Expressiveness of Visibly Pushdown Transducers

Mathieu Caralp\textsuperscript{1}, Emmanuel Filiot\textsuperscript{2}, Pierre-Alain Reynier\textsuperscript{1}, Frédéric Servais\textsuperscript{3} and Jean-Marc Talbot\textsuperscript{1}

\textsuperscript{1} Laboratoire d’Informatique Fondamentale de Marseille, Aix-Marseille Université & CNRS, France
\textsuperscript{2} Laboratoire d’Algorithmique, Complexité et Logique, Université Paris-Est Créteil, France
\textsuperscript{3} Hasselt University & transnational University of Limburg, Belgium

Abstract

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. 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 \cite{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) \cite{9, 10, 5, 11} extend visibly pushdown automata \cite{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 strict subclass of pushdown transducers 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) \cite{5}. VPTs are closed by regular look-ahead which makes them a robust class of transformations \cite{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 transformation model in a streaming context, as shown in \cite{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 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 \cite{8}. 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 of 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 \( \Sigma \) is denoted by
\( \Sigma^* \), and the empty word is denoted by \( \epsilon \). A structured alphabet is a pair \( \Sigma = (\Sigma_c, \Sigma_r) \) of disjoint alphabets, of call and return symbols respectively. Given a structured alphabet \( \Sigma \), we always
denote by \( \Sigma_c \) and \( \Sigma_r \) its implicit structure, and identify \( \Sigma \) 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 \( \Sigma \) is the least set, denoted by \( W_\Sigma \), that satisfies (i) \( \epsilon \in W_\Sigma \), (ii) for all
\( w, w' \in W_\Sigma \), \( w w' \in W_\Sigma \) (closure under concatenation), and (iii) for all \( w \in W_\Sigma \), \( c \in \Sigma_c \), \( r \in \Sigma_r \),
cw r \in W_\Sigma . E.g. on \( \Sigma = (\{ c, r \}, \{ r \}) \), the nested word \( c r c r \) is well-nested while \( r c \) is not.
Finally, note that any well-nested word \( w \) is either empty or can be decomposed uniquely as
\( w = cw_1 rw_2 \) where \( c \in \Sigma_c \), \( r \in \Sigma_r \), \( w_1, w_2 \in W_\Sigma \).

Hedges Let \( \Lambda \) 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 \( H_\Lambda \) over \( \Lambda \) 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 \( h \in H_\Lambda \). Note that any hedge \( h \) is either empty or can be decomposed as \( h = a(h_1) \cdot h_2 \).

Hedges over \( \Lambda \) can be naturally encoded as well-nested words over the structured alphabet
\( \Lambda_c = (\Lambda_c, \Lambda_r) \) where \( \Lambda_c \) and \( \Lambda_r \) are new alphabets respectively defined by \( \Lambda_c = \{ c_a \mid a \in \Lambda \} \) and
\( \Lambda_r = \{ r_a \mid r \in \Lambda \} \). This correspondence is given via a morphism \( \text{lin} : H_\Lambda \rightarrow W_{\Lambda_c} \) inductively
defined by:
\[
\text{lin}(0) = \epsilon \quad \text{and} \quad \text{lin}(a(h_1), h_2) = c_a \text{lin}(h_1) r_a \text{lin}(h_2).
\]
E.g. for \( \Lambda = \{ a, b \} \), we have
\[
\text{lin}(ab) = c_a r_a c_b r_b r_a c_b r_b.
\]
Conversely, any well-nested word over a structured alphabet \( \Sigma \) can be encoded as an hedge
over the product alphabet \( \Sigma_c \times \Sigma_r \), via the mapping \( \text{hedge} : W_\Sigma \rightarrow H_{\Sigma_c \times \Sigma_r} \) defined as
\( \text{hedge}(\epsilon) = 0 \) and \( \text{hedge}(cw_1 rw_2) = (c, r)(\text{hedge}(w_1)) \cdot \text{hedge}(w_2) \) for all \( (c, r) \in \Sigma_c \times \Sigma_r \) and all \( w_1, w_2 \in W_\Sigma \).

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

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
\( \text{fcns} \) defined as: (i) \( \text{fcns}(0) = \bot \), (ii) \( \text{fcns}(f(h_1)h_2) = f(\text{fcns}(h_1) \text{fcns}(h_2)) \) for all \( h_1, h_2 \in H_\Lambda \).

