

STREAMABILITY OF NESTED WORD TRANSDUCTIONS

EMMANUEL FILIOT^a, OLIVIER GAUWIN^b,
PIERRE-ALAIN REYNIER^c, AND FRÉDÉRIC SERVAIS^d

^a Université Libre de Bruxelles

^b LaBRI, CNRS, Université de Bordeaux

^c Aix-Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

^d Ecole Supérieure d'Informatique de Bruxelles

ABSTRACT. We consider the problem of evaluating in streaming (i.e., in a single left-to-right pass) a nested word transduction with a limited amount of memory. A transduction T is said to be height bounded memory (HBM) if it can be evaluated with a memory that depends only on the size of T and on the height of the input word. We show that it is decidable in coNPTIME for a nested word transduction defined by a visibly pushdown transducer (VPT), if it is HBM. In this case, the required amount of memory may depend exponentially on the height of the word. We exhibit a sufficient, decidable condition for a VPT to be evaluated with a memory that depends quadratically on the height of the word. This condition defines a class of transductions that strictly contains all determinizable VPTs.

INTRODUCTION

Memory analysis is an important tool for ensuring system robustness. In this paper we focus on the analysis of programs processing *nested words* [AM09], *i.e.*, words with a recursive structure, like program traces, XML documents, or more generally unranked trees. On huge inputs, a *streaming* mode is often used, where the nested word is read only once, from left to right. This corresponds to a depth-first left-to-right traversal when the nested word is considered as a tree. For such programs, *dynamic analysis* problems have been addressed in various contexts. For instance, runtime verification detects dynamically, and as early as possible, whether a property is satisfied by a program trace [KV01, BLS11]. On XML streams, some algorithms outputting nodes selected by an XPath expression at the earliest

Key words and phrases: visibly pushdown transducers, streaming, nested words, online algorithms.

A preliminary version of this work has been presented at the FSTTCS'11 conference [FGRS11]. Links with this version are discussed in the introduction.

Partially supported by the ESF project GASICS, by the FNRS, by the PAI program Moves funded by the Federal Belgian Government, the ANR projects ExStream (ANR-13-JS02-0010-01) and DeLTA (ANR-16-CE40-0007), and the FET project FOX (FP7-ICT-233599).

possible event have also been proposed [BJ07, GNT09]. These algorithms allow minimal buffering [BYFJ05].

In this paper, we investigate *static analysis* of memory usage for a special kind of programs on nested words, namely programs defined by *transducers*. We assume that the transducers are *functional* and *non-deterministic*. Non-determinism is required as input words are read from left to right in a single pass and some actions may depend on the future of the stream. For instance, the XML transformation language XSLT [Cla99] uses XPath for selecting nodes where local transformations are applied, and XPath queries relies on non-deterministic moves along tree axes, such as a move to any descendant. We require our transducers to be *functional*, as we are mainly interested by transformation languages like XSLT [Cla99], XQuery [BCF⁺07] and XQuery Update Facility, [RCD⁺], for which any transformation maps each XML input document to a unique output document.

Visibly pushdown transducers (VPTs) form a subclass of pushdown transducers adequate for dealing with nested words and streaming evaluation, as the input nested word is processed from left to right. They are visibly pushdown automata [AM09] extended with arbitrary output words on transitions. VPTs capture interesting fragments of the aforementioned XML transformation languages that are amenable to efficient streaming evaluation, such as all editing operations (insertion, deletion, and relabeling of nodes, as used for instance in XQuery Update Facility [RCD⁺]) under all regular tests. Like for visibly pushdown automata, the stack behavior of VPTs is imposed by the type of symbols read by the transducer. Those restrictions on stack operations allow to decide functionality and equivalence of functional VPTs in PTIME and EXPTIME respectively [FRR⁺18].

Some transductions defined by (functional and non-deterministic) VPTs cannot be evaluated efficiently in streaming. For instance, swapping the first and last letter of a word can be defined by a VPT as follows: guess the last letter and transform the first letter into the guessed last letter, keep the value of the first letter in the state, and transform any value in the middle into itself. Any deterministic machine implementing this transformation requires to keep the entire word in memory until the last letter is read. It is not reasonable in practice as for instance XML documents can be very huge.

Our aim is thus to identify decidable classes of transductions for various memory requirements that are suitable to space-efficient streaming evaluation. We first consider the requirement that a transducer can be implemented by a program using a *bounded memory* (BM), *i.e.* computing the output word using a memory independent of the size of the input word. However when dealing with nested words in a streaming setting, the bounded memory requirement is quite restrictive. Indeed, even performing such a basic task as checking that a word is well-nested or checking that a nested word belongs to a regular language of nested words requires a memory dependent on the height (the level of nesting) of the input word [SS07]. This observation leads us to the second question: decide, given a transducer, whether the transduction can be evaluated with a memory that depends only on the size of the transducer and the height of the word (but not on its length). In that case, we say that the transduction is *height bounded memory* (HBM). This is particularly relevant to XML transformations as XML documents can be very long but have usually a small depth [BMV06]. HBM does not specify *how* memory depends on the height. A stronger requirement is thus to consider HBM transductions whose evaluation can be done with a memory that depends *polynomially* on the height of the input word.

Contributions. First, we give a general space-efficient evaluation algorithm for functional VPTs. After reading a prefix of an input word, the number of configurations of the (non-deterministic) transducer as well as the number of output candidates to be kept in memory may be exponential in the size of the transducer and the height of the input word (but not in its length). Our algorithm produces as output the longest common prefix of all output candidates, and relies on a compact representation of sets of configurations and remaining output candidates (the original output word without the longest common prefix). We prove that it uses a memory linear in the height of the input word, and linear in the maximal length of remaining output candidates.

We prove that BM is equivalent to sequentializability for finite state transducers (FSTs), which is known to be decidable in PTIME. BM is however undecidable for arbitrary pushdown transducers but we show that it is decidable for VPTs in CONPTIME.

Like BM, HBM is undecidable for arbitrary pushdown transductions. We show, via a non-trivial reduction to the emptiness of pushdown automata with bounded reversal counters, that it is decidable in CONPTIME for transductions defined by VPTs. In particular, we show that the previously defined algorithm runs in HBM iff the VPT satisfies some property, which is an extension of the so called *twinning property* for FSTs [Cho77] to nested words. We call it the *horizontal twinning property*, as it only cares about configurations of the transducers with stack contents of identical height. This property only depends on the transduction, *i.e.* is preserved by equivalent transducers.

When a VPT-transduction is height bounded memory, the memory needed may be exponential in the height of the word. We introduce a stronger notion of height bounded memory, called *online bounded memory* (OBM). Roughly, an algorithm is OBM if the amount of memory it uses after reading a prefix u of the input nested word only depends on the “current” height of u , *i.e.*, the height of the stack a visibly pushdown machine would be in after reading u . For instance, *ccrrc* has current height 1 but height 2 (where c is a call symbol and r a return symbol). We refine the horizontal twinning property into a so called *matched twinning property*, which we prove to effectively characterize the class of all VPTs which can be evaluated in OBM, and to be decidable in CONPTIME. We call such class the class of *twinned VPTs*. We prove that twinned VPTs only require a quadratic (in the current height) amount of memory to be evaluated. It is simple to see that any sequentializable VPT can be evaluated in OBM (and thus is twinned). However, we show that some non-sequentializable VPT are twinned, in a way making twinned VPT the right class of VPT when it comes to efficient streaming evaluation. Let us mention that the decidability status of the class of sequentializable VPT is open.

Related Work. In the XML context, visibly pushdown automata based streaming processing has been extensively studied for validating XML streams [KMV07, BLS06, SS07]. The validation problem with bounded memory is studied in [BLS06] when the input is assumed to be a well-nested word and in [SS07] when it is assumed to be a well-formed XML document (this problem is still open). Querying XML streams has been considered in [GKS07]. It consists in selecting a set of tuples of nodes in the tree representation of the XML document. For monadic queries (selecting nodes instead of tuples), this can be achieved by a functional VPT returning the input stream of tags, annotated with Booleans indicating selection by the query. However, functional VPTs cannot encode queries of arbitrary arities. The setting for functional VPTs is in fact different to query evaluation, because the output has to be produced on-the-fly in the right order, while query evaluation algorithms can output nodes

in any order: an incoming input symbol can be immediately output, while another candidate is still to be confirmed. This makes a difference with the notion of concurrency of queries, measuring the minimal amount of candidates to be stored, and for which algorithms and lower bounds have been proposed [BYFJ05]. VPTs also relate to tree transducers [FRR⁺18], for which no comparable work on memory requirements is known. However, the height of the input word is known to be a lower bound for Core XPath filters [GKS07]. As VPTs can express them, this lower bound also applies when evaluating VPTs. When allowing two-way access on the input stream, space-efficient algorithms for XML validation [KM13] and querying [MV09] have been proposed. Approximate space-efficient streaming validation algorithms of nested word properties, given as visibly pushdown automata, have been considered in [FMdRS16]. Finally, another related problem is the sliding window validation problem [GJL18, GHL18]: in this context, a window scans the input and each window must satisfy some property, and the goal is to use as little memory as possible.

An approach based on weighted automata for the analysis of online algorithms has been proposed in [AKL10]. In this work, the existence of online algorithms is related to determinism and look-ahead removal. The analysis boils down to checking, given a weighted automaton, whether it can be determinized or approximatively determinized into some automaton, homomorphically embeddable into the original one. While this problem could be adapted in our context and is an interesting question, we did not take determinization as the yardstick notion of streamability because, as we show, for programs transforming nested words, deterministic VPTs are too restrictive to capture all streamable VPT transformations.

Differences with conference version. This version improves the results of the conference version [FGRS11] both by proving stronger results, and by simplifying proofs. Perhaps the strongest improvement is the introduction of the class OBM, characterized by the matched twinning property (MTP). The MTP was already introduced in [FGRS11], but it was only shown to be a sufficient condition for a VPT to admit polynomially height bounded evaluation. The main technical result of [FGRS11], based on heavy arguments of word combinatorics, was to show that the MTP satisfaction is invariant under equivalent VPT, making MTP a proper class of transductions rather than just a class of transducers. In this journal version, we show that this class of transductions corresponds to the class OBM, giving a full characterization in terms of memory requirements. The word combinatorics arguments have been greatly simplified, thanks to a recent result by Saarela [Saa15] about systems of word equations. The proof of Saarela’s result is done in a very elegant way that even completely avoids word combinatorics, by embedding words into polynomials.

1. VISIBLY PUSHDOWN LANGUAGES AND TRANSDUCTIONS

Words and nested words. We consider a finite alphabet Σ partitioned into three disjoint sets Σ_c , Σ_r and Σ_l , denoting respectively the *call*, *return* and *internal* alphabets. We denote by Σ^* the set of (finite) words over Σ and by ϵ the empty word. The length of a word u is denoted by $|u|$. Given $1 \leq i \leq |u|$, $u[i]$ denotes the i -th letter of u . For all words $u, v \in \Sigma^*$, we denote by $u \wedge v$ the longest common prefix of u and v . More generally, for any non-empty finite set of words $V \subseteq \Sigma^*$, the longest common prefix of V , denoted by $\text{lcp}(V)$, is inductively defined by $\text{lcp}(\{u\}) = u$ and $\text{lcp}(V \cup \{u\}) = \text{lcp}(V) \wedge u$. We call v a *factor* of u whenever there exist words v' and v'' such that $u = v'vv''$. The set of *well-nested*

words Σ_{wn}^* is the smallest subset of Σ^* such that $\Sigma_l^* \subseteq \Sigma_{\text{wn}}^*$ and for all $c \in \Sigma_c$, all $r \in \Sigma_r$, all $u, v \in \Sigma_{\text{wn}}^*$, $cur \in \Sigma_{\text{wn}}^*$ and $uv \in \Sigma_{\text{wn}}^*$. Let $u = \alpha_1 \dots \alpha_n \in \Sigma^*$ be a prefix of a well-nested word. We define the *current height* of u as the number of pending calls: $\text{hc}(u) = 0$ if u is well-nested, and $\text{hc}(ucv) = \text{hc}(u) + 1$ if $c \in \Sigma_c$ and v is well-nested. The *height* of u is the maximal number of pending calls on any prefix of u , i.e., $\text{h}(u) = \max_{1 \leq i \leq n} \text{hc}(\alpha_1 \dots \alpha_i)$. For instance, if c is a call and r a return symbol, then we have $\text{h}(crcrcc) = \text{h}(ccrcrr) = 2$, while $\text{hc}(crcrcc) = 2$ and $\text{hc}(ccrcrr) = 0$. In particular, for well-nested words, the height corresponds to the usual height of the nesting structure of the word.

Given two words $u, v \in \Sigma^*$, the *delay* of u and v , denoted by $\Delta(u, v)$, is the unique pair of words (u', v') such that $u = (u \wedge v)u'$ and $v = (u \wedge v)v'$. For instance, $\Delta(abc, abde) = (c, de)$. Informally, in a word transduction, if there are two output candidates u and v during the evaluation, we are sure that we can output $u \wedge v$ and $\Delta(u, v)$ is the remaining suffixes we still keep in memory. We extend the concatenation to pairs of words and denote it by \cdot , i.e. $(u, v) \cdot (u', v') = (uu', vv')$. We will use the following property of delays (Lemma 5 in [BCPS03]).

Lemma 1.1. *For all $u, u', v, v' \in \Sigma^*$, $\Delta(uu', vv') = \Delta(\Delta(u, v) \cdot (u', v'))$.*

A *transduction* is a binary relation $R \subseteq \Sigma^* \times \Sigma^*$. For any input word $u \in \Sigma^*$, we denote by $R(u)$ the set $\{v \mid (u, v) \in R\}$. A transduction R is *functional* if for all $u \in \Sigma^*$, $R(u)$ has size at most one. If R is functional, we identify $R(u)$ with the unique image of u if it exists.

Visibly pushdown transducers (VPTs). As finite-state transducers extend finite-state automata with outputs, visibly pushdown transducers extend visibly pushdown automata [AM09] with outputs [FRR⁺18]. To simplify notations, we suppose that the output alphabet is Σ , but our results still hold for an arbitrary output alphabet. Informally, the stack behavior of a VPT is similar to that of visibly pushdown automata. On a call symbol, the VPT pushes a symbol on the stack and produces some output word (possibly empty), on a return symbol, it must pop the top symbol of the stack and produce some output word (possibly empty) and on an internal symbol, the stack remains unchanged and it produces some output word. We do not require the output of a VPT to be well-nested. This is not a restriction but a more general setting as well nestedness in the output can be enforced on the VPT. However, this more general setting comes for free as our proofs would be the same.

