Expressiveness of Visibly Pushdown Transducers

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

Emmanuel Filiot
Laboratoire d’Algorithmique,
Complexité et Logique
Université Paris-Est Créteil, France
efiliot@lacl.fr

Frédéric Servais
Hasselt University & transnational University of Limburg, Belgium
fservais@gmail.com

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. 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-valuefulness (in NPTIME) and functional equivalence (EXPTIME-c) [5]. VPTs are closed by regular look-ahead which makes them a robust class of transformations.

Unranked trees and more generally hedges can be linearized into well-nested words over a structured alphabet (such as XML documents). VPTs 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 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. TRANSDUCTOR 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 Σ = (Σc, Σr) of disjoint alphabets, of call and return symbols respectively. Given a structured alphabet Σ, we always denote by Σc and Σr its implicit structure, and identify Σ with Σc ∪ Σr.

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 WΣ, that satisfies (i) ϵ ∈ WΣ, (ii) for all w, w′ ∈ WΣ, ww′ ∈ WΣ (closure under concatenation), and (iii) for all w ∈ WΣ, c ∈ Σc, r ∈ Σr, cwr ∈ WΣ. E.g. on Σ = (c1, c2), the nested word c1c2c1 is well-nested while 110 is not. Finally, note that any well-nested word w is either empty or can be decomposed uniquely as w = cw1rw2 where c ∈ Σc, r ∈ Σr, w1, w2 ∈ WΣ.

Hedges Let Λ be an alphabet. We let S(Λ) be the signature {0, ·} ∪ {a | a ∈ Λ} where 0 is a constant symbol, a ∈ Λ are unary symbols and · is a binary symbol. The set of hedges HΛ over Λ is the quotient of the free S(Λ)-algebra by the associativity of · and the axioms 0 · h = h · 0 = h. The constant 0 is called the empty hedge. We may write a instead of a(0), and omit · when it is clear from the context. Unranked hedges are particular hedges of the form a(h) where h ∈ HΛ. Note that any hedge h is either empty or can be decomposed as h = a(h1) · h2.

Hedges over Λ can be naturally encoded as well-nested words over the structured alphabet Λ∗ = (Λc, Λr) where
$\Lambda_1$ and $\Lambda_2$ are new alphabets respectively defined by $\Lambda_1 = \{a^i \mid a \in A\}$ and $\Lambda_2 = \{a^i \mid a \in A\}$. This correspondence is given via a morphism $f : \Lambda_2 \rightarrow \Lambda_1$ inductively defined by: $f(0) = \epsilon$ and $f(a(h_1)h_2) = c(a)h_1f(h_1)h_2$. E.g. for $\Lambda = \{a, b, c\}$, we have $f(abab) = cca_{\text{top}}ca_{\text{top}}ca_{\text{top}}ca_{\text{top}}$.

Conversely, any well-nested word over a structured alphabet $\Sigma$ can be encoded as an hedge over the product alphabet $\Sigma_1 \times \Sigma_2$, via the mapping $\text{hedge} : \mathcal{W}_\Sigma \rightarrow \mathcal{H}_{\Sigma_1 \times \Sigma_2}$ defined as $\text{hedge}(\epsilon) = 0$ and $\text{hedge}(\epsilon a_1\epsilon a_2\epsilon) = (c, r)(\text{hedge}(a_1)) \cdot \text{hedge}(a_2)$ for all $c, r \in \Sigma_1 \times \Sigma_2$ and all $w_1, w_2 \in \mathcal{W}_\Sigma$.

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

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 $h_1, h_2 \in \mathcal{H}_\Lambda$.