The strong relationship between hedges and well-nested words can be considered when re-
stricted to binary trees: we define \( BW_\Lambda \), the set of binary well-nested words over the structured
alphabet \( (\Lambda_c \cup \{ \bot_c \}, \Lambda_r \cup \{ \bot_r \}) \) as the least set satisfying: (i) \( \bot_c \bot_r \in BW_\Lambda \) and (ii) for all
\( f_c \in \Lambda_c \), \( f_r \in \Lambda_r \), if \( w^1 w^2 \in BW_\Lambda \) then \( f_c w^1 w^2 f_r \in BW_\Lambda \). Note that the morphism \( \text{lin} \)
applied on binary trees from \( B_\Lambda \) yields binary nested words in \( BW_\Lambda \).

Finally, we can define the first-child next-sibling encoding of hedges as binary trees, directly
on linearizations; consider a structured alphabet \( \Sigma \) extended as \( \Sigma_\bot = (\Sigma_c \cup \{ \bot_c \}, \Sigma_r \cup \{ \bot_r \}) \). For all well-nested words \( w \) over \( \Sigma \), we define \( \text{fcns}(w) \) over the alphabet \( \Sigma_\bot \) recursively as (i)
\( \text{fcns}(cw_1 rw_2) = c \text{fcns}(w_1) \text{fcns}(w_2) r \) for all \( w_1, w_2 \in W_\Sigma \) and (ii) \( \text{fcns}(\epsilon) = \bot_c \bot_r \).
Visibly Pushdown Transducers Let $\Sigma$ be a structured alphabet, and $\Delta$ be an alphabet. A visibly pushdown transducer from $\Sigma$ to $\Delta$ (the class is denoted $VPT(\Sigma, \Delta)$) 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, $\Gamma$ the (finite) stack alphabet, $\perp \notin \Gamma$ is the bottom stack symbol, and $\delta = \delta_c \cup \delta_r$ 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 $(q, \sigma)$ where $q \in Q$ and $\sigma \in \perp \cdot \Gamma^*$ is a stack content. Let $w = a_1 \ldots a_l$ be a (nested) word on $\Sigma$, and $(q, \sigma), (q', \sigma')$ be two configurations of $A$. A run of the VPT $A$ over $w$ from $(q, \sigma)$ to $(q', \sigma')$ is a (possibly empty) sequence of transitions $\rho = t_1 t_2 \ldots t_l \in \delta^*$ such that there exist $q_0, q_1, \ldots, q_l \in Q$ and $\sigma_0, \ldots, \sigma_l \in \perp \cdot \Gamma^*$ with $(q_0, \sigma_0) = (q, \sigma), (q_l, \sigma_l) = (q', \sigma')$, and for each $0 \leq k \leq l$, we have either (i) $t_k = (q_{k-1}, a_k, \gamma, w_k, q_k) \in \delta_c$ and $\sigma_k = \sigma_{k-1} \cdot \gamma$, and (ii) $t_k = (q_{k-1}, a_k, \gamma, w_k, q_k) \in \delta_r$, and $\sigma_{k-1} = \sigma_k \cdot \gamma$. When the sequence of transitions is empty, $(q, \sigma) = (q', \sigma')$.

The output of $\rho$ is the word $w \in \Delta^*$ defined as the concatenation $w = w_1 \ldots w_l$ when the sequence of transitions is not empty and $\epsilon$ 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 $\mathcal{W}_2 \times \Delta^*$.

Hedge-to-string Transducers We present a model of hedge-to-string transducers (H2S) that run directly on hedges. They can be understood as parameter-free macro forest transducers (MFT) \cite{8} which produce strings.

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

$$
q(0) \rightarrow \epsilon \quad q(f(x_1) \cdot x_2) \rightarrow w_1 q_1(x_1) w_2 q_2(x_2) w_3
$$

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 $\llbracket q \rrbracket : \mathcal{H}_\Lambda \rightarrow 2^{\Delta^*}$ for all $q \in Q$ as follows:

$$
\llbracket q \rrbracket(0) = \begin{cases} 
\{ \epsilon \} & \text{if } q(0) \rightarrow \epsilon \in \delta \\
\emptyset & \text{otherwise}
\end{cases}
$$

$$
\llbracket q \rrbracket(f(h) \cdot h') = \bigcup_{q(f(x_1) \cdot x_2) \rightarrow w_1 q_1(x_1) w_2 q_2(x_2) w_3} w_1 \cdot \llbracket q_1 \rrbracket(h) \cdot w_2 \cdot \llbracket q_2 \rrbracket(h') \cdot w_3
$$

The transduction of a H2S $T = (Q, I, \delta)$ is defined as the relation $\{(h, s) \mid \exists q \in I, s \in \llbracket q \rrbracket(h)\}$. When $s \in \llbracket q \rrbracket(h)$ 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 $w_3 = \epsilon$. We denote by $H2S_{\text{tr}}$ the class of tail-recursive H2Ss.

