



# Equivalence of Deterministic Nested Word to Word Transducers

Slawomir Staworko, Grégoire Laurence, Aurélien Lemay, Joachim Niehren

## ► To cite this version:

Slawomir Staworko, Grégoire Laurence, Aurélien Lemay, Joachim Niehren. Equivalence of Deterministic Nested Word to Word Transducers. Fundamentals of Computation Theory, Maciej Gebala and Malgorzata Korzeniowska, Sep 2009, Wroclaw, Poland. pp.310-322, 10.1007/978-3-642-03409-1\_28 . inria-00423961

**HAL Id: inria-00423961**

**<https://inria.hal.science/inria-00423961>**

Submitted on 13 Oct 2009

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Equivalence of Deterministic Nested Word to Word Transducers

Sławomir Staworko<sup>13</sup> and Grégoire Laurence<sup>23</sup> and Aurélien Lemay<sup>23</sup> and  
Joachim Niehren<sup>13</sup>

<sup>1</sup> INRIA, Lille

<sup>2</sup> University of Lille

<sup>3</sup> Mostrare project, INRIA & LIFL (CNRS UMR8022)

**Abstract.** We study the equivalence problem of deterministic nested word to word transducers and show it to be surprisingly robust. Modulo polynomial time reductions, it can be identified with 4 equivalence problems for diverse classes of deterministic non-copying order-preserving transducers. In particular, we present polynomial time back and fourth reductions to the morphism equivalence problem on context free languages, which is known to be solvable in polynomial time.

**Keywords:** trees, transducers, automata, context-free grammars, XML.

## 1 Introduction

Nested word automata (NAs) [1] are tree automata, that operate on linearizations of unranked trees in streaming manner. All nodes of the tree are visited twice, as usual in preorder traversals. At the first visit (opening event), a symbol is to be pushed onto a stack that is popped at the second visit (closing event). NAs were introduced as a reformulation of visibly pushdown automata [2] and proved equivalent to pushdown forest automata [11] and streaming tree automata [6].

More formally, a nested word over  $\Sigma$  is a word of parenthesis in  $\hat{\Sigma} = \{\text{op}, \text{cl}\} \times \Sigma$ , such that all opening parenthesis are properly closed. We consider nested words as linearizations of unranked trees. For instance, the linearization of  $a(b(c), d)$  is the nested word  $(\text{op}, a) \cdot (\text{op}, b) \cdot (\text{op}, c) \cdot (\text{cl}, c) \cdot (\text{cl}, b) \cdot (\text{op}, d) \cdot (\text{cl}, d) \cdot (\text{cl}, a)$ , or in XML notation  $\langle a \rangle \langle b \rangle \langle c \rangle \langle /c \rangle \langle /b \rangle \langle d \rangle \langle /d \rangle \langle /a \rangle$ . NAs process nested words from left to right, while passing finite state information from opening to matching closing parenthesis.

In this paper, we study nested word to word transducers (N2Ws). These process input nested words such as NAs, while producing output letters in parallel, both from left to right. N2Ws are pushdown transducers that must push at opening and pop at closing parenthesis (see Fig. 2 for an example of a N2W-transduction). N2Ws were first introduced in [15], where they were called *visibly pushdown transducers*. Our notion is slightly more general in that we do not impose any well-nesting conditions on output words. Furthermore, we do not assume *synchronization*, i.e., that deletion and renaming operations on matching

opening and closing parenthesis are in sync (see Fig. 3). Synchronization is a sufficient restriction to make type checking decidable.

N2Ws  $T$  with input alphabet  $\Sigma$  and output alphabet  $\Delta$  define relations  $\llbracket T \rrbracket : \mathcal{T}_\Sigma \times \Delta^*$  mapping unranked trees with labels in  $\Sigma$  to words with letters in  $\Delta$ . They have rules of the following two forms:

$$q \xrightarrow{\text{op } a/w:\gamma} q' \quad \text{or} \quad q \xrightarrow{\text{cl } a/w:\gamma} q'$$

where  $q, q'$  are states,  $\gamma$  a stack symbol,  $a \in \Sigma$  an input label, and  $w \in \Delta^*$  a word of output letters. An opening rule applies in state  $q$ , consumes an opening parenthesis ( $\text{op}, a$ ) from the input, pushes a symbol  $\gamma$  onto the stack, concatenates  $w$  to the output word, and goes into state  $q'$ . Closing rules are applied similarly, except that they pop symbol  $\gamma$  from the stack. Note that we do not permit rules with  $\epsilon$  input (blind insertion) as they are incompatible with determinism.

We call an N2W *deterministic* or a dn2W, if 1. it has a unique initial state, 2. opening rules are determined by the current state  $q$  and input label  $a$ , and 3. closing rules are determined by  $q$ ,  $a$ , and the current stack symbol  $\gamma$ . Every dn2W  $T$  with input alphabet  $\Sigma$  and output alphabet  $\Delta$  defines a partial function  $\llbracket T \rrbracket : \mathcal{T}_\Sigma \rightarrow \Delta^*$ . Since dn2Ws can be identified with deterministic push-down transducer, it follows from the very general result of Szenergues [16] that equivalence of dn2Ws is decidable.

We call an N2W *top-down*, if it is top-down as an NA, i.e., if all its closing rules have the form  $q \xrightarrow{\text{cl } a/w:\gamma} q'$ , so that the closing state is already determined by the stack symbol pushed at opening time. A *top-down deterministic* N2W or a  $\text{dn2W}^\downarrow$  is a deterministic N2W that is top-down. Similarly to top-down deterministic tree automata,  $\text{dn2W}^\downarrow$  are less expressive than dn2Ws.

We consider two kinds of standard transducers on *ranked* trees [8, 4], ranked tree to word transducers (R2Ws) which may either be top-down deterministic ( $\text{dr2W}^\downarrow$ s) or bottom-up deterministic ( $\text{dr2W}^\uparrow$ s). Our main results are polynomial time reductions between all of the following problems:

1. equivalence of dn2Ws,
2. equivalence of  $\text{dn2W}^\downarrow$ s,
3. equivalence of  $\text{dr2W}^\downarrow$ s that are non-copying and order-preserving,
4. equivalence of  $\text{dr2W}^\uparrow$ s that are non-copying and order-preserving,
5. equivalence of morphisms on context free languages.

Plandowski [13, 14] has shown that the last problem is decidable in polynomial time. Since all of our reductions are in polynomial time, all of the above problems are decidable in polynomial time too. This result has not been stated before for any of the first 4 problems. It should be noticed, that equivalence of tree-to-tree transducers can be reduced to equivalence of tree-to-word transducers that linearize output trees. As a consequence, we obtain polynomial time algorithms for deciding equivalence of top-down and bottom-up deterministic tree-to-tree transducers that are non-copying and order-preserving.