Definition 1.2. A *visibly pushdown transducer* (VPT) on finite words over Σ is a tuple $T = (Q, I, F, \Gamma, \delta)$ where Q is a finite set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ the set of final states, Γ is the stack alphabet, $\delta = \delta_c \uplus \delta_r \uplus \delta_l$ the (finite) transition relation, with $\delta_c \subseteq Q \times \Sigma_c \times \Sigma^* \times \Gamma \times Q$, $\delta_r \subseteq Q \times \Sigma_r \times \Sigma^* \times \Gamma \times Q$, and $\delta_l \subseteq Q \times \Sigma_l \times \Sigma^* \times Q$.

A *configuration* of a VPT is a pair $(q, \sigma) \in Q \times \Gamma^*$. A *run* of T on a word $u = a_1 \dots a_l \in \Sigma^*$ from a configuration (q, σ) to a configuration (q', σ') is a finite sequence $\rho = \{(q_k, \sigma_k)\}_{0 \leq k \leq l}$ such that $q_0 = q$, $\sigma_0 = \sigma$, $q_l = q'$, $\sigma_l = \sigma'$ and for each $1 \leq k \leq l$, there exist $v_k \in \Sigma^*$ and $\gamma_k \in \Gamma$ such that either $(q_{k-1}, a_k, v_k, \gamma_k, q_k) \in \delta_c$ and $\sigma_k = \sigma_{k-1}\gamma_k$ or $(q_{k-1}, a_k, v_k, \gamma_k, q_k) \in \delta_r$ and $\sigma_{k-1} = \sigma_k\gamma_k$, or $(q_{k-1}, a_k, v_k, q_k) \in \delta_l$ and $\sigma_k = \sigma_{k-1}$. The word $v = v_1 \dots v_l$ is called an *output* of ρ . We write $(q, \sigma) \xrightarrow{u/v} (q', \sigma')$ when there exists a run on u from (q, σ) to (q', σ') producing v as output. We denote by \perp the empty word on Γ . A configuration (q, σ) is *accessible* (resp. is *co-accessible*) if there exist $u, v \in \Sigma^*$ and $q_0 \in I$ (resp. $q_f \in F$) such that $(q_0, \perp) \xrightarrow{u/v} (q, \sigma)$ (resp. such that $(q, \sigma) \xrightarrow{u/v} (q_f, \perp)$).

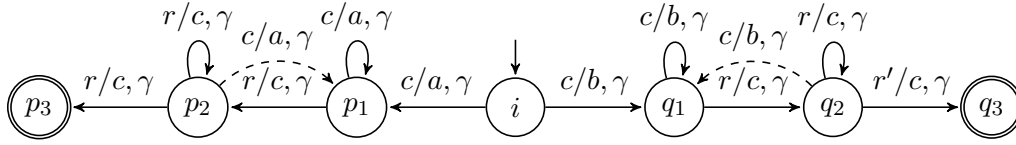


Figure 1: A functional VPT with $\Sigma_c = \{c\}$, $\Sigma_r = \{r, r'\}$ and $\Sigma_l = \{a, b\}$

A transducer T defines a transduction

$$\llbracket T \rrbracket = \{(u, v) \in \Sigma^* \times \Sigma^* \mid \exists q \in I, q' \in F, (q, \perp) \xrightarrow{u/v} (q', \perp)\}.$$

We say that a transduction R is a VPT-transduction if there exists a VPT T such that $R = \llbracket T \rrbracket$. We denote by $T(u)$ the set $\llbracket T \rrbracket(u)$.

Two transducers T_1, T_2 are said to be *equivalent* if $\llbracket T_1 \rrbracket = \llbracket T_2 \rrbracket$. A transducer T is *reduced* if every accessible configuration is co-accessible. Given any VPT, computing an equivalent reduced VPT can be performed in polynomial time [CRT15]. A VPT T is *functional* if $\llbracket T \rrbracket$ is functional, and this can be decided in PTIME [FRR⁺18]. The class of functional VPTs is denoted by fVPT. The *domain* of T (denoted by $Dom(T)$) is the domain of $\llbracket T \rrbracket$. The domain of T contains only well-nested words, which is not necessarily the case of the codomain.

Example 1.3. Consider the VPT T of Fig. 1 represented in plain arrows. The left and right parts accept the same input words except for the last letter of the word. The domain of T is $Dom(T) = \{c^n r^n \mid n \geq 2\} \cup \{cc^n r^n r' \mid n \geq 1\}$. Any word $c^n r^n$ is translated into $a^n c^n$, and any word $cc^n r^n r'$ is translated into $b^{n+1} c^{n+1}$. Therefore the translation of the first sequence of calls depends on the last letter r or r' . As we will see later, this transduction cannot be evaluated with a bounded amount of memory, but with a memory which depends on the height n of the input word.

Finite state transducers (FSTs). A *finite state transducer* (FST) on an alphabet Σ is a tuple (Q, I, F, δ) where Q is a finite set, $I, F \subseteq Q$ and $\delta \subseteq Q \times \Sigma \times \Sigma^* \times Q$ with the standard semantics. This definition corresponds to the usual definition of *real-time FSTs* [Sak09], as there is no ϵ -transitions. We always consider real-time FSTs in this paper, so we just call them FSTs.

Sequential transducers. The *underlying automaton* of a given FST (resp. VPT) is the automaton obtained by ignoring the output. A *sequential FST* (resp. VPT) is a pair (T, Ψ) where T is an FST (resp. VPT) whose underlying automaton is deterministic, and $\Psi : F \rightarrow \Sigma^*$ is a mapping that associates a word with each final state. The output of an input word u by (T, Ψ) is the word $v.\Psi(q)$ if the unique run of T on u produces v and ends in some accepting state q .

VPTs on words of bounded height. Given a natural number $k \in \mathbb{N}$ and a VPT T , one can define an FST, denoted by $FST(T, k)$, which is the restriction of T to input words of height less than k . The transducer $FST(T, k)$ is naturally constructed by taking as states the configurations (q, σ) of T such that $|\sigma| \leq k$. In particular, its initial (resp. final) states are the pairs (q, \perp) where q is initial (resp. final), and there is a transition in $FST(T, k)$ from

state (q, σ) to state (p, γ) on reading $u \in \Sigma$, producing $v \in \Sigma^*$, if there is a (single) transition in T from the configuration (q, σ) to the configuration (p, γ) on input u and output v .

Turing Transducers. In order to formally define the complexity classes for evaluation that we target, we introduce a *deterministic* computational model for word transductions that we call *Turing Transducers*. Turing transducers, a special case of Turing machines, have three tapes: one read-only left-to-right input tape over some alphabet Σ , one write-only left-to-right output tape over Σ , and one standard working tape over some alphabet Σ' . Their transitions are assumed to be deterministic, to model deterministic algorithms. They have accepting and rejecting states. A functional transduction $f : \Sigma^* \rightarrow \Sigma^*$ is *computable* by a Turing transducer T if for all $u \in \text{Dom}(f)$, the machine T halts in some accepting state and the content of the output tape is $f(u)$, and for all $u \notin \text{Dom}(f)$, the machine halts in some rejecting state. The space complexity of a Turing transducer is measured on the working tape only.

2. BOUNDED MEMORY EVALUATION

In this section, we consider the class of transductions that can be evaluated with a constant amount of memory if we fix the machine that defines the transduction, and the problem of deciding whether a transducer (finite-state, pushdown, or visibly pushdown) defines a transduction in this class.

Definition 2.1. A (functional) transduction $f : \Sigma^* \rightarrow \Sigma^*$ is *bounded memory (BM)* if there exists $K \in \mathbb{N}$ such that it is computable by a Turing transducer M that runs in space complexity at most K .

Example 2.2. Let $\Sigma = \{a, b\}$ be an alphabet and let f be the transduction that maps any word of the form $w\sigma$, for $\sigma \in \Sigma$ and $w \in \Sigma^*$, to σw . Clearly, f is not BM: any Turing transducer that computes this transduction, since it reads the input from left-to-right, and produces the output from left-to-right, must wait until the last letter of the word before outputting anything, and therefore has to store on the working tape the word w .

As a positive example, any function f on a finite domain D is BM, by taking K as the length of the longest output word of f on D . The domain D needs not to be finite in general for f to be BM. Indeed, as we show, all sequential functions are BM, and conversely.

2.1. Finite state transducers. It is not difficult to verify that for FST-transductions, bounded memory is characterized by sequentializability, which is decidable in PTIME. An FST T is sequentializable if there exists a sequential transducer T_d such that $\llbracket T \rrbracket = \llbracket T_d \rrbracket$. Sequentializable transducers have been characterized by a structural property of their runs, called the *twinning property* [Cho77], which is decidable in PTime [WK95, BCPS03]. Intuitively, this property requires that no delay can be accumulated along loops synchronized on the same input.

Definition 2.3 (Twinning property for FSTs). Let $T = (Q, I, F, \delta)$ be a reduced FST. T satisfies the twinning property if for all $q_0, q'_0 \in I$, for all $q, q' \in Q$, for all words $u_1, v_1, w_1, u_2, v_2, w_2 \in \Sigma^*$, if:

$$q_0 \xrightarrow{u_1/v_1} q \xrightarrow{u_2/v_2} q \qquad q'_0 \xrightarrow{u_1/w_1} q' \xrightarrow{u_2/w_2} q'$$

then $\Delta(v_1, w_1) = \Delta(v_1v_2, w_1w_2)$.

Proposition 2.4. *Let T be a functional FST. The following statements are equivalent:*

- (1) $\llbracket T \rrbracket$ is BM
- (2) T is sequentializable
- (3) T satisfies the twinning property.

Moreover, it is decidable in PTIME whether $\llbracket T \rrbracket$ is BM.

Proof. The equivalence between (2) and (3) has been shown in [Cho77]. We show the equivalence between (1) and (2). Clearly, if $\llbracket T \rrbracket$ is definable by a sequential transducer T_d , then evaluating T_d on any input word u can be done with a space complexity that depends on the size of T_d only.

Conversely, if $\llbracket T \rrbracket$ is BM, there exists $K \in \mathbb{N}$ and a Turing transducer M that transforms any input word u into $\llbracket T \rrbracket(u)$ in space complexity K . Any word on the working tape of M is of length at most K . As M is deterministic, we can therefore see M as a sequential FST, whose states are pairs (q, w) where q is a state of T and w a word on the working tape (modulo some elimination of ϵ -transitions).

Since sequentializability is decidable in PTIME, as first shown in [WK95], and later on with a different proof in [BCPS03], the result follows from the equivalence between (1) and (2). \square

2.2. Pushdown transducers. Similarly to finite-state transducers that extend finite-state automata with outputs, pushdown transducers extend pushdown automata with outputs. Bounded Memory is undecidable for pushdown transducers, since it is at least as difficult as deciding whether a (non-deterministic) pushdown automaton defines a regular language (the reduction is immediate).

Proposition 2.5. *It is undecidable whether a functional transduction defined by a (non-deterministic) pushdown transducer is BM.*

Prop.2.5 holds for non-deterministic pushdown transducers. It is open whether it holds too for deterministic pushdown transducers. The same reduction cannot be applied since testing the regularity of the language defined by a deterministic pushdown automaton is decidable [Ste67].

2.3. Visibly pushdown transducers. For VPTs, BM is quite restrictive as it imposes to verify whether a word is well-nested by using a bounded amount of memory. This can be done only if the height of the words of the domain is bounded by some constant which depends on the transducer only:

Proposition 2.6. *Let T be a functional VPT with n states.*

- (1) $\llbracket T \rrbracket$ is BM iff (i) for all $u \in \text{Dom}(T)$, $h(u) \leq n^2$, and (ii) $\llbracket \text{FST}(T, n^2) \rrbracket$ is BM;
- (2) It is decidable in CONPTIME whether $\llbracket T \rrbracket$ is BM.

Proof. If $\llbracket T \rrbracket$ is BM, there exist K and a Turing transducer M computing $\llbracket T \rrbracket$, and such that M evaluates any input word in space at most K . We can easily extract from M a finite automaton that defines $\text{Dom}(T)$, whose number of states m only depends on M and K . By a simple pumping argument, it is easy to show that the words in $\text{Dom}(T)$ have a height bounded by m . If the height of the words in $\text{Dom}(T)$ is bounded, then it is bounded by n^2 .

Indeed, assume that there exists a word $u \in \text{Dom}(T)$ whose height is strictly larger than n^2 . Consider all decompositions of u into nested well-nested factors, i.e., $u = u_1 u_2 u_3 u_4 u_5$ where u_3 and $u_2 u_3 u_4$ are well-nested, and $\text{hc}(u_2) > 0$. As the height of u is strictly larger than n^2 , there exists one of such decompositions for which the states q, p reached respectively before u_2 and after u_4 will repeat around u_3 . In other words at least one run of T on u has the following form:

$$(i, \perp) \xrightarrow{u_1/v_1} (q, \sigma) \xrightarrow{u_2/v_2} (q, \sigma\sigma') \xrightarrow{u_3/v_3} (p, \sigma\sigma') \xrightarrow{u_4/v_4} (p, \sigma) \xrightarrow{u_5/v_5} (f, \perp)$$

with σ' non-empty, and i (resp. f) an initial (resp. final) state of T . Then one can iterate the matching loops around q and p to generate words $u_1 u_2^k u_3 u_4^k u_5$ in $\text{Dom}(T)$ with arbitrarily large heights, yielding a contradiction. Therefore $\text{FST}(T, n^2)$ is equivalent to T . As in the proof of Proposition 2.4, we can consider M as a sequential FST T_M whose set of states are configurations of the machine. The FST T_M is equivalent to T , and therefore to $\text{FST}(T, n^2)$. Since T_M is sequential, $\text{FST}(T, n^2)$ is sequentializable and therefore by Proposition 2.4, $\llbracket \text{FST}(T, n^2) \rrbracket$ is BM. The converse is obvious.

Therefore to check whether $\llbracket T \rrbracket$ is BM, we first decide if the height of all input words accepted by T is less or equal than n^2 . This can be done in PTIME $O(|T| \cdot n^2)$ by checking emptiness of the projection of T on the inputs (this is a visibly pushdown automaton) extended with counters up to $n^2 + 1$ that count the height of the word. One can then construct $\text{FST}(T, n^2)$, resulting in an exponentially larger FST equivalent to T , and check whether $\llbracket \text{FST}(T, n^2) \rrbracket$ is BM using the procedure of Theorem 2.4. The time complexity of the overall algorithm is exponential. However, using results which are proved later to characterise a more general class of transductions (namely the VPT-transductions which can be evaluated with height bounded memory, forming the class called HBM — Definition 4.1), one can lower the complexity to CONPTIME. By definition of HBM, a VPT-transduction whose input words have bounded height (i.e., a height which only depends on the transducer itself) is HBM iff it is BM. It is shown in Theorem 4.7 that HBM can be tested in CONPTIME, yielding the result.