The strong relationship between hedges and well-nested words can be considered when restricted to binary trees: we define $\mathbf{BW}_\Lambda$, the set of binary well-nested words over the structured alphabet $(\Lambda \cup \{\bot\}, \Lambda \cup \{\bot\})$ as the least set satisfying: (i) $\bot \in \mathbf{BW}_\Lambda$, and (ii) for all $f \in \Lambda$, if $w \in \mathbf{BT}_\Lambda$ then $f(w_1, w_2) \in \mathbf{BT}_\Lambda$. Note that the morphism $\text{fcns}$ applied on binary trees from $\mathbf{BT}_\Lambda$ yields binary nested words in $\mathbf{BW}_\Lambda$.

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 $\mathbf{VPT}(\Sigma, \Delta)$) is a tuple $\mathcal{T} = (Q, I, \delta, \omega, \gamma)$ 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:

$q(0) \rightarrow \epsilon \quad q(f(x_1) \cdot x_2) \rightarrow w_1q_1(x_1)w_2q_2(x_2)w_3$

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

The semantics of $\mathcal{T}$ is defined via mappings $\text{m} : \mathcal{H}_\Lambda \rightarrow 2^{\Delta^*}$ for all $q \in Q$ as follows:

$\text{m}(q(0)) = \{q\} \quad \text{if } q(0) \rightarrow \epsilon \quad \text{otherwise}$

$\text{m}(q(f(h) \cdot h')) = \bigcup_{q(0) \rightarrow \epsilon} w_1 \cdot q([f(x_1) \cdot x_2]w_1q_1(x_1)w_2q_2(x_2)w_3)$

The transduction of an H2S $T = (Q, I, \delta)$ is defined as the relation $(h, s) \mapsto q \in I, s \in [q(h)]$. We 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 $\mathbf{H2S}_\tau$ the class of tail-recursive H2S.

**Example 1.** Let $\Lambda$ be a finite alphabet. Consider $T_2 \in \mathbf{H2S}(\Sigma, \Lambda)$ defined by $Q = I = \{q, q'\}$ and the following rules, for all $f \in \Lambda$:

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

The domain of $T_2$ is the set of strings over $\Lambda$ (viewed as a particular case of hedges) and $T_1$ defines the mirror image of strings.

**Example 2.** Let $\Lambda$ be a finite alphabet and $\Delta_1$ be its structured version. We define $T_q \in \mathbf{H2S}(\Lambda, \Delta_q)$ 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 $f(\#(ab)(bc)(cd))$. Formally, $T_q$ is defined by $Q = \{q_0, q_1, q_2\}$, $I = \{q_0\}$ and $\delta$ defined as the following set of rules: (observe that $T_q \in \mathbf{H2S}_\tau$):

$q(0) \rightarrow \epsilon \quad \forall i \in \{0, 2\}$

$q_0(f(x_1) \cdot x_2) \rightarrow c_rq_0(x_1)r_fq_0(x_2)$

$q_0(f(x_1) \cdot x_2) \rightarrow c_\gamma c_rq_2(x_1)r_fq_0(x_2)$

$q_0(f(x_1) \cdot x_2) \rightarrow c_\gamma c_rq_2(x_2)r_fq_0(x_2)$

$q_0(f(x_1) \cdot x_2) \rightarrow c_rq_2(x_1)r_fq_0(x_2)$

$q_1(f(x_1) \cdot x_2) \rightarrow c_rq_2(x_2)r_fq_0(x_2)$

$q_1(f(x_1) \cdot x_2) \rightarrow c_rq_2(x_2)r_fq_0(x_2)$

$q_2(f(x_1) \cdot x_2) \rightarrow c_rq_2(x_1)r_fq_0(x_2)$

We consider linear and order-preserving rules only.
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 $(\Delta_1, \Delta_r)$. We define an $H2S(\Lambda, \Delta)$ to be a hedge-to-hedge transducer $(H2H(\Lambda, \Delta))$ if any rhs $w_1 q(x) w_2 q(x) v_1 v_2$ of its transition rules satisfies $w_1 w_2 v_1 v_2 \in \mathcal{W}_\Delta$. We denote $H2H_r$ 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 \[^3]\]. This synchronization is enforced syntactically on stack symbols, these symbols being shared by matching call and return transitions.

