Expressiveness of Visibly Pushdown Transducers*

Mathieu Caralp Pierre-Alain Reynier Jean-Marc Talbot
Laboratoire d'Informatique Fondamentale de Marseille
Aix-Marseille Université & CNRS, France
{mathieu.caralp,pierre-alain.reynier,jean-marc.talbot}@lif.univ-mrs.fr

Emmanuel Filiot†
CS Department
Université Libre de Bruxelles, Belgium
efiliot@ulb.ac.be

Frédéric Servais
Hasselt University and transnational University of Limburg
Belgium
frederic.servais@gmail.com

Visibly pushdown transducers (VPTs) are visibly pushdown automata extended with outputs. They have been introduced to model transformations of nested words, i.e. words with a call/return structure. As trees and more generally hedges can be linearized into (well) nested words, VPTs are a natural formalism to express tree transformations evaluated in streaming. This paper aims at characterizing precisely the expressive power of VPTs with respect to other tree transducer models.

1 Introduction

Visibly pushdown machines [1], automata (VPA) or transducers, are pushdown machines such that stack behavior is synchronized with the structure of the input word. Precisely, the input alphabet is partitioned into call and return symbols. When reading a call symbol the machine must push a symbol onto the stack, and when reading a return symbol it must pop a symbol from the stack.

Visibly pushdown transducers (VPTs) [10, 11, 5, 12] extend visibly pushdown automata [1] with outputs. Each transition is equipped with an output word that is appended to the output tape whenever the transition is triggered. A VPT thus transforms an input word into an output word obtained as the concatenation of all the output words produced along a successful run on that input. VPTs are a strict subclass of pushdown transducers (PTs) and strictly extend finite state transducers. Several problems that are undecidable for PTs are decidable for VPTs, most notably: functionality (in PTIME), k-valuedness (in NPTIME) and functional equivalence (EXPTIME-C) [5]. VPTs are closed by regular look-ahead which makes them a robust class of transformations [6].

Unranked trees and more generally hedges can be linearized into well-nested words over a structured alphabet (such as XML documents). VPT are therefore a suitable formalism to express hedge transformations. In particular, they can express operations such as node deletion, renaming and insertion. As they process the linearization from left to right, they are also an adequate formalism to model and analyze transformations in streaming, as shown in [4]. VPTs output strings, therefore on well-nested inputs they define hedge-to-string transformations, and if the output strings are well-nested too, they define hedge-to-hedge transformations.

In this paper, we characterize the expressive power of VPTs w.r.t. their ability to express hedge-to-string (H2S), and hedge-to-hedge (H2H) transformations. To do so, we define a top-down model

*This work has been supported by the PEPS project SOSP ("Synthesis of Stream Processors") funded by CNRS and by the project ECSPER (ANR-09-JCJC-0069) funded by the ANR.

†FNRS Research Associate ("Chercheur Qualifié")

S. Maneth (Ed.): Workshop on Trends in Tree Automata and Tree Transducers (TTATT 2013).
EPTCS 134, 2013, pp. 17–26, doi:10.4204/EPTCS.134.3

of hedge-to-string transducers, inspired by classical top-down tree transducers. They correspond to parameter-free linear order-preserving macro forest transducers that output strings [9]. We define a syntactic restriction of H2S that captures exactly VPTs, and show that if the VPTs runs on binary encodings of hedges, then they have exactly the same expressive power as H2S. We show that those results still hold when both models are restricted to hedge-to-hedge transformations. Based on those results, we compare VPTs with classical ranked tree transducers, such as top-down tree transducers [2] and macro tree transducers [3].

2 Transducer Models for Nested Words and Hedges

Words and Nested Words The set of finite words over a (finite) alphabet Σ is denoted by Σ^*, and the empty word is denoted by ε. A *structured alphabet* is a pair $\Sigma = (\Sigma_c, \Sigma_r)$ of disjoint alphabets, of call and return symbols respectively. Given a structured alphabet Σ, we always denote by Σ_c and Σ_r its implicit structure, and identify Σ with $\Sigma_c \cup \Sigma_r$.

A *nested word* is a finite word over a structured alphabet. The set of *well-nested words* over a structured alphabet Σ is the least set, denoted by $\mathscr{W}_\Sigma$, that satisfies (i) $\varepsilon \in \mathscr{W}_\Sigma$, (ii) for all $w, w' \in \mathscr{W}_\Sigma$, $ww' \in \mathscr{W}_\Sigma$ (closure under concatenation), and (iii) for all $w \in \mathscr{W}_\Sigma$, $c \in \Sigma_c$, $r \in \Sigma_r$, $cwr \in \mathscr{W}_\Sigma$. E.g. on $\Sigma = (\{c_1, c_2\}, \{r\})$, the nested word $c_1 r c_2 r$ is well-nested while $r c_1$ is not. Finally, note that any well-nested word w is either empty or can be decomposed uniquely as $w = c w_1 r w_2$ where $c \in \Sigma_c, r \in \Sigma_r$, $w_1, w_2 \in \mathscr{W}_\Sigma$.

Hedges Let Λ be an alphabet. We let $S(\Lambda)$ be the signature $\{0, \cdot\} \cup \{a \mid a \in \Lambda\}$ where 0 is a constant symbol, $a \in \Lambda$ are unary symbols and $\cdot$ is a binary symbol. The set of *hedges* $\mathscr{H}_\Lambda$ over Λ is the quotient of the free $S(\Lambda)$-algebra by the associativity of $\cdot$ and the axioms $0 \cdot h = h \cdot 0 = h$. The constant 0 is called the empty hedge. We may write a instead of $a(0)$, and omit $\cdot$ when it is clear from the context. *Unranked trees* are particular hedges of the form $a(h)$ where $h \in \mathscr{H}_\Lambda$. Note that any hedge h is either empty or can be decomposed as $h = a(h_1) \cdot h_2$.