Our main motivation is to define XML document transformation in a deterministic manner, as for instance by the W3C standard XSLT. Various classes

of tree transducers have been proposed to this end before [10,9]. Also the class of  $\text{dN2Ws}^\downarrow$  does neither allow tree copying nor tree permutation operations. However, we believe this is a good compromise between complexity results and expressive power. For instance, it can express more than non-copying and order-preserving transducers from [10]. In particular, it can express most usual “stylesheet-like” XML to HTML transformations (e.g. Fig. 2). Also, it is suitable to model streamed-like processing.

*Related work.* As shown by [5], the equivalence of top-down deterministic tree-to-tree transducers ( $\text{dR2Rs}^\downarrow$ ) can be decided in polynomial time. This result is orthogonal to ours. It is more general in that copying and permutations are permitted, but cannot capture word output of  $\text{dR2Ws}^\downarrow$ s, since lacking the word concatenation operation. Every tree-to-tree transducer can be transformed into a tree-to-word transducer by composition with the yield function. This has been done for instance in order to measure the generating capacities of tree transducers [8]. However, since different trees may have the same yield, one cannot reduce the equivalence problem of tree-to-tree transducers to that of tree-to-word transducers this way (if not adding concatenation operations or macros).

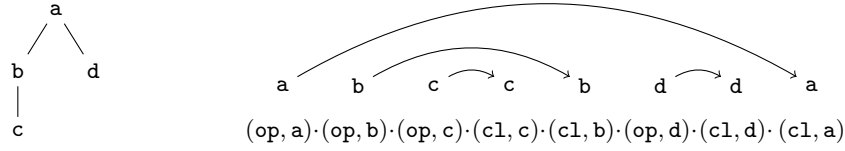
The methods used in this paper are reminiscent of those used by Culik and Karhumäki to show decidability of equivalence of synchronized deterministic pushdown automata and transducers, e.g. see [7]. Intuitively, two deterministic pushdown automata are synchronized if their stacks have *almost* the same height throughout every computation. Then, their execution can be simulated with a single pushdown automaton, very much in the spirit of Lemma 3. In this setting the complexity of this approach is exponential. We take advantage of the observation that two N2Ws can be viewed as synchronized pushdown transducers, and moreover, the stacks always have *exactly* the same height.

## 2 Nested Word to Word Transducers

Let  $\Sigma$  be a finite set of labels. We define the set of unranked trees  $\mathcal{T}_\Sigma$  to be the least set that contains all pairs  $a(t_1, \dots, t_n)$  consisting of a label  $a \in \Sigma$  and a tuple of unranked trees  $(t_1, \dots, t_n) \in (\mathcal{T}_\Sigma)^n$  for some  $n \geq 0$ , i.e.  $\mathcal{T}_\Sigma = \Sigma \times \bigcup_{n \geq 0} (\mathcal{T}_\Sigma)^n$ . When writing XML documents into files, unranked trees are usually linearized in a preorder traversal to words  $\text{lin}(t) \in \hat{\Sigma}^*$  where  $\hat{\Sigma} = \{\text{op}, \text{cl}\} \times \Sigma$ .

$$\text{lin}(a(t_1, \dots, t_n)) = (\text{op}, a) \cdot \text{lin}(t_1) \cdot \dots \cdot \text{lin}(t_n) \cdot (\text{cl}, a)$$

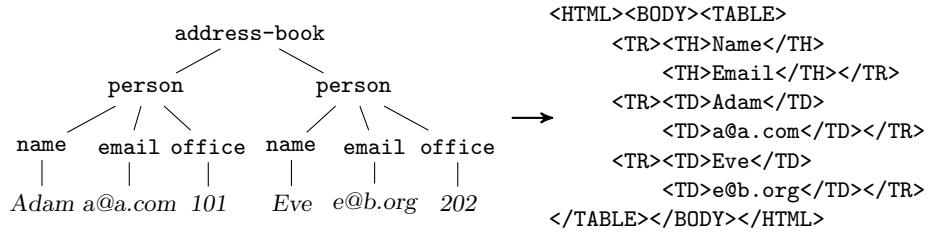
Linearizations of unranked trees are words in  $\hat{\Sigma}^*$  that are well-nested in that all opening parenthesis are properly closed. See Fig. 1 for an example. We define the set of all *nested words* over  $\Sigma$  by  $N_\Sigma = \{\text{lin}(t) \mid t \in \mathcal{T}_\Sigma\}$ . More general definitions can be found in the literature. We impose 4 restrictions, of which only the first matters technically, while the others simplify presentation: (1) No dangling opening or closing parenthesis. As a consequence, segments of nested words between two positions do not need to be nested words. (2) No internal letters, which are neither opening nor closing. (3) No corresponding parenthesis



**Fig. 1.** Example of a tree and its linearization into a nested word.

with different labels (for instance,  $(\text{op } a)(\text{cl } b)$  is not well nested). (4) No hedges, i.e. sequences of unranked trees: these can be represented by unranked trees by adding artificial roots.

A *nested word to word transducer* (N2W)  $T$  is a tuple of finite sets  $T = (\Sigma, \Delta, Q, \Gamma, \text{rul}, \text{init}, \text{fin})$  such that  $\text{init}, \text{fin} \subseteq Q$  and  $\text{rul} \subseteq Q^2 \times \hat{\Sigma} \times \Delta^* \times \Gamma$ . There are labels  $a \in \Sigma$  of input trees, an alphabet for output words  $u \in \Delta^*$ , states  $q \in Q$ , and stack symbols  $\gamma \in \Gamma$ . We denote rules  $r$  equal to  $((q, q'), (\alpha, a), u, \gamma)$  as  $q \xrightarrow{\alpha a/u:\gamma} q'$ . We denote  $\text{lhs}(r) = q$  (for left hand side),  $\text{rhs}(r) = q'$  (for right hand side),  $\text{act}(r) = (\alpha, a)$  (for action),  $\text{output}(r) = u$  and  $\text{ssy}(r) = \gamma$  (for stack symbol). Note that our definition of N2Ws excludes rules with  $\epsilon$ -input. A *nested word automaton* (NA) is a N2W which always outputs the empty word, so that all rules are of the form  $q \xrightarrow{\alpha a/\epsilon:\gamma} q'$ . The main interest of NAs compared to tree automata [1] is that they combine top-down and bottom-up processing by operating on preorder linearizations of unranked trees (nested words). N2W is a suitable choice to model transformations that do not require copying or reordering, for instance simple XML to HTML transformations (Fig. 2).



