes}((q, \downarrow)) \\
(q, \varphi) & \to & \texttt{recipe}((q, \downarrow)) \\
(q, \sigma) & \to & \sigma((q, \downarrow)) \qquad (\sigma \in \{\texttt{description}, \texttt{ingredients}, \texttt{br}, \texttt{instructions}\}) \\
(q, \texttt{item}) & \to & (q, \downarrow) \\
(q, \texttt{text}) & \to & \texttt{text}
\end{array}
$$

where

$$
\varphi = \texttt{recipe} \wedge \big\langle \downarrow[\texttt{comments}]/\downarrow[\texttt{positive}] /\downarrow[\texttt{comment}]/\to[\texttt{comment}]/\to[\texttt{comment}]\big\rangle.
$$

As in Section 5.3, we will reduce testing whether a $\text{DTL}^{\text{XPath}}$ transducer is text-preserving to the emptiness test of an automaton. The beginning of our construction is similar to the one in Section 5.3. We will simulate partial computations of a $\text{DTL}^{\text{XPath}}$ transducer by a *tree-jumping automaton with XPath transitions* ($\text{TJA}^{\text{XPath}}$). A $\text{TJA}^{\text{XPath}}$ is a TJA^{MSO} in which all unary and binary formulas are XPath node and XPath path expressions, respectively.

Fix a $\text{DTL}^{\text{XPath}}$ transducer $T = (Q, \Sigma, \Delta, q_0, R_\Sigma, R_{\mathsf{Text}})$. For states $q, q' \in Q$ and $\sigma, \sigma' \in \Sigma$, we denote by $A^{q,q'}_{T,\sigma,\sigma'}$ the $\text{TJA}^{\text{XPath}}$ which, when started in state q on some node v in a tree t ends in state q' at node v' if and only if $(q,v) \rightsquigarrow^* (q',v')$ in T. Furthermore, it checks whether v is labelled by σ and v' is labelled by σ'. Furthermore, each of these automata can be constructed from T in linear time.

It remains to show that we can perform a similar construction as in Section 5.3. However, in this section, we also have to mind the complexity bounds. In order to avoid a large blow-up when performing unions and intersections of automata, we will translate $\text{TJA}^{\text{XPath}}$ to two-way alternating tree walking automata using techniques from [3].

LEMMA 5.16. *For each* $\text{TJA}^{\text{XPath}}$*, we can construct an equivalent* 2ATWA *in polynomial time.*

Two-way alternating tree walking automata have the interesting property that their intersections and unions can be constructed very efficiently. More precisely, for 2ATWA*s* $A_1, \ldots, A_n$, we can construct a 2ATWA for $\mathcal{L}(A_1) \cup \cdots \cup \mathcal{L}(A_n)$ and a 2ATWA for $\mathcal{L}(A_1) \cap \cdots \cap \mathcal{L}(A_n)$ in time linear in $|A_1| + \cdots + |A_n|$. This implies that we can perform the constructions from Section 5.3 on the 2ATWA*s* in polynomial time.

LEMMA 5.17. *Given a* $\text{DTL}^{\text{XPath}}$ *transducer* T *and an* NTA N*, we can construct, in polynomial time, a* 2ATWA A *such that* T *is text-preserving over* $\mathcal{L}(N)$ *if and only if* $\mathcal{L}(A) = \emptyset$.

Since testing emptiness of a 2ATWA is in EXPTIME [5], we now know that testing whether a $\text{DTL}^{\text{XPath}}$ transducer is text-preserving is in EXPTIME as well. Furthermore, since XPath satisfiability w.r.t. a DTD is EXPTIME-complete [17, 21], testing whether a $\text{DTL}^{\text{XPath}}$ transducer is text-preserving is also EXPTIME-hard.

THEOREM 5.18. *Given a* $\text{DTL}^{\text{XPath}}$ *transducer* T *and an* NTA N*, deciding whether* T *is text-preserving over* $\mathcal{L}(N)$ *is* EXPTIME*-complete.*

Actually, EXPTIME-hardness can also be obtained through a reduction from emptiness of deterministic tree-walking automata known to be EXPTIME-complete [20, 23].

6. RELATED WORK

The advent of XML induced a renewed interest in tree transducers. Milo, Suciu, and Vianu used k-pebble transducers in their seminal paper [18] as a general model for XML transformations, encompassing a variety of languages like XML-QL, XQuery, and XSLT. Engelfriet and Maneth showed that these form a subclass of the macro tree transducers [7].

In the present paper, we focus on XSLT-like transformations. The simplest of these are the top-down uniform transducers introduced in [13]. A more accurate formalization capturing the navigational power of XSLT as well, is given by DTL as introduced by Maneth and Neven [11]. We extend DTL to deal with text values, albeit in a modest way in accordance with our focus on simple transformations: when encountering Text-values, they can only be immediately output or discarded. Transducer models for XSLT equipped to employ text- or data values during computation have been considered in [2, 19]. To simplify presentation, we disregarded the construction functions employed in [11]. Intuitively, the difference is as follows. DTL as defined here associates one state with every binary pattern, while through the use of construction functions, the selection mechanism is generalized, by associating several states to every binary pattern. As this level of expressiveness is not present in XSLT, we decided to neglect it.

