Two-Way Visibly Pushdown Automata and Transducers

Luc Dartois  Emmanuel Filiot
Université Libre de Bruxelles, Belgium
{idartois,efiliot}@ulb.ac.be

Pierre-Alain Reynier  Jean-Marc Talbot
LIF, Aix-Marseille University, CNRS, France
firstname.lastname@lif.univ-mrs.fr

Abstract
Automata-logic connections are pillars of the theory of regular languages. Such connections are harder to obtain for transducers, but important results have been obtained recently for word-to-word transformations, showing that the three following models are equivalent: deterministic two-way transducers, monadic second-order (MSO) transducers, and deterministic one-way automata equipped with a finite number of registers. Nested words are words with a nesting structure, allowing to model unranked trees as their depth-first-search linearisations. In this paper, we consider transformations from nested words to words, allowing in particular to produce unranked trees if output words have a nesting structure. The model of visibly pushdown transducers allows to describe such transformations, and we propose a simple deterministic extension of this model with two-way moves that has the following properties: i) it is a simple computational model, that naturally has a good evaluation complexity; ii) it is expressive: it subsumes nested word-to-word MSO transducers, and the exact expressiveness of MSO transducers is recovered using a simple syntactic restriction; iii) it has good algorithmic/closure properties: the model is closed under composition with a unambiguous one-way letter-to-letter transducer which gives closure under regular look-around, and has a decidable equivalence problem.

Keywords  Transductions, Pushdown automata, Logic.

1. Introduction

Pillars of word language theory  The theory of languages is one of the deepest and richest theory in computer science, with successful applications such as, computer-aided verification and synthesis. A major reason for this success is the strong connections between models of languages, with quite different flavours, that are based on two important pillars: computation and logic. Perhaps one of the most famous example is the effective correspondence for regular languages of finite words between a low-level computational model, finite state automata, and a high-level declarative formalism, monadic second-order logic (MSO). Similar connections have been obtained for other structures (e.g. infinite words, finite and infinite trees, nested words) (Thomas 1997; Comon-Lundh et al. 2007). In some cases, it has been even possible to build a third pillar based on algebra. The class of regular languages for instance is known to be the class of languages with finite syntactic congruence.

The logic/two-way/one-way trinity of word transductions  To model functions from (input) words to (output) words, i.e. word transductions, and more generally word binary relations, automata have been extended to transducers, i.e. automata with outputs. Whenever a transducer reads an input symbol, it can produce on the output a finite word, the final output word being the right concatenation of all the finite words produced along the way. To capture functions mirroring or copying twice the input word, transducers need to read the input word in both directions: this yields the class of two-way finite state transducers (2FST). Two-way transducers have appealing properties: they are closed under composition (Chytil and Jäkl 1977) and if they are deterministic, their equivalence problem is decidable (in PSpace) (Gurari 1982; Culik and Karhumaki 1987) and the transduction can be evaluated in constant space (for a fixed transducer), the output being produced on-the-fly. Impressively, in the late 90s, deterministic two-way transducers have been shown in (Engelfriet and Hoogeboom 2001) to correspond to monadic second-order transducers (MSOT), a powerful logical formalism introduced in (Courcelle 1994) in a more general context, with independent motivations. It was the first logic-transducer connection obtained for a class of word transductions expressive enough to capture interesting and desirable transductions. This correspondence has been extended to finite tree transductions (Engelfriet and Maneth 1999, 2003; Bloem and Engelfriet 2000).

Recently, an MSOT-expressive one-way model, streaming string transducers (SST), has been introduced in (Alur and Cerný 2010, 2011): it uses registers that can store output words and can be combined and updated along the run in a linear (copyless) manner (a register content cannot be used twice in an update). The main advantage of this model is its one-wayness, but the price to pay is the space complexity of evaluation: it depends also on the size of the register contents.

The models MSOT, deterministic 2FST and deterministic SST have the same expressive power, and we refer to this correspondence as the logic/two-way/one-way trinity. This trinity has been extended to transductions of infinite words (Alur et al. 2012) and ranked trees (Courcelle and Engelfriet 2012; Alur and D’Antoni 2012). For trees, bi-directionality is replaced by a tree walking ability: the transducer can move along the edges of the tree in any direction. However, to capture MSOT, the transducer needs to have regular look-around, i.e. needs to be able to test regular properties of the context of the tree node in which it is currently positioned (Courcelle and Engelfriet 2012). Look-arounds can be removed at
the price of adding a pushdown store (Courcelle and Engelfriet 2012). For one-way machines, uni-directionality is modeled by fix-
ing the traversal of the tree to be a depth-first left-to-right traversal and, as for words, to capture MSOT, the transducer needs to have registers (Alur and D’Antoni 2012). Tree-walking transducers with look-around, and tree transducers with registers are strictly more expressive than MSOT, but restrictions have been defined that capture exactly MSOT. Finally, let us mention the macro tree transducers, the first computational model shown to capture, with suit-

able restrictions, MSOT ranked tree transductions (Engelfriet and Maneth 1999, 2003; Bloem and Engelfriet 2000). This model has parallel computations, like a top-down tree automaton, and regis-

ters.

**Nested words** In this paper, we consider transductions of nested words to words. *Nested words* are words with a nesting structure, built over symbols of two kinds: call and return symbols\(^1\). In partic-

ular, nested words can model ordered unranked trees, viewed as their depth-first, left-to-right, linearisation, and in turn are a natural model of tree-structured documents, such as XML documents. Vis-
ibly pushdown automata (VPA) have been introduced in (Alur and Madhusudan 2009) as a model of regularity for languages of nested words. They are pushdown automata with a constrained stack pol-

icy: whenever a call symbol is read, exactly one symbol is pushed onto the stack, and when reading a return symbol, exactly one sym-

bol is popped from the stack. Therefore, at any point, the height of the stack corresponds to the nesting level (call depth) of the word. Roughly, VPA are tree automata over linearised trees, and as such they inherit all the good closure and algorithmic prop-

erties of tree automata. However, viewing trees as nested words has raised motivating questions in the context of tree streams, such as streaming XML validation (Picalausa et al. 2011; Segoufin and Sir-

angelo 2007), streaming XML queries (Kumar et al. 2007; Gauvin et al. 2011), as well as streaming XML transformations (Filiot et al. 2011) (see also (Alur 2016) for other applications of VPA).

By using a matching predicate \(M(x, y)\) that holds true if \(x\) is a call symbol, \(y\) is a return symbol and is the matching return of \(x\), MSO logic can be extended from words to nested words, and it is known to correspond to regular nested word languages (Alur and Madhusudan 2009).

**Nested word to word transductions** Besides the motivations given before for considering nested words instead of unranked trees, we argue that seeing unranked trees as nested word yields a natural and simple two-way model for transductions of nested words, presented later. On the output, we do not require the words to have a particular structure. It is not a weakness: nested words are words, and the model we introduce in this paper can as well produce output words that are nested.

VPA have been extended with output, yielding the class of visibly pushdown transducers (VPT, (Filiot et al. 2010)). When reading an input symbol, VPT can generate a word on the output. VPT have good algorithmic and closure properties, and are well-
suited to a streaming context (Filiot et al. 2011). However, VPT suffer from a low expressive power, as they are only one-way, without registers.

Based on MSO for nested words, one can define MSO transducers à la Courcelle to define nested word to word transductions. From now on, we refer to such MSO transducers as MSOT. A one-

way model has already been defined in (Alur and D’Antoni 2012) that captures exactly MSOT. They extend VPA with registers that can store partial output words. Whenever a call symbol is read, the

\(^1\) Sometimes, internal symbols are also considered but in this paper, to ease the presentation, we omit them. This is wlog as an internal symbol \(a\) can be harmlessly replaced by a call symbol \(ca\) followed by a return symbol \(ra\).

**Objective and two-way visibly pushdown transducers** Our main goal in this paper is to establish a logic/two-way/one-way trinity for nested word to word transductions. Since the logic/one-way connection has already been shown in (Alur and D’Antoni 2012), we want in particular to define a two-way computational model with the following requirements: it must be conceptually simple, at least as expressive as MSOT and have decisive equivalence problem.

To this aim, we introduce deterministic two-way visibly push-
down transducers (D2VPT) and show it meets the later require-
ments. D2VPT read their input in both directions, and their stack behaviour not only depends on the type of symbols they read, but also on the reading mode they are in, either backward or forward. In a forward mode, they behave just like VPT. On the backward mode, they behave like VPT where the call and return types are swapped: when reading a return symbol backward, they push a symbol onto the stack, and when reading a call symbol backward, they pop a symbol from the stack. They can change their mode at any moment, and produce words on the output.

Let us give now an illustrating example of a transduction \(f_2\) of nested words, which will be formalised in Example 1. Assume a set of call symbols \(\{1, \ldots, n\}\) ordered by the total order on natural numbers, and one return symbol \(\{r\}\). The transduction \(f_2\) sorts an input nested word in ascending order, recursively nest-
ing level by nesting level, according to the order on calls. We assume inputs start and end with special symbols \(>\) and \(<\) (call and return resp.). E.g., \(f_2\) maps \(22r1r33r\) to \(r1r2r3r\) and \(23r1r2r3r\) to \(r1r2r3r2r3r\). To make \(f_2\) a function in case the same call symbol occurs twice at the same level, \(f_2\) preserves their order of appearance. The tree repre-

sentation of this mapping is given in Figure 1 (omitting return sym-

bols). The transduction \(f_2\) is easily implemented with a D2VPT \(T_\alpha\). To process a sequence of siblings at level \(k\), \(T_\alpha\) works as fol-

lows: for \(i\) from 0 to \(n\), \(T_\alpha\) performs a forward pass on the siblings (note that a sibling is actually a tree whose linearisation is of the form \(wur\) where \(w\) is again a sequence of linearised trees). During this forward pass, \(T_\alpha\) transforms a sibling \(wur\) into \(s\) if \(j \neq i\), and into \(w^r\) otherwise, where \(w^r\) is the result of sorting recursively \(w\). To implement the loop, when \(T_\alpha\) has finished the \(i\)-th forward pass, i.e., when it reads a return symbol at level \(j = \alpha\), it comes back to its matching call and starts from there the \((i+1\)-th forward pass, if \(i < n\).

**Contributions** By linearising input trees, the simple and well-

known concept of bi-directionality can be generalised naturally from words to trees. While D2VPT, as we show in this paper, allow one to lift known results from word transductions to nested word to word transductions, we think that D2VPT are an appealing model for the following reasons:

**memory efficiency** Regarding the complexity of evaluation, for a fixed D2VPT, computing the output word of an input nested word \(w\) can be done in space \(O(d(w))\), where \(d(w)\) is the depth of \(w\). Indeed, only the stack and current state need to be kept in memory when processing an input nested word. It is an appealing property when transforming large but not deep tree-structured documents, such as XML documents in general.

**expressiveness** At the same time, we show that this efficiency does not entail expressive power: D2VPT can express all MSOT transductions. They are strictly more expressive than MSOT.
as they can for instance express transduction of exponential size increase, while MSOT are only of linear size increase. By putting a simple decidable restriction on D2VPT, called single-useness, D2VPT capture exactly MSOT transductions.

**Algorithmic properties**  Despite their high expressive power, D2VPT still have **decidable equivalence problem**. We also prove that preprocessing the input of a D2VPT by a letter-to-letter unambiguous VPT does not increase its expressive power, as their composition is again a D2VPT.

The proof of expressiveness relies on an existing correspondence between tree-walking and MSO transducers of ranked trees to words (Courcelle and Engelfriet 2012), and on the classical first child-next sibling (fcsn for short) encoding of unranked trees into binary trees. As in (Courcelle and Engelfriet 2012), we use an intermediate automata model equipped with MSO look-around, and then show that these look-around tests can be removed. For the latter property, our proof differs from that of (Courcelle and Engelfriet 2012) in which a pushdown stack is used to update information on MSO-types. On binary trees, their model pushes the stack while moving to the first-child, but also while moving to the second child. This latter push corresponds, through the fcsn encoding, to pushing a symbol while moving to the next sibling, an operation that is not allowed with a visibly pushdown stack. Hence, in order to prove that look-around tests can be removed in our model, we need a more involved construction, that extends a non-trivial result proven in (Hopcroft and Ullman 1967) for two-way automata on words. Decidability of D2VPT equivalence is done by reduction to deterministic top-down tree to word transducer equivalence, a problem which was opened for long and recently solved in (Seidl et al. 2015).

**Application 1: Unranked tree to word walking transducers** D2VPT can easily be translated into a pushdown walking model of unranked tree to word transductions. It works exactly as in the ranked tree case of (Courcelle and Engelfriet 2012): one stack symbol is pushed while going downward and popped while going upward. While moving along sibling relations, the stack is untouched. As a consequence of our results, this model, with single-use restriction, captures exactly MSOT. This model is discussed in the last section.

**Application 2: Query 2VPA** Deterministic two-way VPA have been introduced in (Madhusudan and Viswanathan 2009) as an equi-expressive model for MSO-definable unary queries on nested words. Using (Neven and Schwentick 2002; Niehren et al. 2005), such queries can be shown to be equivalent to unambiguous VPA with special states which select the nested word positions that are answers to the query. As shown in (Madhusudan and Viswanathan 2009), unambiguity can be traded for determinism, at the price of adding two-wayness. This result comes as a consequence of ours: a one-way unambiguous selecting VPA can be seen as a deterministic VPT with look-around, that annotates the input positions selected by the VPA (look-around resolves nondeterminism), which can be transformed into a D2VPT using our results. The main ingredient of the proof of (Madhusudan and Viswanathan 2009) is also a Hopcroft-Ullman construction, but in a setting simpler than ours.

**Organisation of the paper**  In Section 2, we introduce two-way VPA and two-way VPA with look-around, define the notion of transition algebra for 2VPA and use this to show that they are equivalent to one-way VPA. As a consequence, they have decidable (emptiness-c) emptiness problem. In Section 3, we introduce D2VPT and D2VPT with look-around, show that they are equivalent, and study their algorithmic properties. Section 4 is devoted to the expressiveness of D2VPT, with a comparison to MSOT and to other known models of nested word to word transductions. Due to lack of space, some results are proved in Appendix. Finally, all our expressiveness equivalences are effective.

# 2. Two-way visibly pushdown automata

## 2.1 Definitions

We introduce in this section two-way visibly pushdown automata, following the definition of (Madhusudan and Viswanathan 2009).

We consider a structured alphabet $\Sigma$ defined as the disjoint union of call symbols $\Sigma_c$, and return symbols $\Sigma_r$. The set of words over $\Sigma$ is $\Sigma^*$. As usual, $\epsilon$ denotes the empty word. Amongst words, the set of nested words $N(\Sigma)$ is defined as the least set such that $\epsilon \in N(\Sigma)$ and if $w_1, w_2 \in N(\Sigma)$ then both $w_1w_2$ and $cw_1r$ (for all $c \in \Sigma_c$ and $r \in \Sigma_r$) belong to $N(\Sigma)$. In the following, we assume that input words of our models are always nested words. This is not restrictive as all our models can recognize and filter nested words.