**Fig. 2.** An example of XML to HTML transformation.

An N2W defines a relation  $\llbracket T \rrbracket \subseteq \mathcal{T}_\Sigma \times \Delta^*$ . We will present two equivalent definitions of  $\llbracket T \rrbracket$ . The first interprets  $T$  as a pushdown transducer, which inputs nested words in  $N_\Sigma$  and outputs words in  $\Delta^*$ . The pushdown is “visible” in that it pushes (resp. pops) when reading opening (resp. closing) parenthesis. We define the set of configurations of an N2W by  $\hat{\Sigma}^* \times Q \times \Gamma^* \times \Delta^*$ . Every configuration

$C = (w, q, S, u)$  has an input word  $w$ , a state  $q$ , a stack  $S$  and an output word  $u$ . We call  $C$  initial for  $t \in \mathcal{T}_\Sigma$  if  $w = \text{lin}(t)$ ,  $q \in \text{init}$ ,  $S = \epsilon$ , and  $u = \epsilon$ . We call  $C$  a final configuration for  $u$  if  $w = \epsilon$ ,  $q \in \text{fin}$ , and  $S = \epsilon$ . An opening rule  $q \xrightarrow{\text{op } a/u':\gamma} q'$  applies in state  $q \in Q$ , reads an opening parenthesis  $(\text{op}, a)$  where  $a \in \Sigma$  pushes  $\gamma \in \Gamma$  to the stack, concatenates  $u'$  to the end of the current output word, and goes to state  $q' \in Q$ . A closing rule  $q \xrightarrow{\text{cl } a/u':\gamma} q'$  applies in state  $q \in Q$ , reads a closing parenthesis  $(\text{cl}, a)$  under the condition that the top-most symbol on the stack is  $\gamma$ . It then pops  $\gamma$  from the stack, concatenates  $u'$  to the end of the current output word, and goes to state  $q'$ . The transitions between configurations is defined as follow:

$$\frac{q \xrightarrow{\text{op } a/u':\gamma} q' \in \text{rul}}{((\text{op}, a) \cdot w, q, S, u) \rightarrow (w, q', \gamma \cdot S, u \cdot u')} \quad \frac{q \xrightarrow{\text{cl } a/u':\gamma} q' \in \text{rul}}{((\text{cl}, a) \cdot w, q, \gamma \cdot S, u) \rightarrow (w, q', S, u \cdot u')}$$

We say that a N2W  $T$  transforms unranked tree  $t$  to word  $u$  if  $T$  licences a transition sequence  $C \rightarrow^* C'$  where  $C$  is initial for  $t$  and  $C'$  final for  $u$ , and define the relation  $\llbracket T \rrbracket = \{(t, u) \in \mathcal{T}_\Sigma \times \Delta^* \mid T \text{ transforms } t \text{ to } u\}$ . Also, an NA  $A$  recognizes the tree language  $L(A) = \{t \in \mathcal{T}_\Sigma \mid (t, \epsilon) \in \llbracket A \rrbracket\}$ .

The second characterization of  $\llbracket T \rrbracket$  is based on runs, which annotate opening and closing events of nodes of unranked trees by rules. As usual, a single run of an automata on a tree is supposed to capture all information of all intermediate configurations leading to acceptance. We define the set of nodes of an unranked trees by  $\text{nod}(a(t_1, \dots, t_n)) = \{\epsilon\} \cup \{i \cdot \pi \mid \pi \in \text{nod}(t_i)\}$ . An event is a member of the set  $\widehat{\text{nod}(t)}$ , i.e. an element of a preorder traversal of  $t$ . The set  $\widehat{\text{nod}(t)}$  is totally ordered: the first event is the opening of the root  $(\text{op}, \epsilon)$ , the last event is the closing of the root  $(\text{cl}, \epsilon)$ , etc. We write  $(\alpha, \pi) < (\alpha', \pi')$  if  $(\alpha, \pi)$  is properly before  $(\alpha', \pi')$  in this order and define  $\text{pr}(\alpha, \pi) \in \widehat{\text{nod}(t)}$  to be the immediate predecessor of  $(\alpha, \pi) \in \widehat{\text{nod}(t)}$  in that order.

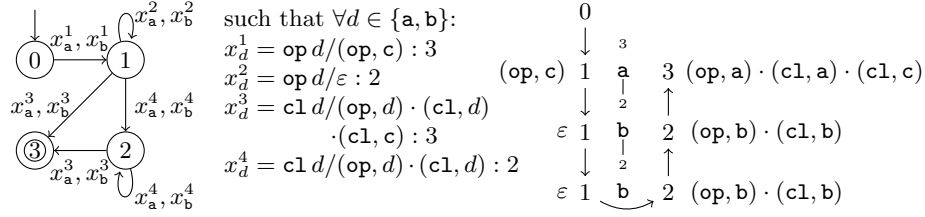
A run of an N2W on a tree  $t \in \mathcal{T}_\Sigma$  is a function  $R : \widehat{\text{nod}(t)} \rightarrow \text{rul}$  such that  $\text{lhs}(R(\text{op}, \epsilon)) \in \text{init}$ , and for any  $(\alpha, \pi) \in \widehat{\text{nod}(t)}$ ,  $\text{ssy}(R(\text{op}, \pi)) = \text{ssy}(R(\text{cl}, \pi))$ ,  $\text{rhs}(R(\text{pr}(\alpha, \pi))) = \text{lhs}(R(\alpha, \pi))$ , and  $\text{act}(R(\alpha, \pi)) = (\alpha, a)$ , where  $a$  is the label of the node  $\pi$  in  $t$ . We call  $R$  *successful* if  $\text{rhs}(R(\text{cl}, \epsilon)) \in \text{fin}$ .

See Fig. 3 for an example of a successful run.

**Lemma 1.**  $\llbracket T \rrbracket = \{(t, \text{output}(R(e_1)) \cdot \dots \cdot \text{output}(R(e_n))) \mid t \in \mathcal{T}_\Sigma, R \text{ successful run of } T \text{ on } t, \text{ the events of } t \text{ are } e_1 < \dots < e_n\}$ .

An N2W  $T$  is *deterministic* or an dN2W if it satisfies (1) for all  $q \in Q$  and  $a \in \Sigma$  there exists at most one  $\gamma \in \Gamma$ ,  $u \in \Delta^*$ , and  $q' \in Q$  such that  $q \xrightarrow{\text{op } a/u:\gamma} q'$  belongs to *rul*; (2) for all  $q \in Q$ ,  $a \in \Sigma$ , and  $\gamma \in \Gamma$  there is at most one  $u \in \Delta^*$  and  $q' \in Q$  such that  $q \xrightarrow{\text{cl } a/u:\gamma} q'$  belongs to *rul*; (3) there exists at most one state in *init*.

An N2W is *top-down* if  $Q = \Gamma$  and all closing rules have the form  $q \xrightarrow{\text{cl } a/u:q'} q'$ . An N2W is *top-down deterministic* (dN2W<sup>↓</sup>) if it is top-down and deterministic. A dNA is a dN2W and an NA, and a dNA<sup>↓</sup> is a dN2W<sup>↓</sup> and an NA.



