Streamability of Nested Word Transductions

Emmanuel Filiot[1], Olivier Gauwin[2], Pierre-Alain Reynier[3], Frédéric Servais[1]

[1]Université Libre de Bruxelles [2]Université de Mons [3]Université de Provence

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 NPTIME 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.

1 Introduction

Memory analysis is an important tool for ensuring system robustness. In this paper we focus on the analysis of programs processing *nested words* [2], *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 [18, 6]. On XML streams, some algorithms outputting nodes selected by an XPath expression at the earliest possible event have also been proposed [7, 13]. These algorithms allow minimal buffering [3].

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 [11] 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 [11], XQuery [8] and XQuery Update Facility [20], 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 [2] 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 [20]) 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 [12].

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. This transformation requires to keep the entire word in memory until we can verify that the guess was correct. 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 [22]. 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 [5]. 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 polynomial in the size of the transducer, linear in the height of the input word, and linear in the maximal length of a remaining output candidate.

We prove that BM is equivalent to subsequentiability 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 NPTIME.

Like BM, HBM is undecidable for arbitrary pushdown transductions. We show that it is decidable in NPTIME 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 [10] 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 thus refine VPT-transductions to *twinned transductions* for which performing the transformation with our algorithm uses a memory *quadratic* in the height of the input word. This class is characterized by a twinning property that takes the height of the configurations into account. A VPT satisfying this twinning property is called *twinned*. We show, via a non-trivial reduction to the emptiness of pushdown automata with bounded reversal counters, that it is

decidable in NPTIME whether a VPT is twinned. Moreover, the most challenging result of this paper is to show that being twinned depends only on the transduction and not on the VPT that defines it. Thus, this property indeed defines a class of transductions. As a consequence of this result, all subsequentializable VPTs are twinned, because subsequential VPTs trivially satisfy the twinning property. The class of twinned transductions captures a strictly larger class than subsequentializable VPTs while staying in the same complexity class for evaluation, i.e. polynomial space in the height of the input word when the transducer is fixed.

Related Work In the XML context, visibly pushdown automata based streaming processing has been extensively studied for validating XML streams [17, 22, 4, 21]. The validation problem with bounded memory is studied in [4] when the input is assumed to be a well-nested word and in [22, 21] when it is assumed to be a well-formed XML document (this problem is still open). Querying XML streams has been considered in [14]. 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 [3]. VPTs also relate to tree transducers [12], for which no comparable work on memory requirements is known. When allowing two-way access on the input stream, more space-efficient algorithms for XML validation [16] and querying [19] have been proposed.

2 Visibly Pushdown Languages and Transductions

Words and nested words In this paper, we consider nested words accessed in streaming. Their nesting structure is thus discovered on-the-fly, so we consider a finite alphabet Σ partitioned into three disjoint sets Σ_c, Σ_r and Σ_ι, 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 $|u|$. 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 $\mathsf{lcp}(V)$, is inductively defined by $\mathsf{lcp}(\{u\}) = u$ and $\mathsf{lcp}(V \cup \{u\}) = \mathsf{lcp}(V) \wedge u$. The set of *well-nested* words Σ^*_{wn} is the smallest subset of Σ^* such that $\Sigma_\iota^* \subseteq \Sigma^*_{\mathsf{wn}}$ and for all $c \in \Sigma_c$, all $r \in \Sigma_r$, all $u, v \in \Sigma^*_{\mathsf{wn}}$, $cur \in \Sigma^*_{\mathsf{wn}}$ and $uv \in \Sigma^*_{\mathsf{wn}}$. Let $u = a_1 \dots a_n \in \Sigma^*$ be a prefix of a well-nested word. A position $i \in \{1, \dots, n\}$ is a *pending call* if $a_i \in \Sigma_c$ and for all $j \geq i$, $a_i \dots a_j \notin \Sigma^*_{\mathsf{wn}}$. The *height* of u is the maximal number of pending calls on any prefix of u, i.e.

$$h(u) = \max_{1 \leq i \leq n} |\{k \mid 1 \leq k \leq i, a_k \text{ is a pending call of } a_1 \dots a_i\}|$$

For instance, $h(crcrcc) = h(ccrcrr) = 2$. 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 have to keep in memory.

Visibly pushdown transducers (VPTs) As finite-state transducers extend finite-state automata with outputs, visibly pushdown transducers extend visibly pushdown automata [2] with outputs [12]. 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 the stack behavior of visibly pushdown automata (VPA). 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. Formally:

DEFINITION 1. *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, $\delta = \delta_c \uplus \delta_r \uplus \delta_\iota$ 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 $\delta_\iota \subseteq Q \times \Sigma_\iota \times \Sigma^* \times Q$.*

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 $\rho = \{(q_k,\sigma_k)\}_{0\le k\le l}$ such that $q_0 = q$, $\sigma_0 = \sigma$, $q_l = q'$, $\sigma_l = \sigma'$ and for each $1 \le k \le l$, 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 $(q_{k-1}, a_k, v_k, q_k) \in \delta_\iota$ 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 $(q,\sigma) \xrightarrow{u/v} (q_f,\perp)$). 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 [9]*. In this paper, we assume all VPTs to be reduced. A transducer T defines the binary word relation $[\![T]\!] = \{(u,v) \in \Sigma^* \times \Sigma^* \mid \exists q \in I, q' \in F,\ (q,\perp) \xrightarrow{u/v} (q',\perp)\}$.