For the sake of completeness, let us give the main arguments to get the CONPTIME bound. We use pushdown counter machines which make a bounded number of reversals (a bounded number of moves from an increasing to a decreasing mode, and from a decreasing to an increasing mode). Such machines are known to have decidable emptiness problem in CONPTIME [FRR⁺18]. This counter machine accepts nested words on which there are two runs witnessing the non-satisfiability of the twinning property by $\text{FST}(T, n^2)$. It is not necessary to encode the stack explicitly in the state, using the pushdown mechanism of the counter machine, and hence we can keep the size of the counter machine polynomial (in T). To witness the non-satisfiability of the twinning property, one uses combinatorics properties of the output words produced by those runs in case the delays are different. There are several conditions to be checked (taken in disjunction), one of them being that there is a mismatch between the output v_1 and the output of v_2 , i.e., there is a position i such that $v_1[i] \neq v_2[i]$. The counter machine simulates the behaviour of T (without producing anything) and the difficulty is that the i -th position of v_1 and v_2 may not be produced when reading *different* input positions. The machine instead non-deterministically guesses to output positions c_1, c_2 (whose values are stored in two different counters), check that $v_1[c_1] \neq v_2[c_2]$ and later on checks that $c_1 = c_2$. This can be done using only one reversal. The details can be found in the proof of Proposition 4.6. \square

Algorithm 1 Algorithm LCPIN.

```

procedure LCPIN(fVPT  $T$ , function  $next()$ )
2:   reduce( $T$ )           // in PTIME using [CRT15]
   initialize( $S$ )       // DAG with edges  $\# \xrightarrow{\epsilon} (q_0, \perp, 0)$  for all initial states  $q_0$  of  $T$ 
4:    $\alpha \leftarrow next()$  // read first input symbol
   while  $\alpha \neq \dagger$  do
6:     if  $\alpha$  is a return symbol then
       if  $S.height() \leq 1$  then           // pop on empty stack
8:         reject this input word
       else
10:        update_return( $S, T, \alpha$ )      // see Algorithm 3
       else                               //  $\alpha$  is a call symbol
12:        update_call( $S, T, \alpha$ )       // see Algorithm 2
       output_lcp( $S$ )                     // see Algorithm 4
14:         $\alpha \leftarrow next()$         // read next input symbol
       if  $S.height() = 1$  and  $\# \xrightarrow{v} (q, \perp, 0)$  in  $S$  with  $q$  final state of  $T$  then
16:         output  $v$ 
           accept this input word
18:       else
           reject this input word

```

Note that in order to decide whether a functional VPT T with n states defines a transduction in BM, one could proceed as follows: first decide whether all the nested words of the domain have height at most n^2 , then construct $FST(T, n^2)$, and then decide whether $FST(T, n^2)$ is sequentializable using Prop.2.4. This would however gives an **ExpTime** procedure, as $FST(T, n^2)$ has exponential size, since there are exponentially many stack contents of height n^2 in general.

3. ONLINE EVALUATION ALGORITHM OF VPT-TRANSDUCTIONS

We present an online algorithm LCPIN to evaluate functional word transductions defined by **fVPTs**¹. For clarity, we present this algorithm under some assumptions, without loss of generality. First, input words of our algorithms are words $u \in \Sigma^*$ concatenated with a special symbol $\dagger \notin \Sigma$, denoting the end of the word. Second, we only consider input words without internal symbols ($\Sigma_i = \emptyset$), as they can easily be encoded by successive call and return symbols. Third, we assume an implementation of VPTs such that the set H of transitions with a given left-hand side can be retrieved in time $O(|H|)$.

The core task of this algorithm, presented in Algorithm 1, is to maintain the configuration for each run of the **fVPT** T on the input u , and produce its output on-the-fly. These configurations are efficiently stored in a data structure S . The first step of the algorithm LCPIN is to transform T into a reduced **fVPT** in polynomial time, using [CRT15]. Indeed, when T is reduced, functionality ensures that, for a given input word u , and for every accessible configuration (q, σ) of T , there is at most one v such that $(q_i, \perp) \xrightarrow{u/v} (q, \sigma)$ with q_i an initial state. Hence, we define a notion called *d-configuration*, as triples (q, σ, w) , where

¹We remind the reader that **fVPTs** stand for the class of functional VPTs.

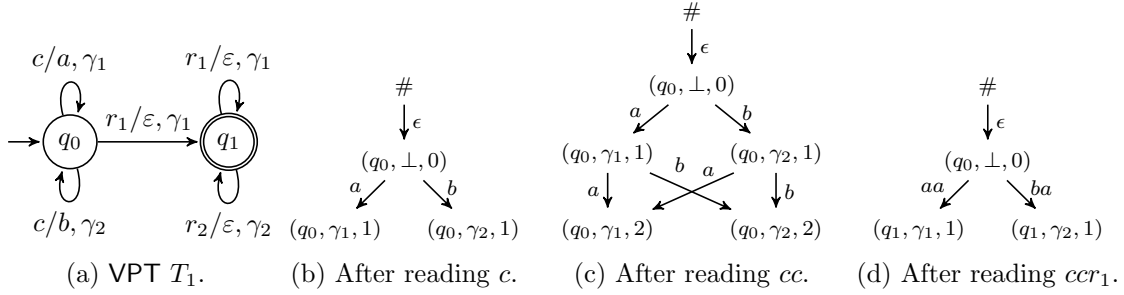


Figure 2: Data structure used by LCPIN.

q is the current state of the run, σ its corresponding stack content, and w is a suffix of v , which has not been output yet.

The set C of d-configurations of T on an input word u can be incrementally computed, starting from the set $\{(q_i, \perp, \epsilon) \mid q_i \in I\}$, and updated in the following way after reading a call symbol $c \in \Sigma_c$:

$$\text{update}(C, c) = \bigcup_{(q, \sigma, v) \in C} \{(q', \sigma\gamma, vv') \mid (q, c, v', \gamma, q') \in \delta_c\}$$

and, for a return symbol $r \in \Sigma_r$:

$$\text{update}(C, r) = \bigcup_{(q, \sigma\gamma, v) \in C} \{(q', \sigma, vv') \mid (q, r, v', \gamma, q') \in \delta_r\}$$

This provides us a first algorithm: update the set of d-configurations until the last letter \dashv , and then output the (unique) word v shared by all d-configurations in this set. This algorithm is inefficient in two aspects. First, it explicitly stores all d-configurations, and this set may grow exponentially. Second, it does not output anything before the end, and thus stores parts of the output that could have been released before the end, saving memory. Algorithm LCPIN addresses these two flaws thanks to the two following features.

3.1. Compact representation. First, the set of current d-configurations is stored in a compact structure that shares common stack contents. Consider for instance the VPT T_1 in Fig. 2 (a). After reading cc , current d-configurations are $\{(q_0, \gamma_1\gamma_1, aa), (q_0, \gamma_1\gamma_2, ab), (q_0, \gamma_2\gamma_1, ba), (q_0, \gamma_2\gamma_2, bb)\}$. Hence after reading c^n , the number of current d-configurations is 2^n . However, d-configurations share a lot of information. For instance, the previous set is the set of tuples $(q_0, \eta_1\eta_2, \alpha_1\alpha_2)$ where (η_i, α_i) is either (γ_1, a) or (γ_2, b) .

Based on this observation, we propose a data structure avoiding this blowup. As illustrated in Fig. 2 (b) to (d), this structure is a directed acyclic graph (DAG). The root of this DAG is denoted by $\#$, and the other nodes are tuples (q, γ, i) where $q \in Q$, $\gamma \in \Gamma$ and $i \in \mathbb{N}$ is the depth of the node in the DAG. Each edge of the DAG, denoted by \hookrightarrow , is labelled with a word, so that a branch of this DAG, read from the root $\#$ to the leaf, represents a d-configuration (q, σ, v) : q is the state in the leaf, σ is the concatenation of stack symbols in traversed nodes, and v is the concatenation of words on edges. For instance, in the DAG of Fig. 2 (c), the branch $\# \xrightarrow{\epsilon} (q_0, \perp, 0) \xrightarrow{b} (q_0, \gamma_2, 1) \xrightarrow{a} (q_0, \gamma_1, 2)$ encodes the d-configuration $(q_0, \gamma_2\gamma_1, ba)$ of the VPT of Fig. 2 (a). However, this data structure cannot

Algorithm 2 Updating structure S with a call symbol.

```

procedure UPDATE_CALL(structure  $S$ , transducer  $T$ , call symbol  $c$ )
2:    $newEdges \leftarrow \emptyset$ 
    $orphans \leftarrow \emptyset$ 
4:   for  $(q, \gamma, i) \in S.leaves()$  do
     if  $\exists v, \gamma', q' \mid (q, c, v, \gamma', q') \in \delta_T$  then
6:       for  $(v, \gamma', q') \mid (q, c, v, \gamma', q') \in \delta_T$  do
          $newEdges.add((q, \gamma, i) \xrightarrow{v} (q', \gamma', i + 1))$ 
8:       else
          $orphans.add((q, \gamma, i))$ 
10:   $remove\_edges(S, orphans)$ 
    $S.edges().add(newEdges)$ 
12:
procedure REMOVE_EDGES(structure  $S$ , set  $orphans$ )
14:  while  $orphans \neq \emptyset$  do
     $n \leftarrow orphans.pop()$ 
16:    for  $m \mid \exists v, m \xrightarrow{v} n$  do
       $S.removeEdge(m \xrightarrow{v} n)$ 
18:    if  $\nexists n', v', m \xrightarrow{v'} n'$  then  $orphans.add(m)$ 

```

store any set of accessible d-configurations of arbitrary functional VPTs: at most one delay w has to be assigned to a d-configuration. This is why we need T to be reduced.

We denote by G_u^T the DAG obtained after reading u . When a call letter $c \in \Sigma_c$ is read, the structure G_u^T is updated such that, for every leaf of G_u^T , a child is added for every way of updating the corresponding configuration according to a rule of T . If a leaf cannot be updated, it is removed and this removal is propagated upwards to its ascendants becoming leaves (procedure REMOVE_EDGES). Algorithm 2 describes how G_{uc}^T is computed from G_u^T . For sake of clarity, we only show how edges are updated, not nodes (nodes without incoming edges are automatically removed). For a return letter $r \in \Sigma_r$, we try to pop every leaf: if it is possible, the leaf is removed and the new leaves updated, otherwise we remove the leaf and propagate the removal upwards (procedure REMOVE_EDGES). This is described in Algorithm 3, where the future level $i - 1$ is stored in $newEdges$, then levels i and $i - 1$ are removed by two calls to REMOVE_LEAVES, and finally the new level $i - 1$ is added.

The correctness of this construction can be established by proving the following invariant by induction on $|u|$:

For every $0 \leq i \leq hc(u)$, there is a path labelled by v from the root $\#$ to the node (q, σ, i) in G_u^T iff there exists $q_0 \in I$ such that $(q_0, \perp) \xrightarrow{u_1 \cdots u_k / v} (q, \sigma)$ where $k = \max\{j \mid hc(u_1 \cdots u_j) = i\}$.

3.2. Computing outputs. The second main feature of LCPIN is that it ensures that after reading a prefix u' of a word u , it will have output the longest common prefix of all corresponding runs, i.e., the word $\text{lcp}_{\text{in}}(u', T) = \text{lcp}(\text{reach}(u'))$ where

$$\text{reach}(u') = \{v \mid \exists (q_0, q, \sigma) \in I \times Q \times \Gamma^*, (q_0, \perp) \xrightarrow{u'/v} (q, \sigma)\}.$$

Algorithm 3 Updating structure S with a return symbol.

```

procedure UPDATE_RETURN(structure  $S$ , transducer  $T$ , return symbol  $r$ )
2:    $newEdges \leftarrow \emptyset$ 
    $orphans \leftarrow \emptyset$ 
4:   for  $(q_\ell, \gamma_\ell, i) \in S.leaves()$  do
   if  $\exists v, q \mid (q_\ell, r, v, \gamma_\ell, q) \in \delta_T$  then
6:     for  $(v, q) \mid (q_\ell, r, v, \gamma_\ell, q) \in \delta_T$  do
       for  $(q_0, \gamma_0, v_0) \mid (q_0, \gamma_0, i-1) \xrightarrow{v_0} (q_\ell, \gamma_\ell, i) \in S.edges()$  do
8:         for  $(n, v_1) \mid n \xrightarrow{v_1} (q_0, \gamma_0, i-1) \in S.edges()$  do
            $newEdges.add(n \xrightarrow{v_1 v_0 v} (q, \gamma_0, i-1))$ 
10:    else
       $orphans.add((q_\ell, \gamma_\ell, i))$ 
12:     $remove\_edges(S, orphans)$ 
       $remove\_leaves(S)$  // level  $i$ 
14:     $remove\_leaves(S)$  // level  $i-1$ 
       $S.edges().add(newEdges)$ 
16:
procedure REMOVE_LEAVES(structure  $S$ )
18:  for  $n \in S.leaves()$  do
    for  $(m, v) \mid m \xrightarrow{v} n \in S.edges()$  do
20:       $S.removeEdge(m \xrightarrow{v} n)$ 

```



(a) Internal node n of the DAG. (b) Node n after update by *factorize*.

Figure 3: Effect of *factorize* on a node.

As detailed in Algorithm 1, when a new input symbol is read, the DAG is first updated as described in the previous section, using Algorithm 2 for a call symbol, and Algorithm 3 for a return symbol.

Then, a bottom-up pass on this DAG computes and outputs $\text{lcp}_{\text{in}}(u', T)$ as described by Algorithm 4. This one starts with the procedure *factorize*, that processes every node in a bottom-up manner (from leaves to the root $\#$). For each node (except the root), let ℓ be the longest common prefix of labels of outgoing edges. Then ℓ is removed from these outgoing edges, and concatenated at the end of labels of incoming edges. This is illustrated in Fig. 3. At the end, the longest common prefix of all output words on u' is the longest common prefix of the words labelling the edges outgoing from the root node $\#$. It can be easily shown by induction on the DAG that *factorize* preserves the set of d-configurations stored in this DAG.