For a word $w \in \Sigma^*$, its length is denoted by $|w|$ and we denote by $w(i)$ its $i$th symbol. Its set of positions is $pos(w) = \{1, \ldots, |w|\}$, and for $i, j \in pos(w)$ such that $i < j$, we say that $(i, j)$ is a **matching pair** of $w$ if $w(i) \in \Sigma_c$, $w(j) \in \Sigma_r$ and $w$ can be decomposed into $w = w_1w(i)w_2w(j)w_3$, where $w_1, w_2, w_3 \in \Sigma_c^*$, $w_2 \in N(\Sigma)$ and $|w_1| = j - i - 1$. Note that if $w \in N(\Sigma)$, then necessarily, $w_1w_3 \in N(\Sigma)$.

When dealing with two-way machines, we assume the structured alphabet $\Sigma$ to be extended into $\Sigma$ by adding two special symbols $>, \prec$ in $\Sigma_c$, and $\Sigma_r$, respectively, and we consider words with left and right markers from $\Sigma$. In the following, we assume that, $\Sigma = \Sigma_r \cup \Sigma_c$.

**Definition 1.** A two way visibly pushdown automaton (2VPA for short) $A$ over $\Sigma$ is given by $(Q, q_0, \Gamma, \delta)$ where $Q$ is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a set of final states and $\Gamma$ is a finite stack alphabet. Given the set $D = \{\leftarrow, \rightarrow\}$ of directions, the **transition relation** $\delta$ is defined by $\delta^{pop} \cup \delta^{push}$ where

- $\delta^{pop} = ((Q \times \{\leftarrow\} \times \Sigma_c) \cup (Q \times \{\rightarrow\} \times \Sigma_r)) \times ((Q \times D) \times \Gamma)$
- $\delta^{push} = ((Q \times \{\leftarrow\} \times \Sigma_r \cup (Q \times \{\rightarrow\} \times \Sigma_c)) \times (Q \times D))$

Additionally, we require that for any states $q, q'$ and any stack symbol $\gamma$, if $(q, \leftarrow, \gamma, q', d) \in \delta^{pop}$ then $d = \rightarrow$ and if $(q, \rightarrow, \gamma, q', d) \in \delta^{pop}$ then $d = \leftarrow$.

Informally, a 2VPA has a reading head pointing between symbols (and possibly on the left of $>$ and on the right of $\prec$). A configuration of the machine is given by a state, a direction $d$ and a
any configuration is actually a run on the empty word $\epsilon$.

By the special treatment of $\triangledown$ symbol leads to push onto the stack while a return symbol pops a head if $d$ move from $(q, i, d, \sigma)$ form a family of unary operations and $\epsilon$ is a constant. The semantics of $\epsilon$ is the empty word, $\triangledown$ is concatenation and for any $w \in \mathcal{N}(\Sigma)$, $f_{e,\epsilon}(w) = \epsilon \ast w$. Obviously, the operators finitely generate $\mathcal{N}(\Sigma)$ which can be seen as the free generated algebra over this signature quotiented by the associativity of $\cdot$ and the neutrality of $\epsilon$ w.r.t the concatenation $\ast$.

The traversal congruence $\sim$ Inspired by works on two-way automata on words (Péchuchet 1985; Shepherdson 1959), we study traversals of a 2VPA $A$. A traversal of some nested word $w$ abstracts a run of $A$ keeping track only of the fact that it starts reading the word from the left or from the right (depending on the initial direction) in some state $p$ and leaves it in some state $q$. Now, formally, for any states $p, q$ and any two directions $d_1, d_2 \in \mathbb{D}$, $(p, d_1, q), (q, d_2, r)$ belongs to the traversal of $w$ if there exists a run of $A$ on $w$ starting in the configuration $(p, \text{pos}(d_1), d_1, \bot)$ and ending in $(q, \text{pos}(d_2), d_2, \bot)$, where

$$\begin{align*}
\text{pos}(d_1) &= 0 \text{ if } d_1 = \rightarrow \text{ and } \text{pos}(d_1) = |w| \text{ otherwise} \\
\text{pos}(d_2) &= |w| \text{ if } d_2 = \rightarrow \text{ and } \text{pos}(d_2) = 0 \text{ otherwise}
\end{align*}$$

Note that the reading starts either at the beginning or at the end of $w$ depending on the initial current direction and that the final direction indeed leads to leave the word. One may associate with a nested word the set of its traversals and define a relation $\sim$ on nested words such that $u \sim v$ if $u$ and $v$ have the same traversals.

Obviously, $\sim$ is an equivalence relation over $\mathcal{N}(\Sigma)$ and we denote by $[w]_\sim$ the set of traversals of a nested word $w$. We prove that $\sim$ is actually a congruence, that is if $w_1 \sim w_2$ and $w'_1 \sim w'_2$ then $f_{e,\epsilon}(w_1) \sim f_{e,\epsilon}(w_2)$ and $w_1 w'_1 \sim w_2 w'_2$ for any nested words $w_1, w'_1, w_2, w'_2 \in \mathcal{N}(\Sigma)$.

**Proposition 1.** The relation $\sim$ is a congruence of finite index.

The transition algebra $\mathcal{T}_A$ Based on Proposition 1, the congruence relation $\sim$ induces a finite algebra $\mathcal{T}_A = (\text{Trav}_A, \sim, \{f_{c,r}^{\epsilon,\triangledown} \mid c \in \Sigma, r \in \Sigma, \epsilon^\star\})$ where the support is Trav$_A$ the set of all traversals induced by $A$. $f_{c,r}^{\epsilon,\triangledown}$ is a binary operation which is associative, each $f_{c,r}^{\epsilon,\triangledown}$ is a unary operation and $\epsilon^\star$ is a constant from Trav$_A$ and a neutral element for $f_{c,r}^{\epsilon,\triangledown}$. More specifically, $\epsilon^\star = \epsilon \ast \epsilon$ and $f_{c,r}^{\epsilon,\triangledown}(w, w) = \epsilon \ast w$. These operations are well-defined since $\sim$ is a congruence.

Hence, there exists a unique and canonical morphism $\mu_{\mathcal{T}_A}$ from the algebra of nested words, onto $\mathcal{T}_A$, that satisfies $\mu_{\mathcal{T}_A}(w) = [w]_\sim$. We also denote $[w]_\sim$ as $w^A$ since it can be considered as the interpretation of $w$ (which is an element $w$) in $\mathcal{T}_A$.

The correction of this morphism $\mu_{\mathcal{T}_A}$ directly implies:

**Proposition 2.** Let $A = (\mathbb{Q}, q_0, \mathbb{D}, \Gamma, \delta)$ be a 2VPA. $\mathcal{L}(A) = \mu_{\mathcal{T}_A}^\mathcal{L}(\{m \in \text{Trav}_A \mid m \cap (\{(q, \rightarrow) \times F \times (\rightarrow)\} \neq \emptyset\}).$

Note that this statement corresponds to the classical notion of recognizability by some finite algebra.

2.3 From two-way visibly pushdown automata to visibly pushdown automata

In this subsection we give a reduction from 2VPA to VPA. While this result can be inferred from (Madhusudan and Viswanathan 2009), our Shepherdson-inspired approach gives an upper bound on the complexity of the procedure. We first recall the notion of recognizability by finite algebra and show that this notion is
equivalent to recognizability by DVPA. Then we prove the main result of this section, which is stated as the transition algebra $T_A$.

Let $A = (D_A, ε^A, L^A_{(c,r)}(c,r) ∈ Σ^∗ × V)$ be a finite algebra such that $ε^A$ is associative having $ε^A$ as neutral element. There exists a unique morphism $µ_A$ from the algebra of nested words $W$ onto $A$.

**Definition 3.** A language $L ⊆ N(Σ)$ is recognized by $A$ if there exists a run $L ∈ D_A$ such that $L = µ_A^{-1}(L_A)$.

As an example, as shown in Proposition 2, a language $L$ defined by a 2VPA is recognized by the transition algebra $T_A$. We show that recognizability by finite algebra implies DVPA recognizability.

**Lemma 1.** If $L$ is recognized by a finite algebra $A$, then it is recognizable by a 2VPA $B_A$. Moreover, the size of $B_A$ is polynomial in the size of $D_A$, the support of $A$.

**Proof.** For $A$ and the set $L_A ⊆ D_A$, we define the DVPA $B_A = (D_A, ε^A, L_A, Σ ∗ × D_A, δ_B_A)$ where $δ_B_A = δ_B_A^n ∪ δ_B_A^o$ and $δ_B_A^n(m^A, c) = (ε^A, (c, m^A))$, $δ_B_A^o(m^A, r, (c, m^A)) = m^A o f^A(c, m^A)$. Obviously, $B_A$ is deterministic. Its correctness can be proved by induction on nested words showing for all $w ∈ N(Σ)$, there exists a run $B_A$ on $w$ or $(m^A, 0_l, 0_l) ⊆ (m^A, w, L)$ iff $m^A = m^A, µ_A(w)$. And so, for an accepting run on $(m^A, 0_l, 0_l) ⊆ (m^A, w, L)$ with $m^A ∈ L_A$, $m^A = m^A, µ_A(w)$. Hence, $L(B_A) = µ_A^{-1}(L_A)$. Finally, note that the number of states of $B_A$ is precisely the cardinality of the support of $A$.

We can now come to the main result of this section.

**Theorem 1.** For any 2VPA $A$, one can compute (in exponential time) a 2VBA $B$ such that $L(A) = L(B)$ and the size of $B$ is exponential in the size of $A$.

**Proof.** One can build from the 2VPA $A$ the elements of $\{[w], w ∈ N(Σ)\}$ and thus, the transition algebra $T_A$, in exponential time. Then, by Lemma 1, a 2VPA $B_t_A$ is built from $T_A$. The correctness follows from Proposition 2 for $L(A) = \sum_1^A ∈ T_A$. A, $TrAV$ $|m^A ∩ \{(q_i, →)\} × \{F\} × Σ^∗\}$.

**Corollary 1.** For any 2VPA $A$, deciding the emptiness of $A$ (i.e., $L(A) = \emptyset$) is EXP-TIME-C. The same result holds for D2VPA.

**Proof.** We prove the upper-bound for 2VPA and the lower bound for D2VPA. For the upper-bound, it suffices to build from $A$ in exponential time a equivalent DVPA $B$ possibly exponentially larger than $A$ (Theorem 1). Then, emptiness of $B$ can be tested in polynomial time (Alur and Madhusudan 2009).

The proof of the lower bound proceeds by a reduction of the emptiness problem of intersection of $k$ deterministic top-down tree automata, that is known to be EXP-TIME-C.

**2.4 2VPA with look-around**

As we will later on need the notion of look-around for transducers, we introduce it first for automata to ease the presentation. Hence, we extend the model of 2VPA with look-around. The feature will add a guard for each transition of the machine. This guard will require to be satisfied for the transition to be applied.

**Definition 4.** A 2VPA with look-around (2VPA$^{LA}$ for short) is given by a triple $(A, λ, B)$ such that $A$ is a 2VPA and $B$ a unambiguous VPA and $λ$ is a mapping from the transitions of $A$ to the states of $B$.

The notion of runs is adapted to take into account look-around as follows: in any run on some nested word $w$, for any two successive configurations $(q_j, i_j, d_j, σ_j)$ and $(q_{j+1}, i_{j+1}, d_{j+1}, σ_{j+1})$ obtained by a transition $t$, we require that there exists a unique accepting run on $w$ in $B$ and that this run contains a configuration of the form $(λ(t), read(w, d_j, i_j, σ_j)) = (τ_j)$.

The definition of accepting runs remains the same and the language defined by such machines is defined accordingly.

The notion of one-wayness extends trivially to 2VPA with look-around. For determinism, we ask the look-around to be disjoint on transitions with the same left hand-side: for any two different transitions of $A$, $t_1 = (q, d, a, q_1, d_1, γ_1)$, $t_2 = (q, d, a, q_2, d_2, γ_2)$ in $δ$, (resp. $t_1 = (q, d, a, q_1, d_1, γ_1)$, $t_2 = (q, d, a, q_2, d_2, γ_2)$ in $δ$), it holds that $λ(t_1) ≠ λ(t_2)$.

Non-surprisingly, 2VPA are closed under look-around:

**Theorem 2.** Given a 2VPA$^{LA}$ $(A, λ, B)$, there exists a VPA $A'$ such that $L((A, λ, B)) = L(A')$.

**3. Two-way visibly pushdown transducers**

**3.1 Definitions**

Let $Σ, Δ$ be two finite alphabets such that $Σ$ is structured. Two-way visibly pushdown transducers (2VPT) from $Σ$ to $Δ$ extend 2VPA over $Σ$ with a one-way-left-to-right output tape. They are defined as a pair $T = (A, O)$ where $A$ is a 2VPA over $Σ$ and $O$ is a morphism from the set of rules of $A$ to words in $Δ^∗$.

A run of a 2VPT $T = (A, O)$ on an input word $w ∈ N(Σ)$ is a run $ρ$ of $A$ on $w$. We say the run is accepting if it is in $A$. A run $ρ$ may be simultaneously a run on a word $w$ and on a word $w' ≠ w$, however, when the underlying input word $w$ is given, there is a unique sequence of transitions $t_1, t_2, ..., t_n$ related with $ρ$ and $w$. In this case, the output produced by the run $ρ$ on $w$ is defined as the word $v = O(t_1)O(t_2)...O(t_n) ∈ Δ^∗$. This word is denoted by $out^w(ρ)$. If $ρ$ contains a single configuration, then we let $out^w(ρ) = ε$. The transduction defined by $T$ is the relation $\{(w, out^w(ρ)) ∈ N(Σ) × Δ^∗ | ρ$ is an accepting run of $T$ on $w$\}.

We say that $T$ is functional if $|T|$ is a function, and that $T$ is deterministic (resp. unambiguous) if $A$ is deterministic (resp. unambiguous). The class of deterministic two-way visibly pushdown transducers is denoted D2VPT. Observe that if $T$ is deterministic or unambiguous, then it is trivially functional. Last, when $T$ is functional, we may interpret the relation $|T|$ as a partial function on $N(Σ)$: given a word $w ∈ N(Σ)$, denote by $|T|(w)$ the unique word $v ∈ Δ^∗$ such that $(w, v) ∈ |T|$, whenever it exists. To ease readability, we may simply write $T$ to denote $|T|$ when it is clear from the context, for example when considering composition of functions.