A *transduction* is a binary relation $R \subseteq \Sigma^* \times \Sigma^*$. We say that a transduction R is a VPT-transduction if there exists a VPT T such that $R = [\![T]\!]$. For any input word $u \in \Sigma^*$, we denote by $R(u)$ the set $\{v \mid (u,v) \in R\}$. Similarly, for a VPT T, we denote by $T(u)$ the set $[\![T]\!](u)$. 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. A VPT T is functional if $[\![T]\!]$ is functional, and this can be decided in PTIME [12]. The class of functional VPTs is denoted by fVPT. The *domain* of T (denoted by $Dom(T)$) is the domain of $[\![T]\!]$. Note that the domain of T contains only well-nested words, which is not necessarily the case of the codomain.

EXAMPLE 2. *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 $Dom(T) = \{c^n r^n \mid n \ge 2\} \cup \{cc^n r^n r' \mid n \ge 1\}$. 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'. This transformation cannot be evaluated with a bounded amount of memory, but with a memory which depends on the height n of the input word.*

*The reduction of VPAs in [9] trivially extends to VPTs.

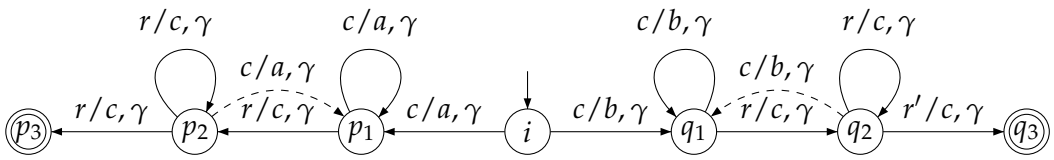


Figure 1: A functional VPT with $\Sigma_c = \{c\}$, $\Sigma_r = \{r, r'\}$ and $\Sigma_\iota = \{a, b\}$

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, as there is no ϵ-transitions. We always consider real-time FSTs in this paper, so we just call them FSTs.

A *subsequential* FST (resp. VPT) is a pair (T, Ψ) where T is an (input) deterministic FST (resp. VPT) and $\Psi : F \to \Sigma^*$. The outputs of u by (T, Ψ) are the words $v.\Psi(q)$ whenever there is a run of T on u producing v and ending up in some accepting state q.

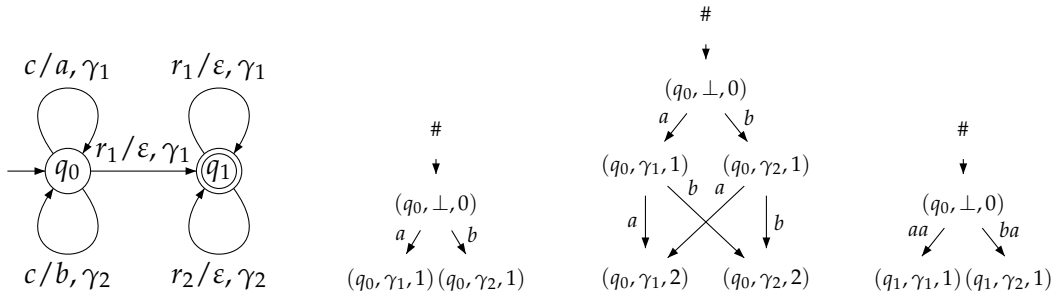
Given an integer $k \in \mathbb{N}$ and a VPT T, one can define an FST, denoted by $\mathsf{FST}(T, k)$, which is the restriction of T to input words of height less than k. The transducer is naturally constructed by taking as states the configurations (q, σ) of T such that $|\sigma| \leq k$.

3 Online Evaluation Algorithm of VPT-Transductions

We present an online algorithm LCPIN to evaluate functional word transductions defined by fVPTs. 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 $\$ \notin \Sigma$, denoting the end of the word. Second, we only consider input words without internal symbols, as they can easily be encoded by successive call and return symbols. Third, input words are supposed to be valid, in the sense that they produce an output. It is indeed easy to extend our algorithms in order to raise an error message when the input is not in the domain: this happens when no run of the VPT applies on the input.

The core task of this algorithm is to maintain the configuration for each run of the fVPT T on the input u, and produce its output on-the-fly. As 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 $(q_i, \bot) \xrightarrow{u/v} (q, \sigma)$ with $q_i \in I$. Hence, a configuration is a triple (q, σ, w) where q is the current state of the run, σ its corresponding stack content, and w the part of the output that has been read but not output yet. We call such a configuration *d-configuration* and write $\mathsf{Dconfs}(T) = Q \times \Gamma^* \times \Sigma^*$ for the set of d-configurations of T. Algorithm LCPIN relies on two main features.