Algorithm 4 Compute and output the longest common prefix of words in S , and remove it from all branches of S .

```

procedure OUTPUT_LCP(structure  $S$ )
2:   factorize( $S, \#, \emptyset$ )
    $\ell \leftarrow \text{lcp}(\{v \mid \exists n, \# \xrightarrow{v} n\})$ 
4:   output  $\ell$ 
   for  $n, v \mid \# \xrightarrow{\ell \cdot v} n$  do
6:     replace  $\# \xrightarrow{\ell \cdot v} n$  by  $\# \xrightarrow{v} n$  in  $S$ 

8: function FACTORIZE(structure  $S$ , node  $n$ , set done)
   if  $n \notin S.\text{leaves}()$  then
10:    for  $m, v \mid n \xrightarrow{v} m$  and  $m \notin \text{done}$  do
       $\text{done} \leftarrow \text{factorize}(S, m, \text{done})$ 
12:    if  $n \neq \#$  then
       $\ell \leftarrow \text{lcp}(\{v \mid \exists m, n \xrightarrow{v} m\})$ 
14:    for  $m, v \mid n \xrightarrow{\ell \cdot v} m$  do
      replace  $n \xrightarrow{\ell \cdot v} m$  by  $n \xrightarrow{v} m$  in  $S$ 
16:    for  $m, v \mid m \xrightarrow{v} n$  do
      replace  $m \xrightarrow{v} n$  by  $m \xrightarrow{v \cdot \ell} n$  in  $S$ 
18:    return  $\text{done} \cup \{n\}$ 

```

Let $\text{out}_{\neq}(u')$ be the maximal length of outputs of T on u' to which the longest common prefix has been removed: $\text{out}_{\neq}(u') = \max_{v \in \text{reach}(u')} |v| - |\text{lcp}_{\text{in}}(u', T)|$. We prove the following complexity result:

Proposition 3.1. *Given an fVPT T , one can build in PTIME a Turing transducer, denoted $M_{\text{LCPIN}}(T)$, which computes $\llbracket T \rrbracket$, and such that, after reading a prefix u' of a well-nested word $u \in \Sigma^*$, uses space in $O((\text{hc}(u') + 1) \cdot \text{out}_{\neq}(u'))$ on the working tape.*

Proof. The first step of Algorithm LCPIN is the reduction of the fVPT, in polynomial time [CRT15]. This procedure eliminates runs starting in the initial configuration but which can not be completed into accepting runs. As a consequence, the value of $\text{out}_{\neq}(u')$ can only decrease. Given a reduced fVPT T , Algorithm LCPIN uses, after reading a prefix u' of an input word u , space in $O(|Q|^2 \cdot |\Gamma|^2 \cdot (\text{hc}(u') + 1) \cdot \text{out}_{\neq}(u'))$ on the working tape. Indeed, the depth of the DAG obtained after reading u' is $\text{hc}(u') + 1$, each level has at most $|Q| \cdot |\Gamma|$ nodes, and each edge is labelled with a word of length less than $\text{out}_{\neq}(u')$ (as each edge participates in a useful d-configuration, T being reduced). \square

4. HEIGHT BOUNDED MEMORY EVALUATION

As we have seen, bounded memory is too restrictive in the context of nested words as it does not allow one to process well-nested words of unbounded height. In this section, we define a notion of bounded memory which takes into account the height of the input word.

4.1. HBM transductions.

Definition 4.1. A (functional) transduction $f : \Sigma^* \rightarrow \Sigma^*$ is *height bounded memory (HBM)* if there exists a function $\lambda : \mathbb{N} \rightarrow \mathbb{N}$ such that f is computable by some Turing transducer M that runs in space complexity at most $\lambda(h(u))$.

Note that this definition ensures that the Turing transducer cannot store the whole input word on the working tape in general, because the length of an input word is not necessarily bounded by some function of its height.

Example 4.2. Given some alphabet A , one can encode any A -labelled ranked tree into some nested word over the set of internal symbols A , the set of call symbols $\{c_a \mid a \in A\}$ and the set of return symbols $\{r_a \mid a \in A\}$ naturally by considering a depth-first traversal of the tree. For instance, $f(g(a, b), a)$ is encoded as $c_f c_g a b r_g a r_f$. Any functional VPT T whose domain is included in the set of encodings of A -labelled ranked trees of arity at most k , where k is a fixed constant, is in HBM. Indeed, the length of any input word u is then at most exponential in $h(u)$, and the number of runs of T on u is at most exponential in $|u|$, hence the result.

Another example of VPT-transduction is that of Fig. 2 (a): it is not in BM, but is in HBM: the stack content suffices (and is necessary) to determine the output.

We have seen that a functional transduction defined by an FST T is BM iff T is sequentializable. We give an example illustrating that for VPTs, being sequentializable is too strong to characterize HBM. Consider the VPT of Fig. 1 defined by the plain arrows. The transduction it defines is in HBM as its domain only contains ranked trees. However it is not sequentializable, as the transformation of c into a or b depends on the last return.

When the structured alphabet contains only internal letters, HBM and BM coincide, thus it is undecidable whether a pushdown transducer is HBM. In the remainder of this section, we prove that HBM is decidable for fVPTs.

4.2. Horizontal twinning property. As we have seen in Section 2, BM functional FST-transductions are characterized by the twinning property. We introduce a similar characterization of HBM fVPTs-transductions, called the *horizontal twinning property (HTP)*. Intuitively, the HTP requires that two runs on the same input cannot accumulate increasing output delay on loops on well-nested input words.

Definition 4.3. Let T be an fVPT. T satisfies the *horizontal twinning property (HTP)* if for all $u_1, u_2, v_1, v_2, w_1, w_2 \in \Sigma^*$ such that u_2 is well-nested, for all $q_0, q'_0 \in I$, for all $q, q' \in Q$, and for all $\sigma, \sigma' \in \Gamma^*$ such that (q, σ) and (q', σ') are co-accessible,

$$\text{if } \begin{cases} (q_0, \perp) \xrightarrow{u_1/v_1} (q, \sigma) \xrightarrow{u_2/v_2} (q, \sigma) \\ (q'_0, \perp) \xrightarrow{u_1/w_1} (q', \sigma') \xrightarrow{u_2/w_2} (q', \sigma') \end{cases} \text{ then } \Delta(v_1, w_1) = \Delta(v_1 v_2, w_1 w_2).$$

Example 4.4. Consider the VPT of Fig. 1 (including dashed arrows). It does not satisfy the HTP, as the delays increase when looping on $crcr\dots$. Without the dashed transitions, the HTP is trivially satisfied. Indeed, for any input word there is no loop between configurations, that is any two reached configurations differ either on the stack or on the state.

4.3. Deciding HTP.

Lemma 4.5. *Let T be an fVPT. T does not satisfy the HTP iff there exist two runs satisfying the premises of the HTP such that either (i) $|v_2| \neq |w_2|$ or (ii) $v_2w_2 \neq \epsilon$ and there exists $1 \leq i \leq \min(|v_1|, |w_1|)$ such that $v_1[i] \neq w_1[i]$.*

Proof. First, we prove the 'if' direction. Let us take states, words and stack contents as in the premises of the HTP:

$$\begin{cases} (q_0, \perp) \xrightarrow{u_1/v_1} (q, \sigma) \xrightarrow{u_2/v_2} (q, \sigma) \\ (q'_0, \perp) \xrightarrow{u_1/w_1} (q', \sigma') \xrightarrow{u_2/w_2} (q', \sigma') \end{cases} \quad (1)$$

and suppose that (i) or (ii) hold. By iterating the loop, i.e., by considering input u_2^i for all $i \geq 1$, we can rewrite the latter pattern as:

$$\begin{cases} (q_0, \perp) \xrightarrow{u_1 u_2^{i-1} / v_1 v_2^{i-1}} (q, \sigma) \xrightarrow{u_2/v_2} (q, \sigma) \\ (q'_0, \perp) \xrightarrow{u_1 u_2^{i-1} / w_1 w_2^{i-1}} (q', \sigma') \xrightarrow{u_2/w_2} (q', \sigma') \end{cases} \quad (\star)$$

If (i) holds, then the difference between the lengths of $v_1 v_2^{i-1}$ and $w_1 w_2^{i-1}$ gets arbitrarily large when i increases, and cannot be compensated by just outputting one more v_2 and w_2 . Hence, there must necessarily exist i such that $\Delta(v_1 v_2^{i-1}, w_1 w_2^{i-1}) \neq \Delta(v_1 v_2^i, w_1 w_2^i)$. The runs (\star) witness the non-satisfiability of the HTP. Similarly, if (ii) holds, since $v_2 w_2 \neq \epsilon$, at least one of v_2, w_2 is non-empty, hence by iterating the loop the delays will accumulate after the mismatch between u_1 and v_1 . In other words, we have $\Delta(v_1 v_2^{i-1}, w_1 w_2^{i-1}) \neq \Delta(v_1 v_2^i, w_1 w_2^i)$ for all $i \geq 1$, hence again witnessing the non-satisfiability of the HTP.

Conversely, suppose that the HTP does not hold for the runs depicted in (1), i.e. $\Delta(v_1, w_1) \neq \Delta(v_1 v_2, w_1 w_2)$. Assume that $|v_2| = |w_2|$. Since $\Delta(v_1, w_1) \neq \Delta(v_1 v_2, w_1 w_2)$, we necessarily have $v_2 w_2 \neq \epsilon$ and since $|v_2| = |w_2|$ we get that both v_2 and w_2 are non-empty. Suppose that there exists i such that there is a mismatch between $v_1 v_2^i$ and $w_1 w_2^i$, then we are done, by taking as witness of the right statement of the lemma the following runs:

$$\begin{cases} (q_0, \perp) \xrightarrow{u_1 u_2^i / v_1 v_2^i} (q, \sigma) \xrightarrow{u_2/v_2} (q, \sigma) \\ (q'_0, \perp) \xrightarrow{u_1 u_2^i / w_1 w_2^i} (q', \sigma') \xrightarrow{u_2/w_2} (q', \sigma') \end{cases}$$

Otherwise, for all i , there is no mismatch between $v_1 v_2^i$ and $w_1 w_2^i$. This implies that by iterating ω -times the loop we obtain the equality $v_1 v_2^\omega = w_1 w_2^\omega$. Wlog assume that v_1 is a prefix of w_1 , i.e. $w_1 = v_1 w'_1$ for some w'_1 (the other case is symmetric). Then we get $v_2^\omega = w'_1 w_2^\omega$. Using simple arguments of word combinatorics (based on Fine and Wilf's theorem, which can be applied since v_2 and w_2 are non-empty, see for instance Lemma 3 of [FRR⁺10]), we get that there exist two words t_1, t_2 and $\alpha > 0, \beta \geq 0$, such that $v_2 = (t_1 t_2)^\alpha$, $w_2 = (t_2 t_1)^\alpha$ and $w'_1 = (t_1 t_2)^\beta t_1$. Therefore, $\Delta(v_1, w_1) = \Delta(v_1, v_1 w'_1) = (\epsilon, w'_1) = (\epsilon, (t_1 t_2)^\beta t_1)$. On the other hand, we have

$$\begin{aligned} \Delta(v_1 v_2, w_1 w_2) &= \Delta(v_2, w'_1 w_2) = \Delta((t_1 t_2)^\alpha, (t_1 t_2)^\beta t_1 (t_2 t_1)^\alpha) \\ &= \Delta((t_1 t_2)^\alpha, (t_1 t_2)^{\beta+\alpha} t_1) = (\epsilon, (t_1 t_2)^\beta t_1) = \Delta(v_1, w_1), \end{aligned}$$

contradicting our assumption. □

Proposition 4.6. *The HTP is decidable in CONPTIME for fVPTs.*

Proof. Let T be an fVPT. We reduce HTP decidability to checking the emptiness of a non-deterministic reversal-bounded pushdown counter machine M , in polynomial time. Such a machine is a pushdown automaton extended with counters which can be incremented, decremented, and tested to zero. Counters can be in two modes, either increasing or decreasing. A *reversal* is a change of mode. Given a fixed constant r and a fixed number of counters k , the emptiness problem of any pushdown k -counter machine whose runs (on any input) make at most r reversals, is decidable in CONPTIME [FRR⁺18].

In our reduction, one only needs to take $r = 1$ and $k = 2$. Our pushdown counter machine M accepts any word of the form $u = u_1\#u_2\#u_3\#u_4$, where $\#$ is a special separator symbol, such that there exist runs of T of the form:

$$\left\{ \begin{array}{l} (q_0, \perp) \xrightarrow{u_1/v_1} (q, \sigma) \xrightarrow{u_2/v_2} (q, \sigma) \xrightarrow{u_3/v_3} (q_f, \perp) \\ (q'_0, \perp) \xrightarrow{u_1/w_1} (q', \sigma') \xrightarrow{u_2/w_2} (q', \sigma') \xrightarrow{u'_3/w_3} (q'_f, \perp) \end{array} \right.$$

where q_f, q'_f are accepting, and such that either (i) $|v_2| \neq |w_2|$ or (ii) $v_2w_2 \neq \epsilon$ and there exists $1 \leq i \leq \min(|v_1|, |w_1|)$ such that $v_1[i] \neq w_1[i]$ (i.e., the HTP does not hold by Lemma 4.5). The words u_3 and u'_3 are used to check that the configurations (q, σ) and (q', σ') are co-accessible. Therefore the HTP holds if and only if no word is accepted by the automaton M .

When reading u , M non-deterministically guesses whether condition (i) or condition (ii) holds. For each of them, it will simulate the behaviour of T (ignoring the outputs), by guessing non-deterministically a run of T on $u_1u_2u_3$ and a run on $u_1u_2u'_3$ (hence the states of M contain pairs of states of T), making sure that those runs loop on u_2 . To do so, when reading the first $\#$, the states (q, q') reached by T on the two runs are remembered, then u_2 is checked to be well-nested (using the pushdown stack), and once the second $\#$ is met, it suffices for M to verify that the pairs of states reached so far is (q, q') , otherwise the run of M rejects.

Let us now explain how condition (i) is checked. It suffices to have two counters c_1, c_2 counting the length of v_2 and w_2 respectively. To do so, the two counters c_1, c_2 are initially set to 0 and after reading the first $\#$, when guessing the two runs of T on u_2 , the lengths of the outputs of any two respective simulated transitions of T are added to c_1 and c_2 . When the second $\#$ is met, M checks whether $c_1 \neq c_2$, using ϵ transitions (whose use is allowed in the model) to decrement in parallel c_1, c_2 until the point where one of them reaches 0. At this point, it suffices to check that the other one is not zero (in which case the two lengths are different), otherwise M rejects.