Hedges over Λ can be naturally encoded as well-nested words over the structured alphabet $\Lambda_s = (\Lambda_c, \Lambda_r)$ where Λ_c and Λ_r are new alphabets respectively defined by $\Lambda_c = \{c_a \mid a \in \Lambda\}$ and $\Lambda_r = \{r_a \mid a \in \Lambda\}$. This correspondence is given via a morphism $\mathsf{lin} : \mathscr{H}_\Lambda \to \mathscr{W}_{\Lambda_s}$ inductively defined by: $\mathsf{lin}(0) = \varepsilon$ and $\mathsf{lin}(a(h_1).h_2) = c_a \mathsf{lin}(h_1) r_a \mathsf{lin}(h_2)$. E.g. for $\Lambda = \{a, b\}$, we have $\mathsf{lin}(ab(ab)) = c_a r_a c_b c_a r_a c_b r_b r_b$.

Conversely, any well-nested word over a structured alphabet Σ can be encoded as an hedge over the product alphabet $\Sigma_c \times \Sigma_r$, via the mapping $\mathsf{hedge} : \mathscr{W}_\Sigma \to \mathscr{H}_{\Sigma_c \times \Sigma_r}$ defined as $\mathsf{hedge}(\varepsilon) = 0$ and $\mathsf{hedge}(c w_1 r w_2) = (c, r)(\mathsf{hedge}(w_1)) \cdot \mathsf{hedge}(w_2)$ for all $(c, r) \in \Sigma_c \times \Sigma_r$ and all $w_1, w_2 \in \mathscr{W}_\Sigma$.

Binary Trees We consider here an alphabet Λ augmented with some special symbol $\perp$. We define the set of *binary trees* $\mathscr{B}_\Lambda$ as a particular case of unranked trees over $\Lambda \cup \{\perp\}$. Binary trees are defined recursively as: (i) $\perp \in \mathscr{B}_\Lambda$, and (ii) for all $f \in \Lambda$, if $t_1, t_2 \in \mathscr{B}_\Lambda$ then $f(t_1\, t_2) \in \mathscr{B}_\Lambda$.

There is a well-known correspondence between hedges and binary trees by means of an encoding called the first-child next-sibling encoding. This encoding is given by the mapping fcns defined as: (i) $\mathsf{fcns}(0) = \perp$, (ii) $\mathsf{fcns}(f(h_1)h_2) = f(\mathsf{fcns}(h_1)\mathsf{fcns}(h_2))$ for all h_1, h_2 in $\mathscr{H}_\Lambda$.

The strong relationship between hedges and well-nested words can be considered when restricted to binary trees: we define $\mathscr{BW}_{\Lambda_s}$ the set of binary well-nested words over the structured alphabet $(\Lambda_c \cup \{\perp_c\}, \Lambda_r \cup \{\perp_r\})$ as the least set satisfying: (i) $\perp_c \perp_r \in \mathscr{BW}_{\Lambda_s}$ and (ii) for all $f_c \in \Lambda_c$, $f_r \in \Lambda_r$, if $w_1^b, w_2^b \in \mathscr{BW}_{\Lambda_s}$ then $f_c\, w_1^b\, w_2^b\, f_r \in \mathscr{BW}_{\Lambda_s}$. Note that the morphism lin applied on binary trees from $\mathscr{B}_\Lambda$ yields binary nested words in $\mathscr{BW}_{\Lambda_s}$.

Finally, we can define the first-child next-sibling encoding of hedges as binary trees, directly on linearizations; consider a structured alphabet Σ extended as $\Sigma_\perp = (\Sigma_c \cup \{\perp_c\}, \Sigma_r \cup \{\perp_r\})$. For all well-nested words w over Σ, we define $\mathsf{fcns}(w)$ over the alphabet $\Sigma_\perp$ recursively as (i) $\mathsf{fcns}(\varepsilon) = \perp_c \perp_r$ and (ii) $\mathsf{fcns}(cw_1rw_2) = c\,\mathsf{fcns}(w_1)\,\mathsf{fcns}(w_2)\,r$ for all $w_1, w_2 \in \mathscr{W}_\Sigma$.

Visibly Pushdown Transducers Let Σ be a structured alphabet, and Δ be an alphabet. A *visibly pushdown transducer* from Σ to Δ (the class is denoted $\mathsf{VPT}(\Sigma,\Delta)$) is a tuple $A = (Q, I, F, \Gamma, \delta)$ where Q is a finite set of states, $I \subseteq Q$ the set of initial states, $F \subseteq Q$ the set of final states, Γ the (finite) stack alphabet, $\perp \notin \Gamma$ is the bottom stack symbol, and $\delta = \delta_c \uplus \delta_r$ is the transition relation where:

- $\delta_c \subseteq Q \times \Sigma_c \times \Gamma \times \Delta^* \times Q$ are the *call transitions*,
- $\delta_r \subseteq Q \times \Sigma_r \times \Gamma \times \Delta^* \times Q$ are the *return transitions*.