**Fig. 3.** An example  $\text{dn2w}^1$  with a successful run on  $a(b(b))$ . For a node  $\alpha$  of the input tree, we put  $\text{output}(R(\text{op}, \alpha))$  and  $\text{rhs}(R(\text{op}, \alpha))$  on its left,  $\text{rhs}(R(\text{cl}, \alpha))$  and  $\text{output}(R(\text{cl}, \alpha))$  on its right, and  $\text{ssy}(R(-, \alpha))$  above it.

It is known that dNAs can recognize all regular languages of unranked trees, while  $\text{dNA}^\perp$ s are properly less expressive. This is similar to the case of ranked trees, where bottom-up deterministic tree automata can recognize all regular languages, while top-down deterministic tree automata capture only path closed languages. Emptiness of NAs can be checked in quadratic time. Intersection and complementation for dNAs can be performed in polynomial time. We can thus check inclusion and equivalence of dNAs in polynomial time too. We also remark that the set of all possible outputs of a N2W can be defined by a context-free grammar of size polynomial in the size of the N2W. Using the results of [17] we show that verifying that a N2W outputs only well-balanced words is in PTIME.

As an example, we consider a transformation  $\tau : \mathcal{T}_\Sigma \rightarrow \hat{\Delta}$  where  $\Sigma = \{a, b\}$  and  $\Delta = \Sigma \cup \{c\}$ .  $\tau$  “turns 90 degree clockwise” the input tree:  $a(b(b(a(b))))$  is mapped to (the linearization of)  $c(b, a, b, b, a)$ . More generally, transformation  $\tau$  maps tree  $a_1(a_2(\dots(a_n)\dots))$  to  $\text{lin}(c(a_n, \dots, a_1))$  where  $a_1, \dots, a_n \in \Sigma$  and  $n \geq 0$ . We define  $\tau$  with a  $\text{dn2w}^\perp$  in Fig. 3 with  $Q = \Gamma = \{0, 1, 2, 3\}$ ,  $\text{init} = \{0\}$  and  $\text{fin} = \{3\}$ . Except for the opening  $c$  parenthesis, all output is produced at closing time. A run of  $T$  on the input tree  $a(b(b))$  is shown in Fig. 3 too. The inverse transformation modulo linearization  $\tau' : \mathcal{T}_\Delta \rightarrow \hat{\Sigma}$  cannot be expressed by any N2W since we would have to read a horizontal word from the right to the left. This is impossible, similarly to word inversion by one-way string transducers.

For every class  $\mathcal{C}$  of transducers, we define the  $\mathcal{C}$ -equivalence problem. Its input are two transducers  $T_1, T_2 \in \mathcal{C}$  with the same alphabets, and its output is yes if  $\llbracket T_1 \rrbracket = \llbracket T_2 \rrbracket$  and no otherwise.

### 3 Morphism Equivalence on CFGs

We relate  $\text{dn2ws}$  and  $\text{dn2ws}^\perp$ -equivalence to word morphism equivalence on CFGs [14, 13]. A (word) *morphism* is a total function  $M : \Sigma^* \rightarrow \Delta^*$  such that  $M(v \cdot u) = M(v) \cdot M(u)$ . It is uniquely specified by the values taken on letters of  $\Sigma$ . We use the set  $\{(a, M(a)) \mid a \in \Sigma\}$  as the representation of  $M$  with size  $|M| = \sum_{a \in \Sigma} |M(a)|$ .

A *context-free grammar* (CFG) over  $\Sigma$  is a tuple  $G = (\Sigma, Q, \text{init}, \text{rul})$ , where  $\Sigma$  is the set of terminals,  $Q$  is the set of nonterminals or states,  $\text{init} \subseteq Q$  the set

of start states,  $rul \subseteq Q \times (\Sigma \cup Q^*)$  a set of rules. We denote rules  $r$  as  $q \rightarrow \omega$  where  $q \in Q$  is its left-hand side ( $lhs(r)$ ) and  $\omega \in \Sigma \cup Q^*$  its right-hand side ( $rhs(r)$ ). The size of a rule  $|r|$  is the length of its  $rhs$ . The size of a grammar  $G$  is  $|G| = |\Sigma| + |Q| + \sum_{r \in rul} |r|$ . The set  $L(G)$  of words recognized by  $G$  is defined in the standard way.

*Morphism equivalence on CFGs* is the decision problem, which inputs two finite sets  $\Sigma$  and  $\Delta$ , two morphisms  $M_1, M_2 : \Sigma^* \rightarrow \Delta^*$ , and a CFG  $G$  with alphabet  $\Sigma$ , and outputs yes iff  $M_1(w) = M_2(w)$  for all  $w \in L(G)$ .

We need extended parse trees, whose inner nodes are labeled by rules instead of nonterminals. Formally, an *extended parse tree* of a grammar  $G$  is a tree  $t \in \mathcal{T}_{rul \cup \Sigma}$ , such that for all nodes  $\pi \in nod(t)$ : 1. inner nodes are labeled by rules and leafs in  $\Sigma$ , 2. if the label of  $\pi$  is a rule  $q \rightarrow q_1 \cdots q_n$  then  $\pi$  has exactly  $n$  children, and for all  $1 \leq i \leq n$  child  $\pi \cdot i$  is labeled by a rule with  $lhs$   $q_i$ , and 3. if the label of  $\pi$  is a rule  $q \rightarrow a$  then  $\pi$  has exactly one child which is a leaf labeled  $a$ . Extended parse trees are ranked trees, when using the size of a rule as its arity. As usually, given a tree  $t$  we define  $yield(t)$  to be the concatenation of the labels of all leaves in left-to-right order. Clearly,  $w \in L(G)$  if and only if there exists an extended parse tree  $t$  of  $G$  with  $w = yield(t)$ .

**Lemma 2.** *For every morphism  $M : \Sigma^* \rightarrow \Delta^*$  and CFG  $G$  with alphabet  $\Sigma$ , we can construct in time  $O(|M| + |G|^2)$  a  $dN2W^\downarrow$   $T$  with alphabet  $\Sigma$  and  $\Delta$  such that  $\llbracket T \rrbracket = \{(t, M(yield(t))) \mid t \text{ extended parse tree of } G\}$ .*

*Proof (sketch).* We take the grammar  $G = (\Sigma, Q_G, init_G, rul_G)$  and construct an  $dN2W^\downarrow$   $T = (\Sigma', \Delta, Q_T, \Gamma_T, init_T, rul_T, fin_T)$  that takes as an input extended parse tree of  $G$   $t$  and outputs  $M(yield(t))$ . This can be done top-down deterministically since extended parse trees contain all necessary information. Let  $\Sigma' = Q_G \cup \Sigma$ ,  $Q_T = \Gamma_T = \{(r, j) \mid r \in rul_G, 0 \leq j \leq |r| \} \cup \{o, d, f\}$ ,  $init_T = \{o\}$ ,  $fin_T = \{f\}$ , and  $rul_T$  consists of the following transitions:

$$\begin{array}{c}
\frac{r \in rul_G \quad lhs(r) \in init_G}{o \xrightarrow{op \ r/\varepsilon:f} (r, 0)} \quad \frac{r \in rul_G \quad rhs(r) = a}{(r, 0) \xrightarrow{op \ a/M(a):(r,1)} d} \\
(r, |r|) \xrightarrow{cl \ r/\varepsilon:f} f \quad d \xrightarrow{cl \ a/\varepsilon:(r,1)} (r, 1) \\
\\
\frac{r, r' \in rul_G \quad rhs(r) = q_1 \cdots q_k \quad 1 \leq j \leq |r| \quad lhs(r') = q_j}{(r, j-1) \xrightarrow{op \ r'/\varepsilon:(r,j)} (r', 0)} \quad \square \\
(r', |r'|) \xrightarrow{cl \ r'/\varepsilon:(r,j)} (r, j)
\end{array}$$