Let $A = (Q, I, F, \Gamma, \delta) \in VPT(\Sigma, \Delta)$. Then $A$ is well-nested if for all $(q, c, r, w, q') \in \delta$, and $(p, r', \gamma', w', p') \in \delta_r$, $\gamma' \equiv \gamma$ implying $w w' \in \mathcal{W}_\Delta$.

We denote by $\text{vwVPT}$ 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 $\Delta^* = (\Delta^+_t, \Delta^+_r)$ be a structured output alphabet such that $\Delta^+_t, \Delta^+_r$ contain two special symbols $\bot, \perp$, respectively. We define a $H2H(\Lambda, \Delta^*)$ to be an hedge-to-binary tree transducer ($H2B(\Lambda, \Delta^*)$) if any right-hand-side $w_1 q_1(x_1) w_2 q_2(x_2) v_3$ of its transition rules satisfies $w_1 = c w_1', w_2 = w_2' \triangledown w_3 = w_3', r$ for some $c \in \Delta_t^+, r \in \Delta_r^+$, $w_1^\perp w_1', w_2^\perp w_2', w_3^\perp w_3'$ in $\mathcal{B}_\mathcal{V} \mathcal{A}_\mathcal{D}$.

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

Formally, given $A \in \text{VPT}(\Sigma, \Delta)$ and $T \in \text{H2S}(\Sigma \times \Sigma_r, \Delta)$, we say that $A$ and $T$ are equivalent, denoted $A \equiv T$, whenever $A \circ \text{hedge} \equiv T \circ \text{hedge}$. Given a subclass $C_1$ of $\text{VPT}(\Sigma, \Delta)$ and a subclass $C_2$ of $\text{H2S}(\Sigma \times \Sigma_r, \Delta)$, we say that $C_1 \subseteq C_2$ (resp. less expressive), denoted $C_1 \preceq C_2$.

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 $\Delta = (\Delta_r, \Delta_s)$ and $\Delta^\perp = (\Delta \cup \{\bot\}, \Delta_r \cup \{\perp\})$ be two structured alphabets.

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

**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 $(q(x))$ unchanged) to obtain $T'$. For Point 2, for any well-nested word $u$ we define its size $|u|$ as the number of symbols occurring in it and its height $\text{height}(u)$:

\[ |u| = 0 \quad \text{if} \quad \epsilon \in u \quad \text{and} \quad (i) \quad |u| = \text{max}(1 + |v|, |w|) \quad \text{if} \quad u = cvw. \]

Size and height can be defined on hedges $h$ accordingly by considering size and heigth of $\text{lin}(h)$. The following facts can easily be proved:

- (Fact 1) One can devise a transducer $T'$ that flattens its input into a sequence $(T'(f(b_1))_{b_1} = c(f)_{r1} T'(h_1)_{T'(h_2)})$. Then, $|T'(h)| = 2|h|$ and $|T|(h)| = 1$. (Fact 2) For all $T \in H2B(\Lambda, \Delta^\perp)$, there exists $k_r \in \mathbb{N}$ such that for all hedges $h$, $|T(h)| \leq k_r |h|$. (Fact 3) If $w \in \mathcal{W}_\Delta$, $|w| = 1$ and $|w| = n$ then $|\text{fcns}(w)| = n$. Consider the family $H_{n \in \mathbb{N}}$ 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, $|\text{fcns}(T'(h))| = 2^{n+1}$. Assuming that $T$ exists yields $|\text{fcns}(T'(h))| = 2^{n+1} = |T(h)| \leq k_r n$ for some constant $k_r$ for all $n$. Contradiction.