We consider classes of one-way visibly pushdown transducers, obtained by considering the corresponding classes of one-way visibly pushdown automata. The notions of functional, deterministic and unambiguous transducers are naturally defined for these transducers, and we denote by (D)VPT the class of (deterministic) one-way visibly pushdown transducers. Last, we say that a VPT $T = (A, O)$ from $Σ$ to $Δ$ is letter-to-letter if $Δ$ is a structured alphabet and if $O$ maps every call transition of $A$ to an element of $Δ$, and every return transition of $A$ to an element of $Δ$.

2VPT (resp. D2VPT) can be extended with look-around, as we did for 2VPA. Formally, a two-way visibly pushdown transducer with look-around (2VPT$^{LA}$ for short) is a pair $T = (A, O)$ where $A' = (A, λ, B)$ is a 2VPA$^{LA}$ and $O$ is a morphism from the set of rules of $A$ to words in $Δ^∗$. We say that such a machine is deterministic if the 2VPA$^{LA}$ $A$ is deterministic, the resulting class being denoted by D2VPT$^{LA}$.

**Example 1.** We now formally express the transduction given in the introduction (see Figure 1). Let $Q = \{q_1, ..., q_n\} ∪ \{q_j | 1 ≤ i, j ≤ n \}$ or $i = v\}$. Then $\{q_j | 1 ≤ i, j ≤ n \}$ or $i = v\}$ be the set of states with initial state $q_1$ and final state $q_f$, a set of stack symbols $T = \{\⊥\} ∪ \{i | i = v\}$.
1. . . . n, and for all i, j, k ∈ {>, 1, . . . , n}, we have the rules:

\[ q_i \rightarrow i_{k}^{i_{k+j}} \quad q_i \rightarrow q_j \quad q_i \rightarrow r_{j}^{r_{j-i}} \quad r_i \rightarrow \]

The markers are treated as letters, except that they push ⊥ instead of ⊤ and upon popping ⊥ in state \( q_0 \), the transducer goes to \( q_1 \) and accepts. The transitions labeled by \( (j, r) \) are macros corresponding to moving along matching relation, which can easily be implemented.

Evaluation: Observe here that if a transformation is given as a D2VPT \( T \), then one can evaluate it using a memory linear in the depth of the input word \( w \) (we assume \( w \) can be accessed as we want on some media). Indeed, one simply needs to store the current configuration of \( T \), given as a state and a stack content.

3.2 Closure under composition

We prove in this subsection that 2VPT are closed by composition with a letter-to-letter unambiguous VPT, extending a similar result for transducers on words (Hopcroft and Ullman 1967). This will reveal useful to show that D2VPT are closed under look-around. First, we extend to nested words a result that was known for finite transducers:

Lemma 2. Any unambiguous VPT \( T \) can be written as the composition of two VPT \( T_1 \circ T_2 \), where \( T_1 \) is deterministic and \( T_2 \) is letter-to-letter and co-deterministic. Furthermore, if \( T \) is letter-to-letter, so is \( T_1 \).

Theorem 3. Given a letter-to-letter DVPT \( A \) and a 2VPT \( B \), we can construct a 2VPT \( C \) that realizes the composition \( C = B \circ A \). If furthermore \( B \) is deterministic, then so is \( C \).

Proof. We first notice that since we are considering visibly pushdown machines and the first machine is letter-to-letter, the stacks of both machines are always synchronized, meaning that they have the same height on each position. Then, let us remark that when the 2VPT moves to the right, we can do the simulation in a straight forward fashion by simulating it on the production of the one-way. It becomes more involved when it moves to the left. We then need to rewind the run of the one-way, and nondeterminism can arise. To bypass this, let us recall that a similar construction from (Hopcroft and Ullman 1967) exists for classical transducers, and that the rewinding is done through a back and forth reading of the input, backtracking the run up to a position where the nondeterminism is cleared, and then moving back to the current position. The method is to compute the set of possible candidates for the previous state, and keep moving to the left until we reach a position \( i \) where there is only one path left leading to the starting position \( j \). Afterward, we simply follow this path along another one from position \( i + 1 \). As we know that they will merge at position \( j \), we can stop at position \( j - 1 \) with the correct state. If we reach the beginning of the word with multiple candidates, we do the same procedure, the correct path being the one starting from the initial state.

This cannot be done as such on pushdown transducers since rewinding the run might lead to popping the stack, and losing information. However, if at each push position, we push not only the stack symbols but also the current state, we are able, when rewinding the run, to clear the nondeterminism as soon as we pop this information by using it as a local initial state, limiting the back and forth reading to the current subhedge. The overall construction can be seen as a classical Hopcroft-Ullman construction on hedges, abstracted as words over the left-to-right traversals of their subhedges, which are called summaries in (Alur and Madhusudan 2009) (see Figure 2). These summaries can be computed on-demand by a one-way automaton.

Finally, note that to apply this construction, we need to push this local initial state each time we enter a subhedge, whether we enter from the right or from the left. This can be maintained since when entering from the left, it simply corresponds to the current state and when entering from the right, this state is computed by the Hopcroft-Ullman construction. Note also that the Hopcroft-Ullman routine is deterministic, and consequently the construction preserves determinism.

Theorem 4. Let \( A \) be a D2VPT and \( relab \) be an unambiguous letter-to-letter VPT. Then the composition \( A \circ relab \) can be defined by a D2VPT.

Proof. The proof is straightforward using previous results. First, Lemma 2 states that \( relab \) can be decomposed in \( T_1 \circ T_2 \), where \( T_1 \) is a deterministic VPT and \( T_2 \) is a co-deterministic one, and both are letter-to-letter, i.e \( A \circ relab = A \circ T_1 \circ T_2 \). Now Theorem 3 states that we can construct a D2VPT \( A' \) that realizes the composition \( A \circ T_1 \). Finally, as a co-deterministic VPT can be seen as a deterministic one going right-to-left, a symmetric construction of Theorem 3 on \( A' \circ T_2 \) gives a D2VPT that realizes \( A \circ relab \).

A look-around can be viewed as an MSO formula with one free variable, and it is satisfied iff the formula is satisfied at this position. In (Madhusudan and Viswanathan 2009), the authors consider MSO queries on nested words. An MSO query is an MSO formula with one free variable that annotates the positions of the input word that satisfies it. They proved, using a Hopcroft-Ullman argument, that MSO queries were also implemented by D2VPA. Theorem 4 proves that looks-around can be done on the fly while following the run of an other D2VPA. Since a look-around can be encoded as an unambiguous letter-to-letter VPT, we get the following corollary, that subsumes the result by (Madhusudan and Viswanathan 2009).

Corollary 2. D2VPT = D2VPT-LA.

3.3 Decision problems

We consider the following type-checking problem: given a VPA \( A_1 \) on \( \Sigma \), a finite-state automaton \( A_2 \) on \( \Delta \), and a D2VPT \( T \) from \( \mathcal{N}(\Sigma) \) to \( \mathcal{D}' \), decide whether for every word \( w \) in \( \mathcal{L}(A_1) \), \( \exists \mathcal{L}(T)(w) \) belongs to \( \mathcal{L}(A_2) \). This property is denoted by \( T(A_1) \subseteq A_2^T \). The equivalence problem asks whether given two D2VPT as input, they define the same transduction. We prove the following result:

Theorem 5. 1. The inverse image of a regular language of words by a D2VPT is recognizable by a VPA.
2. The type-checking problem for D2VPT is ExpTime-complete.
3. The equivalence problem for D2VPT is decidable.

1 If \( A_2 \) is a VPA, the problem is known to be undecidable even for \( T \) a DVPT (Raskin and Servais 2008).
4. Expressiveness of Two-Way Visibly Pushdown Transducers

In this section, we study the expressiveness of D2VP T by comparing it with Courcelle’s MSO-transductions casted to nested words, the one-way model of (Alur and D’Antoni 2012), and a top-down model for hedges, inspired by top-down tree-to-string transducers.

4.1 MSO-definable Transductions

We first define MSO for nested words and words, as done in (Alur and Madhusudan 2009), and then MSO-transductions from nested words to words, based on Courcelle’s MSO-definable graph transductions (Courcelle 1994).

MSO on nested words and words

Let \( \Sigma \) be a structured alphabet. A nested word \( w \in N(\Sigma) \) is viewed as a structure with \( pos(w) \) as domain, over the successor predicate \( S(x,y) \) interpreted as pairs \( (i,i+1) \) for \( i \in pos(w) \) \( \{[w]\} \), the label predicates \( \sigma(x) \) for \( \sigma \in \Sigma \), interpreted by the positions labeled by \( \sigma \), and the matching predicate \( M(x,y) \) interpreted as the set of matching pairs in \( w \).

Monadic second-order logic (MSO) extends first-order logic with quantification over sets. First-order variables \( x,y,\ldots \) are interpreted by positions of words, while second-order variables \( X,Y,\ldots \) are interpreted by sets of positions. MSO formulas for nested words over \( \Sigma \) are defined by the following grammar:

\[
\varphi ::= \sigma(x) \mid x \in X \mid S(x,y) \mid M(x,y) \mid \neg \varphi \mid \varphi \lor \varphi \mid \exists x. \varphi \mid \exists X. \varphi
\]

where \( \sigma \in \Sigma \). The semantics of an MSO formula is defined in a classical way, and for \( \varphi \) an MSO formula, \( w \in N(\Sigma) \), \( \nu \) a valuation of the free variables of \( \varphi \) into positions and sets of positions of \( w \), we write \( w\models \varphi \) to mean that \( w \) is a model of \( \varphi \) under the valuation \( \nu \). When \( \varphi \) is a sentence, we just write \( w \models \varphi \). We denote by MSO_{nw}[\Sigma] the set of MSO formulas for nested words over \( \Sigma \) (and just MSO_{nw} when \( \Sigma \) is clear from the context). Since we are interested in transductions from nested words to words, we also define MSO for words. Similarly as nested words, words are seen as structures but in that case we do not have the matching pair predicate \( M(x,y) \). MSO formulas on words are defined accordingly to this smaller signature.

Example 2. We interpret MSO_{nw}[\Sigma] on nested words rather that on words in \( \Sigma^* \). It is not a restriction since checking whether a given relation \( M(x,y) \) is a valid matching relation is definable by an MSO formula \( \phi_{uw} \). This formula expresses that \( M \) is a bijection between call and return symbols, and that it is well-nested (there is no crossing), as follows:

\[\neg \exists x_c, x_r, y_c, y_r. M(x_c, x_r) \land M(y_c, y_r) \land x_c < y_c < x_r < y_r \land bij(M) \land \forall x, y. M(x, y) \Rightarrow x < y \]

where \( < \) is the transitive closure of \( S \) (well-known to be MSO-definable) and bij(M) expresses that M maps bijectively call and return symbols (it is trivially MSO-definable).

MSO transducers from nested words to words

MSO-transducers define (partial) functions from nested words to word structures. The output word structure is defined by taking a fixed number of \( k \) copies of the input structure domain. Nodes of these copies can be filtered out by MSO_{nw} formulas with one free first-order variable. In particular, the nodes of the \( e \)-th copy are the input positions that satisfy some given MSO_{nw} formula \( \phi_{pos}(x) \). The label predicates \( \sigma(x) \) and the successor predicate \( S(x,y) \) of the output structure are defined by MSO_{nw} formulas with respectively one and two free first-order variables, interpreted over the input structure. Formally, a MSO-transducer from nested words to words is a tuple \( T = (k, \phi_{dom}, (\phi_{pos}^e(x))_{1 \leq e \leq k}, (\phi_{ls}^e(y, x))_{1 \leq e \leq k}, (\phi_{rd}^d(x,y))_{1 \leq e \leq d \leq k}) \) where \( k \in \mathbb{N} \) and the formulas \( \phi_{dom}, \phi_{pos}^e, \phi_{ls}^e \) and \( \phi_{rd}^d \) are MSO_{nw} formulas. We denote by MSO_{nw2w} the class of MSO-transducers from nested words to words.

An MSO-transducer \( T \) defines a function from nested word structures over \( \Sigma \) to word structures over \( \Sigma \), denoted by \( [T] \). The domain of \( [T] \) consists of all nested word structures \( u \) such that \( u \models \phi_{dom} \). Given a nested word structure \( u \in dom([T]) \), the output structure \( v \) such that \( (u,v) \in [T] \) is defined by the domain \( D^v \subseteq pos(u) \times \{1,\ldots,k\} \) such that \( D^v = \{ (i,c) \mid i \in pos(u), c \in \{1,\ldots,k\}, u \models \phi_{pos}(i) \} \), a node \( (i,d) \in D^v \) of the output structure is labeled \( a \in \Sigma \) if \( u \models \phi_{ls}(i) \) and a node \( (j,d) \in D^v \) is the successor of a node \( (i,c) \in D^v \) if \( u \models \phi_{rd}^d(i,j) \). Note that the output structure is not necessarily a word, because for instance, nothing guarantees that an output node is labeled by a unique symbol, or that the successor relation forms a linear order on the positions. However, it is not difficult to see that it is definable whether an MSO_{nw2w} transducer produces only words (see for instance (Filotti 2015)).

We say that a function \( f \) from nested words to words is MSO-definable if there exists an \( T \in MSO_{nw2w} \) such that \( [T] = f \). By definition of MSO_{nw2w} transducers, for any MSO-definable function \( f \) there exists \( k \in \mathbb{N} \) such that for all \( u \in Dom(f) \), \( f(u) \subseteq [k;u] \) (by taking \( k \) as the number of copies of the MSO_{nw2w} transducer defining \( f \)). We say in that case that \( f \) is of linear-size increase.

Example 3. This example transforms a nested word into the sequence of calls of maximal depth (the leaves). E.g., \( c_1 c_2 c_3 c_4 c_5 T_1 \) is mapped to \( c_2 c_4 \). This transformation is MSO-definable. The domain is defined by the formula \( \phi_{uw} \) (see Example 2). One needs only one copy of the input word, whose positions are filtered out by the formula \( \phi_{pos}^e = g_3 \cdot M(x,y) \land S(x,y) \) holds true iff \( x \) is a call position and its successor position \( y \) is its matching return position. The labels are preserved: \( \phi_{ls}^e(x) = a(x) \) for all \( a \in \Sigma \). Finally, the successor relation is defined by \( \phi_{rd}^d(x,y) = \phi_{pos}^e(x) \land \phi_{ls}^e(y) \land x < y \land \neg \exists z. \phi_{pos}^e(z) \land x < z < y \).

4.2 Logical equivalences

An MSO_{nw2w} \( T \) is said to be order-preserving if for any word \( u \) of the domain of \( T \), any positions \( i,j \) of \( u \) and any copies \( c,d \) of \( T \),...
if \( u \models \phi_{q,i,j}^c \) then \( i \leq j \). This means that the output arrows can not point to the right. It is emphasized by the next theorem, which echoes a similar result on words proved in (Bojanczyk 2014; Filiot 2015).