A configuration of A is a pair (q, σ) where $q \in Q$ and $\sigma \in \perp \cdot \Gamma^*$ is a stack content. Let $w = a_1 \dots a_l$ be a (nested) word on Σ, and $(q,\sigma), (q',\sigma')$ be two configurations of A. A *run* of the VPT A over w from (q,σ) to (q',σ') is a (possibly empty) sequence of transitions $\rho = t_1 t_2 \dots t_l \in \delta^*$ such that there exist $q_0, q_1, \dots q_l \in Q$ and $\sigma_0, \dots \sigma_l \in \perp \cdot \Gamma^*$ with $(q_0,\sigma_0) = (q,\sigma)$, $(q_l,\sigma_l) = (q',\sigma')$, and for each $0 < k \leq l$, we have either (i) $t_k = (q_{k-1}, a_k, \gamma, w_k, q_k) \in \delta_c$ and $\sigma_k = \sigma_{k-1}\gamma$, or (ii) $t_k = (q_{k-1}, a_k, \gamma, w_k, q_k) \in \delta_r$, and $\sigma_{k-1} = \sigma_k \gamma$. When the sequence of transitions is empty, $(q,\sigma) = (q',\sigma')$.

The *output* of ρ is the word $w \in \Delta^*$ defined as the concatenation $w = w_1 \dots w_l$ when the sequence of transitions is not empty and ε otherwise. Initial (resp. final) configurations are pairs $(q, \perp)$ with $q \in I$ (resp. with $q \in F$). A run is *accepting* if it starts in an initial configuration and ends in a final configuration. The transducer A defines a relation from nested words to words defined as the set of pairs $(u,w) \in \Sigma^* \times \Delta^*$ such that there exists an accepting run on u producing w as output. From now on, we confuse the transducer and the transduction it represents. Note that since we accept by empty stack and there is no return transition on empty stack, A accepts only well-nested words, and thus is included into $\mathscr{W}_\Sigma \times \Delta^*$.

Hedge-to-string Transducers We present now a model of hedge-to-string transducers ($\mathsf{H2S}$) that run directly on hedges, and is closer to classical transducers than VPTs are. In particular, this model is a syntactic subclass of macro forest transducers (MFT) [9] with no parameters, no swapping and no copy.

Let Λ and Δ be two finite alphabets. An *hedge-to-string transducer* from Λ to Δ (the class is denoted $\mathsf{H2S}(\Lambda,\Delta)$) is a tuple $T = (Q, I, \delta)$ where Q is a set of states, $I \subseteq Q$ is a set of initial states and δ is a set of rules of the form: [1]

$$q(0) \to \varepsilon \qquad\qquad q(f(x_1) \cdot x_2) \to w_1 q_1(x_1) w_2 q_2(x_2) w_3$$

where $q, q_1, q_2 \in Q$, $f \in \Lambda$ and $w, w_1, w_2, w_3 \in \Delta^*$.

The semantics of T is defined via mappings $[\![q]\!] : \mathscr{H}_\Lambda \to 2^{\Delta^*}$ for all $q \in Q$ as follows:

$$[\![q]\!](0) = \begin{cases} \{\varepsilon\} & \text{if } q(0) \to \varepsilon \in \delta \\ \emptyset & \text{otherwise} \end{cases}$$

$$[\![q]\!](f(h) \cdot h') = \bigcup_{\substack{q(f(x_1)\cdot x_2) \to \\ w_1 q_1(x_1) w_2 q_2(x_2) w_3}} w_1 \cdot [\![q_1]\!](h) \cdot w_2 \cdot [\![q_2]\!](h') \cdot w_3$$

[1] We consider linear and order-preserving rules only.

The transduction of an H2S $T=(Q,I,\delta)$ is defined as the relation $\{(h,s) \mid \exists q \in I,\ s \in [\![q]\!](h)\}$. When $s \in [\![q]\!](h)$ for some H2S T, we may say that the computation of the H2S T on the hedge h leads to q producing s.
We say that T is *tail-recursive* whenever in any rule, we have $w_3 = \varepsilon$. We denote by $\mathsf{H2S}_{\mathsf{tr}}$ the class of tail-recursive H2S.

Example 1. *Let Λ be a finite alphabet. Consider $T_1 \in \mathsf{H2S}(\Lambda,\Lambda)$ defined by $Q = I = \{q,q'\}$ and the following rules, for all $f \in \Lambda$:*

$$q(0) \to \varepsilon \qquad q'(0) \to \varepsilon \qquad q(f(x_1)\cdot x_2) \to q'(x_1)q(x_2)f$$

The domain of T_1 is the set of strings over Λ (viewed as a particular case of hedges) and T_1 defines the mirror image of strings.

Example 2. *Let Λ be a finite alphabet and Λ_s be its structured version. We define $T_2 \in \mathsf{H2S}(\Lambda,\Lambda_s)$ which can non-deterministically root any subhedge of the input hedge under a new symbol # and output the linearization of the new hedge. For instance, the input tree $f(abcd)$ can be non-exhaustively translated into the string $\mathsf{lin}(f(a\#(bc)d))$ or the string $\mathsf{lin}(f(\#(ab)\#(cd)))$. Formally, T_2 is defined by $Q=\{q_0,q_1,q_2\}$, $I=\{q_0\}$ and δ defined as the following set of rules: (observe that $T_2 \in \mathsf{H2S}_{\mathsf{tr}}$)*

$$\begin{array}{ll}
q_i(0) \to \varepsilon \quad \forall i \in \{0,2\} & q_0(f(x_1)\cdot x_2) \to c_f q_0(x_1) r_f q_0(x_2) \\
q_0(f(x_1)\cdot x_2) \to c_\# c_f q_2(x_1) r_f r_\# q_0(x_2) & q_0(f(x_1)\cdot x_2) \to c_\# c_f q_2(x_1) r_f q_1(x_2) \\
q_1(f(x_1)\cdot x_2) \to c_f q_2(x_1) r_f r_\# q_0(x_2) & q_1(f(x_1)\cdot x_2) \to c_f q_2(x_1) r_f q_1(x_2) \\
q_2(f(x_1)\cdot x_2) \to c_f q_2(x_1) r_f q_2(x_2) &
\end{array}$$

Hedge-to-hedge Transducers We consider now transducers running on hedges but producing (representations of) hedges as well-nested words. We define them as restrictions of the two models we have considered so far.