We now detail how to check condition (ii). Initially, c_1 and c_2 are filled with an arbitrary value i . This can be done again by using some ϵ -loop which increments both counters in parallel. Then, in parallel to simulating two runs of T , M decrements c_1, c_2 by the length of the outputs of the transitions taken by the two simulated runs respectively. When one of them reaches 0, say c_1 , it means that the output v_1 of the first run has reached position i , it suffices to store the current symbol $v_1[i]$ in the state of M . When later on the other counter, say c_2 , reaches 0, it means that the second run of T has reached position i of v_2 , and M can therefore check whether $v_1[i] \neq v_2[i]$. \square

4.4. Deciding HBM. We now show that HTP characterizes HBM fVPTs-transductions.

Theorem 4.7. *Let T be an fVPT. $\llbracket T \rrbracket$ is HBM iff the HTP holds for T , which is decidable in CONPTIME. In this case, the Turing transducer $M_{\text{LCPIN}}(T)$ runs, on an input stream u , in space complexity exponential in the height of u .*

We can state more precisely the space complexity of $M_{\text{LCPIN}}(T)$ when T is reduced. In this case, it is in $O(3(\text{h}(u) + 1)^3 \cdot |Q|^{2(\text{h}(u)+1)} \cdot M)$, where $M = \max\{|v| \mid (q, a, v, \gamma, q') \in \delta\}$.

Proof. Let T be an fVPT. If $\llbracket T \rrbracket$ is HBM, then the HTP holds for T by Lemma 4.8 (proved in this section). Conversely, if T satisfies the HTP, thanks to Theorem 6.1 of [CRT15], we build an equivalent reduced fVPT T' in polynomial time (more precisely, we use the construction **reduce** of [CRT15]). In this construction, the states and the stack symbols of T' are obtained from those of T by enriching them with a state of T . In addition, given a run in T' , one recovers a run in T (with the same input and output words) when projecting away this additional component. As a consequence, the fact that T satisfies the HTP implies that T' also does. We thus assume now that T is reduced.

Then we apply Lemma 4.10 (proved in this section) which bounds the maximal difference between outputs of T on a prefix u' of the input u : $\text{out}_{\neq}(u') \leq 3(\text{h}(u') + 1)^2 |Q|^{2(\text{h}(u')+1)} M$. Proposition 3.1 gives the complexity of the evaluation algorithm: $O((\text{hc}(u') + 1) \cdot \text{out}_{\neq}(u'))$ on the working tape after reading a prefix u' of u . We know that $\text{hc}(u') \leq \text{h}(u') \leq \text{h}(u)$, so the space is in $O(3(\text{h}(u) + 1)^3 \cdot |Q|^{2(\text{h}(u)+1)} \cdot M)$, and finally $\llbracket T \rrbracket$ is HBM. Hence deciding HBM reduces to deciding HTP, and this is in CONPTIME by Proposition 4.6. \square

Lemma 4.8. *Let T be an fVPT. If $\llbracket T \rrbracket$ is HBM, then the HTP holds for T .*

Proof. Suppose that the HTP does not hold for T . Therefore there are words

$$u_1, u_2, u_3, u'_3, v_1, v_2, v_3, w_1, w_2, w_3, w_3 \in \Sigma^*,$$

stacks $\sigma, \sigma' \in \Gamma^*$ and states $q, q' \in Q$, $q_0, q'_0 \in I$ and $q_f, q'_f \in F$ such that:

$$\left\{ \begin{array}{l} (q_0, \perp) \xrightarrow{u_1/v_1} (q, \sigma) \xrightarrow{u_2/v_2} (q, \sigma) \xrightarrow{u_3/v_3} (q_f, \perp) \\ (q'_0, \perp) \xrightarrow{u_1/w_1} (q', \sigma') \xrightarrow{u_2/w_2} (q', \sigma') \xrightarrow{u'_3/w_3} (q'_f, \perp) \end{array} \right.$$

and $\Delta(v_1, w_1) \neq \Delta(v_1 v_2, w_1 w_2)$. Let $K = \max(\text{h}(u_1 u_2 u_3), \text{h}(u_1 u_2 u'_3))$. By definition of $\text{FST}(T, K)$ (states are configurations of T) and Definition 2.3, the twinning property for FSTs does not hold for $\text{FST}(T, K)$. Therefore, by [Cho77] (see Lemma 2.4), $\text{FST}(T, K)$ is not sequentializable. By Proposition 2.4, $\llbracket \text{FST}(T, K) \rrbracket$ is not BM. Therefore $\llbracket T \rrbracket$ is not HBM, otherwise $\llbracket T \rrbracket$ could be evaluated in space complexity $f(\text{h}(u))$ on any input word u , for some function f . This corresponds to bounded memory if we fix the height of the words to K at most. \square

For the converse, we can apply the evaluation algorithm of Section 3, whose complexity depends on the maximal delay between all the candidate outputs of the input word. This maximal delay is exponentially bounded by the height of the word.

In order to prove this result, we introduce a notion of arity by analogy with trees that can be encoded by well-nested words. The *arity* ar of a well-nested word is inductively defined by: $\text{ar}(i) = 1$ if $i \in \Sigma_\iota$, $\text{ar}(uv) = \text{ar}(u) + \text{ar}(v)$ if u and v are well-matched, and $\text{ar}(cur) = 1$ if u is well-matched, and c and r are call and return symbols, respectively. We say that a well-nested word u is k -narrow if $\text{ar}(v) \leq k$ for all well-nested factors v of u .

Lemma 4.9. *If u is a k -narrow well-nested word with $k \geq 2$ then $|u| \leq 3(k^{\text{h}(u)+1} - 1)$.*

Proof. We show, by induction on $h(u)$, that $|u| \leq \frac{2k}{k-1}(k^{h(u)+1} - 1)$. Note that this implies $|u| \leq 3(k^{h(u)+1} - 1)$, as $k \geq 2$. Let us consider the unique decomposition $u = u_1 u_2 \cdots u_n$ with $\text{ar}(u_i) = 1$ for all $1 \leq i \leq n$ and $n = \text{ar}(u) \leq k$. The basic case is $h(u) = 0$, i.e., every word u_i is an internal symbol. In that case

$$|u| = n \leq k \leq 2k = \frac{2k}{k-1}(k^{h(u)+1} - 1).$$

In the general case, every word u_i is either an internal symbol (and thus of length 1), or of the form $c_i v_i r_i$ where c_i (resp. r_i) is a call (resp. return) symbol and v_i a well-nested word such that $h(v_i) < h(u)$ so, by induction hypothesis,

$$|v_i| \leq \frac{2k}{k-1}(k^{h(v_i)+1} - 1) \leq \frac{2k}{k-1}(k^{h(u)} - 1).$$

Hence,

$$|u| \leq k(2 + \frac{2k}{k-1}(k^{h(u)} - 1)) = 2k \frac{k-1 + k(k^{h(u)} - 1)}{k-1} = \frac{2k}{k-1}(k^{h(u)+1} - 1). \quad \square$$

Lemma 4.10. *Let T be a reduced fVPT. If the HTP holds for T , then for all well-nested words $u \in \Sigma^*$, and all prefixes u' of u , we have*

$$\text{out}_{\neq}(u') \leq 3(h(u') + 1)^2 |Q|^{2(h(u')+1)} M,$$

where $M = \max\{|t| \mid (q, a, t, \gamma, q') \in \delta\}$.

Proof. Let $u \in \Sigma^*$ be a well-nested word, and $u' \in \Sigma^*$ be a prefix of u . The proof is similar to that of [BC02] for FST. It proceeds by induction on the length of u' . If $|u'| \leq 3(h(u') + 1)^2 |Q|^{2(h(u')+1)}$, then the result is trivial. Otherwise, consider the unique decomposition of u' such that $u' = u_0 c_1 u_1 c_2 \cdots u_{n-1} c_n u_n$ where every u_i is well-nested, and every c_i is a call symbol. Hence $n = \text{hc}(u') \leq h(u')$.

Let us first consider the case where every u_i is $|Q|^2$ -narrow. If $|Q| > 1$ then, by Lemma 4.9, $|u_i| \leq 3(|Q|^{2(h(u_i)+1)} - 1)$ and thus

$$|u'| \leq (n + 1)(3(|Q|^{2(h(u')+1)} - 1) + 1),$$

since $h(u_i) \leq h(u')$. As $n \leq h(u')$,

$$|u'| \leq 3(h(u') + 1) |Q|^{2(h(u')+1)} \leq 3(h(u') + 1)^2 |Q|^{2(h(u')+1)},$$

which means that we are in the basic case. If $|Q| = 1$ then each u_i is 1-narrow and thus of the form $c'_1 \cdots c'_\ell v'_\ell \cdots r'_1$ with $\iota \in \{\epsilon\} \cup \Sigma_\iota$ and, for every j , $c'_j \in \Sigma_c$, $r'_j \in \Sigma_r$. So $|u_i| \leq 2h(u_i) + 1$ and

$$|u'| \leq (n + 1)(2h(u_i) + 2) \leq 2(h(u') + 1)^2 \leq 3(h(u') + 1)^2 |Q|^{2(h(u')+1)}.$$

So we are also in the basic case.

Assume now that one of the u_i is not $|Q|^2$ -narrow, i.e., $\text{ar}(u'') > |Q|^2$ for some well-nested factor u'' of u_i . Let $(q, \sigma, w), (q', \sigma', w') \in Q \times \Gamma^* \times \Sigma^*$ be such that there exist runs $\rho : (i, \perp) \xrightarrow{u'/v} (q, \sigma)$ and $\rho' : (i', \perp) \xrightarrow{u'/v'} (q', \sigma')$, with $i, i' \in I$, $v = \text{lcp}_{\text{in}}(u', T) \cdot w$, $v' = \text{lcp}_{\text{in}}(u', T) \cdot w'$, and such that $\text{out}_{\neq}(u') = |w|$. Consider the decomposition of u'' into well-nested words u''_i of arity one: $u'' = u''_1 u''_2 \cdots u''_k$, and consider the states p_i (resp. p'_i) encountered in ρ (resp. ρ') after reading u''_i . As $k = \text{ar}(u'') > |Q|^2$, there exist i, j such that

$1 \leq i < j \leq k$ and $(p_i, p'_i) = (p_j, p'_j)$. Let $y = u''_{i+1} u''_{i+2} \cdots u''_j$. We can decompose runs ρ and ρ' as follows:

$$\begin{cases} \rho : (i, \perp) & \xrightarrow{x/v_1} & (p_i, \sigma_1) & \xrightarrow{y/v_2} & (p_i, \sigma_1) & \xrightarrow{z/v_3} & (q, \sigma) \\ \rho' : (i', \perp) & \xrightarrow{x/v'_1} & (p'_i, \sigma'_1) & \xrightarrow{y/v'_2} & (p'_i, \sigma'_1) & \xrightarrow{z/v'_3} & (q', \sigma') \end{cases}$$

In addition, we have $u' = xyz$, and $y \neq \varepsilon$ and well-nested, $v = \text{lcp}_{\text{in}}(u', T) \cdot w = v_1 v_2 v_3$, and $v' = \text{lcp}_{\text{in}}(u', T) \cdot w' = v'_1 v'_2 v'_3$. Moreover, (q, σ) and (q', σ') are co-accessible, as T is reduced. By the HTP property, we obtain $\Delta(v_1 v_2, v'_1 v'_2) = \Delta(v_1, v'_1)$. By Lemma 1.1, this entails the equality

$$\Delta(v_1 v_2 v_3, v'_1 v'_2 v'_3) = \Delta(\Delta(v_1 v_2, v'_1 v'_2) \cdot (v_3, v'_3)) = \Delta(\Delta(v_1, v'_1) \cdot (v_3, v'_3)) = \Delta(v_1 v_3, v'_1 v'_3).$$

Thus, we obtain

$$\Delta(w, w') = \Delta(v, v') = \Delta(v_1 v_2 v_3, v'_1 v'_2 v'_3) = \Delta(v_1 v_3, v'_1 v'_3).$$

As $v_1 v_3$ and $v'_1 v'_3$ are possible output words for the input word $u_1 u_3$, whose length is strictly smaller than $|u'|$, we obtain $\text{out}_{\neq}(u') = |w| \leq \text{out}_{\neq}(u_1 u_3)$ and the result holds by induction. \square

Example 4.11 (HBM is tight). Theorem 4.7 shows that the space complexity of a VPT in HBM is at most exponential. We give here an example illustrating the tightness of this bound. We describe here a transduction on trees, and use the well-known encoding of trees by well-nested words defined by: $\text{encode}(a(t_1, \dots, t_n)) = c_a \cdot \text{encode}(t_1) \cdots \text{encode}(t_n) r_a$. The idea is to encode the tree transduction $(f(t, a) \mapsto f(t, a)) \cup (f(t, b) \mapsto f(\bar{t}, b))$ by a VPT, where t is a binary tree over $\{0, 1\}$ and \bar{t} is the mirror of t , obtained by replacing the 0 by 1 and the 1 by 0 in t . Thus taking the identity or the mirror depends on the second child of the root f . To evaluate this transformation in a streaming manner, one has to store the whole subtree t in memory before deciding to transform it into t or \bar{t} . The evaluation of this transduction cannot be done in space polynomial in the height of the input as there are a doubly exponential number of trees of height n , for all $n \geq 0$.

5. ONLINE BOUNDED MEMORY EVALUATION

In the previous section, we have shown that a VPT-transduction is in HBM iff the horizontal twinning property holds. The notion of height-bounded memory is quite permissive as for instance, any transduction of *ranked* tree linearizations (given a fixed rank) is HBM, because in this case, the length of the tree linearization functionally depends on the height of the tree.

In this section, we introduce a stronger constraint on memory: the amount of memory must depend only, at each moment (i.e., at any position in the nested word), on the current height. We call this requirement *online bounded memory* (OBM).