**Theorem 6.** An order-preserving transduction is definable in \( \text{MSO}[\text{nw2w}] \) if, and only if, it is definable by a functional\(^4\) VPT.

In the following, we show that D2VPT are strictly more expressive than \( \text{MSO}[\text{nw2w}] \), and define a restriction that capture exactly \( \text{MSO}[\text{nw2w}] \). The fact that D2VPT are more expressive than \( \text{MSO}[\text{nw2w}] \) can be easily shown, based on a similar result for ranked trees established in (Courc najle and Engelfriet 2012). Since D2VPT can, using their stack, express transductions of exponential-size increase, while MSO-transductions are of linear-size increase, they are strictly more expressive than \( \text{MSO}[\text{nw2w}] \).

To capture exactly \( \text{MSO}[\text{nw2w}] \), one defines the single-use restriction for D2VPT (and D2VPT\(^{LA}\)). Intuitively, this restriction requires that when a D2VPT passes twice at the same position with the same state, then necessarily the transitions fired from these states produces \( \epsilon \).

**Definition 5** (Single-use restriction). A D2VPT (resp. D2VPT\(^{LA}\)) \( T = (A, O) \) with \( A = (\Sigma, q_0, q_f, \eta) \) a 2VPA (resp. 2VPA\(^{LA}\)) is single-use with respect to a set \( P \subseteq Q \) if any transition \( t \) from a state \( q \not\in P \) satisfies \( O(t) = \epsilon \), and if for all runs \( r = (q_0, i_0, a_0, \sigma_0) \ldots (q_i, i, a, \sigma) \) of \( T \) on a word \( w \) and all states \( p \in P \), \( r \) does not visit twice the same position in state \( p \), i.e., if \((q_i, i_0, a_0, \sigma_0) = (q_j, i, a, \sigma) \) for \( a \neq \beta \), then \( q_i = q_j \not\in P \).

A D2VPT (resp. D2VPT\(^{LA}\)) is single-use if it is single-use w.r.t. some set \( P \subseteq Q \), and strongly single-use if it is single-use w.r.t. \( Q \).

We denote by D2VPT\(_{su}\) (resp. D2VPT\(_{su}^{LA}\)) the class of single-use D2VPT (resp. D2VPT\(^{LA}\)). By reduction to the 2VPA emptiness, we get:

**Proposition 3.** Deciding the single-use property on a 2VPT is Exptime-c.

A single-use restriction was already defined in (Courc najle and Engelfriet 2012) for deterministic tree-walking transducers with look-around to capture MSO-transductions from trees to trees (and words). It requires that in any accepting run, every node is visited at most once by a state. It is therefore more restrictive than our single-use restriction and, as a matter of fact, corresponds to what we call the strongly single-use restriction. However, the following result shows that the strongly single-use restriction is not powerful enough, in our context, to capture all MSO-definable transductions, even with regular look-arounds.

**Lemma 3.** There is an MSO-definable nested word to word transduction \( f \) which is not definable by strongly single-use D2VPT\(^{LA}\).

We now proceed to the first logical equivalence, between our model and MSO-transductions, which is mainly a consequence of results from (Courc najle and Engelfriet 2012).

**Theorem 7.** Let \( f \) be a transduction from nested words to words. Then \( f \) is MSO-definable if, and only if, it is definable by a (look-around) D2VPT\(_{su}\), i.e.,

\[
\text{MSO}[\text{nw2w}] = \text{D2VPT}_{su}^{LA} = \text{D2VPT}_{su}.
\]

**Sketch of proof.** We show that both other models are equivalent to D2VPT\(_{su}^{LA}\). We have already seen that look-around can be removed from D2VPT\(^{LA}\) (Theorem 2), while preserving their expressive power. Our Hopcroft-Ullman’s construction can add exponentially more visits to the same positions, but these visits are only \( \epsilon \)-producing. In other words, our Hopcroft-Ullman’s construction does not preserve the strongly single-use restriction, but it preserves the single-use restriction. As a consequence of this observation and Corollary 2, we obtain that D2VPT\(_{su} = \text{D2VPT}_{su}^{LA}\).

To show \( \text{MSO}[\text{nw2w}] \subseteq \text{D2VPT}_{su}^{LA}\), we rely on the equivalence of (Courc najle and Engelfriet 2012) between deterministic binary tree to word walking transducers with look-around (DTWT\(^{LA}\)) and MSO-transductions from binary trees to words (MSO[\(b2w]\)). Informally, DTWT\(^{LA}\) can follow the directions of binary trees (1st child, 2nd child and parent) and take their transitions based on regular look-around information. Due to determinism, they are always strongly single-use, in the sense that any position is not visited twice by the same state. Such a machine, running on first-child next-sibling encoding of nested words, is easily encoded into an equivalent D2VPT\(_{su}\). In this encoding, a nested word over \( \Sigma \) is encoded as a binary tree over \((\Sigma_1 \times \Sigma_2) \cup \{ \bot \}\), inductively defined as \( \text{fcns}(cw_1 \cdot rw_2) = (c, r)(\text{fcns}(w_1), \text{fcns}(w_2)) \) and \( \text{fcns}(\epsilon) = \bot \). In this encoding, moving to a 1st child corresponds to moving from \( c \) to \( w_1 \), which can be done by a D2VPT\(_{su}\), and moving to a 2nd child corresponds to moving from \( c \) to \( w_2 \). This can be done also by a D2VPT\(_{su}^{LA}\) but it needs to traverse all the word \( cw_1 \cdot r \), while producing \( \epsilon \) only. Similarly, one can encode moves to parent nodes. The two latter moves implies that the D2VPT\(_{su}^{LA}\) is not strongly single-use anymore, but it remains single-use: the extra moves are all \( \epsilon \)-producing. The result follows as \( \text{MSO}[\text{nw2w}] = \text{MSO}[b2w] \circ \text{fcns} \).

To show \( \text{D2VPT}_{su}^{LA} \subseteq \text{MSO}[\text{nw2w}] \), we rely on another correspondence shown in (Courc najle and Engelfriet 2012), between \( \text{MSO}[b2w] \) and deterministic (visibly) pushdown binary tree to word walking transducers with look-around of linear-size increase (DPTWT\(_{la}\)). These transducers extend DTWT\(_{la}\) with a pushdown store with a visibly condition: when moving to a child, they push one symbol, and moving up, they pop one symbol. The lsi restriction is semantical: they restrict the class to transducers that define lsi transductions. Any D2VPT\(_{su}^{LA}\) defines an lsi transduction, and can be easily encoded into a DPTWT\(_{la}\) running on fencs encodings, which mimics the moves of the D2VPT\(_{su}\). Again, the result follows by the equality \( \text{MSO}[\text{nw2w}] = \text{MSO}[b2w] \circ \text{fcns} \).

**4.3 Comparison with other transducer models**

In this section, we relate D2VPT to two other transducer models, namely streaming tree-to-string transducers and deterministic hedge-to-string transducers with look-ahead. Streaming tree-to-string transducers with a simple copyless restriction of updates will serve as the third edge of our trinity. Deterministic hedge-to-string transducers with look-ahead is a natural model for which equivalence is known to be decidable.

Streaming tree-to-string transducers are deterministic one-way machines (Alur and D’Antoni 2012) equipped with registers storing words. We fix a finite alphabet \( \Delta \) and, given two finite sets \( X \) and \( Y \), denote by \( \mathcal{U}(X, Y) \) the set of mappings from \( X \) to \((\Delta \cup Y)^*\).

**Definition 6.** A streaming tree-to-string transducer \( S \) (STST for short) is a deterministic machine defined over a structured alphabet \( \Sigma \) and given by the tuple \((Q, q_0, \Gamma, X, \delta, \mu, \nu)\) where \( Q \) is a finite set of states, \( q_0 \in Q \) is the initial state, \( \Gamma \) is a finite set of stack symbols and \( X \) is a finite set of registers. Finally, \( \mu, \nu \) is a partial mapping from \( Q \) to \((\Delta \cup X)^* \) and \( \delta = \delta^{\text{push}} \cup \delta^{\text{pop}} \) where \( \delta^{\text{push}} : Q \times \Sigma \rightarrow Q \times \Gamma \times X \) and \( \delta^{\text{pop}} : Q \times \Sigma \rightarrow Q \times X \cup \Delta \times X \), \( X' \) being a disjoint copy of \( X \).
Let $\mathcal{V}_\Delta^X$ be the set of mappings from from $X$ to $\Delta^*$. These mappings are extended to $(X \cup \Delta)^+$ by considering them as identity over $\Delta$. An accepting run of a STST $S$ on a nested word $w$ is a (non-empty) sequence $(q_0, \theta_1, \sigma_1, w_1) \ldots (q_t, \theta_t, \sigma_t, w_t)$ of quadruples from $Q \times \mathcal{V}_\Delta^X \times (I \times \mathcal{V}_\Delta^X)^* \times \Sigma^*$ such that $q_0 = q$, $w_0 = w$, $w_t = \epsilon$, $\theta_t$ is the mapping $\delta$ which associates $\epsilon$ to any $X$ in $X$, $\sigma_i, \sigma_t$ are equal to $\bot$ the empty stack and for all $0 \leq i < t$, one has either

- $w_i = cw_{i+1}$ and there exists $(q_i, c, q_{i+1}, \gamma, \nu) \in \delta^{push}$, $\theta_{i+1} = \theta_i$, and $\sigma_{i+1} = \sigma_i (\gamma, \theta) \circ \nu$,
- $w_i = rw_{i+1}$ and there exists $(q_i, \gamma, q_{i+1}, \nu) \in \delta^{pop}$, $\sigma_{i+1} = \sigma_i (\gamma, \theta)$ and $\theta_{i+1} = \theta'_i \circ \nu$, where $\theta'_i (X) = \theta(X)$ for all $X \in X$.

The semantics $[S]$ of the STST $S$ is a partial mapping from $N(\Sigma) \to \Delta^*$ such that $[S][w] = \nu$ if there exists an accepting run on $w$ in $S$ ending in some configuration $(q_c, \theta, \bot, \nu)$ and $v = \theta_{\epsilon}(q_f(q_1))$.

Using a restriction on the updates $U$ used in STST (so-called copyless updates), (Alur and D’Antoni 2012) proved that copyless STST and MSO[nw2w] are expressively equivalent. As a consequence, we obtain the logic-two-way/one-way trinity announced in the introduction:

**Theorem 8.** MSO[nw2w] = D2VPT_{uw} = copyless STST

A well-known class of transducers running on ranked trees is the class of deterministic top-down tree transducers with look-ahead. This class can be defined to output strings. We consider now the extension of this class to unranked trees, or more precisely sequences of unranked trees, that is, hedges.

**Definition 7.** An hedge automaton (HA for short) over the structured alphabet $\Sigma^\Delta$ is a tuple $(Q, F, \delta)$ where $Q$ is a finite set of states, $F \subseteq Q$ is a set of final states and $\delta$ is a transition relation such that $\delta \subseteq Q \times \Sigma^\cdot \Sigma^* \times \Sigma \times Q$.

An hedge automaton is said to be bottom-up deterministic if whenever $(q, c, r, q_1, q_2)$ and $(q', c, r, q_1, q_2)$ belong to $\delta$, it holds that $q = q'$. The semantics of an HA $B$ is given by means of sets $L^B_\Delta \subseteq \mathcal{V}^\Delta$ defined for each $q \in Q$ inductively as follows: (i) $\epsilon \in L^B_\Delta$ for all $q$ and (ii) $cw w' \in L^B_\Delta$ if $(q, c, r, q_1, q_2) \in \delta$ and $w \in L^B_\Delta$, $w' \in L^B_\Delta$. The language defined by an HA $B$ is then $\bigcup_{q \in F} L^B_\Delta$. Note that when $B$ is bottom-up deterministic whenever $q_1 \neq q_2$, it holds that $L^B_{q_1} \cap L^B_{q_2} = \emptyset$.

**Definition 8.** A deterministic hedge-to-string transducer with look-ahead (dHST^{LA}) $H$ over the structured alphabet $\Sigma$ and the output alphabet $\Delta$ is given by a tuple $(Q, \mathcal{I}, F, \delta, B)$ where $Q$ is a finite set of states, $q_1 \in Q$ is an initial state, $F \subseteq Q$ is a set of final states, $B$ is a deterministic bottom-up hedge automaton with states $Q'$, and $\delta$ is a transition relation given by a partial mapping $\delta : Q \times \Sigma_c \times \Sigma_r \times Q' \times Q' \to \Delta_\Delta$.

The semantics of a dHST^{LA} is first given by a partial mapping $[H]$ from $N(\Sigma) \times Q$ onto $\Delta^*$ defined inductively as: (i) $[H][\epsilon][q] = \epsilon$ if $q \in F$, and (ii) for $w = cw_{i+1}rw_{i+1}$ with $w_1, w_2 \in N(\Sigma)$, $[H][w][q] = w_2 \rho(q_{x_1}) \rho(q_{x_2}) \rho([H][w_1]) \rho(q_{x_1}) \rho(q_{x_2})$, where $\omega(q_{x_1}) \rho([H][w_1]) \rho(q_{x_2})$ denotes the word $\omega$ in which each occurrence of $q_{x_1}$ has been replaced by $[H][w_1]$ if

\[\delta(q, c, r, q', q'') = \omega, w_1 \in L^B_{q_1}, w_2 \in L^B_{q_2}\]

and undefined otherwise.

Then, the transduction $[H]$ defined by $H$ is given by $\{(w, s) \mid w \in N(\Sigma), s = [H](w, q_1)\}$.

**Theorem 9.** D2VPT \subseteq STST and D2VPT \subseteq dHST^{LA}

**Sketch of proof.** The two results rely on a same intermediate model that extends the transition algebra described in Section 2. This algebra allows to describe the possible traversals of a D2VPA. One can extend it to D2VPT by storing in matrices the words produced by traversals. This yields an infinite algebra, realized by a finite set of operations. We use this to describe effective translations into STST and dHST^{LA}.

As an illustration, in order to build of an equivalent STST, the set of variables considered is the set $\Xi = \{q^{(n,d)}(p', d') \mid (p, d, p', d') \in \mathcal{X} \times \Delta\}$, i.e. one variable for each traversal. This generalizes the construction described in (Alur and Cerný 2010; Alur et al. 2012) in order to translate a deterministic two-way transducer (on words) into a streaming string transducer.

The fact that the inclusions are strict relies on a simple argument based on size increase: on nested words of bounded depth, D2VPT are linear-size increase, while STST and dHST^{LA} are not.

\[\square\]

5. Discussion

**Unranked tree to word transductions** Since unranked trees $t$ can be linearised into nested words $lin(t)$, our result also gives a model for unranked tree to word transductions. If one denotes by MSO[u2w] the transductions from unranked trees to words definable by an MSO transducer (over the signature of unranked trees that has the child and next-sibling predicates), it is easy to show that MSO[u2w] = D2VPT_{uw} \circ lin.