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

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

**Proof.** Consider the variant over the input alphabet $\Sigma \times \Sigma_r$ of the transducer $T$ defined in Example \[^1\]. It is easy to see this transducer produces an output (after an $\text{hedge}$ application) only on nested words from $(\Lambda, \Lambda_r)^*$. 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.** $\text{VPT}(\Sigma, \Delta) \not\equiv \text{H2S}_h(\Sigma \times \Sigma_r, \Delta)$.

**Proof (Sketch).** Intuitively, in order to transform $A \in \text{VPT}(\Sigma, \Delta)$ into $T \in H2S_h(\Sigma \times \Sigma_r, \Delta)$, 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 transition on \( c \) from \( p \) to \( p_1 \), and 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 \( q_2 \)):

\[
(p, q)((c, r)(x_1) \cdot x_2) \rightarrow w_1 \cdot (p_1, p_2)(x_1) \cdot w_2 \cdot (q_1, q_2)(x_2)
\]

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 \( H2B \), because the transducer defining the transduction of Example 1 is actually an \( H2B \). Similarly, Lemma 3 also holds when restricted to hedge-to-hedge transductions (the same constructions apply):

**Lemma 4.** \( \text{wnVPT}(\Sigma, \Delta) \equiv H2B(\Sigma_c \times \Sigma_r, \Delta) \).  

**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 \( w_1 \) 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 \( w_2 \) can be output when processing this symbol. Formally, we have:

**Lemma 5.** \( \text{VPT}(\Sigma_c, \Delta) \circ \text{fcns} \equiv H2S(\Sigma_c \times \Sigma_r, \Delta) \).

**Proof (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 leaves. 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 leaves is handled using the \( \perp_c, \perp_r \) symbols, and dedicated rules. Details can be found in the Appendix.

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

3.3 Comparison with other tree transducers

\( H2S \) correspond to parameter-less macro forest transducers \( \text{H2S} \) without swapping nor copying, that output strings. Therefore by Lemma 4 VPTs are strictly less expressive than mfts. However, macro tree transducers (mfts) are transducers on ranked trees \( \text{H2S} \). To compare them with VPTs, which run on (linearization of) hedges, we use the first-child next-sibling encoding. As shown in \( \text{H2S} \), any mft is equivalent to the composition of two mfts 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 \( \text{H2S} \), it is shown that any linear-size increase transformation defined by an arbitrary composition of mfts is definable by a single linear-size increase mft. Therefore, linear-size increase mfts are equivalent to linear-size increase mfts. Since VPTs clearly define linear-size increase transformations, they are also strictly included in mfts.

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 \( \text{wnVPTs} \), 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 \( \text{H2S} \), 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.

**APPENDIX**

A. **Proof of Lemma 3 and 4**

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

- \( Q' = \{(q_1, q_2) \in Q^2 \mid \exists w \in W_c \text{ s.t. } (q_1, \perp) \xrightarrow{w} (q_2, \perp)\} \)
- \( I' = Q' \cap (I \times F) \)

- 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 \((q_1, c, r, w_1, p_1) \in \delta_c, (p_2, r, \gamma, w_2, q_1') \in \delta_r\), we build the rule:

\[
(q_1, q_2)((c, r)(x_1) \cdot x_2) \rightarrow w_1 \cdot (p_1, p_2)(x_1) \cdot w_2 \cdot (q_1', q_2)(x_2)
\]
In addition, we also have:

\[(q_1, q_2)(0) \to \epsilon \in \delta' \iff q_1 = q_2\]

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

Notice that if \(A\) is a \(\text{wnVPT}\), then we have \(w_1 w_2 \in W_{\Sigma}\), and thus \(T \in H2\). This proves one direction of Lemma 3.