\footnote{We consider linear and order-preserving rules only.}
Example 1. Let $\Lambda$ be a finite alphabet. Consider $T_1 \in H2S(\Lambda, \Lambda)$ defined by $Q = I = \{q, q'\}$ and the following rules:

\[
q(0) \rightarrow \epsilon \quad q'(0) \rightarrow \epsilon \quad q(f(x_1) \cdot x_2) \rightarrow q'(x_1)q(x_2)f
\]

$T_1$ defines the mirror image on strings (viewed as a particular case of hedges).

Example 2. Let $\Lambda$ be a finite alphabet and $\Lambda_s$ be its structured version. We define $T_2 \in H2S(\Lambda, \Lambda_s)$ which can embed any subhedge under a new symbol $\#$. Formally, the transducer builds the linearization of the resulting hedge. $T_2$ is defined by $Q = \{q_0, q_1, q_2\}$, $I = \{q_0\}$ and $\delta$ defined as the following set of rules: (observe that $T_2 \in H2S_v$)

\[
\begin{align*}
q_1(0) &\rightarrow \epsilon & q_0(f(x_1) \cdot x_2) &\rightarrow c_f q_0(x_1)rf_q q_0(x_2) \\
q_0(f(x_1) \cdot x_2) &\rightarrow c_f q_2(x_1)rf_q q_1(x_2) & q_1(f(x_1) \cdot x_2) &\rightarrow c_f q_2(x_1)rf_q q_1(x_2) \\
q_1(f(x_1) \cdot x_2) &\rightarrow c_f q_2(x_1)rf_q q_0(x_2) & q_2(f(x_1) \cdot x_2) &\rightarrow c_f q_2(x_1)rf_q q_2(x_2)
\end{align*}
\]

For instance, the input tree $f(abcd)$ can be translated into $\text{lin}(f(a\#(bc)d))$ or into $\text{lin}(f(\#(ab)\#(cd)))$.

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 $\Delta$ to be structured as $(\Delta_c, \Delta_s)$. We define a $H2S(\Lambda, \Delta)$ to be hedge-to-hedge transducer (H2H $(\Lambda, \Delta)$) if any right hand-side $w_1q_1(x_1)w_2q_2(x_2)w_3$ of its transition rules satisfies $w_1w_2w_3 \in W_\Delta$. We denote $H2H_v$ 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 $\text{VPT}_s^\Delta$. This synchronization is enforced syntactically on stack symbols, these symbols being shared by matching call and return transitions.

Let $A = (Q, I, F, \Gamma, \delta) \in \text{VPT}(\Sigma, \Delta)$. Then $A$ is well-nested if for all $(q, c, \gamma, w, q') \in \delta_c$ and $(p, r, \gamma', w', p') \in \delta_s$, $\gamma = \gamma'$ implies that $ww' \in W_\Delta$.

We denote by $\text{wnVPT}$ the class of well-nested $\text{VPT}_s$. 

Hedge-to-binary tree Transducers We consider here 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 $\Delta^\perp = (\Delta^\perp_c, \Delta^\perp_s)$ be a structured output alphabet such that $\Delta^\perp_c, \Delta^\perp_s$ contain two special symbols $\perp_c, \perp_s$, respectively. We define a H2H$(\Lambda, \Delta^\perp)$ to be an hedge-to-binary tree transducer (H2B$(\Lambda, \Delta^\perp)$) if any right hand-side $w_1q_1(x_1)w_2q_2(x_2)w_3$ of its transition rules satisfies $w_1 = c w'_1$, $w_2 = w'_2\perp s$, $w_3 = w'_3 r$ for some $c$ in $\Delta^\perp_c$, $r$ in $\Delta^\perp_s$, $w'_1 \perp c \perp l$, $w'_2 \perp s$ and $w'_3 \perp s \perp l$ in $BW_{\Delta^\perp}$.