We assume the output alphabet Δ to be structured as (Δ_c,Δ_r). We define an $\mathsf{H2S}(\Lambda,\Delta)$ to be an hedge-to-hedge transducer ($\mathsf{H2H}(\Lambda,\Delta)$) if any rhs $w_1 q_1(x_1) w_2 q_2(x_2) w_3$ of its transition rules satisfies $w_1 w_2 w_3 \in \mathscr{W}_\Delta$. We denote $\mathsf{H2H}_{\mathsf{tr}}$ the class of H2H that are additionally tail-recursive.

Using the direct relationship between well-nested words and hedges, we may define hedge-to-hedge transducers by means of a restriction in the definition of VPT: this restriction asks the nesting level of the input and the output words to be synchronized, that is the nesting level of the output just before reading a call (on the input) must be equal to the nesting level of the output just after reading the matching return (on the input). This simple syntactic restriction yields a subclass of VPTs [5].

This synchronization is enforced syntactically on stack symbols, these symbols being shared by matching call and return transitions.

Let $A=(Q,I,F,\Gamma,\delta) \in \mathsf{VPT}(\Sigma,\Delta)$. Then A is *well-nested* if for all $\gamma \in \Gamma$, all transitions $(q,c,\gamma,w,q') \in \delta_c$ and $(p,r,\gamma,w',p') \in \delta_r$, it holds that $ww' \in \mathscr{W}_\Delta$. We denote by wnVPT the class of well-nested VPTs.

Hedge-to-binary tree Transducers We consider transducers running on hedges and producing (representations of) binary trees as binary well-nested words. We define them as restrictions of hedge-to-hedge transducers.

Let $\Delta^\perp = (\Delta_c^\perp,\Delta_r^\perp)$ be a structured output alphabet such that $\Delta_c^\perp,\Delta_r^\perp$ contain two special symbols $\perp_c,\perp_r$ respectively. We define a $\mathsf{H2H}(\Lambda,\Delta^\perp)$ to be an hedge-to-binary tree transducer ($\mathsf{H2B}(\Lambda,\Delta^\perp)$) if any right hand-side $w_1 q_1(x_1) w_2 q_2(x_2) w_3$ of its transition rules satisfies $w_1 = c\,w_1'$, $w_2 = w_1'' w_2'$, $w_3 = w_3' r$ for some c in $\Delta_c^\perp$, r in $\Delta_r^\perp$, $w_1' \perp_c \perp_r w_1''$ and $w_2' \perp_c \perp_r w_3'$ in $\mathscr{BW}_{\Delta^\perp}$.

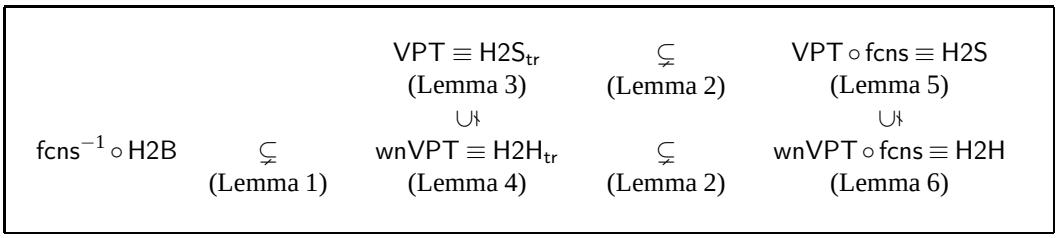

Figure 1: Expressivness results in a nutshell

Hedge-to-binary tree transducers are close to linear and order-preserving top-down ranked tree transducers. They will serve us to compare the expressiveness of H2H to this latter class of transducers defined on the first-child next-sibling encoding of input and output hedges.

3 Some Results on Expressiveness

In the sequel, we assume that input hedges accepted by transducers are non-empty. This restriction is done without loss of generality. We depict on Figure 1 the results we obtained.

3.1 Definitions of expressiveness

Let Σ be a structured alphabet and Δ be a finite alphabet. We denote by $\mathscr{T}(\mathscr{W}_\Sigma, \Delta^*)$ the set of transductions from $\mathscr{W}_\Sigma$ to Δ^*. First observe that the semantics of a transducer $A \in \mathsf{VPT}(\Sigma, \Delta)$ is an element of $\mathscr{T}(\mathscr{W}_\Sigma, \Delta^*)$. Second, given a transducer $T \in \mathsf{H2S}(\Sigma_c \times \Sigma_r, \Delta)$, we have that $T \circ \mathsf{hedge} \in \mathscr{T}(\mathscr{W}_\Sigma, \Delta^*)$. Hence, up to the mapping hedge, we can thus compare the expressiveness of a subclass $\mathscr{C}_1$ of $\mathsf{VPT}(\Sigma, \Delta)$ and of a subclass $\mathscr{C}_2$ of $\mathsf{H2S}(\Sigma_c \times \Sigma_r, \Delta)$, by their interpretation as transductions from $\mathscr{W}_\Sigma$ to Δ^*.