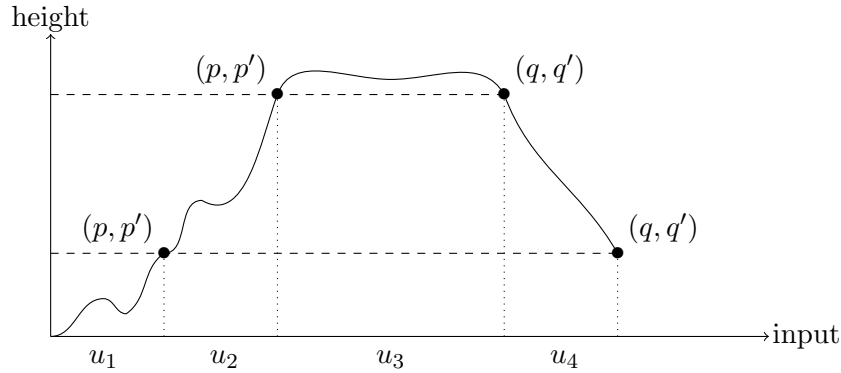


Figure 4: Premises of the matched twinning property (MTP)

5.1. OBM transductions.

Definition 5.1. A (functional) transduction $f : \Sigma^* \rightarrow \Sigma^*$ is *online bounded memory* (OBM) if there exists a function $\lambda : \mathbb{N} \rightarrow \mathbb{N}$ such that f is computable by a Turing transducer M satisfying the following property: on any input word $u \in \Sigma^*$, if M has read the prefix v of u (but not more), then the amount of memory of M on the working tape is less than $\lambda(\text{hc}(v))$.

Example 5.2. Consider the transduction that maps any word of the form $c^n r^n$ to $a^n c^n$, and any word of the form $c^n r' r^{n-2} r'$ to $b^n c^n$. This transduction is OBM: it suffices to compute the number n of c symbols till we read a symbol r or r' . During this phase, the memory is in $O(\log(n))$ and depends on the current height of the word. When r (resp. r') is read for the first time, the word $a^n c$ (resp. $b^n c$) is output and the memory flushed. Then, whenever a symbol is read, nothing is stored on memory and c symbols are output whenever a symbol is read.

In this section, we give an effective characterization of OBM transductions definable by VPT, using a new twinning property, called the *matched twinning property* (or *MTP* for short). Since it is a characterization of transductions, this property does not depend on the VPT that implements them: two equivalent VPTs that implements the same transduction both satisfy, or both do not satisfy, this twinning property. Another appealing property of OBM, compared to HBM, is that the maximal amount memory needed when running the algorithm of Section 3 is at most *quadratic* in the current height of the input nested word, while it is *exponential* for HBM transductions, and this latter bound is tight. In other words, any OBM transduction defined by a VPT can be evaluated with quadratic memory in the height of the input nested word.

5.2. Matched twinning property (MTP). The matched twinning property is a strengthening of the horizontal twinning property obtained by adding some new delay constraints on the well-matched loops. Intuitively, the MTP requires that two runs on the same input cannot accumulate increasing output delay on well-matched loops. They can accumulate delay on loops with increasing stack but this delay has to be caught up on the matching loops with descending stack. We show that this property is decidable, and that sequential VPTs satisfy it. Therefore the class of OBM VPT-transductions *subsumes* the class of sequentializable VPTs. We illustrate the following definition in Figure 4.

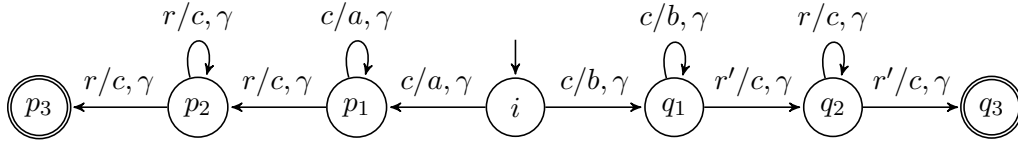


Figure 5: A non-sequentializable VPT that defines an OBM-transduction.

Definition 5.3. Let $T = (Q, I, F, \Gamma, \delta)$ be an \mathfrak{f} VPT. T satisfies the *matched twinning property* (MTP) if for all $u_i, v_i, w_i \in \Sigma^*$ ($i \in \{1, \dots, 4\}$) such that u_3 is well-nested, and u_2u_4 is well-nested, for all $i, i' \in I$, for all $p, q, p', q' \in Q$, and for all $\sigma_1, \sigma_2 \in \perp.\Gamma^*$, for all $\sigma'_1, \sigma'_2 \in \Gamma^*$, such that (q, σ_1) and (q', σ_2) are co-accessible:

$$\text{if } \begin{cases} (i, \perp) \xrightarrow{u_1/v_1} (p, \sigma_1) \xrightarrow{u_2/v_2} (p, \sigma_1\sigma'_1) \xrightarrow{u_3/v_3} (q, \sigma_1\sigma'_1) \xrightarrow{u_4/v_4} (q, \sigma_1) \\ (i', \perp) \xrightarrow{u_1/w_1} (p', \sigma_2) \xrightarrow{u_2/w_2} (p', \sigma_2\sigma'_2) \xrightarrow{u_3/w_3} (q', \sigma_2\sigma'_2) \xrightarrow{u_4/w_4} (q', \sigma_2) \end{cases}$$

then $\Delta(v_1v_3, w_1w_3) = \Delta(v_1v_2v_3v_4, w_1w_2w_3w_4)$. We say that a VPT T is *twinning* whenever it satisfies the MTP.

Note that any twinned VPT also satisfies the HTP (with $u_3 = u_4 = \epsilon$).

Example 5.4 (MTP does not imply sequentializable). The VPT of Fig. 1 with plain arrows does not satisfy the MTP, as the delay between the two branches increases when iterating the loops. Consider now the VPT of Fig. 5. It implements the transduction of Example 5.2. It is obviously twinned, as we cannot construct two runs on the same input which have the form given in the premises of the MTP. However this transducer is not sequentializable, as the output on the call symbols cannot be delayed to the matching return symbols.

5.3. Deciding MTP. As for the HTP, one can decide the MTP using a reduction to the emptiness of a reversal-bounded pushdown counter machines. First, one needs a technical lemma about the delays of iterated words, whose proof strongly relies on some word combinatorics result by Saarela [Saa15].

Lemma 5.5. Let v_1, \dots, v_{2n+1} and w_1, \dots, w_{2m+1} be two sequences of finite words over Σ , for two positive integers n and m . We define, for all $i \geq 0$, the two following words:

$$\begin{aligned} V_i &= v_1(v_2)^i v_3(v_4)^i \dots v_{2n-1}(v_{2n})^i v_{2n+1} \\ W_i &= w_1(w_2)^i w_3(w_4)^i \dots w_{2m-1}(w_{2m})^i w_{2m+1} \end{aligned}$$

If $\Delta(V_0, W_0) \neq \Delta(V_1, W_1)$, then for any $i \geq 0$, the set $\{j \geq 0 \mid \Delta(V_i, W_i) = \Delta(V_j, W_j)\}$ is finite.

Proof. The proof relies on a recent result of word combinatorics proved by Saarela in [Saa15]. It states that if the equality

$$\alpha_0(\alpha_1)^i \alpha_2 \dots \alpha_{2n-1}(\alpha_{2n})^i \alpha_{2n+1} = \beta_0(\beta_1)^i \beta_2 \dots \beta_{2m-1}(\beta_{2m})^i \beta_{2m+1}$$

holds for $m+n$ values of i , then it holds for all $i \geq 0$, where $n, m \geq 0$ and the α_j, β_j are arbitrary words. To simplify notations and case study, we work in the free group, i.e., the free monoid extended with the inverse u^{-1} , quotiented by the equalities $\sigma^{-1}\sigma = \sigma\sigma^{-1} = \epsilon$ for all σ .

First observe that w.l.o.g., we can assume that v_{2n} and w_{2m} are non-empty. Indeed, if one of them is empty, say $v_{2n} = \epsilon$, we set v_{2n-1} to $v_{2n-1}v_{2n+1}$.

We proceed by contradiction and assume that there exists a delay d such that $d = \Delta(V_i, W_i)$ for infinitely many i 's. We fix such a d and we let $J \subseteq \mathbb{N}$ be the set of indices i such that $d = \Delta(V_i, W_i)$. We write $d = (x, y)$ and distinguish two cases:

- (1) if $x = \epsilon$ or $y = \epsilon$, for simplicity suppose that $x = \epsilon$, the other case being symmetric. For all $i \in J$, we have $W_i = V_i y$. Since J is infinite, we can use Saarela's result to conclude that $W_i = V_i y$ for all $i \geq 0$. In particular, we get $W_0 = V_0 y$ and $W_1 = V_1 y$. Hence, $\Delta(W_0, V_0) = (\epsilon, y) = \Delta(W_1, V_1)$ which is a contradiction.
- (2) if $x \neq \epsilon$ and $y \neq \epsilon$, we again distinguish three cases:
 - (a) if $|x| \leq |v_{2n+1}|$, then $v_{2n+1} = zx$ for some z . By definition of x and y , we know that for all $i \in J$, there exists $\ell_i \in \Sigma^*$ such that

$$V_i = \ell_i x = v_1(v_2)^i v_3(v_4)^i \dots v_{2n-1}(v_{2n})^i zx \quad \text{and} \quad W_i = \ell_i y$$

Hence, we obtain:

$$\begin{aligned} V_i x^{-1} y &= v_1(v_2)^i v_3(v_4)^i \dots v_{2n-1}(v_{2n})^i zy \\ &= W_i \\ &= w_1(w_2)^i w_3(w_4)^i \dots w_{2m-1}(w_{2m})^i w_{2m+1} \end{aligned}$$

for all $i \in J$. By Saarela's result and since J is infinite, the above equation holds for all $i \geq 0$. Now, for all $i \geq 0$, let $(x_i, y_i) = \Delta(V_i, W_i)$ and ℓ_i such that $V_i = \ell_i x_i$ and $W_i = \ell_i y_i$. Then, for all $i \geq 0$, $y_i = \ell_i^{-1} W_i = \ell_i^{-1} V_i x_i^{-1} y = \ell_i^{-1} \ell_i x_i x_i^{-1} y = x_i x_i^{-1} y$, hence $x_i^{-1} y_i = x^{-1} y$ for all $i \geq 0$. By definition of the delay, x and y (which are both non-empty by assumption), start with different symbols. Hence $x^{-1} y \in (\Sigma^{-1})^{|x|} \Sigma^{|y|}$. Since $|x|, |y| > 0$, it cannot be the case that x_i or y_i is empty. Hence, they both start with different symbols (as delays), and we get $x_i = x$ and $y_i = y$ for all $i \geq 0$. In particular, it means that $\Delta(V_0, W_0) = \Delta(V_1, W_1)$.

- (b) if $|y| \leq |w_{2m+1}|$. This case is symmetric to the case $|x| \leq |v_{2n+1}|$.
- (c) if $|x| > |v_{2n+1}|$ and $|y| > |w_{2m+1}|$. Since $\Delta(V_i, W_i) = (x, y)$ for infinitely many i , by taking $i \in J$ sufficiently large, we have for some k, k' which can be assumed to be greater than $\max(|v_{2n}|, |w_{2m}|)$ (the reason why we take such values will be clear later), and some $v', w' \in \Sigma^*$ such that v' and w' are prefixes of v_{2n} and w_{2m} respectively:

$$\begin{aligned} V_i &= v_1(v_2)^i v_3(v_4)^i \dots v_{2n-1}(v_{2n})^k v' x \\ W_i &= w_1(w_2)^i w_3(w_4)^i \dots w_{2m-1}(w_{2m})^{k'} w' y \\ v_1(v_2)^i v_3(v_4)^i \dots v_{2n-1}(v_{2n})^k v' &= w_1(w_2)^i w_3(w_4)^i \dots w_{2m-1}(w_{2m})^{k'} w' \end{aligned}$$

In other words, the first letter of x and y corresponds to the first mismatch between V_i and W_i . Moreover, $v'x = v_{2n}^j x'$ for some j, x' and $w'y = w_{2m}^{j'} y'$ for some j', y' and since $|x| > |v_{2n+1}|$ and $|y| > |w_{2m+1}|$, we necessarily have $j, j' > 0$. Since v' and w' are prefixes of v_{2n} and w_{2m} respectively, we get the existence of two different symbols $a, b \in \Sigma$ and two words v'', w'' such that $v_{2n} = v' a v''$ and $w_{2m} = w' b w''$. Replacing these values in the third equation above, we get:

$$v_1(v_2)^i v_3(v_4)^i \dots v_{2n-1}(v' a v'')^k v' = w_1(w_2)^i w_3(w_4)^i \dots w_{2m-1}(w' b w'')^{k'} w'$$

which can be rewritten into

$$v_1(v_2)^i v_3(v_4)^i \dots v_{2n-1} v' (av''v')^k = w_1(w_2)^i w_3(w_4)^i \dots w_{2m-1} w' (bw''w')^{k'}$$

Let $U = v_1(v_2)^i v_3(v_4)^i \dots v_{2n-1} v' (av''v')^k$ and ℓ its length. We have $U[\ell - j | v_{2n} | + 1] = a$ for all $1 \leq j \leq k$. Similarly, we have $U[\ell - j' | w_{2m} | + 1] = b$ for all $1 \leq j' \leq k'$. By taking $j = |w_{2m}|$ and $j' = |v_{2n}|$ (which is possible since $k, k' \geq \max(|v_{2n}|, |w_{2m}|)$), we get a contradiction, as $a \neq b$. \square

The following lemma gives an alternative characterisation of the MTP which can be exploited to decide the MTP using counter pushdown machines.

Lemma 5.6. *Let $T = (Q, I, F, \Gamma, \delta)$ be an fVPT. T does not satisfy the MTP iff there exist words $u_i, v_i, w_i, i = 1, \dots, 4$, states $i, i' \in I$, $p, q, p', q' \in Q$, stack contents $\sigma_1, \sigma_2 \in \perp \cdot \Gamma^*$ and $\sigma'_1, \sigma'_2 \in \Gamma^*$ such that the following conditions hold:*

- (1) (q, σ_1) and (q', σ_2) are co-accessible, u_3 and $u_2 u_4$ are well-nested,
- (2)
$$\left\{ \begin{array}{l} (i, \perp) \xrightarrow{u_1/v_1} (p, \sigma_1) \xrightarrow{u_2/v_2} (p, \sigma_1 \sigma'_1) \xrightarrow{u_3/v_3} (q, \sigma_1 \sigma'_1) \xrightarrow{u_4/v_4} (q, \sigma_1) \\ (i', \perp) \xrightarrow{u_1/w_1} (p', \sigma_2) \xrightarrow{u_2/w_2} (p', \sigma_2 \sigma'_2) \xrightarrow{u_3/w_3} (q', \sigma_2 \sigma'_2) \xrightarrow{u_4/w_4} (q', \sigma_2) \end{array} \right.$$
- (3) $|v_2 v_4| \neq |w_2 w_4|$, or $|v_2 v_4| = |w_2 w_4|$, $|v_2 v_4 w_2 w_4| \neq 0$, and there is a mismatch between $v_1 v_3$ and $w_1 w_3$ (i.e. $v_1 v_3 = x \alpha v$ and $w_1 w_3 = x \beta w$ for some $x, v, w \in \Sigma^*$ and some $\alpha \neq \beta \in \Sigma$).