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, the transition used to update a d-configuration relates the stack symbol and the output word. 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 2(d), this structure is a directed acyclic graph (DAG). Nodes of this DAG 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 is labelled with a word, so that a branch of



(a) VPT T_1. (b) After reading c. (c) After reading cc. (d) After reading ccr_1.

Figure 2: Data structure used by LCPIN.

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 $\# \to (q_0, \bot, 0) \xrightarrow{b} (q_0, \gamma_2, 1) \xrightarrow{a} (q_0, \gamma_1, 2)$ encodes the d-configuration $(q_0, \gamma_2\gamma_1, ba)$ of the VPT of Fig. 2.(a). However, this data structure cannot store any set of accessible d-configurations of arbitrary functional VPTs: at most one delay w has to be assigned to a d-configuration. Thus we assume all fVPTs to be reduced in this paper.

Computing outputs Second, after reading a prefix u' of a word u, LCPIN will have output the common prefix of all corresponding runs, i.e. $\mathsf{lcp}_{\mathsf{in}}(u', T) = \mathsf{lcp}(\mathsf{reach}(u'))$ where $\mathsf{reach}(u') = \{v \mid \exists (q_0, q, \sigma) \in I \times Q \times \Gamma^*,\ (q_0, \bot) \xrightarrow{u'/v} (q, \sigma)\}$. When a new input symbol is read, the DAG is first updated. Then, a bottom-up pass on this DAG computes $\mathsf{lcp}_{\mathsf{in}}(u', T)$ in the following way. For each node, let ℓ be the largest 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. At the end, the largest common prefix of all output words on branches is the largest common prefix of words on edges outgoing from the root node #.

Let $\mathsf{out}_{\neq}(u')$ be the maximal size of outputs of T on u' where their common prefix is removed: $\mathsf{out}_{\neq}(u') = \max_{v \in \mathsf{reach}(u')} |v| - |\mathsf{lcp}_{\mathsf{in}}(u', T)|$ and $\mathsf{out}^{\max}_{\neq}(u)$ its maximal value over prefixes of u: $\mathsf{out}^{\max}_{\neq}(u) = \max_{u' \text{ prefix of } u} \mathsf{out}_{\neq}(u')$. We prove the following complexity results for LCPIN:

PROPOSITION 3. *Let T be an* fVPT, *and $u \in \Sigma^*$. The space used by* LCPIN *for computing $T(u)$ is in $O(|Q|^2 \cdot |\Gamma|^2 \cdot (h(u)+1) \cdot \mathsf{out}^{max}_{\neq}(u))$, and processing each symbol of u is in time polynomial in $|Q|$, $|\Gamma|$, $|\delta|$, $h(u)$, $\mathsf{out}^{max}_{\neq}(u)$ and $|\Sigma|$.*

4 Bounded Memory Evaluation Problems

In this section, we consider two classes of transductions: bounded memory and height bounded memory transductions. Intuitively, the first one corresponds to transductions whose evaluation is in constant memory if we fix the machine that defines the transduction, while for the second one the evaluation is in constant memory if we fix both the machine and the height of the input word.

Turing Transducers In order to formally define the complexity classes we target, we introduce a *deterministic* computational model for word transductions that we call *Turing Transducers (TT)*.

Turing transducers have three tapes: one read-only left-to-right input tape, one write-only left-to-right output tape, and one standard working tape. Such a machine naturally defines a transduction: the input word is initially on the input tape, and the result of the transduction is the word written on the output tape after the machine terminates in an accepting state. We denote by $[\![M]\!]$ the transduction defined by M. The space complexity is measured on the working tape only.

Bounded Memory Transductions We first consider transductions that can be evaluated with a constant amount of memory:

DEFINITION 4. *A (functional) transduction $R \subseteq \Sigma^* \times \Sigma^*$ is* bounded memory (BM) *if there exists a Turing transducer M and $K \in \mathbb{N}$ such that $[\![M]\!] = R$ and on any input word $u \in \Sigma^*$, M runs in space complexity at most K.*

It is not difficult (see Appendix B) to verify that for FST-transductions, bounded memory is characterized by subsequentializability, which is decidable in PTIME [23]. Moreover, BM is undecidable for pushdown transducers, since it is as difficult as deciding whether a pushdown automaton defines a regular language. 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 5. *Let T be a functional* VPT *with n states.*

1. *$[\![T]\!]$ is BM iff (i) for all $u \in Dom(T)$, $h(u) \leq n^2$, and (ii) $\mathsf{FST}(T, n^2)$ is BM;*
2. *It is decidable in* NPTIME *whether $[\![T]\!]$ is BM.*

PROOF. [Sketch] The first assertion is obvious by using simple pumping techniques to show that bounded memory implies bounded height. In the sequel, we define the class of height bounded memory transductions, and show it is decidable in NPTIME. On words of bounded height, this class collapses with bounded memory transductions. ∎

Height Bounded Memory Transductions As we have seen, bounded memory is too restrictive to still benefit from the extra expressiveness of VPT compared to FST, namely the ability to recognize nested words of unbounded height. In this section, we define a notion of bounded memory which is well-suited to VPTs.