Formally, given $A \in \mathsf{VPT}(\Sigma, \Delta)$ and $T \in \mathsf{H2S}(\Sigma_c \times \Sigma_r, \Delta)$, we say that A and T are equivalent, denoted $A \equiv T$, whenever $A = T \circ \mathsf{hedge}$. Given a subclass $\mathscr{C}_1$ of $\mathsf{VPT}(\Sigma, \Delta)$ and a subclass $\mathscr{C}_2$ of $\mathsf{H2S}(\Sigma_c \times \Sigma_r, \Delta)$, we say that $\mathscr{C}_1$ is more expressive than $\mathscr{C}_2$ (resp. less expressive), denoted $\mathscr{C}_1 \supseteq \mathscr{C}_2$ (resp. $\mathscr{C}_1 \subseteq \mathscr{C}_2$), whenever we have:

- for every $T \in \mathscr{C}_2$, there exists $A \in \mathscr{C}_1$ such that $A \equiv T$
- for every $A \in \mathscr{C}_1$, there exists $T \in \mathscr{C}_2$ such that $A \equiv T$, respectively

Last, we write $\mathscr{C}_1 \equiv \mathscr{C}_2$ whenever $\mathscr{C}_1$ and $\mathscr{C}_2$ are expressively equivalent meaning that both $\mathscr{C}_1 \supseteq \mathscr{C}_2$ and $\mathscr{C}_1 \subseteq \mathscr{C}_2$ hold.

3.2 Comparing expressiveness

We first recall in the framework we proposed here a known expressiveness result [11] comparing H2H and H2B.

Lemma 1. *Let $\Delta = (\Delta_c, \Delta_r)$ and $\Delta^\perp = (\Delta_c \cup \{\perp_c\}, \Delta_r \cup \{\perp_r\})$ be two structured alphabets.*

1. *For any $T \in \mathsf{H2B}(\Lambda, \Delta^\perp)$, there exists $T' \in \mathsf{H2H}(\Lambda, \Delta)$ such that $T' = \mathsf{fcns}^{-1} \circ T$.*
2. *There exists $T' \in \mathsf{H2H}(\Lambda, \Delta)$ such that there is no $T \in \mathsf{H2B}(\Lambda, \Delta^\perp)$ satisfying $T' = \mathsf{fcns}^{-1} \circ T$.*

Proof. For Point (1), it is enough to apply fcns^{-1} to the right-hand side of transition rules of T (keeping sub-expressions $(q(x_i)$ unchanged) to obtain T'. For Point (2), for any well-nested word u let us define its size $|u|$ as the number of symbols occurring in it and its height $||u||$ as: (i) $||u|| = 0$ if $u = \varepsilon$ and (ii) $||cvrw|| = max(1 + ||v||, ||w||)$ if $u = cvrw$. Size and height can be defined on hedges h accordingly by considering size and heigth of $\mathsf{lin}(h)$. The following facts can easily be proved: (Fact 1) One can devise a transducer T' that flattens its input into a sequence $(T'(f(h_1)h_2) = c_f r_f T'(h_1)T'(h_2))$. Then, $|T'(h)| = 2|h|$ and $||T'(h)|| = 1$. (Fact 2) For all T in $\mathsf{H2B}(\Lambda, \Delta^{\perp})$, there exists k_T in $\mathbb{N}$ such that for all hedges h, $||T(h)|| \leq k_T||h||$; (Fact 3) If $w \in \mathscr{W}_{\Delta}$, $||w|| = 1$ and $|w| = n$ then $||\mathsf{fcns}(w)|| = n$.

Now, consider the family $H_{n|n\in\mathbb{N}}$ of hedges h such that $||h|| = n$ and $|h| = 2^n$. For any $h \in H_n$, $|T'(h)| = 2^{n+1}$ and $||T'(h)|| = 1$. Hence, $||\mathsf{fcns}(T'(h))|| = 2^{n+1}$. Assuming that T exists yields $||\mathsf{fcns}(T'(h))|| = 2^{n+1} = ||T(h)|| \leq k_T n$ for some constant k_T for all n. Contradiction. □

It turns out that $\mathsf{H2S}$ are stricly more expressive than VPTs. Formally:

Lemma 2. *There exists* $T \in \mathsf{H2S}(\Sigma_c \times \Sigma_r, \Delta)$ *such that for all* $A \in \mathsf{VPT}(\Sigma, \Delta)$, $T \circ \mathsf{hedge} \not\equiv A$.

Proof. Consider the variant over the input alphabet $\Sigma_c \times \Sigma_r$ of the transducer T defined in Example 1. It is easy to see this transducer produces an output (after an hedge application) only on nested words from $(\Lambda_c.\Lambda_r)^*$. Over such input words, any VPT admits only finitely many configurations in its accepting runs and thus, is equivalent to some finite state transducer. But it is well known that finite state transducer can not compute the mirror image of its inputs. □

Informally, this is due to the abitility that $\mathsf{H2S}$ have to "complete" the output once the current hedge is processed. This ability vanishes when tail-recursive $\mathsf{H2S}$ are considered.

Lemma 3. $\mathsf{VPT}(\Sigma, \Delta) \equiv \mathsf{H2S}_{\mathsf{tr}}(\Sigma_c \times \Sigma_r, \Delta)$.