Hedge-to-binary tree transducers are closed 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 will assume that input hedges accepted by transducers are non-empty. This restriction is done without loss of generality. We depict below the results we obtained:

| VPT $\equiv$ H2S, \text{(Lemma 3)} | $\subseteq$ \text{(Lemma 2)} | VPT $\circ$ fcns $\equiv$ H2S, \text{(Lemma 5)} |
| \text{fcns}^{-1} \circ \text{H2B} | $\subseteq$ \text{(Lemma 1)} | \text{wnVPT} $\equiv$ H2H, \text{(Lemma 4)} | \text{wnVPT} $\circ$ fcns $\equiv$ H2H, \text{(Lemma 2)} |

#### 3.1 Definitions of expressiveness

Let $\Sigma$ be a structured alphabet and $\Delta$ be a finite alphabet. We denote by $\mathcal{T}(\mathcal{W}_\Sigma, \Delta^*)$ the set of transductions from $\mathcal{W}_\Sigma$ to $\Delta^*$. First observe that the semantics of a transducer $A \in \text{VPT}(\Sigma, \Delta)$ is an element of $\mathcal{T}(\mathcal{W}_\Sigma, \Delta^*)$. Second, given a transducer $T \in \text{H2S}(\Sigma_c \times \Sigma_r, \Delta)$, we have that $T \circ \text{hedge} \in \mathcal{T}(\mathcal{W}_\Sigma, \Delta^*)$. Hence, up to the mapping $\text{hedge}$, we can thus compare the expressiveness of a subclass $C_1$ of $\text{VPT}(\Sigma, \Delta)$ and of a subclass $C_2$ of $\text{H2S}(\Sigma_c \times \Sigma_r, \Delta)$, by their interpretation as transductions from $\mathcal{W}_\Sigma$ to $\Delta^*$.

Formally, given $A \in \text{VPT}(\Sigma, \Delta)$ and $T \in \text{H2S}(\Sigma_c \times \Sigma_r, \Delta)$, we say that $A$ and $T$ are equivalent, denoted $A \equiv T$, whenever $A = T \circ \text{hedge}$. Given a subclass $C_1$ of $\text{VPT}(\Sigma, \Delta)$ and a subclass $C_2$ of $\text{H2S}(\Sigma_c \times \Sigma_r, \Delta)$, we say that $C_1$ is more expressive than $C_2$ (resp. less expressive), denoted $C_1 \supseteq C_2$ (resp. $C_1 \subset C_2$), whenever we have:

- for every $T \in C_2$, there exists $A \in C_1$ such that $A \equiv T$
- for every $A \in C_1$, there exists $T \in C_2$ such that $A \equiv T$, respectively

Last, we write $C_1 \equiv C_2$ whenever $C_1$ and $C_2$ are expressively equivalent meaning that both $C_1 \supseteq C_2$ and $C_1 \subseteq C_2$ hold.

#### 3.2 Comparing expressiveness

We first recall in the framework we proposed here a known expressiveness result [10] 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 $T \in \text{H2B}(\Lambda, \Delta^\perp)$, there exists $T' \in \text{H2H}(\Lambda, \Delta)$ such that $T' = \text{fcns}^{-1} \circ T$.

2. There exists $T' \in \text{H2H}(\Lambda, \Delta)$ such that no $T \in \text{H2B}(\Lambda, \Delta^\perp)$ satisfies $T' = \text{fcns}^{-1} \circ T$.

**Proof.** For Point 1, it is enough to apply $\text{fcns}^{-1}$ 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 $\|u\|$ of a well-nested word $u$ as: (i) $|u| = 0$ if $u = \epsilon$ and (ii) $\|uvw\| = \max(1 + \|v\|, \|w\|)$ if $u = cvw$. Definitions for size and height can be defined on hedges $h$ accordingly by considering size and height of $\text{lin}(h)$. The following facts can easily be proved: (Fact 1) One can devise a transducer $T'$ flattening its input into a sequence $(T'(f(h_1)h_2) = c_{rf}T'(h_1)T'(h_2))$. Then, $\|T'(h)\| = 2|h|$ and $\|T'(h)\| = 1$. (Fact 2) For all $T$ in $\text{H2B}(\Lambda, \Delta^\perp)$, there exists $k_T$ in $\mathbb{N}$ such that for all hedges $h$, $\|T(h)\| \leq k_T|h|$. (Fact 3) If $u \in \mathcal{W}_\Delta$, $\|u\| = 1$ and $|u| = n$ then $||\text{fcns}(w)|| = n$.

Now, consider the family $H_n$ 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, $||\text{fcns}(T'(h))|| = 2^{n+1}$. Assuming that $T$ exists yields $||\text{fcns}(T'(h))|| = 2^{n+1} = ||T(h)||$ for some constant $k_T$ for all $n$. Contradiction. 