Proof. Suppose that the MTP is not satisfied. By definition of the MTP, there exist words and states that satisfy conditions (1) and (2), and such that $\Delta(v_1 v_3, w_1 w_3) \neq \Delta(v_1 v_2 v_3 v_4, w_1 w_2 w_3 w_4)$. If $|v_2 v_4| \neq |w_2 w_4|$ then we are done. Let us assume that $|v_2 v_4| = |w_2 w_4|$. Clearly, if $|v_2 v_4 w_2 w_4| = 0$, then $\Delta(v_1 v_3, w_1 w_3) = \Delta(v_1 v_2 v_3 v_4, w_1 w_2 w_3 w_4)$ which is impossible. By Lemma 5.5, the set $\{\Delta(v_1(v_2)^i v_3(v_4)^i, w_1(w_2)^i w_3(w_4)^i) \mid i \geq 0\}$ is infinite. We show that there exists i_0 such that there is a mismatch between $v_1(v_2)^{i_0} v_3(v_4)^{i_0}$ and $w_1(w_2)^{i_0} w_3(w_4)^{i_0}$. Suppose that no such i_0 exists and that $|v_1 v_3| \leq |w_1 w_3|$ (the other case is symmetric). Then, since $|v_2 v_4| = |w_2 w_4|$, for all $i \geq 0$, we have $|v_1(v_2)^i v_3(v_4)^i| = |w_1(w_2)^i w_3(w_4)^i| - |w_1 w_3| + |v_1 v_3|$, and $v_1(v_2)^i v_3(v_4)^i \preceq w_1(w_2)^i w_3(w_4)^i$. In other words, for all i , there exists $x_i \in \Sigma^*$ such that $w_1(w_2)^i w_3(w_4)^i = v_1(v_2)^i v_3(v_4)^i x_i$. Rephrased with delays, it means that $\Delta(v_1(v_2)^i v_3(v_4)^i, w_1(w_2)^i w_3(w_4)^i) = (\epsilon, x_i)$ for all $i \geq 0$. However, since $|v_1(v_2)^i v_3(v_4)^i| = |w_1(w_2)^i w_3(w_4)^i| - |w_1 w_3| + |v_1 v_3|$ for all $i \geq 0$, we have $|x_i| = |w_1 w_3| - |v_1 v_3|$ for all i . It contradicts the fact that $\{\Delta(v_1(v_2)^i v_3(v_4)^i, w_1(w_2)^i w_3(w_4)^i) \mid i \geq 0\}$ is infinite. Therefore, there exists $i_0 \geq 0$ such that there is a mismatch between $v_1(v_2)^{i_0} v_3(v_4)^{i_0}$ and $w_1(w_2)^{i_0} w_3(w_4)^{i_0}$. Finally, using the following decomposition, one satisfies the three conditions of the Lemma (i.e. by replacing u_3 by $u_2^{i_0} u_3(u_4)^{i_0}$ and so on):

$$\begin{array}{l} (i, \perp) \xrightarrow{u_1/v_1} (p, \sigma_1) \xrightarrow{u_2/v_2} (p, \sigma_1 \sigma'_1) \xrightarrow{(u_2)^{i_0} u_3(u_4)^{i_0} / (v_2)^{i_0} v_3(v_4)^{i_0}} (q, \sigma_1 \sigma'_1) \xrightarrow{u_4/v_4} (q, \sigma_1) \\ (i', \perp) \xrightarrow{u_1/w_1} (p', \sigma_2) \xrightarrow{u_2/w_2} (p', \sigma_2 \sigma'_2) \xrightarrow{(u_2)^{i_0} u_3(u_4)^{i_0} / (w_2)^{i_0} w_3(w_4)^{i_0}} (q', \sigma_2 \sigma'_2) \xrightarrow{u_4/w_4} (q', \sigma_2) \end{array}$$

Conversely, suppose that the conditions (1), (2) and (3) are satisfied. If $|v_2 v_4| \neq |w_2 w_4|$, then clearly, the set $\{\Delta(v_1(v_2)^i v_3(v_4)^i, w_1(w_2)^i w_3(w_4)^i) \mid i \geq 0\}$ is infinite, and therefore there exist i_0, i_1 such that $i_0 < i_1$ and $\Delta(v_1(v_2)^{i_0} v_3(v_4)^{i_0}, w_1(w_2)^{i_0} w_3(w_4)^{i_0})$ is different from $\Delta(v_1(v_2)^{i_1} v_3(v_4)^{i_1}, w_1(w_2)^{i_1} w_3(w_4)^{i_1})$. Then, the following decomposition witnesses the non-satisfiability of the MTP:

$$\begin{array}{l}
(i, \perp) \xrightarrow{u_1/v_1} (p, \sigma_1) \xrightarrow{u_2^{i_1-i_0}/v_2^{i_1-i_0}} (p, \sigma_1(\sigma'_1)^{i_1-i_0}) \xrightarrow{u_2^{i_0}u_3u_4^{i_0}/v_2^{i_0}v_3v_4^{i_0}} (q, \sigma_1(\sigma'_1)^{i_1-i_0}) \xrightarrow{u_4^{i_1-i_0}/v_4^{i_1-i_0}} (q, \sigma_1) \\
(i', \perp) \xrightarrow{u_1/w_1} (p', \sigma_2) \xrightarrow{u_2^{i_1-i_0}/w_2^{i_1-i_0}} (p', \sigma_2(\sigma'_2)^{i_1-i_0}) \xrightarrow{u_2^{i_0}u_3u_4^{i_0}/w_2^{i_0}w_3w_4^{i_0}} (q', \sigma_2(\sigma'_2)^{i_1-i_0}) \xrightarrow{u_4^{i_1-i_0}/w_4^{i_1-i_0}} (q', \sigma_2)
\end{array}$$

Now, suppose that $|v_2v_4| = |w_2w_4|$, $|v_2v_4w_2w_4| \neq 0$ and there is a mismatch between v_1v_3 and w_1w_3 , i.e. $v_1v_3 = x\alpha v$ and $w_1w_3 = x\beta w$ for some $x, v, w \in \Sigma^*$ and some $\alpha \neq \beta \in \Sigma$. Then $\Delta(v_1v_3, w_1w_3) = (\alpha v, \beta w)$. Suppose that $\Delta(v_1(v_2)^i v_3(v_4)^i, w_1(w_2)^i w_3(w_4)^i) = (\alpha v, \beta w)$ for all $i \geq 0$. \square

Lemma 5.7. *The matched twinning property is decidable in CONPTIME for fVPTs.*

Proof. The proof is very similar to the proof of Proposition 4.6 for HTP. From an fVPT T , we also construct in polynomial time a pushdown automaton with a constant number of (one reversal) counters that accepts any word $u = u_1u_2u_3u_4u_5$ such that $u_1u_2u_3u_4$ satisfies the premise of the MTP but such that $\Delta(v_1v_3, w_1w_3) \neq \Delta(v_1v_2v_3v_4, w_1w_2w_3w_4)$ (i.e., the MTP is not verified). Therefore the MTP holds if and only if no word is accepted by the automaton, and this can be checked in CONPTIME [FRR⁺18].

As for HTP, the automaton simulates any two runs and guesses the decomposition $u_1u_2u_3u_4$. It checks that each run is in the same state after reading u_1 and u_2 , and in the same state after reading u_3 and u_4 . Using its stack, it verifies that u_3 and u_2u_4 are well-nested. With two additional counters it checks that $|v_2v_4| = |w_2w_4|$, if it is not the case then it accepts u (the MTP is not verified). Finally the automaton checks that $\Delta(v_1v_3, w_1w_3) \neq \Delta(v_1v_2v_3v_4, w_1w_2w_3w_4)$. This is done using the characterization given in Lemma 5.6 (item 3 in particular), with the technique described in the proof of Proposition 4.6. \square

5.4. Deciding OBM. We show in this section that the MTP and OBM coincide: any twinned fVPT can be evaluated with online bounded memory, and OBM transductions can only be realized by twinned fVPTs. In particular, this shows that being twinned is not only a property of the transducer, but also of the transduction it defines. Another consequence is that OBM is decidable for fVPTs.

Theorem 5.8. *Let T be an fVPT. $\llbracket T \rrbracket$ is OBM iff the T is twinned, which is decidable in CONPTIME. In this case, the Turing transducer $M_{\text{LCPIN}}(T)$ runs, on an input stream u , in space complexity quadratic in the height of u .*

Proof. Using Proposition 5.10 and Proposition 5.11 (both proved in this section), an fVPT T is twinned iff $\llbracket T \rrbracket$ is OBM. The former is decidable in CONPTIME by Lemma 5.7. Proposition 3.1 proves that the algorithm LCPIN uses at most $O((\text{hc}(u') + 1) \cdot \text{out}_{\neq}(u'))$ space on the working tape after reading a prefix u' of u . Proposition 5.10 shows that

$$\text{out}_{\neq}(u') \leq (\text{hc}(u') + 1) \cdot \left(3(|Q|^{2(|Q|^4+1)} - 1) \right) \cdot M$$

when T is twinned. Thus the space used by the algorithm is quadratic in $\text{hc}(u')$. \square

Twinned fVPTs define a class of transductions (and not just a class of transducers):

Corollary 5.9. *Let T, T' be two equivalent fVPTs. Then T is twinned iff T' also is.*

Proof. By Theorem 5.8, T is twinned iff $\llbracket T \rrbracket$ is OBM iff $\llbracket T' \rrbracket$ is OBM iff T' is twinned. \square

Proposition 5.10. *If an fVPT T is twinned then $\llbracket T \rrbracket$ is in OBM.*

Proof. Let T be a twinned fVPT. We show that, for all input words u and all prefixes u' of u ,

$$\text{out}_{\neq}(u') \leq (\text{hc}(u') + 1) \cdot \left(3(|Q|^{2(|Q|^4+1)} - 1)\right) \cdot M,$$

where M is the length of the longest output occurring on the transitions of T . Using Proposition 3.1, this shows that $\llbracket T \rrbracket$ is in OBM.

Let $u \in \text{Dom}(T)$, and u' be a prefix of u . There exists a unique decomposition of u' as follows: $u' = u_0 c_1 u_1 c_2 \dots u_{n-1} c_n u_n$, where $n = \text{hc}(u')$ (the current height of u'), and for any i , $c_i \in \Sigma_c$ and u_i is well-nested. If each of the u_i 's is such that $|u_i| \leq 3(|Q|^{2(|Q|^4+1)} - 1)$, then the property holds as the length of u' can be bounded by

$$(\text{hc}(u') + 1) \cdot \left(3(|Q|^{2(|Q|^4+1)} - 1)\right).$$

Otherwise, we prove that there exists a strictly shorter input word that produces the same delays as u' when evaluating the transduction on it. If $|Q| = 1$ then we can apply the HTP (implied by the MTP) on any of the u_i to show that removing it would not change the delays (as in the proof of Lemma 4.10). If every u_i is empty, then $|u'| = \text{hc}(u')$ and we get the result. Now, assume that $|Q| > 1$. Let $(q, \sigma, w), (q', \sigma', w') \in Q \times \Gamma^* \times \Sigma^*$ be such that there exist runs $\rho : (q_0, \perp) \xrightarrow{u'/v} (q, \sigma)$ and $\rho' : (q'_0, \perp) \xrightarrow{u'/v'} (q', \sigma')$, with $q_0, q'_0 \in I$, $v = \text{lcp}_{\text{in}}(u', T) \cdot w$, $v' = \text{lcp}_{\text{in}}(u', T) \cdot w'$, and such that $\text{out}_{\neq}(u') = |w|$. Consider the smallest index i such that $|u'_i| > 3(|Q|^{2(|Q|^4+1)} - 1)$. We distinguish two cases:

- (1) if $\text{h}(u_i) \leq |Q|^4$, then u'_i is not $|Q|^2$ -narrow. Indeed, if it was, we could apply Lemma 4.9 (as $|Q| > 1$) and get $|u_i| \leq 3(|Q|^{2(\text{h}(u_i)+1)} - 1) \leq 3(|Q|^{2(|Q|^4+1)} - 1)$, a contradiction. So u_i is not $|Q|^2$ -narrow. In this case we “pump horizontally” like in the proof of Lemma 4.10 as there exists a well-nested factor u'' from which we can remove a non-empty factor while preserving the delays.
- (2) if $\text{h}(u_i) > |Q|^4$, we prove that we can “pump vertically” u_i , and thus reduce its length too. Indeed, let k be the first position in word u_i at which height $\text{h}(u_i)$ is obtained. As u_i is well-nested, we can define for each $0 \leq j < \text{h}(u_i)$ the unique position $\text{left}(j)$ (resp. $\text{right}(j)$) of u_i as the largest index, less than k (resp. the smallest index, larger than k), whose height is j (see Figure 6). As $\text{h}(u_i) > |Q|^4$, there exist two heights j and j' such that configurations reached at positions $\text{left}(j)$, $\text{left}(j')$, $\text{right}(j)$ and $\text{right}(j')$ in runs ρ and ρ' satisfy the premises of the matched twinning property, considering a prefix of $u_0 c_1 \dots c_i u_i$. Thus, one can replace in this prefix u_i by a shorter word u'_i and hence reduce its length, while preserving the delays reached after it. Let u'' be the word obtained from u' by substituting u'_i to u_i , hence $|u''| < |u'|$. By Lemma 1.1, this entails that the delays reached after u' and u'' are the same, proving the result. \square

Proposition 5.11. *Let T be an fVPT. If $\llbracket T \rrbracket$ is in OBM then T is twinned.*

Proof. Consider a VPT T that does not satisfy the MTP, and assume for contradiction that there exists an OBM Turing transducer A computing the transduction of T . As T does not satisfy the MTP, we can find two runs as in the definition of the MTP, that accumulate different delays:

$$\begin{array}{cccccccc} (i, \perp) & \xrightarrow{u_1/v_1} & (p, \sigma_1) & \xrightarrow{u_2/v_2} & (p, \sigma_1 \sigma'_1) & \xrightarrow{u_3/v_3} & (q, \sigma_1 \sigma'_1) & \xrightarrow{u_4/v_4} & (q, \sigma_1) \\ (i', \perp) & \xrightarrow{u_1/w_1} & (p', \sigma_2) & \xrightarrow{u_2/w_2} & (p', \sigma_2 \sigma'_2) & \xrightarrow{u_3/w_3} & (q', \sigma_2 \sigma'_2) & \xrightarrow{u_4/w_4} & (q', \sigma_2) \end{array}$$

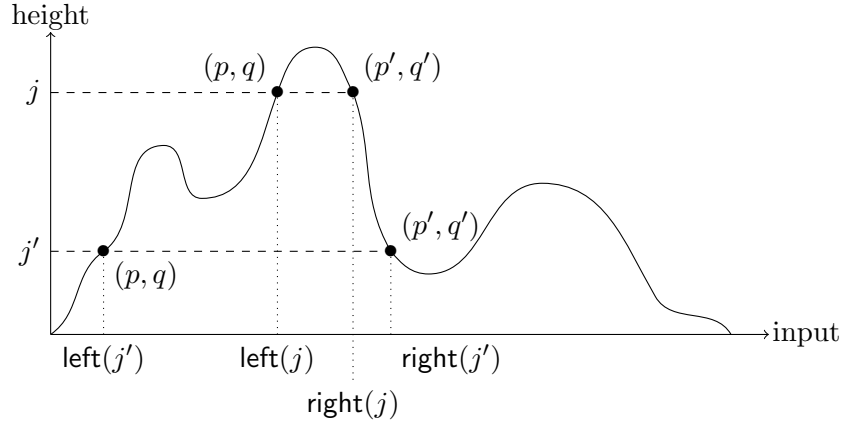


Figure 6: Vertical pumping in a well-nested word

and $\Delta(v_1v_3, w_1w_3) \neq \Delta(v_1v_2v_3v_4, w_1w_2w_3w_4)$.