One could argue that D2VPT for realising transductions of unranked trees is not an adequate model, because it performs unnecessary $\epsilon$-producing moves to navigate, for instance, from a node $n$ to its next-sibling. Indeed, the D2VPT needs to walk through the whole subtree rooted at $n$.

First, while it is true from an operational point of view, we think that the simplicity of D2VPT makes it a good candidate as a specification model of unranked tree transductions, and to this aim, it is easy to define, as we did for next-sibling moves (rules $q \mapsto p$, macros that realise moves given by the predicates of unranked trees (and their inverse). Second, for instance in the context of stream processing of XML documents, it cannot be always assumed that the input document is given by its DOM (with the unranked tree predicates) as sometimes, it is just stored as plain text, i.e. as its linearisation.

Finally and most importantly, our result allows one to get an extension of a known model of ranked tree to word transductions, to unranked tree to word transductions, namely, deterministic pushdown unranked tree to word walking transducers (DPUWT). To avoid technical details, we define formally this model only in Appendix, and rather give intuitions here. DPUWT can walk through the unranked tree following the next-sibling and first-child predicates (and their inverse), while producing words on the output. They are also equipped with a pushdown store with a visibly condition: whenever they go down the tree by one level, they have to push one symbol onto the stack, and going up, they pop one symbol. They let the stack unchanged when moving horizontally between siblings. With the single-use restriction, defined similarly as for D2VPT, we get that MSO[u2w] = DPUWT_{uw}. Therefore, if the input is given by an unranked tree, one can rather use a DPUWT or a D2VPT on the linearisation.
Nested word to nested word transductions As we claimed earlier, D2VPT\textsubscript{su} can be used to define unranked tree transformations represented as nested word to nested word transducers, that is, as nested word to word transducers with a structured output alphabet. On the logical side, MSO[\text{nw2w}] transductions can be extended with binary formulas \( \phi(x, y) \) aiming at representing the matching relation existing on output nested words. As checking whether a relation denotes a matching relation is MSO definable (see Example 2), one can decide whether any input nested word is indeed transformed by the MSO[\text{nw2w}] transducer into a nested word by testing the validity of the sentence obtained from the logical definition of the matching \( M \) (Example 2) by replacing the predicate \( M \) with \( \bigvee_{x, y \in \text{nw}^c} \phi(x, y) \). So, starting from an MSO[\text{nw2w}] transducer with a matching relation defined on its output, one may forget this matching and view this transducer as an ordinary MSO[\text{nw2w}] transducer; this machine turns out to be equivalent in the sense that remaining call and returns symbols induce uniquely the erased matching. Finally, by the results presented in this paper, one can from this MSO[\text{nw2w}] transducer build an equivalent D2VPT\textsubscript{su} whose range will indeed contain only nested words and thus, defines an unranked tree transformation.

Let us point out that our results do not entail the trinity for tree-to-tree transformations: the class of D2VPT which produce only nested words/trees as output may be a good candidate to complete the missing part (the equivalence between MSO transformations and streaming tree transducers has already been established in (Alur and D’Antoni 2012)). Nonetheless, deciding this class seems to be challenging and moreover, there is actually no guarantee that it corresponds to the other two cited members of this trinity.

Input streaming In an input streaming scenario, one assumes that the input nested word is given as a stream of call and return symbols. In such a scenario, one wants to transform the input stream as soon as possible, on-the-fly, and it is not reasonable to load the whole stream in memory. An interesting question is whether a given D2VPT really needs its two-way ability? In other words, can we decide whether a given D2VPT is equivalent to a (one-way) VPT? For words and two-way finite transducers, this question has been shown to be decidable in (Filiot et al. 2013). As future work, we want to extend this result to D2VPT.

References


A. Appendix

A.1 Two-way visibly pushdown automata

Proposition 1. The relation \( \sim \) is a congruence of finite index.

Proof. We consider \( R \) the set of binary relations over \( Q \times \mathbb{D} \). Obviously, \( R \) is finite. As traversals are subsets of \( R, \sim \) is of finite index. Let us now prove that \( \sim \) is a congruence relation for the binary operation \( \cdot \) and the unary ones, \( f_{c,r} \) (for \( c \in \Sigma, r \in \Sigma_r \)).

From \( \sim \), we define four equivalence relations \( \sim_{\ll}, \sim_{\rr}, \sim_{\rl}, \sim_{\lr} \) on \( Q \times Q \) such that for \( (\alpha, \beta) \in \{1, r\} \), we have \( u \sim_{\alpha \beta} v \) if

\[
[u]_{\sim_{\alpha \beta}} \cap (Q \times \{bdir(\alpha)\}) = [v]_{\sim_{\alpha \beta}} \cap (Q \times \{bdir(\beta)\})
\]

where \( bdir(1) = edir(r) = \Rightarrow \) and \( bdir(r) = edir(1) = \Leftarrow \).

Intuitively, \( (p, q) \) belongs to \( [u]_{\sim_{\alpha \beta}} \) (respectively to \( [v]_{\sim_{\alpha \beta}} \)) if there exists a run of \( A \) on \( w \) starting reading \( w \) from the left side, ie, with direction \( \rightarrow \) in state \( p \) and leaves the word on the left, ie, with direction \( \Leftarrow \) (resp. on the right, ie, with direction \( \rightarrow \)) in state \( q \).

The relation \( \sim \) is uniquely determined by the four relations \( \sim_{\ll}, \sim_{\rr}, \sim_{\rl}, \sim_{\lr} \) and in particular, \( \sim \) is a congruence iff all the \( \sim_{\ll}, \sim_{\rr}, \sim_{\rl}, \sim_{\lr} \) are congruences.

Let us first notice that for \( c \), one has \( [\varepsilon]_{\sim_{\ll}} = [\varepsilon]_{\sim_{\rr}} = \emptyset \) whereas \( [\varepsilon]_{\sim_{\rl}} = [\varepsilon]_{\sim_{\lr}} \) are the identity relation.

Let us consider \( u, u', v, v' \in \mathcal{N}(\Sigma) \) and assume that \( u \sim u' \) (and thus, \( u \sim_{\ll} u', u \sim_{\rr} u', u \sim_{\rl} u', u \sim_{\lr} u' \)) and \( v \sim v' \). We consider \( u.v \) and \( u'.v' \) and prove that \( u.v \sim u'.v' \).

From the definition of runs and traversals, one has

\[
[u.v]_{\sim_{\alpha \beta}} = [u]_{\sim_{\alpha \beta}} \cup [u]_{\sim_{\alpha \beta}} \circ [v]_{\sim_{\alpha \beta}} \circ ([u]_{\sim_{\alpha \beta}})^\ast \circ [v]_{\sim_{\alpha \beta}} \circ [u]_{\sim_{\alpha \beta}}
\]

Hence, \( [u.v]_{\sim_{\alpha \beta}} = [u'.v']_{\sim_{\alpha \beta}} \) for all \( \alpha, \beta \in \{1, r\} \) and so, \( u.v \sim u'.v' \).

Let us point out that these definitions are similar to those defined for words in the case of two-way finite state automata (Shepherdson 1959) and that \( [u.v]_{\sim_{\alpha \beta}} = [u.(v.w)]_{\sim_{\alpha \beta}} \) and \( [u.]_{\sim_{\alpha \beta}} = [\varepsilon]_{\sim_{\alpha \beta}} \) for all \( \alpha, \beta \in \{1, r\} \).

Now, let us consider \( u, u' \in \mathcal{N}(\Sigma) \) and assume that \( u \sim u' \).

We consider \( cur = f_{c,r}(u) \) and \( cur' = f_{c,r}(u') \) and show that \( cur \sim cur' \). Expressing traversals on \( cur \) is much more intricate. To ensure that traversals abstract properly runs, we need to forget about stack contents and thus, reason again only on nested words when composing sub-runs of \( cur \).

Hence, new notations are needed: we let for \( u \in \mathcal{N}(\Sigma) \) and so, \( u.v \sim u'.v' \).

The expressions \( Z_{\alpha \beta}^\mathbb{D} \) and \( Z_{\alpha \beta}^{\mathbb{D}w} \) stands both for left-to-left traversal reading twice the initial letter \( c \); the former one represents a back-and-forth move on \( c \) whereas \( Z_{\alpha \beta}^\mathbb{D} \) implies that between the readings of \( c \) a left-to-left traversal of \( w \) is performed. If the last direction \( d \) is \( \Leftarrow \) then the reading head leaves the word, otherwise the next reading will be \( c \) again. The expressions \( Z_{\alpha \beta}^\mathbb{D} \) and \( Z_{\alpha \beta}^{\mathbb{D}w} \) are defined dualy.

\[
T_{\alpha \beta}^\mathbb{D} = \bigcup_{\gamma \in \mathbb{D}} \left( \bigcup_{\gamma \in \mathbb{D}} \{ (p, q) \} \circ [w]_{\sim_{\alpha \beta}} \circ \left( \bigcup_{\gamma \in \mathbb{D}} \{ (p, q) \} \right) \right)
\]

The expression \( T_{\alpha \beta}^{\mathbb{D}w} \) represents a direct traversal from left-to-right, going once through \( c \) and \( r \).

Finally, the classes \( [curw]_{\sim_{\alpha \beta}}, [cur]_{\sim_{\alpha \beta}} \) and \( [curw]_{\sim_{\alpha \beta}} \) are defined in Figure 3.

Hence, we indeed have that \( f_{c,r}(u) \sim_{\alpha \beta} = [curw]_{\sim_{\alpha \beta}} = [curr']_{\sim_{\alpha \beta}} = [curr(u')]_{\sim_{\alpha \beta}} \) for all \( \alpha, \beta \in \{1, r\} \). \( \blacksquare \)

Corollary 1. For any 2VPA \( A \), deciding the emptiness of \( A \) (ie \( \mathcal{L}(A) = \emptyset \)) is EXPTime-c. The same result holds for 2DVAR.

A.2 Two-way visibly pushdown transducers

Lemma 2. Any unambiguous VPT \( T \) can be written as the composition of two VPT \( T_1 \circ T_2 \) where \( T_1 \) is deterministic and \( T_2 \) is letter-to-letter and co-deterministic. Furthermore, if \( T \) is letter-to-letter, so is \( T_1 \).

Proof. It has been proved in (7) that every unambiguous VPT can be transformed into a DVPPT equipped with a look-ahead limited to the current hedge. Formally, such a transducer is defined as a triple \( (T, A, \lambda) \) where \( T \) is a VPPT, \( A \) is a VPA with no initial states, and \( \lambda \) is a mapping from call transitions of \( T \) to states of \( A \).

Given a state \( p \) of \( A \), we denote by \( A_p \) the VPA defined from \( A \) by letting \( \{ p \} \) be the set of initial states. A call transition \( t \) of \( T \) can then be fired at some position of an input word \( w \) only if the longest nested word of \( w \) from this position belongs to \( \mathcal{L}(A_p) \).

Intuitively, the decomposition of a VPPT with look-ahead works as follows: the co-deterministic letter-to-letter VPPT does a first pass enriching the alphabet with the results of the look-ahead tests. Then the deterministic VPPT simulates the VPPT with look-ahead using this additional information.

Formally, let \( (T, A, \lambda) \) be a VPPT with look-ahead from \( \Sigma \) to \( \Delta \), with \( (Q, F, \Gamma, \beta) \). We first define the structured alphabet \( \Sigma' \) as the disjoint union of the set of call symbols \( \Sigma_c \times Q^d \), and the set of return symbols \( \Sigma_r \). We define the co-deterministic letter-to-letter VPPT \( T_2 = (A_2, Q_2) \) from \( \Sigma \) to \( \Sigma' \), where \( A_2 \) is defined as the co-determinisation of \( A \). Formally, let us denote by \( id_\chi \) the set \( \{ (q, q) \mid q \in X \} \). We define \( A_2 = (2^Q \times Q, I_2, id_F, \Sigma_r \times 2^Q \times Q \times \alpha) \) where \( I_2 = \{ S \subseteq Q \times Q \mid S \cap I \times F \neq \emptyset \} \) and the transitions of \( A_2 \) are defined as follows:
on the enriched alphabet $p$ that the look-ahead constraint is satisfied. Then, let us remark that when the state $w$ of the one-way enters a hedge. Remark that thanks to this, upon moving to the left of a call letter, the state of the one-way is directly given by the information in the stack.

We now explain how we can treat subhedges as letters. First, we consider an automaton not over the subhedge, but over their summaries, which are finite. We can thus compute a finite automaton of the summaries, and apply the Hopcroft-Ullman construction on it. Consequently, we need to be able to compute the summaries of a given subhedge. This is easily done on the fly using the determinisation procedure of the VPAs. Finally, note that applying the Hopcroft-Ullman construction to the automaton of summaries gives the state in which the one-way enters the previous subhedge (when rewinding a run). This allows us to maintain the invariant, and by reading this subhedge we can compute the state of the one-way at the previous position (from where we started).

Note that the Hopcroft-Ullman routine is deterministic, and consequently the construction preserves determinism.

**Theorem 3.** Given a letter-to-letter DVPT $A$ and a 2VPT $B$, we can construct a 2VPT $C$ that realizes the composition $C = B \circ A$.

If furthermore $B$ is deterministic, then so is $C$.

**Proof.** We first notice that since we’re considering visibly pushdown machines, the stacks of both machines are always synchronized, meaning that they have the same height on each position. Then, let us remark that when the 2VPT moves to the right, we can do the simulation in a straight forward fashion by simulating it on the production of the one-way, which we can compute. It becomes more involved when it moves to the left. We then need to rewind the run of the one-way, and nondeterminism can arise. To bypass this, let us recall that a similar construction from (Hopcroft and Ullman 1967) exists for classical transducers, and that the rewinding is done through a back and forth reading of the input, backtracking the run up to a position where the nondeterminism is cleared, and then moving back to the current position.

The main idea is that if we were to consider a hedge as a word over subhedges (see Figure 4), we can use the Hopcroft-Ullman construction, given that we know the initial state, i.e. the state in which the one-way enters the hedge. To overcome this, we will ensure the invariant that the stack contains not only the stack symbols from the two transducers, but also at each step it contains the state in which the one-way enters a hedge. Remark that thanks to this, upon moving to the left of a call letter, the state of the one-way is directly given by the information in the stack.