QED
It turns out that $H2S$ are strictly more expressive than VPTs. Formally:

**Lemma 2.** There exists $T \in H2S(\Sigma_c \times \Sigma_r, \Delta)$ such that for all $A \in VPT(\Sigma, \Delta)$, $T \circ \text{hedge} \neq A$.

**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 ability that $H2S$ have to "complete" the output once the current hedge is processed. This ability vanishes when tail-recursive $H2S$ are considered.

**Lemma 3.** $VPT(\Sigma, \Delta) \equiv H2S_{tr}(\Sigma \times \Sigma, \Delta)$.

**Proof (Sketch).** Intuitively, to transform $A \in VPT(\Sigma, \Delta)$ into $T \in H2S_{tr}(\Sigma \times \Sigma, \Delta)$, 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 a hedge. More formally, the following rule will exist in $T$ iff there exist a call transition on $c$ from $q$ to $p_1$, a matching return on $r$ from $p_2$ to $q$, 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 $q$):

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

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 $H2S$ to $H2H$, because the transducer defining the transduction of Example 1 is actually an $H2H$. Similarly, Lemma 3 also holds when restricted to hedge-to-hedge transductions (the same constructions apply):

**Lemma 4.** $\text{wnVPT}(\Sigma, \Delta) \equiv H2H_{tr}(\Sigma \times \Sigma, \Delta)$.

**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 $\text{fcns}$ encoding of hedges, then we can express any $H2S$-definable transduction. Intuitively, in the $\text{fcns}$ 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.** $VPT(\Sigma_{\perp}, \Delta) \circ \text{fcns} \equiv H2S(\Sigma_c \times \Sigma_r, \Delta)$.

**Proof (Sketch).** A construction similar to that 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 ⊥_c, ⊥_r symbols, and dedicated rules. Details can be found in the Appendix.

**Lemma 6.** \( \text{wnVPT}(\Sigma, \Delta) \circ \text{fcns} \equiv \text{H2H}(\Sigma_c \times \Sigma_r, \Delta) \).

### 4 Conclusion

Clearly, as H2S is a subclass of macro forest transducers (mfts) [8] 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 [8], linear-size increase mtts on those encodings are equivalent to mfts. Therefore mtts are strictly more expressive than VPTs.

Top-down ranked tree transducers with the linear and order-preserving restrictions are equivalent to H2B transducers on first-child next-sibling encodings. By Lemma 1, we get that they are strictly less expressive than \( \text{wnVPTs} \), and therefore VPTs. The arguments on the size of the outputs 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 [7], 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 [10].

### References


A Appendix

A.1 Proof of Lemma 3 and 4

Proof. Let $A = (Q, I, F, \Gamma, \delta) \in \text{VPT}(\Sigma, \Delta)$. We define $T = (Q', I', \delta') \in \text{H2S}_\text{w}(\Sigma, \Delta)$ as follows:

- $Q' = \{(q_1, q_2) \in Q^2 \mid \exists w \in \mathcal{W}_\Sigma \text{ s.t. } (q_1, \perp) \xrightarrow{w} (q_2, \perp)\}$
- $I' = Q' \cap (I \times F)$
- for every $c \in \Sigma_c$, $r \in \Sigma_r$, and for every 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:
  
  $$(q_1, q_2)((c, r)(x_1) \cdot x_2) \rightarrow w_1 : (p_1, p_2)(x_1) \cdot w_2 : (q_1', q_2)(x_2)$$

In addition, we also have:

$$(q_1, q_2)(0) \rightarrow \epsilon \in \delta' \iff q_1 = q_2$$

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

Observe also, for the proof of Lemma 4 that if $A$ is a wnVPT, then we have $w_1w_2 \in \mathcal{W}_\Sigma$, and thus $T \in \text{H2H}$.

Conversely, let us consider $T = (Q, I, \delta) \in \text{H2S}_\text{w}(\Sigma, \Delta)$. We define $A = (Q', I', F', \Gamma', \delta') \in \text{VPT}(\Sigma, \Delta)$ as follows:

- $Q' = Q$
- $I' = I$
- $F' = \{q \in Q \mid q(0) \rightarrow \epsilon \in \delta\}$
- $\Gamma' = \delta$
- for every rule $t = q((c, r)(x_1) \cdot x_2) \rightarrow w_1q_1(x_1)w_2q_2(x_2) \in \delta'$, we add the following rules to $\delta'$:
  
  $$(q, c, t, w_1, q_1) \rightarrow \{q', r, t, w_2, q_2 \mid q' \in F'\}$$

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

Observe also, for the proof of Lemma 4 that if $B \in \text{H2H}$, then we have $w_1w_2 \in \mathcal{W}_\Sigma$, and thus $A$ is a well-nested VPT.