DEFINITION 6. *A (functional) transduction $R \subseteq \Sigma^* \times \Sigma^*$ is* height bounded memory (HBM) *if there exists a Turing transducer M and a function $f : \mathbb{N} \to \mathbb{N}$ such that $[\![M]\!] = R$ and on any input word $u \in \Sigma^*$, M runs in space at most $f(h(u))$.*

Note that this definition ensures that the machine cannot store all the input words on the working tape in general. The VPT in Fig. 2(a) is not in BM, but is in HBM: the stack content suffices (and is necessary) to determine the output. When the structured alphabet contains only internal letters, HBM and BM coincides, thus it is undecidable whether a pushdown transducer is HBM. The remainder of this section is devoted to the proof that HBM is decidable for fVPTs.

BM functional FST-transductions (or equivalently subsequentializable FSTs) are characterized by the so called *twinning property* [10], which is decidable in PTIME [23]. We introduce a similar characterization of HBM fVPTs-transductions, called the *horizontal twinning property* (HTP). The restriction of the horizontal twinning property to FSTs is equivalent to the usual twinning property for FSTs (see Appendix C.1). Intuitively, the HTP requires that two runs on the same input cannot accumulate increasing output delay on loops.

DEFINITION 7. *Let T be an* fVPT. *T satisfies the* horizontal twinning property *(HTP) if for all $u_1, u_2, v_1, v_2, w_1, w_2 \in \Sigma^*$, for all $q_0, q_0' \in I$, for all $q, q' \in Q$, and for all $\sigma, \sigma' \in \Gamma^*$,*

$$\text{if } \begin{cases} (q_0, \bot) & \xrightarrow{u_1/v_1} & (q, \sigma) & \xrightarrow{u_2/v_2} & (q, \sigma) \\ (q_0', \bot) & \xrightarrow{u_1/w_1} & (q', \sigma') & \xrightarrow{u_2/w_2} & (q', \sigma') \end{cases} \quad (1) \quad \text{then } \Delta(v_1, w_1) = \Delta(v_1 v_2, w_1 w_2).$$

EXAMPLE 8. *Consider the VPT of Fig. 1 (including dashed arrows). It does not satisfy the HTP, as the delays increase when looping on crcr... Without the dashed transitions, the HTP is satisfied.*

LEMMA 9. *The HTP is decidable in* NPTIME *for* fVPTs.

PROOF. First, let us show that an fVPT T does not satisfy the HTP if and only if there exist $u_1, u_2, v_1, v_2, w_1, w_2 \in \Sigma^*$, $q_0, q_0' \in I$, $q, q' \in Q$, and $\sigma, \sigma' \in \Gamma^*$ that satisfy (1), and such that either we have (i) $|v_2| \neq |w_2|$, or (ii) $|v_2| = |w_2|$, $|v_1| \leq |w_1|$ and not $v_1 v_2 \preceq w_1 w_2$. Indeed, one can easily check that it is a necessary condition. To prove that it is a sufficient condition, suppose we have elements that satisfy (1) with $\Delta(v_1, w_1) \neq \Delta(v_1 v_2, w_1 w_2)$ but conditions (i) and (ii) do not hold. Wlog, we can assume that $|v_1| \leq |w_1|$, therefore we have $|v_2| = |w_2|$, $|v_1| \leq |w_1|$, $v_1 v_2 \preceq w_1 w_2$ and $\Delta(v_1, w_1) \neq \Delta(v_1 v_2, w_1 w_2)$. One can verify that there exists $k \in \mathbb{N}$ such that replacing u_2 with $u_2' = {u_2}^k$ yields a system that satisfies (ii). We refer the reader to Appendix C.2 for the details.

Second, let T be an fVPT, we define a pushdown automaton with bounded reversal counters [15], A, such that le language of A is empty if and only if T satisfies the HTP. More precisely, A accepts the words $u = u_1 u_2 u_3 \in \mathit{Dom}(T)$ such that there exist $v_1, v_2, w_1, w_2 \in \Sigma^*$, $q_0, q_0' \in I$, $q, q' \in Q$, and $\sigma, \sigma' \in \Gamma^*$ that satisfy (1) and either (i) or (ii). A simulates in parallel any two runs of T on the input word (product automaton). It guesses the end of u_1 and stores the states q and q' of the first and second run (in order to be able to check that the simulated runs of T are in state q, resp. q' after reading u_2). Non-deterministically, it checks whether (i) or (ii) holds. To check (i), it uses two counters, one for each run. It does so by, after reading u_1, increasing the counters by the length of the output word of each transition of the corresponding run. Then, when reaching the end of u_2 it checks that both counters are different (by decreasing in parallel both counters and checking they do not reach 0). Similarly, using two other counters, A checks that (ii) holds as follows. Note that condition (ii) implies that there is a position p such that the p-th letter a_1 of $v_1 v_2$ and the p-th letter a_2 of $w_1 w_2$ are different. The automaton A guesses the position $p \in \mathbb{N}$ of the mismatch, and initializes both counters to the value p. Then, while reading $u_1 u_2$, it decreases each counter by the length of the output words of the corresponding run. When a counter reaches 0, A stores the output letter of the corresponding run. Finally, A checks that $a_1 \neq a_2$. T satisfies the HTP iff the language of A is empty. The later is decidable in NPTIME [12]. ∎