Remind that in the definition of MTP, we require that (q, σ_1) and (q', σ_2) are co-accessible configurations. Therefore there exists two continuations u_5 and u'_5 that lead to accepting configurations from these two configurations respectively. Assume that the runs on u_5 and u'_5 produce two output words v_5 and w_5 respectively.

For all $\alpha \in \mathbb{N}$, we let

$$\begin{aligned} U_\alpha &= u_1u_2^\alpha u_3u_4^\alpha \\ V_\alpha &= v_1v_2^\alpha v_3v_4^\alpha \\ W_\alpha &= w_1w_2^\alpha w_3w_4^\alpha \end{aligned}$$

Note that $U_\alpha u_5$ and $U_\alpha u'_5$ are both accepted for all α , and their respective outputs are $V_\alpha v_5$ and $W_\alpha w_5$.

Now, after reading U_α the stack height $h \in \mathbb{N}$ is always the same for all α . Since A is OBM, the amount of information written on its working tape after reading U_α only depends on h . Hence, there exists a pair (c, q) where c is a word written on the working tape of A and q is a state of A , such that for infinitely many α , A is in the configuration (c, q) after reading U_α . Let Ξ denote the set of such α 's.

We denote by V the output produced by A on u_5 from the configuration (c, q) , and by W the output produced by A on u'_5 from (c, q) . For all α , we also denote by O_α the output of A on U_α (it is unique since the machine is deterministic). Therefore, for all $\alpha \in \Xi$, we have:

$$\begin{aligned} \llbracket T \rrbracket(U_\alpha u_5) &= O_\alpha V = V_\alpha v_5 \\ \llbracket T \rrbracket(U_\alpha u'_5) &= O_\alpha W = W_\alpha w_5 \end{aligned}$$

Hence $O_\alpha = V_\alpha v_5 V^{-1} = W_\alpha w_5 W^{-1}$ for all $\alpha \in \Xi$. We show that it implies that there exists a delay d and infinitely many α such that $\Delta(V_\alpha, W_\alpha) = d$, which contradicts Lemma 5.5 below (applied to $m = n = 2$) since $\Delta(V_0, W_0) \neq \Delta(V_1, W_1)$ by our initial assumption.

Suppose that it is not the case, then we can find an arbitrarily large delay between V_α and W_α . We then choose α such that $\Delta(V_\alpha, W_\alpha) = (x, y)$ with $\max(|x|, |y|) > |V| + |W| + |v_5| + |w_5|$. It means that $V_\alpha = \ell x$ and $W_\alpha = \ell y$ for some ℓ . Then, we know that $V_\alpha v_5 V^{-1} = W_\alpha w_5 W^{-1}$, hence $xv_5 V^{-1} = yw_5 W^{-1}$. We also have $|x| + |v_5| - |V| \leq |xv_5 V^{-1}| \leq |x| + |v_5| + |V|$, and $|y| + |w_5| - |W| \leq |yw_5 W^{-1}| \leq |y| + |w_5| + |W|$. We distinguish three cases:

- $x = \epsilon$: then $|y| + |w_5| - |W| \leq |yw_5W^{-1}| = |xv_5V^{-1}| \leq |v_5| + |V|$, and $|y| \leq |v_5| + |V| + |W| - |w_5|$ which contradicts $|y| > |v_5| + |V| + |W| + |w_5|$,
- $y = \epsilon$: this case is symmetric to the previous one,
- $x = ax'$ and $y = by'$ for some letters $a \neq b$: then we have $ax'v_5V^{-1} = by'w_5W^{-1}$, which implies that either the first a is “erased” by V^{-1} or the first b is erased by W^{-1} , but it cannot be the case that both these letters are erased due to the length of x and y . Suppose that a is erased by V^{-1} , then $ax'v_5V^{-1}$ is of the form β^{-1} for some β (it is an inverse word), while $by'w_5W^{-1}$ is not. The other case is symmetric. \square

Remark 5.12 (Sequential VPTs). Sequential transducers have at most one run per input word, so *sequentializable VPTs are twinned*. The MTP is not a sufficient condition to be sequentializable, as shown for instance by Example 5.4. Therefore the class of transductions defined by transducers which satisfy the MTP is strictly larger than the class of transductions defined by sequentializable transducers. However, these transductions are in the same complexity class for evaluation, i.e., polynomial space in the height of the input word for a fixed transducer.

6. CONCLUSION AND REMARKS

This work investigates the streaming evaluation of nested word transductions defined by visibly pushdown transducers. The main result is the introduction of two classes of VPT-transductions, shown to be decidable (in the class of VPT-transductions): VPT-transductions which can be evaluated in streaming with a memory that depends only on the height of the input nested word (HBM) and on the current height of the prefixes of the input nested word (OBM), respectively. These two classes have been effectively characterized by structural properties of VPTs, respectively called horizontal and matched twinning properties. We have designed a streaming algorithm to evaluate VPT in general and analysed its space complexity. This algorithm, applied to a VPT satisfying the horizontal twinning property, runs in height bounded memory. Applied to a VPT satisfying the matched twinning property, it runs in online bounded memory.

The following inclusions summarize the relations between the different *classes* of transductions we have studied:

$$\text{BM fVPTs} \subsetneq \text{Sequentializable VPTs} \subsetneq \text{OBM fVPTs} \subsetneq \text{HBM fVPTs} \subsetneq \text{fVPTs}$$

Moreover, we have shown that BM, OBM and HBM fVPTs are decidable in CONPTIME.

Further Directions. An important asset of the class of OBM fVPTs w.r.t. the class of sequentializable VPTs is that it is decidable. It would thus be interesting to determine whether or not the class of sequentializable VPTs is decidable, and to characterize the class of sequentializable VPTs in terms of memory requirements. In addition, we also plan to extend our techniques to more expressive transducers, such as two-way visibly pushdown transducers as introduced in [DFRT16], which are equivalent to MSO-transducers from nested words to words. For (flat) words, deciding bounded memory of a transduction given by a finite transducer amounts to decide whether it is sequentializable. If the transduction is given by a two-way (flat) word transducer, or equivalently by an MSO-transducer [EH01], deciding bounded memory can be done by first checking whether the transduction is rational, i.e., whether it is realizable by a (one-way) finite state transducer, and then by deciding

sequentializability of the one-way transducer. The first step has been shown to be decidable in [FGRS13], with an elementary complexity in [BGMP17].

To extend the result of this paper to other models of transducers, say two-way visibly pushdown transducers, we plan to extend the result of [FGRS13] to nested words. I.e., given a two-way visibly pushdown transducer, decide whether it is equivalent to some VPT.

Another line of work concerns the extension of our evaluation procedure beyond functional transductions, or to multi-input and multi-output transductions.

ACKNOWLEDGEMENT

The authors would like to thank Jean-François Raskin and Stijn Vansummeren for their comments on a preliminary version of this work.

REFERENCES

- [AKL10] Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. *ACM Trans. Algorithms*, 6(2):28:1–28:36, 2010.
- [AM09] Rajeev Alur and P. Madhusudan. Adding nesting structure to words. *J. ACM*, 56(3):16:1–16:43, 2009.
- [BC02] Marie-Pierre Béal and Olivier Carton. Determinization of transducers over finite and infinite words. *Theor. Comput. Sci.*, 289(1):225–251, 2002.
- [BCF⁺07] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML query language, W3C recommendation, 2007.
- [BCPS03] Marie-Pierre Béal, Olivier Carton, Christophe Prieur, and Jacques Sakarovitch. Squaring transducers: An efficient procedure for deciding functionality and sequentiality. *Theor. Comput. Sci.*, 292(1):45–63, 2003.
- [BGMP17] Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. Untwisting two-way transducers in elementary time. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, (LICS)*. ACM, 2017.
- [BJ07] Michael Benedikt and Alan Jeffrey. Efficient and expressive tree filters. In *Proceedings of the 27th International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 4855 of *LNCS*, pages 461–472. Springer Verlag, 2007.
- [BLS06] Vince Bárány, Christof Löding, and Olivier Serre. Regularity problems for visibly pushdown languages. In *Proceedings of the 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 420–431. Springer Berlin Heidelberg, 2006.
- [BLS11] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20:14:1–14:64, 2011.
- [BMV06] Denilson Barbosa, Laurent Mignet, and Pierangelo Veltri. Studying the XML web: Gathering statistics from an XML sample. *World Wide Web*, 9(2):187–212, 2006.
- [BYFJ05] Ziv Bar-Yossef, Marcus Fontoura, and Vanja Josifovski. Buffering in query evaluation over XML streams. In *Proceedings of the 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 216–227. ACM-Press, 2005.
- [Cho77] Christian Choffrut. Une Caractérisation des Fonctions Séquentielles et des Fonctions Sous-Séquentielles en tant que Relations Rationnelles. *Theor. Comput. Sci.*, 5(3):325–337, 1977.
- [Cla99] James Clark. XSL Transformations (XSLT) version 1.0, W3C recommendation, 1999.
- [CRT15] Mathieu Caralp, Pierre-Alain Reynier, and Jean-Marc Talbot. Trimming visibly pushdown automata. *Theor. Comput. Sci.*, 578(C):13–29, 2015.
- [DFRT16] Luc Dartois, Emmanuel Filiot, Pierre-Alain Reynier, and Jean-Marc Talbot. Two-way visibly pushdown automata and transducers. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, (LICS)*, pages 217–226. ACM, 2016.
- [EH01] Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.*, 2(2):216–254, 2001.

- [FGRS11] Emmanuel Filiot, Olivier Gauwin, Pierre-Alain Reynier, and Frédéric Servais. Streamability of nested word transductions. In *Proceedings of the IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 13 of *LIPIcs*, pages 312–324. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.
- [FGRS13] Emmanuel Filiot, Olivier Gauwin, Pierre-Alain Reynier, and Frédéric Servais. From two-way to one-way finite state transducers. In *Proceedings of the 28th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 468–477. IEEE Computer Society, 2013.
- [FMdRS16] Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre. Streaming Property Testing of Visibly Pushdown Languages. In Piotr Sankowski and Christos Zaroliagis, editors, *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:17. Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
- [FRR⁺10] Emmanuel Filiot, Jean-François Raskin, Pierre-Alain Reynier, Frédéric Servais, and Jean-Marc Talbot. On functionality of visibly pushdown transducers. *CoRR*, abs/1002.1443, 2010.
- [FRR⁺18] Emmanuel Filiot, Jean-François Raskin, Pierre-Alain Reynier, Frédéric Servais, and Jean-Marc Talbot. Visibly pushdown transducers. *J. Comput. Syst. Sci.*, 97:147–181, 2018.
- [GHL18] Moses Ganardi, Danny Hucce, and Markus Lohrey. Randomized sliding window algorithms for regular languages. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 127:1–127:13, 2018.
- [GJL18] Moses Ganardi, Artur Jez, and Markus Lohrey. Sliding windows over context-free languages. In *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, pages 15:1–15:15, 2018.
- [GKS07] Martin Grohe, Christoph Koch, and Nicole Schweikardt. Tight lower bounds for query processing on streaming and external memory data. *Theor. Comput. Sci.*, 380:199–217, July 2007.
- [GNT09] Olivier Gauwin, Joachim Niehren, and Sophie Tison. Earliest query answering for deterministic nested word automata. In *Proceedings of the 17th International Symposium on Fundamentals of Computation Theory (FCT)*, volume 5699 of *LNCS*, pages 121–132. Springer Berlin Heidelberg, 2009.
- [KM13] Christian Konrad and Frédéric Magniez. Validating XML documents in the streaming model with external memory. *ACM Trans. Database Syst.*, 38(4):27:1–27:36, 2013.
- [KMV07] Viraj Kumar, P. Madhusudan, and Mahesh Viswanathan. Visibly pushdown automata for streaming XML. In *Proceedings of the 16th international conference on World Wide Web (WWW)*, pages 1053–1062. ACM-Press, 2007.
- [KV01] Orna Kupferman and Moshe Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.
- [MV09] P. Madhusudan and Mahesh Viswanathan. Query automata for nested words. In *Proceedings of the 34th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 5734 of *LNCS*, pages 561–573. Springer Berlin Heidelberg, 2009.
- [RCD⁺] Jonathan Robie, Don Chamberlin, Michael Dyck, Daniela Florescu, Jim Melton, and Jérôme Siméon. XQuery Update Facility 1.0, W3C Recommendation 17 March 2011.
- [Saa15] Aleksi Saarela. Systems of word equations, polynomials and linear algebra: A new approach. *Eur. J. Comb.*, 47:1–14, 2015.
- [Sak09] Jacques Sakarovich. *Elements of Automata Theory*. Cambridge University Press, Cambridge, England, 2009.
- [SS07] Luc Segoufin and Cristina Sirangelo. Constant-memory validation of streaming XML documents against DTDs. In *Proceedings of the 11th International Conference on Database Theory (ICDT)*, pages 299–313. Springer Berlin Heidelberg, 2007.
- [Ste67] Richard E. Stearns. A regularity test for pushdown machines. *Information and Control*, 11(3):323–340, 1967.
- [WK95] Andreas Weber and Reinhard Klemm. Economy of description for single-valued transducers. *Inf. Comput.*, 118(2):327–340, 1995.