The most deeply studied problem concerning XML and transducers is undoubtedly the XML typechecking problem [18, 13, 14, 15, 9, 12, 22, 10], where it is asked whether $T(t) \in S_{\text{out}}$ for all $t \in S_{\text{in}}$ for given input and output

schema S_{in} and S_{out} and transducer T. The typechecking problem is quite different from the problem considered in this paper, both in statement as in computational difficulty. For instance, typechecking top-down uniform tree transducers against unranked tree automata is already EXPTIME-complete while testing whether one is text-preserving is in PTIME for the corresponding setting.

Finally, we note that our Text-trees are different from the data trees in, e.g., [6], where each node carries a label from Σ and a label from Text. Moreover, unlike the logics and automata described in [6], the presented transducer model can not perform equality tests on Text-values.

7. CONCLUSION

Text-preserving transformations can be seen as a well-defined simple class of transformations for text-centric XML documents. In this paper, we showed that for XSLT-style transformations this property is decidable and even tractable for the top-down case.

From the employed proof technique it follows that for every transducer in one of the above mentioned classes, the maximal subset of the input schema can be computed on which the transformation is text-preserving. Indeed, we have actually shown that the class of trees on which a transducer is not text-preserving is a regular tree language. It would be interesting to pinpoint the exact size of a representation of that maximal subclass.

Another interesting question for future research is whether there is a normal form for text-preserving transformations in the considered formalisms.

Finally, although we think that mappings which restrict the output to be a subsequence of the input are relevant, there might be other notions that could be interesting in the context of text-centric mappings. It would therefore be interesting to develop more general notions of which text-preserving is a special case.

Actually, our employed proof technique already allows to test stronger properties than just text-preserving. For instance, we can specify a set of tree types (for example, a set of node labels) under which text values in the subtree can not be deleted. In our running example, we could require, e.g., that the transformation is text-preserving and that it does not delete any text values under a node labelled `instructions`. These more flexible tests do not influence the complexity of the problem.

Acknowledgment

We thank Maarten Marx for bringing the notion of text-preserving mappings and its relevance for text-centric documents to our attention.

8. REFERENCES

[1] J. Albert, D. Giammerresi, D. Wood. Normal form algorithms for extended context free grammars. *Theor. Comp. Sc.*, 267(1–2):35–47, 2001.

[2] G. J. Bex, S. Maneth, F. Neven. A formal model for an expressive fragment of XSLT. *Inf. Syst.*, 27(1):21–39, 2002.

[3] H. Björklund, W. Gelade, W. Martens. Incremental XPath evaluation. *ACM Trans. Database Syst.*, 35(4), 2011.

[4] R. Bloem, J. Engelfriet. A comparison of tree transductions defined by monadic second order logic and by attribute grammars. *J. Comput. Syst. Sci.*, 61(1):1–50, 2000.

[5] M. Bojanczyk. Tree-walking automata. In *LATA*, pages 1–2, 2008.

[6] M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin. Two-variable logic on data trees and XML reasoning. *Journal of the ACM*, 56(3), 2009.

[7] J. Engelfriet, S. Maneth. A comparison of pebble tree transducers with macro tree transducers. *Acta Inf.*, 39(9):613–698, 2003.

[8] L. Libkin. *Elements Of Finite Model Theory.* Springer Verlag, 2004.

[9] S. Maneth, A. Berlea, T. Perst, H. Seidl. XML type checking with macro tree transducers. In *PODS*, pages 283–294, 2005.

[10] S. Maneth, S. Friese, H. Seidl. Type checking of tree walking transducers. In *Modern Applications of Automata Theory.* World Scientific Publishing, 2011.

[11] S. Maneth, F. Neven. Structured document transformations based on XSL. In *DBPL*, pages 80–98, 1999.

[12] S. Maneth, T. Perst, H. Seidl. Exact XML type checking in polynomial time. In *ICDT*, pages 254–268, 2007.

[13] W. Martens, F. Neven. On the complexity of typechecking top-down XML transformations. *Theor. Comp. Sc.*, 336(1):153–180, 2005.

[14] W. Martens, F. Neven. Frontiers of tractability for typechecking simple XML transformations. *J. Comput. Syst. Sci.*, 73(3):362–390, 2007.

[15] W. Martens, F. Neven, M. Gyssens. Typechecking top-down XML transformations: Fixed input or output schemas. *Inf. and Comput.*, 206(7):806–827, 2008.

[16] W. Martens, F. Neven, T. Schwentick. Complexity of decision problems for XML schemas and chain regular expressions. *SIAM J. Comput.*, 39(4):1486–1530, 2009.

[17] M. Marx. XPath with conditional axis relations. In *EDBT*, pages 477–494, 2004.

[18] T. Milo, D. Suciu, V. Vianu. Typechecking for XML transformers. *J. Comput. Syst. Sci.*, 66(1):66–97, 2003.

[19] F. Neven. On the power of walking for querying tree-structured data. In *PODS*, pages 77–84, 2002.

[20] F. Neven. Attribute grammars for unranked trees as a query language for structured documents. *J. Comput. Syst. Sci.*, 70(2):221–257, 2005.

[21] F. Neven, T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Log. Meth. in Comp. Sc.*, 2(3), 2006.

[22] T. Perst and H. Seidl. Macro forest transducers. *Inf. Process. Lett.*, 89(3):141–149, 2004.

[23] M. Samuelides and L. Segoufin. Complexity of pebble tree-walking automata. In *FCT*, pages 458–469, 2007.

[24] B. ten Cate and C. Lutz. The complexity of query containment in expressive fragments of XPath 2.0. *Journal of the ACM*, 56(6), 2009.