**Proposition 1.** *Morphism equivalence on CFGs can be reduced in quadratic time to  $dN2W^\downarrow$ -equivalence.*

*Proof.* Take two morphisms  $M_1, M_2 : \Sigma^* \rightarrow \Delta^*$  and a CFG  $G$ , and let  $T_1$  and  $T_2$  be obtained with the construction described in Lemma 2 for  $M_1$  with  $G$  and  $M_2$  with  $G$  respectively, in  $O(|G|^2 + |M_1| + |M_2|)$ . Since  $T_1$  and  $T_2$  have the same domain, it should be clear that  $T_1$  and  $T_2$  are equivalent if and only if  $M_1$  and  $M_2$  are equivalent on  $G$ .  $\square$



We next reduce dN2W-equivalence to morphism equivalence on CFGs. Let  $T_1$  and  $T_2$  be two N2Ws with input alphabet  $\Sigma$  and output alphabet  $\Delta$ . Let  $t \in \mathcal{T}_\Sigma$  be a tree whose events are  $e_1 < \dots < e_n$ . A *successful parallel run* of  $T_1$  and  $T_2$  on tree  $t$  is a word  $s$  with alphabet  $\mathcal{R} = \text{rul}_{T_1} \times \text{rul}_{T_2}$  such that there exist two successful runs  $R_1$  of  $T_1$  and  $R_2$  of  $T_2$  on  $t$  with  $s = (R_1(e_1), R_2(e_1)) \cdot \dots \cdot (R_1(e_n), R_2(e_n))$ . We use two morphisms  $M_1$  and  $M_2$  from  $\mathcal{R}$  to  $\Delta$  such that:

$$M_i((r_1, r_2)) = u \quad \text{if } r_i = q \xrightarrow{\alpha a/u:\gamma} q'$$

If  $s$  is a successful parallel run of  $T_1$  and  $T_2$  on  $t$ , then  $(t, M_i(s)) \in \llbracket T_i \rrbracket$ .

**Lemma 3.** *For any two N2Ws  $T_1$  and  $T_2$  with the same alphabets there exists a CFG  $G$  such that  $L(G)$  is the set of all successful parallel runs of  $T_1$  and  $T_2$ .*

*Proof (sketch).* We construct the grammar  $G = (\mathcal{R}, Q_G, \text{init}_G, \text{rul}_G)$  as follows. The set of nonterminals is  $Q_G = \{\circ\} \cup Q_{T_1}^2 \times Q_{T_2}^2$ . A nonterminal  $((p_1, q_1), (p_2, q_2))$  is supposed to produce a parallel run of  $T_1$  from  $p_1$  to  $q_1$  and  $T_2$  from  $p_2$  to  $q_2$  (on the same input). There is only one start symbol  $\text{init}_G = \{\circ\}$  and the rules in  $\text{rul}_G$  are defined follows:

$$\begin{array}{c} \frac{r_1, r'_1 \in \text{rul}_{T_1} \quad r_1 = p_1 \xrightarrow{\text{op } a/u_1:\gamma_1} q_1 \quad r'_1 = p'_1 \xrightarrow{\text{cl } a/u'_1:\gamma_1} q'_1 \quad p_1 \in \text{init}_{T_1} \quad q'_1 \in \text{fin}_{T_1} \\ r_2, r'_2 \in \text{rul}_{T_2} \quad r_2 = p_2 \xrightarrow{\text{op } a/u_2:\gamma_2} q_2 \quad r'_2 = p'_2 \xrightarrow{\text{cl } a/u'_2:\gamma_2} q'_2 \quad p_2 \in \text{init}_{T_2} \quad q'_2 \in \text{fin}_{T_2}} \\ \circ \rightarrow (r_1, r_2) \cdot ((q_1, p'_1), (q_2, p'_2)) \cdot (r'_1, r'_2) \\ \\ \frac{r_1, r'_1 \in \text{rul}_{T_1} \quad r_1 = p_1 \xrightarrow{\text{op } a/u_1:\gamma_1} q_1 \quad r'_1 = p'_1 \xrightarrow{\text{cl } a/u'_1:\gamma_1} q'_1, \\ r_2, r'_2 \in \text{rul}_{T_2} \quad r_2 = p_2 \xrightarrow{\text{op } a/u_2:\gamma_2} q_2 \quad r'_2 = p'_2 \xrightarrow{\text{cl } a/u'_2:\gamma_2} q'_2} \\ ((p_1, q'_1), (p_2, q'_2)) \rightarrow (r_1, r_2) \cdot ((q_1, p'_1), (q_2, p'_2)) \cdot (r'_1, r'_2) \\ \\ \frac{p_1, p'_1, q_1 \in Q_{T_1} \quad p_2, p'_2, q_2 \in Q_{T_2}}{((p_1, q_1), (p_2, q_2)) \rightarrow ((p_1, p'_1), (p_2, p'_2)) \cdot ((p'_1, q_1), (p'_2, q_2))} \quad \frac{q_1 \in Q_{T_1}, q_2 \in Q_{T_2}}{((q_1, q_1), (q_2, q_2)) \rightarrow \varepsilon} \quad \square \end{array}$$

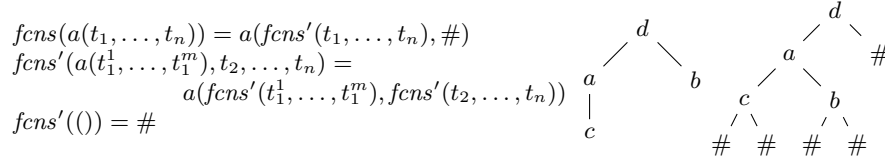
**Proposition 2.** *dN2W-equivalence can be reduced in polynomial time to morphism equivalence on CFGs.*

*Proof.* Let  $T_1$  and  $T_2$  be two dN2Ws with the input alphabet  $\Sigma$  and the output alphabet  $\Delta$ . First, we need to verify that the domains of  $T_1$  and  $T_2$  coincide. To do that, we test equivalence of the dNAs that define the domains of  $T_1$  and  $T_2$  (obtained by removing the output components from transitions). We recall that this can be done in polynomial time due to determinism.