A.2 Proof of Lemma 5 and 6

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

- $X = \{(p, q) \in Q^2 \mid \text{there exists a run } (p, \perp) \overset{\text{wrt}}{\rightarrow} (q, \perp) \text{ in } A, \text{ with } c \in \Sigma_c, r \in \Sigma_r, w \in \mathcal{W}_\Sigma\}$
- $X_\perp = \{(p, q) \in Q^2 \mid \text{there exists a run } (p, \perp) \overset{\text{wrt}}{\rightarrow} (q, \perp) \text{ in } A\}$

We define $B = (Q', I', \delta') \in \text{H2S}(\Sigma, \Delta)$ as follows:
• \( Q' = X \cup X_\perp \)

• \( I' = X \cap (I \times F) \)

• for every \((p, q) \in X_\perp\), and every transitions \((p, \bot_c, \gamma, w, p'), (p', \bot_r, \gamma, w', q)\), we add the following rule to \( \delta' \):

\[
(p, q)(0) \to w w'
\]

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 transitions \((p, c, \gamma, w_1, p_1) \in \delta_c, (p_3, r, \gamma, w_3, q) \in \delta_r\), 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 \( w \in \mathcal{W}_2\), \( B \) has a computation over \( \text{hedge}(w) \) leading to \((q_1, q_2)\) producing \( w' \) iff \( A \) admits a run from \((q_1, \bot)\) to \((q_2, \bot)\) over \( \text{fcns}(w) \) producing \( w' \).

Observe also that \( B \) does not comply with the definition of \( \text{H2S} \) 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 \((p, x, q)(0) \to \epsilon\). 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 \( \text{"} w_2 \text{"} \) words.

In addition, we assumed that input words are non-empty. As a consequence, the \( \text{fcns} \) 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 \( \text{wnVPT} \), then \( B \in \text{H2H} \).

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

• \( Q' = \{(q, i) \mid q \in I, i \in \{0, 1\}\} \cup \{(t, i) \mid t \in \delta, i \in \{0, 1, 2\}\} \cup \{q_\perp\} \)

• \( I' = I \times \{0\} \)

• \( F' = I \times \{1\} \)

• \( \Gamma' = Q' \)

• for every rule \( t = q((c, r)(x_1) \cdot x_2) \to w_1 q_1(x_1) w_2 q_2(x_2) w_3 \in \delta \) such that \( q \in I \), we add the two following rules to \( \delta' \):

\[
((q, 0), c, (q, 0), w_1, (t, 0)) \quad ((t, 2), r, (q, 0), w_3, (q, 1))
\]

In addition, for every rule \( t = q((c, r)(x_1) \cdot x_2) \to w_1 q_1(x_1) w_2 q_2(x_2) w_3 \in \delta \), and for every rule \( t' = q'(c', r')(x_1) \cdot x_2) \to w'_1 q'_1(x_1) w'_2 q'_2(x_2) w'_3 \in \delta \), and \( i \in \{0, 1\} \) such that \( q'_i = q \), we add the two following rules to \( \delta' \):

\[
((t', i), c, (t', i), w_1, (t, 0)) \quad ((t, 2), r, (t', i), w_3 x, (t', i+1)) \text{ where } x = \begin{cases} w'_2 & \text{if } i = 0 \\ \epsilon & \text{otherwise} \end{cases}
\]
Last, we consider rules associated with leaves: for every rule $q(0) \rightarrow \varepsilon$, we add the two following transitions: (provided that the $i$-th state of the rule $t$ is $q$)

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

It can be shown by induction that for all well-nested word $w \in W_\Sigma$, $B$ has a computation over $\text{hedge}(w)$ 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 $\text{fcns}(w)$ producing $w'$
- for every $(t, i) \in \delta \times \{0, 1, 2\}$ such that $t = p((c, r)(x_1 \cdot x_2) \rightarrow w_1q_1(x_1)w_2q_2(x_2)w_3 \in \delta$ and $q_i = q$, $A$ admits a run from $(t, i)$ to $(t, i + 1)$ over $\text{fcns}(w)$ producing $w'x$, where $x = \varepsilon$ if $i = 1$, and $x = w_2$ otherwise.

$\square$