We now explain how we can treat subhedges as letters. First, while the subhedge alphabet is infinite, we are actually interested in their behaviour in the one-way. Thus we consider an automaton not over the subhedge, but over their summaries, which are finite. We can thus compute a finite automaton of the summaries, and apply the Hopcroft-Ullman construction on it. Consequently, we need to be able to compute the summaries of a given subhedge. This is easily done on the fly using the determinisation procedure of the VPAs. Finally, note that applying the Hopcroft-Ullman construction to the automaton of summaries gives the state in which the one-way enters the previous subhedge (when rewinding a run). This allows us to maintain the invariant, and by reading this subhedge we can compute the state of the one-way at the previous position (from where we started).

Note that the Hopcroft-Ullman routine is deterministic, and consequently the construction preserves determinism.

**Formal construction.** Let $A = \left((Q, i, F, \Gamma, \delta), Q_1\right)$ be a letter-to-letter DVPT and $B = \left((P, j, G, \Theta, \alpha), Q_2\right)$ a 2VPT that can be composed with $A$. We assume that $A$ works on the alphabet equipped with left and right markers and preserves them. Note that it can easily be extended if it is not the case.

We construct $C = \left((N, k, H, \Omega, \beta), Q_3\right)$ a 2VPT that realizes the composition.

- $N = N_m \cup N_r \cup N_f \cup N_s$ where $N_m$, $N_r$ and $N_f$ correspond to the classical sets of the Hopcroft-Ullman construction, and $N_s$ is used to compute the summary of a subhedge. We have the main mode $N_m = P \times Q$, the back mode $N_b = P \times Q \cup P \times Q^2$ and the further mode $N_f = P \times Q^2$, while $N_s = Q^2$. Note that there are also other states like $\text{read}$ or states from $P \times Q \times \{\text{end}\}$ that were omitted. The total size of the omitted states is linear in $P$ and $Q$.

- $k = (i, j)$ is the initial state.

- $H = F \times G$ is the set of final state.

- $\Omega$ can similarly to $N$ be written as the disjoint union of stack alphabets for the different modes. We have $\Omega_m = Q \times \Gamma$, $\Omega_r = P \times (Q^2 \cup Q) \times \Sigma_r$, $\Omega_f = P \times ((Q \times \Gamma) \cup (Q \times \Gamma)^2)$ and $\Omega_s = Q^2 \times \Sigma_r$.

Now we give the transition function $\beta$. Lowercase letters denote element of its uppercase counterpart. The direction of a transition is given by the sense of an arrow, and the resulting direction is omitted if it doesn’t change. Push transitions are denoted with a $+$ symbol while pop transitions are denoted by a $-$ symbol. For
example, we write \((q, r, q', r, \gamma)\) in \(\delta^{push}\) as \(q' \xrightarrow{r, \gamma} q\) and \((q, r, q', r, \gamma)\) in \(\delta^{pop}\) as \(q \xrightarrow{r, \gamma} q', r, \gamma\).

- The first three items describe the cases when we are able to directly advance in the two runs. These are the simpler cases. The fourth corresponds to the end of the Hopcroft-Ullman construction, where all the needed information was computed. In these cases, the production of \(O_3\) is the one of the corresponding transition of \(O_2\). Note that in all other cases, the production of \(O_3\) will be empty and thus omitted.

- \((p, q) \xrightarrow{c, +, (q, \gamma)} (p', q')\) if \(O_1(q \xrightarrow{c, +, \gamma} q') = c'\) and \(p \xrightarrow{c', +, \theta} p'\), \(d\).

- \((p, q) \xrightarrow{r, - , (p, q, \gamma)} (p', q')\) if \(O_1(q \xrightarrow{r, - , \gamma} q') = r'\) and \(p \xrightarrow{r', - , \theta} p'\), \(d\).

- \((p', q') \xrightarrow{c, - , (q, \gamma, \theta)} (p, q, q')\) if \(O_1(q \xrightarrow{c, - , \gamma} q) = c'\) and \(p' \xrightarrow{c', - , \theta} p\).

- \((p', q') \xrightarrow{r, - , (p, q, \gamma)} (p, q, q', r, \gamma)\) where there exists \(q''\) such that \(O_1(q' \xrightarrow{r, - , \gamma} q'') = r'\) and \(p' \xrightarrow{r', - , \theta} p\).

- When \(B\) moves to the left on a recall letter, we engage in the Hopcroft-Ullman construction. In order to do that, we need to compute the summary of the subhedge we are about to read. Note that a similar transition happens when the automaton on summaries rewrites one more step. Thus we have the following transitions:

  - \(\text{id}_Q \xrightarrow{r, +, (p, q, r)} (p, q)\).
  - \(\text{id}_Q \xrightarrow{r, +, (p, R, r)} (p, R)\).

- Computing a summary amounts to determining a VPA. Note that we stop when we reach the height we are interested in, which is where the stack first contains a state of \(B\), which is handle by the next item. Given a summary \(S\) and \(c, r\) a call and return letter respectively, we define \(\text{Update}(c, S, r) = \{(q, q') \mid \exists (q_1, q_2) \in S \text{ and } q \xrightarrow{r, \gamma} q_1 \text{ and } q_2 \xrightarrow{r, \gamma} q'\}\). This will reveal to be useful in the remainder of the construction.

- \(\text{id}_Q \xrightarrow{r, +, (S, r)} S\).

- \(S'' \xrightarrow{c, - , (S', r)} S\) where \(S'' = S \circ \text{Update}(c, S, r)\).

- After reading the first subhedge, we get to the point where the top stack symbol is of the form \((p, q, r)\). If there is only one candidate, then there is no ambiguity. Otherwise, we start rewinding the runs.

- \((p, q', \text{end}) \xrightarrow{c, - , (p, q, r)} S\) if \(q'\) is the only state such that \((q', q)\) belongs to \(\text{Update}(c, S, r)\).

- \((p, R) \xrightarrow{c, - , (p, q, r)} S\) where \(R = \{(q', q') \mid (q', q) \in \text{Update}(c, S, r)\}\).

- After reading the following subhedges, similar subcases appear, depending on whether the nondeterminism is cleared or not. If there is only one candidate left, we store a state leading to the next subhedge, as well as a state leading to another candidate. They will be used to know when we got to the correct position. Otherwise we just update the set of partial runs.

- \((p, q, q') \xrightarrow{c, - , (p, R, r)} S\) if \(R\) is defined, \(\text{ReUpdate}(c, S, r)\) \(Q \times \{R(q)\}\) and if \(R(q')\) is defined and different from \(R(q)\).

- \((p, R') \xrightarrow{c, - , (p, R, r)} S\) where \(R' = R \circ \text{Update}(c, S, r)\) and \(R' \not\subseteq Q \times \{q\}\) for any \(q\).

- It can happen that the nondeterminism has not been cleared until we reach the beginning of the hedge. In the same way that the Hopcroft-Ullman uses the initial state, we then use the information on the top of the stack to decide the candidate.

- \((p, q, \theta, q') \xrightarrow{c, +, (p, q, \gamma, \theta)} (p, R)\) if \(q''\) is such that \(q \xrightarrow{c, +, \gamma} q'',\) both \(R(q')\) and \(R(q'')\) are defined and different.

- Due to the definition, the model of 2VPT does not allow for direct u-turns. Consequently, the u-turns have been parametrized by specific states in the previous cases. We know explicit how we handle them:

- \((p, q, \text{end}) \xrightarrow{c, +, (p, q, \gamma)} q'\) where \(q \xrightarrow{c, +, \gamma} q'\). We also have a subroutine that follows run of \(A\) on this subhedge until it ends.

- \((p, q, q') \xrightarrow{c, +, (p, q, \gamma)} \text{read}\) where the state \(\text{read}\) is a subroutine that only reads the subhedge until it pops the stacked information.

- \(\text{read} \xrightarrow{r, - , (p, q, r)} (p, q, q')\). At the end of the \(\text{read}\) subroutine, we start following two runs in parallel, in the same way as in the next subcase.

- \((p, q, \theta, q') \xrightarrow{c, +, (p, q, \gamma, \theta)} (p, q', q')\) where \(q \xrightarrow{c, +, \gamma} q''\).

- When we are in states of \(N_T\), i.d. states of the form \((p, q, q')\), we simply follow the two runs in parallel, stacking \(p\) and the current states on the current height and the stack letters of both runs at each step nonetheless. This subroutine ends upon popping a stack letter that contains \(p\) where the two runs collide, meaning we reached the original position. We now explicit what happens on this position:

- \(q \xrightarrow{c, - , (p, q, q', r, \gamma)} (p, q', q, \gamma)\).

- \((q, q') \xrightarrow{c, +, (p, q_1, q_2, \gamma, r, \theta)} (p, q_1, q, \gamma)\), \(\text{if there exists } q''\) such that \(q \xrightarrow{c, +, \gamma} q''\) and \(q' \xrightarrow{c, +, \gamma} q'\). \(\Box\)

### A.3 Expressiveness of Two-Way Visibly Pushdown Transducers

**Theorem 6.** An order-preserving transduction is definable in MSO\([\text{nw}2\text{w}]]\ if, and only if, it is definable by a functional VPT.

**Proof.** The proof relies on the similar result for finite words from (Filiot 2015) and the equivalence between VPA and MSO\([\text{nw}])\ from (7). Let \(T\) be a functional VPT. From (7), we know that we can construct an equivalent unambiguous VPT \(T'\) realizing the same function. Using (7), we can construct an MSO\([\text{nw}])\ formula \(\varphi\) of the form \(\exists X_1, \ldots, \exists X_n \cdot \psi(X_1, \ldots, X_n)\) that recognizes \(\text{dom}(T')\). Moreover, given \(u \in \text{dom}(T')\), there exists a unique assignment of the variables \(X_i\) satisfying \(\psi\), such that a variable \(x \in X_i\) if, and only if, \(x\) quantify a position \(j\) such that the unique accepting run of \(T'\) on \(u\) is in state \(q_i\) on position \(j\). Using \(\varphi\), we can then easily construct an MSO\([\text{nw}2\text{w}])\ transduction \(T''\) using \(|Q|\) copies. The domain formula is \(\varphi\), position formulas are \(\phi_{pos}^q(x) = \varphi(x \in X_q)\). The successor transition is given by \(\phi_{pos}^q(x, y) = (S(x, y) \land \phi_{pos}^q(x) \land \phi_{pos}^q(y))\) and we label the q copy of a node by the possibly empty production of the transducer.

Within the class of VPT, the class of functional VPT is decidable in PTime (Filiot et al. 2010)
in state $q$ reading the label of the node. We have, for $v$ a production of $T$, $\phi_i^v(x) = \bigwedge_{q \in P} a(x)$ where $a_i(v) = \{a \in A | \exists q^i \overset{a}{\rightarrow} q^i\}$. Note that labeling by possibly empty words is not restrictive as MSO transductions are closed under composition, and a simple transduction can extend words into linear graphs and compress the $c$-labeled paths.

Now given an order-preserving MSO$[nw2w]$ $T$, we construct an unambiguous VPT that recognizes the same function. As $T$ is order-preserving, for every $u = u_1, \ldots, u_n$ in dom($T$), we can decompose $T(u)$ in $v_1, \ldots, v_n$ where $v_i$ corresponds to the production from position $i$. Let us call $B$ the finite set of all possible $v_i$ appearing in such a decomposition. For any $v_i$ in $B$, we use the formulas of $T$ to construct a formula $\phi_i(x)$ that holds on an input word $u$ and a position $i$ if in the decomposition of $T(u)$, $v_i = v_i$. For any sequence $I = (e_1, \ldots, e_k)$ of $|\Sigma|$ different copies of $T$, we define $\phi_I^v(x) = \bigwedge_{e \in B} \phi_{\text{dom}}(x, v) \land \bigwedge_{x \in B} \phi_{\text{trans}}(x, v) \land \bigwedge_{y \in B} \phi_{\text{comp}}(x, v)$ where $\phi_{\text{dom}}$ is obtained from $\phi_{\text{dom}}$ by replacing every predicate $a(x)$ by $\bigwedge_{v \in B} a(v, x)$. Now thanks to (7), we can construct a DVPA that recognizes $L = L(\psi)$. Finally, we transform it into a VPT by replacing transitions reading (a, $v$) into transitions reading $a$ and producing $v$. Since $T$ realizes a function, we obtain a functional VPT, concluding the proof.

**Proposition 3.** Deciding the single use property on a 2VPT is ExpTime-c.

**Proof.** We prove that this problem is equivalent to deciding the emptiness of a D2VPA, which concludes the proof thanks to Corollary 1.

Let us first remark that if $A$ is single use, it is single use with respect to the set of all states that can produce a non empty word. Let $A$ be a 2VPT on an input alphabet $\Sigma$. We define a 2VPA $B$ on the marked alphabet $\Sigma \times \{0, 1\}$ as follows. The transducer $B$ first reads its input to ensure that there is exactly one position with a 1. It then nondeterministically chooses a producing state $q$ and simulates $A$ on its input. It finally accepts if it visits the marked position twice in state $q$. Then $A$ is single use if, and only if, the language recognised by $B$ is empty. Since the size of $B$ is linear in the size of $A$, deciding the single use property is ExpTime.

Conversely, let $B$ be a 2VPA. We construct a 2VPT $A$ as follows. All existing transitions of $B$ are set to produce the empty word, and every accepting transition is replaced by a back and forth move on the last position, producing a single letter. Then the producing transitions can only be fired in $A$ if there is a run of $B$ that fires an accepting transition. If it is the case, then the corresponding run on $A$ will visit the state $q$ twice in the last position while producing non empty words. Thus the language recognised by $B$ is empty, if, and only if, $A$ is single-use. As the size of $A$ is linear in the size of $B$, we get the ExpTime-hardness of the single use problem.

**Lemma 3.** There is an MSO-definable nested word to word transduction $T$ which is not definable by strongly single-use D2VPT$^{LA}$.

**Proof.** We explicitly a transformation that is definable by an MSO$[nw2w]$ transduction but not by a strongly single-use D2VPT$^{LA}$.

Consider an alphabet $\Sigma$ with some special letters $c$ and $r$ from $\Sigma_c$ and $\Sigma_r$, respectively. We define the transformation $T$ which associates to a word $w_0c/w_1c/w_2, w_0cw_1w_2c, \ldots, w_{n−1}cw_nw_{n}′$ where $w_i, w_i′$ are non empty nested words and do not any contain $c$, for $0 \leq i \leq n$, the word $w_0\overline{w}_0w_0w_1w_2, \ldots, w_nw_{n−1}w_{n−1}′$. Its domain is then the set of nested words where any $c$ is matched by an $r$, and all letters $c$ appear successively nested on a given branch. The transformation is illustrated in Figure 5.

Before giving the MSO$[nw2w]$ that defines $f$, we explain how it is not definable by a strongly single use D2VPT$^{LA}$. As the $w_i$ and $w_i′$ are unbounded, they cannot be guessed by look-around. Thus a machine realizing it has to visit these subwords in the order they are output. But each walk from $w_i$ to $w_i′$ has to cross $w_i$. Thus, $w_n$ is read at least $2n$ times. As $n$ is not bounded, $f$ cannot be realized by a strongly single use D2VPT$^{LA}$.