If the domains of  $T_1$  and  $T_2$  are equal, let  $G$  be the grammar constructed in Lemma 3, and  $M_1$  and  $M_2$  the two morphisms defined above. Clearly,  $T_1$  is equivalent to  $T_2$  if and only if  $M_1$  is equivalent to  $M_2$  on  $G$ .  $\square$

Interestingly, composing Proposition 2 with Proposition 1 allows us to reduce dN2W-equivalence to dN2W<sup>↓</sup>-equivalence, the latter dealing with a weaker model than the former. The converse reduction is trivial.

**Corollary 1.** *dN2W-equivalence can be reduced in polynomial time to dN2W<sup>↓</sup>-equivalence, and vice versa.*



**Fig. 4.** First-child next-sibling encoding

## 4 Top-Down Ranked Tree to Word Transducers

We now relate  $\text{dn2ws}^\downarrow$  to standard top-down deterministic transducers on ranked trees, based on binary encoding of unranked trees.

A ranked alphabet consists of a set  $\Sigma$  and a arity function  $ar : \Sigma \rightarrow \mathbb{N}$ . The set  $\mathcal{T}_\Sigma^r$  of ranked trees over  $\Sigma$  is the least subset of  $\mathcal{T}_\Sigma$  that contains all trees  $a(t_1, \dots, t_{ar(a)})$  with  $t_i \in \mathcal{T}_\Sigma^r$ .

We fix an infinite sequence of pairwise distinct tree variables  $(\mathbf{x}_i)_{i \in \mathbb{N}}$ . A *top-down ranked-tree to word transducer* ( $\text{R2W}^\downarrow$ ) is a tuple  $S = (\Sigma, \Delta, Q, \text{init}, \text{rul})$ , where  $\Sigma$  is a ranked alphabet,  $\Delta$  a finite set,  $\text{init} \subseteq Q$ , and  $\text{rul}$  a finite set of rules of the form  $q(a(\mathbf{x}_1, \dots, \mathbf{x}_{ar(a)})) \rightarrow w$  where  $q \in Q$  and  $w$  is a word over alphabet  $\Delta \uplus \{p(\mathbf{x}_i) \mid p \in Q, 1 \leq i \leq ar(a)\}$ . An  $\text{R2W}^\downarrow$   $S$  is *top-down deterministic* or a  $\text{dR2W}^\downarrow$  if for every  $a \in \Sigma$  there exists at most one state  $q \in \text{init}$  such that  $q(a(\mathbf{x}_1, \dots, \mathbf{x}_{ar(a)})) \rightarrow w \in \text{rul}$  for some  $w$ , and if there exist no two rules in  $\text{rul}$  with the same left hand side.

The semantics of  $S$  is defined in terms of the relations  $\llbracket S \rrbracket_q \subseteq \mathcal{T}_\Sigma^r \times \Delta^*$ , where  $q \in Q$ , intuitively representing the transformation performed by  $S$  in state  $q$ . Formally,  $(a(t_1, \dots, t_k), u) \in \llbracket S \rrbracket_q$  if and only if there exists  $q(a(\mathbf{x}_1, \dots, \mathbf{x}_k)) \rightarrow w$  in  $\text{rul}$  such that  $u$  is obtained by replacing in  $w$  every occurrence of  $p(\mathbf{x}_i)$  by  $u_i$  for some  $(t_i, u_i) \in \llbracket S \rrbracket_p$  (if some  $p(\mathbf{x}_i)$  occurs more than once in  $w$ , then different pairs  $(t_i, u_i), (t_i, u'_i) \in \llbracket S \rrbracket_p$  can be used for replacement). Finally, the transformation defined by  $S$  is  $\llbracket S \rrbracket = \bigcup_{q \in \text{init}} \llbracket S \rrbracket_q$ .

From now, we will restrict ourselves to noncopying and order-preserving  $\text{dR2W}^\downarrow$ s. These are  $\text{dR2W}^\downarrow$ s with rules of the form:

$$q(a(\mathbf{x}_1, \dots, \mathbf{x}_k)) \rightarrow u_0 \cdot q_1(\mathbf{x}_1) \cdot u_1 \cdot \dots \cdot u_{k-1} \cdot q_k(\mathbf{x}_k) \cdot u_k,$$

where  $u_i \in \Delta^*$  for all  $i \in \{1, \dots, k\}$ . Given this restriction, we can drop variables from the rules, and simply denote them as  $q(a) \rightarrow u_0 \cdot q_1 \cdot \dots \cdot q_k \cdot u_k$ .

Our first result, shows that any transformation definable with a  $\text{dn2W}^\downarrow$  can be expressed by a noncopying order-preserving  $\text{dR2W}^\downarrow$ s, modulo a binary encoding of input trees. Here we use Rabin's encoding of unranked trees as usual, often called *first-child next-sibling* encoding. An unranked tree over  $\Sigma$  is represented using a binary tree whose inner nodes are labeled with  $\Sigma$  (binary symbol) and leaves with  $\#$  (constant symbol not belonging to  $\Sigma$ ). Formally, the encoding is defined and illustrated by an example in Fig. 4.

**Proposition 3.** *For any  $\text{dn2W}^\downarrow$   $T$  we can construct in time  $O(|\Sigma|^2 * |Q_T|^2)$  a noncopying order-preserving  $\text{dR2W}^\downarrow$   $S$  with  $\llbracket S \rrbracket = \{(fcns(t), u) \mid (t, u) \in \llbracket T \rrbracket\}$ .*

*Proof (sketch).* Let  $T = (\Sigma, \Delta, Q_T, \Gamma_T, \text{init}_T, \text{rul}_T, \text{fin}_T)$  and recall that  $\Gamma_T = Q_T$  because  $T$  is a  $\text{dN2W}^\perp$ . We defined  $S = (\Sigma \cup \{\#\}, \Delta, Q_S, \text{init}_S, \text{rul}_S)$  such that  $Q_S = \text{init}_T \cup \{q_\#\} \cup Q_T \times \Sigma \times \Gamma_T$ ,  $\text{init}_S = \text{init}_T$ , and  $\text{rul}_S$  consists of the following rules:

$$\begin{array}{c} \frac{q_0 \in \text{init}_T \quad q_2 \in \text{fin}_T \quad q_0 \xrightarrow{\text{op } a/u:q_2} q_1 \in \text{rul}_T}{q_0(a) \rightarrow u \cdot (q_1, a, q_2) \cdot q_\#} \quad \frac{q \xrightarrow{\text{cl } a/u:q'} q' \in \text{rul}_T}{(q, a, q')(\#) \rightarrow u} \\[10pt] \frac{b \in \Sigma \quad p \in Q_T \quad q \xrightarrow{\text{op } a/u:q_2} q_1 \in \text{rul}_T}{(q, b, p)(a) \rightarrow u \cdot (q_1, a, q_2) \cdot (q_2, b, p)} \quad \frac{\text{true}}{q_\#(\#) \rightarrow \varepsilon} \end{array}$$

Intuitively, the state  $(q, a, \gamma)$  corresponds to  $T$  being in state  $q$ , having  $\gamma$  on the top of the stack, and  $a$  being the label of the parent of the current node. The introduction of the state  $q_\#$  makes sure that  $S$  accepts on the input only trees that are result of the first-child next-sibling encoding of some unranked tree. One can then prove inductively that  $\llbracket S \rrbracket = \{(fcs(t), u) \mid (t, u) \in \llbracket T \rrbracket\}$ .  $\square$

Next, we show the converse, i.e. that every transformation defined with a  $\text{dR2W}^\perp$  can be expressed by  $\text{dN2W}^\perp$ . This time no encoding is necessary since unranked trees comprise a subset of ranked trees.