Conversely, let us consider the transducer \(T = (Q, I, \delta)\) from \(H2S(\Sigma_{c} \times \Sigma_{r}, \Delta)\). We define \(A = (Q', I', F', \Gamma', \delta')\) in \(\text{VPT}(\Sigma_{c}, \Delta)\) as follows: \(Q' = Q\), \(I' = I\), \(F' = \{q \in Q \mid q(0) = \epsilon \in \delta\}\), \(\Gamma' = \delta\) 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 \(\delta'\):

\[(q, c, t, w_1, q_1) \quad \{q'(r, t, w_2, q_2) \mid q' \in F'\}\]

It can be shown by induction that for all well-nested word \(w \in W_{\Sigma}\), \(B\) has a computation over \(\text{hedge}(w)\) leading to \(q\) producing \(w'\) iff \(A\) admits a run from \((q_1, \bot)\) to \((q_2, \bot)\) over \(w\) producing \(w'\), for some \(q' \in F'\).

Notice that if \(B \in H2\), then we have \(w_1 w_2 \in W_{\Sigma}\), and thus \(A\) is a well-nested \(\text{VPT}\). This proves the other direction of Lemma 3.

B. PROOF OF LEMMA 5 AND 6

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

\[X = \{(p, q) \in Q^2 \mid \text{there exists a run } (p, \bot) \xrightarrow{\text{cut}} (q, \bot) \text{ in } A, \text{ with } c \in \Sigma_{r}, r \in \Sigma, w \in W_{\Sigma}\}\]

\[X_{\bot} = \{(p, q) \in Q^2 \mid \text{there exists a run } (p, \bot) \xrightarrow{\delta_{\leftarrow}} (q, \bot) \text{ in } A\}\]

We define \(B = (Q', I', \delta') \in H2S(\Sigma_{c} \times \Sigma_{r}, \Delta)\) as follows: \(Q' = X \cup X_{\bot}\), \(I' = X \cap (I \times F)\), for all \((p, q) \in X_{\bot}\), and all transitions \((p, \bot, c, x, p', \gamma, w, p')\), \((p', \bot, r, \gamma, w', q')\), we add the following rule to \(\delta'\):

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

It can be shown by induction that for all well-nested word \(w \in W_{\Sigma}\), \(B\) 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 \(H2S\) 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 \((p, q)(0) \to x, \text{ build a state } (p, x, q)\), 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 "\(w'x\)" 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 \(\bot, \bot, \bot\). 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 3 it is easy to verify that if \(A\) is a \(\text{wnVPT}\), then \(B \in H2\).

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

\[Q' = \{(q, i) \mid q \in I, i \in \{0, 1\} \cup \{t, i \mid t \in \delta, i \in \{0, 1\}\}, I' = I \times \{0\}, F' = I \times \{1\}, \Gamma' = Q'\}

and for every rule \(t = q((c, r)(x_1) \cdot x_2) \to w_1 q_1(x_1) w_2 q_2(x_2) w_3 \in \delta\) 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 two rules

\[t = q((c, r)(x_1) \cdot x_2) \to w_1 q_1(x_1) w_2 q_2(x_2) w_3 \in \delta\]

\[t' = q'((c', r')(x_1) \cdot x_2) \to w'_1 q'_1(x_1) w'_2 q'_2(x_2) w'_3 \in \delta\]

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

\[
((t', i), c, (t', i), w_3 x, (t', i + 1)) \text{ where } x = \begin{cases} w'_i & \text{if } i = 0 \\ \epsilon & \text{otherwise} \end{cases}
\]

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

\[((t, i), \bot, (t, i), \epsilon, q_i) \quad (q_i, \bot, (t, i), \epsilon, (t, i + 1))\]

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 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 \((t, i) \in \delta \times \{0, 1\}\) such that \(t = p((c, r)(x_1) \cdot x_2) \to w_1 q_1(x_1) w_2 q_2(x_2) w_3 \in \delta\) and \(q_i = q\). \(A\) admits a run from \((t, i)\) to \((t, i + 1)\) over \(\text{fcns}(w)\) producing \(w'x\), where \(x = \epsilon\) if \(i = 1\), and \(x = w_2\) otherwise.

\[\Box\]

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

C. REFERENCES