Now we define a MSO$[nw2w]$ $T$ that realizes $f$. In order to do that, we define a binary predicate $H(x, y)$ which holds if $x$ and $y$ are call or return positions of a same hedge. Let $H_c(X)$ be defined by the formula:

$$\forall x \in X \exists \sigma(x) \rightarrow (\forall y. M(x, y) \lor (S(y, x) \lor \exists \sigma(y) \rightarrow y \in X)) \land \Sigma(x) \rightarrow (\forall y. M(x, y) \lor (S(x, y) \lor \exists \sigma(y) \rightarrow y \in X))$$

with $\Sigma(x) = \bigvee_{c \in \Sigma} \sigma(x)$ for $c = c, r$. Then a set $X$ satisfies $H_c(X)$ if, and only if, it is closed by the relation $belong$ to the same hedge. We then simply $H(x, y) = \forall x \in X \land H_c(X) \rightarrow y \in X$. We also define the parent relation $P(x, y) = \exists z.H(z, x) \land (\exists \sigma'. H(z, \sigma') \rightarrow z \leq z') \land S(y, z)$ which holds if $y$ is the call corresponding to the parent of $x$.

We can now define the domain formula $\phi_{\text{dom}} = \forall x c(x) \rightarrow (\forall y. M(x, y) \rightarrow (r(y)) \land (P(x, y) \rightarrow (y))) \land (H(x, y) \land (c(x) \rightarrow x = y))$ stating exactly what was mentioned earlier. The transducer $T$ uses 1 copy, the position formula $\phi_{\text{pos}}(x) = (\exists c(x) \lor (\exists y.M(x, y) \land c(y)))$ simply erases the $c$ labeled positions and their matching, the labeling formulas simply maintain the labels, and finally the successor formula $\phi_s(x, y)$ is defined by:

$$\exists z.S(x, z) \land \neg c(z) \lor (\exists \sigma M(w, z) \land c(w)) \land y = z) \lor (c(z) \land \exists z'. M(z', z) \land S(z', y)) \lor (\exists z', z'' M(z', z) \land c(z') \land N_{\text{ext}}(z', z'') \land S(z'', y))$$

where $N_{\text{ext}}(x, y) = x < y \land c(y) \land \forall z. x \leq z < y \land \neg c(z)$.}

**Theorem 7.** Let $f$ be a transduction from nested words to words. Then $f$ is MSO-definable iff it is definable by a (look-around) D2VPT$^{LA}$, i.e.,

$$\text{MSO}[nw2w] = \text{D2VPT}^{LA}_{su} = \text{D2VPT}^{LA}_{su}$$

**Proof.** We prove the equivalence of MSO$[nw2w]$ = D2VPT$^{LA}_{su}$ by using results of Courcelle and Engelfriet 2012, on the class of deterministic tree-to-word walking transducers (DTWT), possibly augmented with visibly pushdown stack (then denoted DPTWT) and a regular look-around ability (denoted by an exponent $\text{ls}$), and possibly restricted to linear-size increase the class of linear-size increase transductions (denoted by subscript $\text{ls}$), or to strongly single-use (denoted by

![Figure 5. The transformation $f$ alterns $n$ times between positions left and right of $w_n$. Thus it has to read $w_n$ at least $2n$ times.](image-url)
subscript $ssu$). We will define the most general model formally in the sequel.

Let us also denote by $\text{MSO}[b2w]$ the class of MSO-definable transductions from (ranked) trees to words. Then, it is shown in (Courcelle and Engelfriet 2012) that

$$\text{MSO}[b2w] = \text{DTWT}^{ti} = \text{DPTWT}^{ti}_{la}$$

The inclusion $\text{MSO}[nw2w] \subseteq \text{D2VPT}^{la}$ is proved using the equality $\text{MSO}[b2w] = \text{DTWT}^{ti}$. Due to determinism, $\text{DTWT}^{ti}$ are always strongly single-use (otherwise they could be stuck in a loop), i.e., $\text{DTWT}^{ti} = \text{DTWT}^{ti}_{ssu}$ (see (Courcelle and Engelfriet 2012), in which it is just called single-use). Using a first-child next-sibling encoding of nested words $w$ into binary trees $\text{fcns}(w)$, we have $\text{MSO}[nw2w] = \text{MSO}[b2w] \circ \text{fcns}$, and therefore $\text{MSO}[nw2w] = \text{DTWT}^{ti}_{ssu} \circ \text{fcns}$. Then, we show that $\text{DTWT}^{ti}_{ssu} \circ \text{fcns}$ are always strongly single-use (otherwise they could be stuck in an internal state to a new state, and produces some partial word on the output. It can also decide to stop the walk by going to a stopping state $q_s$.

Such transducers can be augmented with look-around. We define look-around by an unambiguous bottom-up tree automaton. Prior to starting the computation of the tree walking transducer, the tree, if accepted by the look-around automaton, is labeled by the states of the accepting run of the automaton. Then, transitions are taken depending also on the look-around states.

Finally, walking transducers can be augmented by a (visibly) pushdown store. Initially at the root the pushdown stack contains an initial symbol $\gamma_0$, and whenever the transducer goes one step downward, it has to push one symbol on the stack. If it moves one step upward, it has to pop one symbol. At any moment, it can also read the top symbol of the stack.

Formally, a deterministic pushdown tree to word walking transducer with look-around from $\text{Trees}_{SA}$ to $\Sigma^*$ is a tuple $T = (L, Q, q_0, q_s, \Gamma, \gamma_0, R)$ where $L$ is an unambiguous bottom-up tree automaton$^7$ over a finite set of states $P$ (the look-around automaton), $Q$ is a finite set of states, $q_0$ is the initial state, $q_s$ the stopping state, $\Gamma$ is a finite stack alphabet with initial symbol $\gamma_0$, $R$ is a transition function such that:

$$R : Q \times P \times \Lambda \times \Sigma^* \times (\{q_0\} \cup \Gamma \times \{1, 2\}) \cup \{\{1\}\} \cup \{1\})$$

A configuration of $T$ on a tree $t$ is a triple $(q, n, \beta, u) \in Q \times N \times \Gamma^* \times \Sigma^*$. For all trees $t \in \text{Trees}_{SA}$, if $t$ is accepted by the look-around automaton, we define $\rightarrow$ to be a binary relation between consecutive configurations as follows: for all $q, q' \in Q$, all $n, n' \in N$, all $\beta, \beta' \in \Gamma^*$, all $\gamma \in \Gamma$, all $u, v \in \Sigma^*$, and $i \in \{1, 2\}$, we have

$$\rightarrow (q, n, \beta, u) \rightarrow (q', n', \beta', uv)$$

if the accepting run of $L$ labels $n$ by a state $p \in P$ such that $(q, p, t(n), \gamma) \in \text{Dom}(R)$ and either

- **(stopping move)** $R(q, p, t(n), \gamma) = (v, q_s)$ and $q_s = n, \beta' = \beta \gamma, \gamma$, or
- **(downward move)** $R(q, p, t(n), \gamma) = (v, q', i, i)$ for $i \in \{1, 2\}$ and $\beta' = \beta \gamma, \gamma, t(n) \in \Lambda_2$, and $n'$ is the $i$-th child of $n$, or
- **(upward move)** $R(q, p, t(n), \gamma) = (v, q', -1)$ and $n \neq \epsilon$ (i.e. $n$ is not the root node), $\beta' = \beta, \gamma$, and $n'$ is the father of $n$.

A run of $T$ on a tree $t$ is a finite sequence of configurations $c_0 c_1 \cdots c_m$ such that $c_i \rightarrow c_{i+1}$ for all $i = 0, \ldots, m - 1$. It is accepting if $c_m = (q_0, \epsilon, \gamma_0, \epsilon)$ and $c_m = (q_s, n, \beta, u)$ for some $n \in N_2, \beta \in \Gamma^*$, $n \in \Sigma^*$. Since $R$ is a function and $L$ is unambiguous, there exists at most one accepting run per input tree $t$, and we call $u$ the output of $t$. The transduction realized by $T$ is the set of pairs $(t, u)$ such that $t$ is accepted by the look-around automaton, and there exists an accepting run of $T$ on $t$ whose output is $u$. The class of deterministic pushdown tree to word walking transducers$^8$ is denoted by $\text{DPTWT}^{ti}$.

---

$^7$ We refer the reader to (Comon-Lundh et al. 2007) for a definition of bottom-up tree automata

$^8$ We have slightly changed the definition of (Courcelle and Engelfriet 2012) to simplify our presentation, but in an equivalent way, and have specialized it to the tree-to-word setting. In (Courcelle and Engelfriet 2012), look-around are MSO-formulas on trees, with one first-order-variable, attached to the transitions of the transducer: a transition can be fired only if its look-around formula holds at the current node. It is known that such an MSO formula $\phi(x)$ is equivalent to an unambiguous bottom-up tree automaton $A_\phi$ (Niehren et al. 2005; Neven and Schwentick 2002) in the following sense: the automaton as a special set of selecting states $S$, such that on a tree $t$ accepted by the automaton, a node $n$ is such that $t \models \phi(n)$ if this node is labeled by a state of $S$ in the accepting run of the automaton on $t$. If $\phi_1(x), \ldots, \phi_n(x)$ are the look-around formulas appearing on the transitions of the tree walking transducer, then by taking the product of the unambiguous automata $A_{\phi_i}$, one obtains an unambiguous automaton $A_{\phi_i}$.
move one-step to the right in w. For the second-child move, T′ has to move to the next call symbol at the right of current one, at the same nesting depth: this is done by pushing one special symbol γ 2 when reading the first call, and moving to the right, until γ 1 is popped. For a father move, it suffices to move left: if the previous symbol is a call, then T′ has arrived to the call corresponding to the father node. Otherwise, the left symbol is a return symbol: again, a special symbol γ 1 is pushed, to know, when T′ walks left, whenever it is at the same depth as the initial call symbol. If after popping γ 1, a call symbol is read again, then it corresponds to the father. Note that these walks do not produce anything on the output.

The look-around L of T′ is transformed into a look-around of T′ such that, if L labels a tree node labelled (c, r) by a state p, then T′ will label the call symbol c by the state p, as well as the call symbol r. It is possible, since bottom-up tree automata and visibly pushdown automata correspond modulo first-child next-sibling encodings, while preserving unambiguity (Alur and Madhusudan 2009). Therefore, if P′ is the set of states of L, then the set of states of the look-around automaton of T′ is P × P′. Then, a transition (q, p, u, (q′, d)) where d ∈ \{−1, 1, 2\} is simulated by T′(q, p, (c, r), u, (q′, d′)) where d′ ∈ \{−1, 1\} and ends in state q′, and performs moves as explained before.

There is a last additional technical difficulty: fcns encodings contain the symbol ⊥, unlike the encoded nested words. Therefore, T′ may move to ⊥, while T cannot. Moves to nodes labeled ⊥ can be simulated easily by T′ by adding c-transitions, which can in turn be removed while preserving determinism. It is not difficult but unnecessarily technical.

Finally, since T′ is necessarily single-use (due to non-determinism), T′ is also single-use (the extra states added to simulate one-step moves of T by several moves of T′ may be used several times at the same tree node, but the transitions fired from those states are c-producing).

Proof of inclusion (3) Due to the single-use restriction, any D2VPT(LA) transduction is LSI. It remains to show that a D2VPT(LA) can be simulated by a DPTW\textsuperscript{i}A. By using again the correspondence between (unambiguous) visibly pushdown automata and (unambiguous) bottom-up tree automata, one can simulate their look-arounds. Since DPTW\textsuperscript{i}A have the ability to push stack symbols in both directions (first-child or second-child), it is not difficult to construct a DPTW\textsuperscript{i}A that simulates a D2VPT(LA)\textsuperscript{A}. As a matter of fact, pushing symbols when moving to the second-child is not necessary to simulate D2VPT(LA)\textsuperscript{A}; indeed, a second-child in a fcns encoding correspond to a next-sibling in the nested word, and D2VPT(LA)\textsuperscript{A} do not use their stack for processing symbols that are at the same depth (they do not push “horizontally”).

\section{A.4 Inclusion into streaming transducers and hedge-to-string transducers}

\textbf{Theorem 9.} D2VPT \(\subseteq\) STST and D2VPT \(\subseteq\) dH2S\textsuperscript{LA}

\textbf{Proof.} The proofs of these two inclusions share a same intermediate formal description of transformation. It turns out that this representation will be an extension of the finite transition algebra \(\mathcal{T}_A\) for some D2VPA \(A\).

We recall that elements from the algebra \(\mathcal{T}_A\) are binary relations over \(Q \times \mathbb{D}\) where \(Q\) is the set of states of \(A\) and can thus be depicted as Boolean square matrices \(M_{\mathcal{T}_A}\) over \(Q \times \mathbb{D}\). Hence, the morphism \(\mu_{\mathcal{T}_A}\) associates with each word \(w\) from \(\mathcal{N}(\Sigma)\) a matrix from \(M_{\mathcal{T}_A}\) such that \(\mu_{\mathcal{T}_A}(w)(p_1, d_1), (p_2, d_2)\) is true if there exists a run on \(w\) from \((p_1, d_1)\) to \((p_2, d_2)\) in \(A\).

One may extend this notion to transducers as follows. For a D2VPT \(A\), we consider square matrices \(\mathcal{N}_A\) over \(Q \times \mathbb{D}\) whose
values range over subsets of $\Delta^*$. One can define a mapping $\mu$ from $\mathcal{N}(\Sigma)$ to $\mathcal{N}_A$ such that for all words $w$ from $\mathcal{N}(\Sigma)$, $\mu(w)$ is a matrix $N_A$ satisfying that $N_A((p_1, d_1), (p_2, d_2))$ is equal to $L$ if for each $v$ in $L$, there exists a run on $w$ from $(p_1, d_1)$ to $(p_2, d_2)$ in $A$ producing $v$. Note in fact that $A$ being deterministic, $L$ is either a singleton or the empty set. One can actually prove that one can define an (infinity) algebra $\mathbb{A}_\Sigma = (\mathcal{N}_A, \delta_{\mathbb{A}}, \langle f, r \rangle_{(c, r) \in \Sigma})$ such that $\mathbb{A}_\Sigma$ is associative and $\delta_{\mathbb{A}}$ is its neutral element. Moreover, the considered mapping $\mu$ turns out to $\mu^{A}_\Sigma$ be the canonical morphism from $\mathcal{W}$ to $\mathbb{A}_\Sigma$. It is worth noticing that for all $(p, d), (p', d')$, $\mu^{A}_\Sigma(w)((p, d), (p', d'))$ is false if $\mu^{A}_\Sigma(w)((p, d), (p', d')) \neq \emptyset$.