**Proposition 4.** *noncopying order-preserving  $\text{dR2W}^\perp$   $S$  can be converted to  $\text{dN2W}^\perp$   $T$  with  $\llbracket S \rrbracket = \llbracket T \rrbracket$  in time  $O(|S| * n)$  where  $n = \max(\{ar(x) \mid x \in S\})$ .*

*Proof (sketch).* Let  $S = (\Sigma, \Delta, \text{states}_S, \text{init}_S, \text{rul}_S)$ . We extend the arity function  $ar$  to  $\text{rul}_S$  in the following way:  $ar(q(a) \rightarrow w) = ar(a)$ .

The constructed  $\text{dN2W}^\perp$  is  $T = (\Sigma, \Delta, Q_T, \Gamma_T, \text{init}_T, \text{rul}_T, \text{fin}_T)$ , where  $Q_T = \Gamma_T = \{(r, j) \mid r \in \text{rul}_S, 0 \leq j \leq ar(r)\} \cup \{\circ, \mathbf{f}\}$ ,  $\text{init}_T = \{\circ\}$ ,  $\text{fin}_T = \{\mathbf{f}\}$ , and  $\text{rul}_S$  consists of the following rules

$$\begin{array}{c} \frac{q_0 \in \text{init}_S}{r = q_0(a) \rightarrow u_0 \cdot q_1 \dots q_k \cdot u_k \in \text{rul}_S} \quad \frac{r = q(a) \rightarrow u_0 \cdot q_1 \dots q_j \cdot u_j \dots q_k \cdot u_k \in \text{rul}_S}{r' = q_j(b) \rightarrow v_0 \cdot p_1 \dots p_m \cdot v_m \quad 1 \leq j \leq k} \\[10pt] \frac{\circ \xrightarrow{\text{op } a/u_\circ:\mathbf{f}} (r, 0)}{(r, k) \xrightarrow{\text{cl } a/\varepsilon:\mathbf{f}} \mathbf{f}} \quad \frac{(r, j-1) \xrightarrow{\text{op } b/v_\circ:(r, j)} (r', 0)}{(r', m) \xrightarrow{\text{cl } b/u_j:(r, j)} (r, j)} \end{array}$$

Intuitively, when the transducer is in the state  $(r, j)$ , it processes the rule  $r = q_0(a) \rightarrow u_0 \cdot q_1 \dots q_k \cdot u_k$ , has just written out  $u_j$ , and is about to handle the state  $q_{j+1}$  (or to close the parenthesis  $a$  if  $j = k$ ).  $\llbracket S \rrbracket = \llbracket T \rrbracket$  is proved with an inductive proof.  $\square$

**Corollary 2.** *Equivalence of noncopying order-preserving  $\text{dR2W}^\perp$   $S$  can be reduced in polynomial time to  $\text{dN2W}^\perp$ -equivalence, and vice versa.*

## 5 Bottom-Up Ranked Tree to Word Transducers

Even though the expressive power of bottom-up and top-down deterministic tree transducers are uncomparable, we can show their equivalence problems are the

same. Here we use direct reductions, that are inspired from the reduction of  $\text{dN2W}$ -equivalence to  $\text{dN2W}^\downarrow$ -equivalence.

We fix an infinite sequence of pairwise distinct variables  $(\mathbf{x}_i)_{i \in \mathbb{N}}$  for output words. A *bottom-up ranked-tree to word transducer* ( $\text{R2W}^\uparrow$ ) is a tuple  $S = (\Sigma, \Delta, Q, \text{fin}, \text{rul})$  with rules in  $\text{rul}$  of the form:  $a(q_1(\mathbf{x}_1), \dots, q_k(\mathbf{x}_k)) \rightarrow q(w)$  where  $k = \text{ar}(a)$   $q, q_i \in Q$ , and  $w$  is a word with alphabet  $\Delta \uplus \{\mathbf{x}_1, \dots, \mathbf{x}_k\}$ . The semantics  $\llbracket S \rrbracket \subseteq \mathcal{T}_\Sigma^r \times \Delta^*$  can be defined as follows:

$$\frac{a(q_1(\mathbf{x}_1), \dots, q_k(\mathbf{x}_k)) \rightarrow q(w) \in \text{rul} \quad (t_1, u_1) \in \llbracket S \rrbracket_{q_1} \quad \dots \quad (t_k, u_k) \in \llbracket S \rrbracket_{q_k}}{(a(t_1, \dots, t_k), w[u_1/\mathbf{x}_1, \dots, u_k/\mathbf{x}_k]) \in \llbracket S \rrbracket_q} \quad \frac{\text{true}}{\llbracket S \rrbracket = \cup_{q \in \text{fin}} \llbracket S \rrbracket_q}$$

An  $\text{R2W}^\uparrow S$  is *bottom-up deterministic* (or a  $\text{dR2W}^\uparrow$ ) if no two rules of  $S$  have the same left-hand side. In this case,  $\llbracket S \rrbracket$  defines a partial function.

From now on, we will only consider noncopying and order-preserving  $\text{R2W}^\uparrow$ s, i.e.,  $\text{R2W}^\uparrow$ s with rules restricted to the form  $a(q_1(\mathbf{x}_1), \dots, q_k(\mathbf{x}_k)) \rightarrow q(u_0 \cdot \mathbf{x}_1 \cdot u_1 \cdots \mathbf{x}_k \cdot u_k)$ . Since noncopying, such  $\text{R2W}^\uparrow$ s can be translated to  $\text{R2W}^\downarrow$ s defining the same relation, and vice versa. A rule as above is transformed to  $q(a) \rightarrow u_0 \cdot q_1 \cdot u_1 \cdots q_k \cdot u_k$ , while final states of the bottom-up transducer become initial states of its top-down version. We can thus talk of  $\text{R2W}$ s in the noncopying case, rather than distinguishing  $\text{R2W}^\uparrow$ s and  $\text{R2W}^\downarrow$ s artificially.