(Sketch). Intuitively, in order to transform $A \in \mathsf{VPT}(\Sigma, \Delta)$ into $T \in \mathsf{H2S}_{\mathsf{tr}}(\Sigma_c \times \Sigma_r, \Delta)$, we proceed as follows. States of T are pairs of states of A, corresponding to states reached respectively at the beginning and at the end of the processing of an hedge. More formally, the following rule will exist in T iff there exist a call transition on c from p to p_1, a matching return on r from p_2 to q_1, the hedge represented by x_1 (resp. by x_2) can be processed from state p_1 to state p_2 (resp. from q_1 to q):

$$(p,q)((c,r)(x_1) \cdot x_2) \to w_1 \cdot (p_1,p_2)(x_1) \cdot w_2 \cdot (q_1,q)(x_2)$$

The word w_1 (resp. w_2) is the output of the call transition (resp. of the return transition). It is worth observing that this encoding directly implies the tail-recursive property of T.

The converse construction follows the same ideas. The stack is used to store the transition used on the call symbol, to recover it when reading the return symbol. □

Lemma 2 still holds even if we restrict $\mathsf{H2S}$ to $\mathsf{H2H}$, because the transducer defining the transduction of Example 1 is actually an $\mathsf{H2H}$. Similarly, Lemma 3 also holds when restricted to hedge-to-hedge transductions (the same constructions apply):

Lemma 4. $\mathsf{wnVPT}(\Sigma, \Delta) \equiv \mathsf{H2H}_{\mathsf{tr}}(\Sigma_c \times \Sigma_r, \Delta)$.

Removing the tail-recursive assumption As we have seen in the proof of Lemma 3, the behavior of a VPT is naturally encoded by a tail-recursive H2S. Intuitively, the word w_3 of rules of H2S should be produced after having processed the whole hedge.

We prove now that if we run VPTs on the fcns encoding of hedges, then we can express any H2S-definable transduction. Intuitively, in the fcns encoding, the return symbol of the root of the first tree of the hedge is encountered at the end of the processing of the hedge. As a consequence, the word w_3 can be output when processing this symbol. Formally, we have:

Lemma 5. $\mathsf{VPT}(\Sigma_\perp, \Delta) \circ \mathsf{fcns} \equiv \mathsf{H2S}(\Sigma_c \times \Sigma_r, \Delta)$.

(Sketch). A construction similar to the one presented in the proof of Lemma 3, based on pairs of states of the VPT, can be used to build an equivalent H2S. It will not necessarily be tail-recursive as the output of the return transition will be produced last. Note also that to handle empty subtrees encoded by $\perp_c \perp_r$, the resulting H2S may associate a non-empty output word to leafs. It is however not difficult to simulate such rules.

Conversely, the construction is a bit more complex. States of the VPT store the rule that is applied at the previous level, and the position in this rule (beginning, middle, or end). A special case is that of the first level, as there is no previous level. In this case, we store the initial state we started from. This information is stored in the stack, so as to recover it and faithfully simulate the application of the rule. The case of rules associated with leafs is handled using the $\perp_c, \perp_r$ symbols, and dedicated rules. Details can be found in the Appendix. □

Lemma 6. $\mathsf{wnVPT}(\Sigma_\perp, \Delta) \circ \mathsf{fcns} \equiv \mathsf{H2H}(\Sigma_c \times \Sigma_r, \Delta)$.

3.3 Comparison with other tree transducer models

H2S correspond to parameter-less macro forest transducers [9] without swapping nor copying, that output strings. Therefore by Lemma 3, VPTs are strictly less expressive than mfts. Macro tree transducers (mtts) are transducers on ranked trees [3]. To compare them with VPTs, which run on (linearization of) hedges, we use the first-child next-sibling encoding. As shown in [9], any mft is equivalent to the composition of two mtts on those encodings. Linear-size increase transformations (or transducers) are those transformations such that the size of an output is linearly bounded by the size of the input. In [7] it is shown that any linear-size increase transformation defined by an arbitrary composition of mtts is definable by a single linear-size increase mtt. Therefore, linear-size increase mfts are equivalent to linear-size increase mtts. Since VPTs clearly define linear-size increase transformations, they are also strictly included in mtts.

Top-down ranked tree transducers with the linear and non-swapping restrictions are equivalent to H2B transducers on first-child next-sibling encodings. By Lemma 1, we get that they are strictly less expressive than wnVPTs, and therefore VPTs. The arguments on the size of the ouputs in the proof of Lemma 1 still applies when dropping that restriction (the yield transduction cannot be defined), and therefore top-down ranked tree transducers are incomparable with VPTs. For the same reasons, bottom-up tree transducers are also incomparable with VPTs.

Finally, let us mention the *uniform tree transducers* introduced by Neven and Martens [8], and inspired by the XSLT language. These transducers can duplicate subtrees, but must use the same state to transform all the children of a node. For those reasons they are incomparable with VPTs [11].

A Appendix: Proof of Lemma 3 and 4

Proof. Let $A = (Q, I, F, \Gamma, \delta) \in \mathsf{VPT}(\Sigma, \Delta)$. We define $T = (Q', I', \delta') \in \mathsf{H2S_{tr}}(\Sigma_c \times \Sigma_r, \Delta)$ as follows:

- $Q' = \{(q_1, q_2) \in Q^2 \mid \exists w \in \mathscr{W}_\Sigma \text{ s.t. } (q_1, \bot) \xrightarrow{w} (q_2, \bot)\}$
- $I' = Q' \cap (I \times F)$
- for all $c \in \Sigma_c$, $r \in \Sigma_r$, and all states p_1, p_2, q_1, q_2, q_1' such that $(q_1, q_2), (p_1, p_2), (q_1', q_2) \in Q'$, if there exist transitions $(q_1, c, \gamma, w_1, p_1) \in \delta_c$, $(p_2, r, \gamma, w_2, q_1') \in \delta_r$, we build the rule:

$$(q_1, q_2)((c, r)(x_1) \cdot x_2) \to w_1 \cdot (p_1, p_2)(x_1) \cdot w_2 \cdot (q_1', q_2)(x_2)$$

  In addition, we also have:

$$(q_1, q_2)(0) \to \varepsilon \in \delta' \iff q_1 = q_2$$

It can be shown by induction that for all well-nested words $w \in \mathscr{W}_\Sigma$, T has a computation over $\mathsf{hedge}(w)$ leading to (q_1, q_2) producing w' iff A admits a run from $(q_1, \bot)$ to $(q_2, \bot)$ over w producing w'. Observe also that by definition T is tail-recursive.