We now show that HTP characterizes HBM fVPTs-transductions and therefore by Lemma 9 we get:

THEOREM 10. *Let T be an* fVPT. *It is decidable in* NPTIME *whether $[\![T]\!]$ is HBM. Moreover in this case, Algorithm* LCPIN *runs in space complexity $O(|Q|^4 \cdot |\Gamma|^{2h(u)+2} \cdot (h(u)+1) \cdot M)$ when evaluating u on T, where $M = \max\{|v| \mid (q, a, v, \gamma, q') \in \delta\}$.*

PROOF. [Sketch] We prove that $[\![T]\!]$ is HBM iff the HTP holds for T. To prove that the HTP is a necessary condition to be in HBM, we proceed by contradiction. We find a counter-example for the HTP and we let K be the height of the input word of this counter-example. It implies that the twinning property for FSTs does not hold for $\mathsf{FST}(T, K)$, and therefore $\mathsf{FST}(T, K)$ is not BM by Proposition 5. In particular, T is not HBM.

For the converse, we show that if the HTP holds for T, then for any input word $u \in \Sigma^*$, the maximal delay $\mathsf{out}^{\max}_{\neq}(u)$ between the outputs of u is bounded by $(|Q| \cdot |\Gamma|^{h(u)})^2 M$. This is done by a pumping technique "by width" that relies on the property $\Delta(vv', ww') = \Delta(\Delta(v,w) \cdot (v', w'))$ for any words v, v', w, w'. In particular for an input word for which there are two runs that pass by the same configurations twice at the same respective positions, the delay of the output is equal to the delay when removing the part in between the identical configurations. Finally we apply Proposition 3. ∎

HBM vs Subsequentializable fVPTs We have seen that a functional transduction defined by an FST T is BM iff T is subsequentializable. We give an example illustrating that for VPTs, being subsequentializable is too strong to characterize HBM. Consider the VPT of Fig. 1 defined by the plain arrows. The transduction it defines is in HBM by Proposition 3, as at any time the delay between two outputs is bounded by the height of the input: $\mathsf{out}^{\max}_{\neq}(u) \leq 2h(u)$. However it is not subsequentializable, as the transformation of c into a or b depends on the last return.

5 Quadratic Height Bounded Memory Evaluation

In the previous section, we have shown that a VPT-transduction is in HBM iff the horizontal twinning property holds, and if it is in HBM, the algorithm of Section 3 uses a memory at most exponential in the height of the word. We exhibit in Section 6 an fVPT proving that this exponential bound is tight. To avoid this exponential cost, we identify in this section a subclass of HBM containing transductions for which the evaluation algorithm of Section 3 uses a memory *quadratic in the height of the word*. Therefore, we strengthen the horizontal twinning property by adding some properties for well-matched loops. Some of our main and challenging results are to show the decidability of this property and that it depends only on the transduction, i.e. is preserved by equivalent transducers. We show that subsequential VPTs satisfy this condition and therefore our class *subsumes* the class of subsequentializable transducers.

The property we introduce is a strengthening of the horizontal twinning property that we call the *twinning property (TP)*. Intuitively, the TP 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.

DEFINITION 11. *Let $T = (Q, I, F, \Gamma, \delta)$ be an* fVPT*. T satisfies the* twinning property *(TP) 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 $\sigma_1, \sigma_2 \in \bot.\Gamma^*$, for all $\sigma_1', \sigma_2' \in \Gamma^*$,*

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

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* twinned *whenever it satisfies the TP.*

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

EXAMPLE 12. *The VPT of Fig. 1 with plain arrows does not satisfy the TP, as the delay between the two branches increases when iterating the loops. Consider now the VPT obtained by replacing r by r' in the transition (q_1, r, c, γ, q_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 TP. However this transducer is not subsequentializable, as the output on the call symbols cannot be delayed to the matching return symbols.*

As for the HTP, we can decide the TP using a reduction to the emptiness of a pushdown automaton with bounded reversal counters. A complete proof can be found in Appendix D.

LEMMA 13. *The twinning property is decidable in* NPTIME *for* fVPTs.

The most challenging result of this paper is to show that the TP only depends on the transduction and not on the transducer that defines it. The proof relies on fundamental properties of word combinatorics that allow us to give a general form of the output words $v_1, v_2, v_3, v_4, w_1, w_2, w_3, w_4$ involved in the TP, that relates them by means of conjugacy of their primitive roots. The proof gives a deep insight into the expressive power of VPTs which is also interesting on its own. As many results of word combinatorics, the proof is a long case study, so that we give it in Appendix D.2 only.