Given two noncopying order-preserving  $\text{R2W}$ s  $S_1$  and  $S_2$  with the same alphabets  $\Sigma$  and  $\Delta$ , a *successful parallel run* of  $S_1$  and  $S_2$  is a tree  $s$  over alphabet  $\mathcal{R} = \text{rul}_{S_1} \times \text{rul}_{S_2}$  with nodes  $\pi \in \text{nod}(s)$  labeled by some pair of rules  $(r_1^\pi, r_2^\pi)$  such that (1) the label of the root of  $s$  satisfies  $r_i^\epsilon = q_i(-) \rightarrow -$  for some  $q_i \in \text{fin}_{S_i}$ , and (2) for all nodes  $\pi \in \text{nod}(s)$  with  $r_i^\pi = q_i(a_i) \rightarrow u_i^0 \cdot q_i^1 \cdots q_i^{k_i} \cdot u_i^{k_i}$ , it holds that  $a_1 = a_2$  and thus  $k_1 = k_2 = \text{ar}(a_1) = \text{ar}(a_2)$  and that  $\pi$  has exactly  $k = k_1 = k_2$  children  $\pi \cdot j$  labeled by rules  $r_i^{\pi \cdot j} = q_i^j(-) \rightarrow -$  where  $1 \leq j \leq k$  and  $1 \leq i \leq 2$ . For every successful parallel run  $s$  we define  $\text{input}(s) \in \mathcal{T}_\Sigma^r$  and the  $\text{output}_i(s) \in \Delta^*$  as follows, where  $r_i = q_i(a) \rightarrow u_i^0 \cdot q_i^1 \cdots q_i^k \cdot u_i^k$ :

$$\begin{aligned} \text{input}((r_1, r_2)(s_1, \dots, s_k)) &= a(\text{input}(s_1), \dots, \text{input}(s_k)) \\ \text{output}_i((r_1, r_2)(s_1, \dots, s_k)) &= u_i^0 \cdot \text{output}_i(s_1) \cdots \text{output}_i(s_k) \cdot u_i^k \end{aligned}$$

**Lemma 4.** *Let  $S_1$  and  $S_2$  be noncopying and order-preserving  $\text{R2W}$ s over the alphabets  $\Sigma$  and  $\Delta$ . Then, for all  $t \in \mathcal{T}_\Sigma^r$  and  $u_1, u_2 \in \Delta^*$  we have that  $(t, u_1) \in \llbracket S_1 \rrbracket$  and  $(t, u_2) \in \llbracket S_2 \rrbracket$  if and only if there exists a successful parallel run  $s$  of  $S_1$  and  $S_2$  such that  $\text{input}(s) = t$ ,  $\text{output}_1(s) = u_1$ , and  $\text{output}_2(s) = u_2$ .*

**Proposition 5.** *Equivalence of noncopying and order-preserving  $\text{dR2W}^\uparrow$ s  $S_1$  and  $S_2$  can be reduced in  $O(|S_1| \times |S_2|)$  to equivalence of noncopying and order-preserving  $\text{dR2W}^\downarrow$ s, and vice versa.*

Indeed, taking for example two noncopying and order-preserving  $\text{dR2W}^\uparrow$ s  $S_1$  and  $S_2$  over the same domain (otherwise, it just a problem of equivalence over tree automata that can be dealt in  $O(|S_1| \times |S_2|)$ , see [3]), one can build two  $\text{dR2W}^\downarrow$ s  $S'_1$  and  $S'_2$  with  $\llbracket S'_i \rrbracket$  containing  $(s, \text{output}_i(s))$  where  $s$  is a parallel run of  $S_1$  and  $S_2$ . Lemma 4 then finishes the reduction.

## 6 Conclusion and Future Work

We have shown that various classes of tree-to-word transducers have the same equivalence problem modulo polynomial time reductions. Our results are based on a new relationship to the morphism equivalence problem on CFGs that we established. Our equivalences do not carry over to tree-to-tree transducers without macros or concatenation operations, since these cannot express word morphisms on CFGs in any obvious manner (Lemma 3). The classes of deterministic nested word transducers that we have proposed here, open up quite a number of questions of interest for the XML, formal language, tree automata and grammatical inference communities. For instance, grammatical inference algorithm algorithms often relies on Myhill-Nerode theorem, which is closely linked with equivalence problem (for example, see [12] for a learning algorithm for a class of word transducers). So far, there lack results on type checking, minimization, and learnability.

## References

1. Alur, R.: Marrying words and trees. ACM SIGMOD-SIGACT-SIGART SPDS 26, 233–242 (2007)
2. Alur, R., Madhusudan, P.: Visibly pushdown languages. ACM STC 36, 202–211 (2004)
3. Champavère, J., Gilleron, R., Lemay, A., and Niehren, J.: Efficient Inclusion Checking for Deterministic Tree Automata and DTDs. LATA 184–195 (2008)
4. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., and Tommas, M.: Tree Auto. Tech. and App.online, (Revised 2007).
5. Engelfriet, J., Maneth, S., and Seidl, H.: Deciding Equiv. of Top-Down XML Transformations in Poly. Time. Journal of Comp. and Syst. Sci., (to appear, 2009)
6. Gauwin, O., Niehren, J., and Roos Y.: Streaming tree automata. IPL 109(1), 13–17 (2008)
7. Culik II, K., Karhumäki, J.: Synchronizable deterministic pushdown automata and the decidability of their equivalence. Acta Inf. 23(5), 597–605 (1986)
8. Maneth, S.: Models of Tree Translation. PhD Thesis
9. Maneth, S., Berlea, A., Perst, T., Seidl, H.: type checking with macro tree transducers. ACM SPDS 24, 283–294 (2005)
10. Martens, W., Neven, F.: Typechecking top-down uniform unranked tree transducers. ICDT, LNCS 2572, 64–78 (2003)
11. Neumann, A., Seidl, H.: Locating matches of tree patterns in forests. Found. of Soft. Tech. and Theor. Comp. Sci., 134–145 (1998)
12. Oncina, J., Garcia, P., Vidal, E.: Learn. Subseq. Transd. for Patt. Recogn. and Interpretation Tasks. Trans. on Patt. Anal. and Mach. Intel. 15, 448–458 (1993)
13. Plandowski, W.: Testing Equivalence of Morphisms on Context-Free Languages. ESA, 460–470 (1994)
14. Karhumäki, J., Plandowski, W., Rytter, W.: Polynomial Size Test Sets for Context-Free Languages. J. Comput. Syst. Sci. 50(1), 11–19 (1995)
15. Raskin, J.-F., Servais, F.: Visibly pushdown transducers. Auto., Lang. and Prog., LNCS 5126, 386–397 (2008)
16. Sénizergues, G.: The equivalence problem for deterministic pushdown automata is decidable. Auto., Lang. and Prog., LNCS 1256, 671–681 (1997)
17. Tozawa, A., Minamide, Y.: Complexity results on balanced context-free languages. FoSSaCS, LNCS 4423, 346–360 (2007)