Notice that if A is a wnVPT, then we have $w_1 w_2 \in \mathscr{W}_\Sigma$, and thus $T \in \mathsf{H2H}$. This proves one direction of Lemma 4.

Conversely, let us consider the transducer $T = (Q, I, \delta)$ from $\mathsf{H2S_{tr}}(\Sigma_c \times \Sigma_r, \Delta)$. We define $A = (Q', I', F', \Gamma', \delta') \in \mathsf{VPT}(\Sigma, \Delta)$ as follows: $Q' = Q$, $I' = I$, $F' = \{q \in Q \mid q(0) \to \varepsilon \in \delta\}$, $\Gamma' = \delta$ and for every rule $t = q((c, r)(x_1) \cdot x_2) \to w_1 q_1(x_1) w_2 q_2(x_2) \in \delta$, we add the following rules to δ':

$$(q, c, t, w_1, q_1) \qquad \{(q', r, t, w_2, q_2) \mid q' \in F'\}$$

It can be shown by induction that for all well-nested word $w \in \mathscr{W}_\Sigma$, B has a computation over $\mathsf{hedge}(w)$ leading to q producing w' iff A admits a run from $(q, \bot)$ to $(q', \bot)$ over w producing w', for some $q' \in F'$.

Notice that that if $B \in \mathsf{H2H}$, then we have $w_1 w_2 \in \mathscr{W}_\Sigma$, and thus A is a well-nested VPT. This proves the other direction of Lemma 4. □

B Appendix: Proof of Lemma 5 and 6

Proof. Let $A = (Q, I, F, \Gamma, \delta) \in \mathsf{VPT}(\Sigma_\bot, \Delta)$. We first define the two following sets:

$$X = \{(p, q) \in Q^2 \mid \text{ there exists a run } (p, \bot) \xrightarrow{cwr} (q, \bot) \text{ in } A, \text{ with } c \in \Sigma_c, r \in \Sigma_r, w \in \mathscr{W}_{\Sigma_\bot}\}$$
$$X_\bot = \{(p, q) \in Q^2 \mid \text{ there exists a run } (p, \bot) \xrightarrow{\bot_c \bot_r} (q, \bot) \text{ in } A\}$$

We define $B = (Q', I', \delta') \in \mathsf{H2S}(\Sigma_c \times \Sigma_r, \Delta)$ as follows: $Q' = X \cup X_\bot$, $I' = X \cap (I \times F)$, for all $(p, q) \in X_\bot$, and all transitions $(p, \bot_c, \gamma, w, p')$, $(p', \bot_r, \gamma, w', q)$, we add the following rule to δ': $(p, q)(0) \to ww'$.

In addition, for every $c \in \Sigma_c$, $r \in \Sigma_r$, and for every states p, q, p_1, p_2, p_3 such that $(p, q) \in X$, and $(p_1, p_2), (p_2, p_3) \in Q'$, if there exist a transition $(p, c, \gamma, w_1, p_1) \in \delta_c$ and a transition $(p_3, r, \gamma, w_3, q) \in \delta_r$, we build the rule:

$$(p, q)((c, r)(x_1) \cdot x_2) \to w_1 \cdot (p_1, p_2)(x_1) \cdot (p_2, p_3)(x_2) \cdot w_3$$

It can be shown by induction that for all well-nested word $w \in \mathscr{W}_\Sigma$, B has a computation over $\mathsf{hedge}(w)$ leading to (q_1,q_2) producing w' iff A admits a run from $(q_1,\bot)$ to $(q_2,\bot)$ over $\mathsf{fcns}(w)$ producing w'.

Observe also that B does not comply with the definition of $\mathsf{H2S}$ as the first set of rules may produce non-empty words. However, it is easy to transform B to ensure this property as follows: for every rule $(p,q)(0) \to x$, build a state (p,x,q), and add the rule $(p,x,q)(0) \to \varepsilon$. Then, modify the second set of rules by replacing (p,q) by (p,x,q), and introducing x at the convenient position in the output of the rule. Note that this transformation will result in non-empty "w_2" words.

In addition, we assumed that input words are non-empty. As a consequence, the fcns encodings considered as input are different from the word $\bot_c\bot_r$. This justifies that the initial states can be taken in X only. This also implies that the removing of non-empty leaf rules described before is correct, as every leaf rule will be applied in the context of some rule associated with an internal node.

Last, for the proof of Lemma 6, it is easy to verify that if A is a wnVPT, then $B \in \mathsf{H2H}$.