THEOREM 14. *Let T_1, T_2 be two* fVPTs *such that $[\![T_1]\!] = [\![T_2]\!]$. T_1 is twinned iff T_2 is twinned.*

PROOF. [Sketch] We assume that T_1 is not twinned and show that T_2 is not twinned either. By definition of the TP there are two runs of the form

$$\begin{cases} (i_1, \bot) \xrightarrow{u_1/v_1} (p_1, \sigma_1) \xrightarrow{u_2/v_2} (p_1, \sigma_1\beta_1) \xrightarrow{u_3/v_3} (q_1, \sigma_1\beta_1) \xrightarrow{u_4/v_4} (q_1, \sigma_1) \\ (i'_1, \bot) \xrightarrow{u_1/v'_1} (p'_1, \sigma'_1) \xrightarrow{u_2/v'_2} (p'_1, \sigma'_1\beta'_1) \xrightarrow{u_3/v'_3} (q'_1, \sigma'_1\beta'_1) \xrightarrow{u_4/v'_4} (q'_1, \sigma'_1) \end{cases}$$

such that $\Delta(v_1v_3, v'_1v'_3) \neq \Delta(v_1v_2v_3v_4, v'_1v'_2v'_3v'_4)$. We will prove that by pumping the loops on u_2 and u_4 sufficiently many times we will get a similar situation in T_2, proving that T_2 is not twinned. It is easy to show that there exist $k_2 > 0$, $k_1, k_3 \geq 0$, $w_i, w'_i \in \Sigma^*$, $i \in \{1, \ldots, 4\}$, some states $i_2, p_2, q_2, i'_2, p'_2, q'_2$ of T_2 and some stack contents $\sigma_2, \beta_2, \sigma'_2, \gamma'_2$ of T_2 such that we have the following runs in T_2:

$$\begin{cases} (i_2, \bot) \xrightarrow{u_1u_2^{k_1}/w_1} (p_2, \sigma_2) \xrightarrow{u_2^{k_2}/w_2} (p_2, \sigma_2\beta_2) \xrightarrow{u_2^{k_3}u_3u_4^{k_3}/w_3} (q_2, \sigma_2\beta_2) \xrightarrow{u_4^{k_2}/w_4} (q_2, \sigma_2) \\ (i'_2, \bot) \xrightarrow{u_1u_2^{k_1}/w'_1} (p'_2, \sigma'_2) \xrightarrow{u_2^{k_2}/w'_2} (p'_2, \sigma'_2\beta'_2) \xrightarrow{u_2^{k_3}u_3u_4^{k_3}/w'_3} (q'_2, \sigma'_2\beta'_2) \xrightarrow{u_4^{k_2}/w'_4} (q'_2, \sigma'_2) \end{cases}$$

such that (q_1, σ_1) and (q_2, σ_2) are co-accessible with the same input word u_5, and (q'_1, σ'_1) and (q'_2, σ'_2) are co-accessible with the same input word u'_5. Now for all $i \geq 0$, we let

$$\begin{array}{ll} V^{(i)} = v_1(v_2)^{k_1+ik_2+k_3}v_3(v_4)^{k_1+ik_2+k_3} & W^{(i)} = w_1(w_2)^iw_3(w_4)^i \\ V'^{(i)} = v'_1(v'_2)^{k_1+ik_2+k_3}v'_3(v'_4)^{k_1+ik_2+k_3} & W'^{(i)} = w'_1(w'_2)^iw'_3(w'_4)^i \\ D_1(i) = \Delta(V^{(i)}, V'^{(i)}) & D_2(i) = \Delta(W^{(i)}, W'^{(i)}) \end{array}$$

In other words, $D_1(i)$ (resp. $D_2(i)$) is the delay in T_1 (resp. T_2) accumulated on the input word $u_1(u_2)^{k_1+ik_2+k_3}u_3(u_4)^{k_1+ik_2+k_3}$ by the two runs of T_1 (resp. T_2). There is a relation between the words $V^{(i)}$ and $W^{(i)}$. Indeed, since T_1 and T_2 are equivalent and (q_1, σ_1) and (q_2, σ_2) are both co-accessible by the same input word, for all $i \geq 1$, either $V^{(i)}$ is a prefix of $W^{(i)}$ or $W^{(i)}$ is a prefix of $V^{(i)}$. We have a similar relation between $V'^{(i)}$ and $W'^{(i)}$.