The operations $\delta_{\mathbb{A}}^{A}$, $\delta_{\mathbb{A}}$ and $\cap^{A}$ can be represented as matrices as well. To do so, let us first consider the two sets of symbols $\Xi = \{x_{(p, d), (p', d')} | (p, d), (p', d') \in Q \times D\}$ for $\alpha \in \{1, 2\}$.

Then, let $\mathcal{N}_A$ be the set of matrices defined over $Q \times D$ such that for each $\mathcal{N}_A$ in $\mathcal{N}_A$, for all $(p_1, d_1), (p_2, d_2), \mathcal{N}_A((p_1, d_1), (p_2, d_2))$ is either the empty set $\emptyset$ or a singleton set included into the set of words $(\Delta \cup \Xi \cup \Xi^*)$. Moreover, for $\delta_{\mathbb{A}}^{A}$, the matrix is precisely the one with $(\epsilon)$ on its main diagonal and $\emptyset$ everywhere else. For $\cap^{A}$, the elements of the matrix are actually included into $(\Delta \cup \Xi)$.

The operation $\delta_{\mathbb{A}}^{A}$ deals with two matrices $\mathcal{N}_A$ and $\mathcal{N}_A$ and produces the matrix $\mathcal{N}_A$ satisfying that for all $(p_1, d_1), (p_2, d_2), \mathcal{N}_A((p_1, d_1), (p_2, d_2))$ is obtained from $\mathcal{N}_A$’’ the matrix of $\delta_{\mathbb{A}}^{A}$ by replacing everywhere in $\mathcal{N}_A$’’ $(p_1, d_1), (p_2, d_2)$ the symbol $x_{(p, d), (p', d')}$ by $\mathcal{N}_A((p, d), (p', d'))$ for $\alpha \in \{1, 2\}$. The application for $f_{c,r}^{A}$ represented by some matrix $N^{A}_{c,r}$ is similar with a single matrix as operand.

The matrices $\mathcal{N}_A$ and $\mathcal{N}_A^{c,r}$ can be defined by means of expressions similar to the ones defining recursively the equivalence classes of traversals. Hence, these matrices are defined by means of unions, concatenations and Kleene star entrywise; the fact that entries of the matrices contain at most singletons and that Kleene star can be expressed as finite concatenations relies on the determinism of $A$.

From the infinite algebra $\mathbb{A}_\Sigma$ and more specifically the matrices representation of the operators $\delta_{\mathbb{A}}^{A}$ and $\cap^{A}$ from this algebra, we are going now to build a streaming tree-to-string transducer on the one side and a dHS2S on the other side.

For streaming tree-to-string transducers, the idea is to define from a D2VP $A$ such a machine $S_{A}$ to simulate the computation of $f_{c,r}^{A}(w)$ for any word $w$ or more precisely to compute the value associated to $((q_1, \rightarrow), (q_2, \rightarrow))$ in this matrix, $q_1$ being the initial state of the D2VP $A$ and $q_2$ the (final) state reached by $A$ after reading $w$ from $q_1$.

We recall that we can define from $A$ the finite algebra $\mathbb{T}_A$ whose domain is $\text{Trav}_A$ and we consider $\Xi = \{x_{(p, d), (p', d')} | (p, d), (p', d') \in Q \times D\}$ for $\alpha \in \{1, 2\}$.

For a D2VP $A$, the STST $S_{A}$ is defined by $\langle \text{Trav}_A, \mathbb{T}_A, \Sigma_{c,r} \times \text{Trav}_A, \Xi, \delta_{S_{A}}^{A}, \mu_{S_{A}}^{A} \rangle$ where

for $\delta_{S_{A}}^{A}$:

- $\delta_{S_{A}}^{A}(m^{A}, c, (e, m^{A}), \nu_{td})$ where

Let us consider the case of deterministic hedge-to-string transducer. We first define the bottom-up deterministic look-ahead automaton $B_{A}$ as $\langle \text{Trav}_A, \langle \epsilon \rangle_{A}, \delta_{B_{A}} \rangle$ where $\delta_{B_{A}}$ is the set of rules of the form $\{(m^{T_A}, c, r, m^{T_A}, m^{T_A}) \in \text{Trav}_A\}$.

Now, we define the $\text{dHS}_2 S_{H_A}$ as follows: the set of states $Q_{H_A}$ is $\{q_1\} \cup \text{Trav}_A \times (Q \times D)$, $q_1$ is the initial state, and the set of final states is $Q_{H_A}$. Now, for the transition function, we define

$$
\delta(q_1, c, r, n_{A}^{T_A}, n_{A}^{T_A}) =
\omega(x_{(q_1, d_1), (q_2, d_2)} \leftarrow (n_{A}^{(q_1, (d_1), (q_2, d_2))), (x_0))$$

$$
\delta((m^{T_A}, ((p, d), (p', d'))), c, r, m^{T_A}) =
\omega(x_{(q_1, d_1), (q_2, d_2)} \leftarrow (m^{T_A}_{\epsilon}, (((p, d), (q_2, d))), (x_0))$$

where

- for any $n_{A}^{T_A}$, $n_{A}^{T_A}$ such that $(f_{c,r}^{A}(n_{A}^{T_A}), m^{T_A}m^{T_A}) = m^{T_A}$ and $(p, d), (p', d') \in m^{T_A}$ and

- $\omega$ is equal to the word $N((p, d), (p', d'))$, $N$ being the matrix $f_{c,r}^{A}(N_{A}^{2}), N_{A}^{2}$ such that the matrix $N_{A}^{2}$ satisfies for all $\alpha \in \{1, 2\}$, for all $(p_1, d_1), (p_2, d_2)$. $N_{A}^{2}(1, 2), (p_2, d_2) = (x_{(p, d), (p', d')} \leftarrow (p_{1}, d_1), (p_2, d_2))$.

Let us prove now that the inclusions are strict. The transformation serving as a counter-example is the same for the two cases: we consider a transformation $T$ over the input alphabet $\Sigma = \{c\}$ and $\Sigma_r = \{r\}$ and the output alphabet $\{a\}$. This transformation takes as an input words such as $(c)^n$ for any natural $n$ and outputs $c^{n+1}$. $T$ is given by the dHS2S with a single state $q$ and with a universal look-ahead automaton with $q'$ as unique state by

$\delta(q, c, r, q') = aq(x)(q(x))$

The transformation $T$ can also be defined by a STST with a single state $q$, a unique register variable $X$ and a unique stack symbol $\gamma$. The transitions are given by

$\delta_{T_{PUSH}}(q, c) = (q, q, \{X \mapsto aX\})$

$\delta_{T_{POP}}(q, r, \gamma) = (q, \{X \mapsto X\})$

The transformation $T$ cannot be realized by some D2VP; indeed, for such a machine with stack alphabet $\Gamma$ on the shallow inputs of the domain, the possible stacks occurring in runs are either $\perp$ or the form $\gamma$ for $\gamma \in \Gamma$. Hence, the possible behaviours of such D2VP are similar to the ones of a deterministic finite state transducer. It is known that deterministic finite state transducers realize only functions that are linear-size increase; this is not the case of the transformation $T$.

A.5 Unranked Tree Walking Transducers

Unranked Trees

Let $\Lambda$ be a finite set of symbols. Unranked trees $t$ over $\Lambda$ are defined inductively as $t ::= a | \{a_1, \ldots, a_n\}$, for all $a \in \Lambda$, all $n \geq 1$. Unranked trees over $\Lambda$ can be identified (modulo renaming of nodes) with structures over the signature $\mathcal{U}_\Lambda$ that consists of the first-child predicate $f(x, y)$ that relates a node $x$ to its first-child $y$, the next-sibling predicate $n(x, y)$ that relates a node $x$ to its next-sibling $y$ in a sequence of unranked trees, and $a(x)$, for all $a \in \Lambda$, that holds true in node $x$ if it is labeled $a$. In addition, we also add a parent predicate $\text{parent}(x, y)$ that relates a node to its parent.

For instance, the unranked tree $a(b, c(a), c)$ is identified with the structure whose set of nodes is $\{e, 1, 2, 3, 21\}$, where the first-child predicate is $\{(e, 1), (2, 21)\}$, the next-sibling predicate is $\{(1, 2), (2, 3)\}$, the a predicate is $\{(e, 21), b\}$ the b predicate is $\{1\}$ and the c predicate is $\{2, 3\}$. The parent predicate is given by $\{(1, e), (2, e), (3, e), (21, 2)\}$.
Unranked Tree Walking Transducers They are defined similarly as ranked tree walking transducers, except that they move along the next-sibling and first-child predicates. They are equipped with a (visibly) pushdown store such that whenever they go down the first-child, they have to push some symbol, whenever they go up to the parent of a node, they have pop one symbol from the stack. However, when they move horizontally along next-sibling predicates, they do not touch the stack. Before applying a transition, they can test whether the current node is the root, is the first-child of some node, the last-child, or a leaf. Their move have to be consistent with the result of such a test. They are also equipped with stay moves that stay at the same tree node.

Formally, a deterministic pushdown unranked tree word walking transducer (DPTnWT) from unranked trees over Λ to Σ' is a tuple T = (Q, δ₀, q₁, Γ, γ₀, R) where Q is a finite set of states, q₀ is the initial state, q₁ the stopping state, Γ is a finite stack alphabet with initial symbol γ₀, R is a transition function such that

\[ R : Q \times Λ \times Γ \times \{ \{0,1\} \} \rightarrow \Sigma' \times (\{q₁\} \cup Γ \times \{\}, \{→, ↑, ←, ↓, \} \}) \]

A configuration of T on a tree t with set of nodes N_t is a triple \((q, n, \beta, u) \in Q \times N_t \times Γ^* \times Σ'\). We define → a binary relation between consecutive configurations as follows: Let \(n \in N_t\) labeled \(a \in Λ\). Let \(b = (b₁, b₂, b₃, b₄) \in \{0,1\}^4\) such that \(b₄ = 1\) if \(n\) is a first-child, \(b₁ = 1\) if \(n\) is a last-child, \(b₂ = 1\) if \(n\) is the root, \(b₃ = 1\) if \(n\) is a leaf. Then, for all \(q, q₁, q₂ \in Q\), all \(n, n₁, n₂ \in Ν_t\), all \(β, β₁, β₂, β₃ \in Γ^*\), all \(γ, γ₁ \in Γ\), all \(u, v \in Σ'\), \(q, n, β, u \rightarrow (q₁, n₁, β₁, v₁)\)

- **(stopping move)** \(R(q, a, γ, b) = (v, q, n, γ₁ = q, n₁ = n, β₁ = βγ', or)
- **(downward move)** \(R(q, a, γ, b) = (v, γ', q', ↓, β' = βγ', and fc(n, n₁), or,
- **(upward move)** \(R(q, a, γ, b) = (v, q', ↑) and β' = β, and parent(n, n₁), or,
- **(right sibling move)** \(R(q, a, γ, b₁) = (v, q', ←) and β' = β, and ns(n, n₁), or,
- **(left sibling move)** \(R(q, a, γ, b₁) = (v, q', →) and β' = β, and n₁s(n, n₁), or,
- **(stay move)** \(R(q, a, γ, b₁) = (v, q', ⊙) and β' = β, and n = n₁).

A run of T on an unranked tree t is a finite sequence of configurations \(c₀c₁ \ldots cₙ\) such that \(c_i \rightarrow c_{i+1}\) for all \(i = 0, \ldots, m - 1\). It is accepting if \(c_m = (q₀, r, γ₀, t)\), where \(r\) is the root node of t, and \(c_n = (q_n, n, \beta, u)\) for some node n of \(t\), \(β \in Γ^*\), and \(u \in Σ^*\). Since \(R\) is function, there exists at most one accepting run per input tree t, and we call u the output of t. The transduction realized by T is the set of pairs \((t, u)\) such that there exists an accepting run of T on t whose output is u.

Equivalence between D2VPT and DPTnWT Modulo nested word linearisation of unranked trees, the two models are equivalent. Let us briefly sketch why.

Assume T is a DPTnWT and let us construct an equivalent D2VPT \(T'. First notice that when T is positioned at some node n, its stack height is exactly the depth of node n in the tree, as well as the depth of the call and returns symbols corresponding to n in the linearisation. Also note that T can always read the top symbol of the stack, while \(T'\) only reads it when it pops a symbol. This issue can be overcome by always keeping in the state of \(T'\) the top stack symbol. It remains to see how \(T'\) can simulate the moves of T and its tests (root, leaf, etc.). We assume that if T is positioned at some tree node n, then \(T'\) is positioned at the call position \(c_n\) corresponding to n in the linearisation of the input tree. Then, if T moves from n to its next-sibling \(n₁\), \(T'\) has to traverse the whole linearisation of the subtree rooted at n. It can be easily done by pushing a special symbol when reading \(c_n\), forward, which is popped once the matching return position of \(c_n\) is met. Simulating previous-sibling moves is done symmetrically. Suppose now that a tree node \(n'\) is the parent of a tree node n. In the linearisation, it means that there is a (sub) nested word of the form \(c_n = w₁Uc_nw₂Ucₙ′w₃\), where \(w₁, w₂, w₃\) are nested words. To simulate a move of \(T'\) from n to \(n₁, T'\) has to move backward from \(c_n\) to \(c_n, traverse\(ng\) w₁. Again, by using a special stack symbol when traversing \(w₁\), \(T'\) can detect when it reads \(c_n\): It is the first time it does not popped the special stack symbol. To simulate a stay move, \(T'\) just move one-step forward and one-step backward.

Finally, we have to show how \(T'\) can simulate the tests (root, leaf, etc.). By using a special bottom stack symbol, \(T'\) can know when it is at the root. The other tests can easily be performed by \(T'\): For instance, to detect that \(T'\) is positioned at a call position that corresponds to a first-child, it suffices to go one step backward and check whether the previous symbol is call.

Conversely, let T be a D2VPT whose input are assumed to be linearisations of unranked trees. To construct an equivalent DPTnWT \(T'\), one again has to show how the moves of T are simulated by moves of \(T'\).

If T moves forward by reading a call symbol \(c_n\), then its next position can be either that of a call symbol \(c_n\) (which means that \(n₁\) is the first-child of \(n\)), or that of return symbol \(r_n\) (which means that \(n\) is a leaf). Using a test, T can decide whether it is at a leaf or not. In the first case, it uses a stay transition and in the second case, it uses a first-child transition.

Other cases are treated similarly: For instance, if T moves forward by reading a return symbol \(r_n\), then if the next symbol is a call symbol \(c_n\), it means that \(n₁\) is the next-sibling of \(n\), and if the next symbol is a return symbol \(r_n\), it means that \(n₁\) is a parent of \(n\). Using tests, T can decide what moves to perform, either next-sibling or parent.