Conversely, let us consider the transducer $B = (Q,I,\delta)$ in $\mathsf{H2S}(\Sigma_c \times \Sigma_r, \Delta)$. We define $A = (Q',I',F',\Gamma',\delta')$ in $\mathsf{VPT}(\Sigma_\bot,\Delta)$ as follows:

$Q' = \{(q,i) \mid q \in I, i \in \{0,1\}\} \cup \{(t,i) \mid t \in \delta, i \in \{0,1,2\}\} \cup \{q_\bot\}$, $I' = I \times \{0\}$, $F' = I \times \{1\}$, $\Gamma' = Q'$, and for every rule $t = q((c,r)(x_1)\cdot x_2) \to w_1q_1(x_1)w_2q_2(x_2)w_3 \in \delta$ such that $q \in I$, we add the two following rules to δ':

$$((q,0),c,(q,0),w_1,(t,0)) \qquad ((t,2),r,(q,0),w_3,(q,1))$$

In addition, for every two rules

$$\begin{array}{rcl} t & = & q((c,r)(x_1)\cdot x_2) \to w_1q_1(x_1)w_2q_2(x_2)w_3 \in \delta \\ t' & = & q'((c',r')(x_1)\cdot x_2) \to w'_1q'_1(x_1)w'_2q'_2(x_2)w'_3 \in \delta \end{array}$$

and $i \in \{0,1\}$ such that $q'_i = q$, we add the two following rules to δ':

$$((t',i),c,(t',i),w_1,(t,0))$$

$$((t,2),r,(t',i),w_3x,(t',i+1)) \text{ where } x = \begin{cases} w'_2 & \text{if } i = 0 \\ \varepsilon & \text{otherwise} \end{cases}$$

Last, we consider rules associated with leafs: for every rule $q(0) \to \varepsilon$, we add the two following transitions: (provided that the i-th state of the rule t is q)

$$((t,i),\bot_c,(t,i),\varepsilon,q_\bot) \qquad (q_\bot,\bot_r,(t,i),\varepsilon,(t,i+1))$$

It can be shown by induction that for all well-nested word $w \in \mathscr{W}_\Sigma$, B has a computation over $\mathsf{hedge}(w)$ leading to q producing w' iff the two following properties are verified:

- if $q \in I$, then A admits a run from $(q,0)$ to $(q,1)$ over $\mathsf{fcns}(w)$ producing w'
- for every $(t,i) \in \delta \times \{0,1,2\}$ such that $t = p((c,r)(x_1)\cdot x_2) \to w_1q_1(x_1)w_2q_2(x_2)w_3 \in \delta$ and $q_i = q$, A admits a run from (t,i) to $(t,i+1)$ over $\mathsf{fcns}(w)$ producing $w'x$, where $x = \varepsilon$ if $i = 1$, and $x = w_2$ otherwise.

□

Acknowledgments We are very grateful to Sebastian Maneth and anonymous referees for helpful comments on this paper.

References

[1] Rajeev Alur & P. Madhusudan (2009): *Adding nesting structure to words*. *Journal of the ACM* 56(3), pp. 1–43, doi:10.1145/1516512.1516518.

[2] Hubert Comon, Max Dauchet, Rémi Gilleron, Christof Löding, Florent Jacquemard, Denis Lugiez, Sophie Tison & Marc Tommasi (2007): *Tree Automata Techniques and Applications*. Available at http://tata.gforge.inria.fr/.

[3] Joost Engelfriet & Sebastian Maneth (2003): *Macro tree translations of linear size increase are MSO definable*. *SIAM Journal on Computing* 32, pp. 950–1006, doi:10.1137/S0097539701394511.

[4] Emmanuel Filiot, Olivier Gauwin, Pierre-Alain Reynier & Frédéric Servais (2011): *Streamability of Nested Word Transductions*. In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, 13, pp. 312–324, doi:10.4230/LIPIcs.FSTTCS.2011.312.

[5] Emmanuel Filiot, Jean-François Raskin, Pierre-Alain Reynier, Frédéric Servais & Jean-Marc Talbot (2010): *Properties of Visibly Pushdown Transducers*. In: *Mathematical Foundations of Computer Science*, pp. 355–367, doi:10.1007/978-3-642-15155-2_32.

[6] Emmanuel Filiot & Frédéric Servais (2012): *Visibly Pushdown Transducers with Look-Ahead*. In: *International Conference on Current Trends in Theory and Practice of Computer Science*, pp. 251–263, doi:10.1007/978-3-642-27660-6_21.

[7] Sebastian Maneth (2003): *The Macro Tree Transducer Hierarchy Collapses for Functions of Linear Size Increase*. In: *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, pp. 326–337, doi:10.1007/978-3-540-24597-1_28.

[8] Wim Martens & Frank Neven (2003): *Typechecking Top-Down Uniform Unranked Tree Transducers*. In: *International Conference on Database Theory*, *Lecture Notes in Computer Science* 2572, Springer, pp. 64–78, doi:10.1007/3-540-36285-1_5.

[9] Thomas Perst & Helmut Seidl (2004): *Macro Forest Transducers*. *Information Processing Letter* 89(3), pp. 141–149, doi:10.1016/j.ipl.2003.05.001.

[10] Jean-François Raskin & Frédéric Servais (2008): *Visibly Pushdown Transducers*. In: *International Colloquium on Automata, Languages and Programming*, *Lecture Notes in Computer Science* 5126, pp. 386–397, doi:10.1007/978-3-540-70583-3_32.

[11] Frédéric Servais (2011): *Visibly Pushdown Transducers*. Ph.D. thesis, Université Libre de Bruxelles. Available at http://theses.ulb.ac.be/ETD-db/collection/available/ULBetd-09292011-142239/.

[12] Slawomir Staworko, Grégoire Laurence, Aurélien Lemay & Joachim Niehren (2009): *Equivalence of Deterministic Nested Word to Word Transducers*. In: *Fundamentals of Computer Theory*, *Lecture Notes in Computer Science* 5699, pp. 310–322, doi:10.1007/978-3-642-03409-1_28.