We prove in Appendix the following intermediate results: (i) there exists $i_0 \geq 0$ such that for all $i, j \geq i_0$ such that $i \neq j$, $D_1(i) \neq D_1(j)$; (ii) for all $i, j \geq 1$, if $D_1(i) \neq D_1(j)$, then $D_2(i) \neq D_2(j)$. The proofs of those results rely on fundamental properties of word combinatorics and a non-trivial case study that depends on how the words $v_1(v_2)^{k_1+ik_2+k_3}v_3(v_4)^{k_1+ik_2+k_3}$ and $w_1(w_2)^iw_3(w_4)^i$ are overlapping. Thanks to (i) and (ii), we clearly get that $D_2(i_0) \neq D_2(i_0 + 1)$, which provides a counter-example for the twinning property. ∎

Subsequential transducers have at most one run per input word, so we get the following:

COROLLARY 15. *Subsequentializable* VPTs *are twinned.*

The TP is not a sufficient condition to be subsequentializable, as shown for instance by Example 12. Therefore the class of transductions defined by transducers which satisfy the TP is strictly larger than the class of transductions defined by subsequentializable 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:

THEOREM 16. *Let T be an* fVPT *and $u \in \Sigma^*$. If T is twinned then the evaluation of T on u can be done in space complexity quadratic in $h(u)$ and exponential in $|T|$.*

PROOF. [Sketch] We prove that Algo. LCPIN runs in space complexity $O\left(p(T) \cdot (h(u)+1)^2 \cdot M\right)$ on T and $u \in \Sigma^*$, with $M = \max\{|v| : (q,a,v,\gamma,q') \in \delta\}$ and $p(T) = |Q|^4 \cdot |\Gamma|^{2|Q|^4+2}$. Therefore, we use a pumping technique to show that for any word $u \in \Sigma^*$ on which there is a run of T, we have $\mathsf{out}_{\neq}^{\max}(u) \leq (h(u)+1)q(T)$ for some function q, whenever the TP holds for T. This is done as follows: any such word can be uniquely decomposed as $u = u_0 c_1 u_1 c_2 \dots c_n u_n$ with $n \leq h(u)$, each u_i is well-nested and each c_i is a call. Then if the u_i are long enough, we can pump them vertically and horizontally without affecting the global delay, by using the property $\Delta(vv', ww') = \Delta(\Delta(v,w).(v',w'))$. Then we can apply Proposition 3. ∎

6 Conclusion and Remarks

This work investigates the streaming evaluation of nested word transductions, and in particular identifies an interesting class of VPT-transductions which subsumes subsequentializable transductions and can still be efficiently evaluated. The following inclusions summarize the relations between the different *classes* of transductions we have studied:

$$\text{BM fVPTs} \subsetneq \text{Subsequentializable VPTs} \subsetneq \text{twinned fVPTs} \subsetneq \text{HBM fVPTs} \subsetneq \text{fVPTs}$$

Moreover, we have shown that BM, twinned and HBM fVPTs are decidable in NPTIME.

Remarks:[HBM is tight] We have mentionned that the space complexity of a VPT in HBM is at most exponential. We give here an example illustrating the tightness of this bound. 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 polynomial space as there are a doubly exponential number of trees of height n, for all $n \geq 0$. ∎

Further Directions An important asset of the class of twinned fVPTs w.r.t. the class of subsequentializable VPTs is that it is decidable. It would thus be interesting to determine whether or not the class of subsequentializable VPTs is decidable. In addition, we also plan to extend our techniques to more expressive transducers, such as those recently introduced in [1], which extend VPTs with global variables and are as expressive as MSO-transductions, and can therefore swap or reverse sub-trees. Another line of work concerns the extension of our evaluation procedure, which holds for functional transductions, to finite valued transductions.

References

[1] R. Alur and L. D'Antoni. Streaming tree transducers. Available on: `http://www.cis.upenn.edu/~alur/`, 2011. Submitted.

[2] R. Alur and P. Madhusudan. Adding nesting structure to words. *JACM*, 56(3):16:1–16:43, 2009.

[3] Z. Bar-Yossef, M. Fontoura, and V. Josifovski. Buffering in query evaluation over XML streams. In *PODS*, pages 216–227. ACM-Press, 2005.

[4] V. Bárány, C. Löding, and O. Serre. Regularity problems for visibly pushdown languages. In *STACS*, pages 420–431, 2006.

[5] D. Barbosa, L. Mignet, and P. Veltri. Studying the XML web: Gathering statistics from an xml sample. *World Wide Web*, 8:413–438, 2005.

[6] A. Bauer, M. Leucker, and C. Schallhart. Runtime verification for LTL and TLTL. *ACM TOSEM*, 20, 2011.

[7] M. Benedikt and A. Jeffrey. Efficient and expressive tree filters. In *FSTTCS*, volume 4855 of *LNCS*, pages 461–472. Springer Verlag, 2007.

[8] S. Boag, D. Chamberlin, M. F. Fernàndez, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML query language, W3C recommendation, 2007.

[9] M. Caralp, P.-A. Reynier, and J.-M. Talbot. A polynomial procedure for trimming visibly pushdown automata. Technical Report hal-00606778, HAL, CNRS, France, 2011.

[10] C. Choffrut. Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles. *Theor. Comput. Sci.*, 5(3):325–337, 1977.

[11] J. Clark. XSL Transformations (XSLT) version 1.0, W3C recommendation, 1999.

[12] E. Filiot, J.-F. Raskin, P.-A. Reynier, F. Servais, and J.-M. Talbot. Properties of visibly pushdown transducers. In *MFCS*, volume 6281 of *LNCS*, pages 355–367. Springer, 2010.

[13] O. Gauwin, J. Niehren, and S. Tison. Earliest query answering for deterministic nested word automata. In *FCT*, volume 5699 of *LNCS*, pages 121–132. Springer, 2009.

[14] M. Grohe, A. Hernich, and N. Schweikardt. Lower bounds for processing data with few random accesses to external memory. *J. ACM*, 56(3):12:1–12:58, 2009.

[15] T. Harju, O. H. Ibarra, J. Karhumaki, and A. Salomaa. Some decision problems concerning semilinearity and commutation. *JCSS*, 65, 2002.

[16] C. Konrad and F. Magniez. The streaming complexity of validating XML documents. Technical Report 1012.3311, arXiv, 2010.

[17] V. Kumar, P. Madhusudan, and M. Viswanathan. Visibly pushdown automata for streaming XML. In *WWW*, pages 1053–1062. ACM-Press, 2007.

[18] O. Kupferman and M. Y. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.

[19] P. Madhusudan and M. Viswanathan. Query automata for nested words. In *MFCS*, volume 5734 of *LNCS*, pages 561–573. Springer Berlin / Heidelberg, 2009.

[20] J. Robie, D. Chamberlin, M. Dyck, D. Florescu, J. Melton, and J. Siméon. XQuery Update Facility 1.0, W3C Recommendation 17 March 2011.

[21] L. Segoufin and C. Sirangelo. Constant-memory validation of streaming XML documents against dtds. In *ICDT*, pages 299–313, 2007.

[22] L. Segoufin and V. Vianu. Validating streaming XML documents. In *PODS*, pages 53–64. ACM-Press, 2002.

[23] A. Weber and R. Klemm. Economy of description for single-valued transducers. *Inf. Comput.*, 118(2):327–340, 1995.

A Online Evaluation Algorithm of Visibly Pushdown Transductions

All over this section we assume an implementation of VPTs such that the set S of transitions with a given left-hand side can be retrieved in time $O(|S|)$. We define the current height of a prefix of a nested word in the following way: $hc(u) = 0$ if u is well-nested, and $hc(ucv) = hc(u) + 1$ if $c \in \Sigma_c$ and v is well-nested.

A.1 NAIVE algorithm

We start with the algorithm NAIVE, that we will later improve to obtain LCPIN. The algorithm NAIVE simply computes all the runs (with their respective outputs) of the fVPT T on the input word u, stores them in a data structure and, at the end of u, outputs the only output word: it will be the same in all accepting runs, as T is functional.

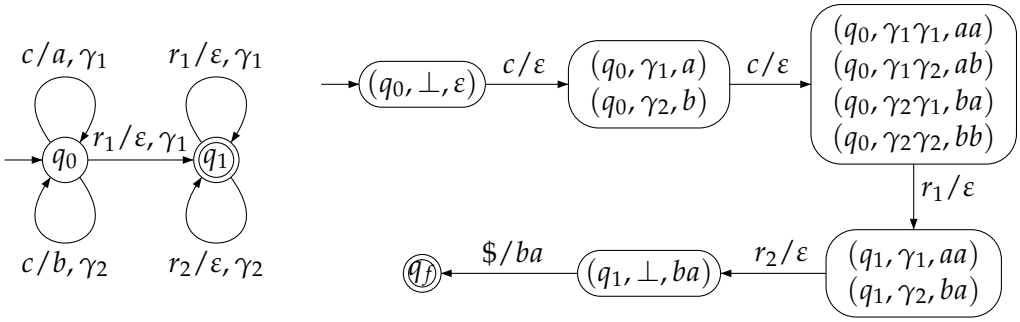
NAIVE consists in maintaining the set of d-configurations corresponding to the runs of T on the input word u. Hence, it is based on the operation $\mathsf{update}(C, a)$ that returns the set of d-configurations obtained after applying rules of T using input symbol a to each d-configuration of C. The function $\mathsf{update} : \mathsf{Dconfs}(T) \times \Sigma \to \mathsf{Dconfs}(T)$ maps a set of d-configurations and an input symbol to another set of d-configurations. For call symbols $c \in \Sigma_c$,

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

and for return symbols $r \in \Sigma_r$,

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

The function update can be considered as the transition function of a transition system with states $\mathsf{Dconfs}(T)$ (i.e. an infinite number of states). We can easily turn it into an infinite state transducer, i.e. an FST with infinitely many states: this transducer returns ϵ at every input symbol, except for the last one \$, where it returns the output word. This is illustrated in Fig. 3. Formally, an infi-



(a) A VPT T_1.

(b) Part of the IST t corresponding to T_1, computed on ccr_1r_2.

Figure 3: Illustration of the computations of NAIVE on an input word.

nite state transducer t (IST for short) is defined exactly like an FST, except that its number of states may be infinite (but countable). In particular, the acceptance condition remains the same, so that the transduction $[\![t]\!]$ is still a set of pairs of finite words (u, v).

Given the functional VPT $T = (Q, I, F, \Gamma, \delta)$, consider the IST $t = (\mathsf{Dconfs}(T) \uplus \{q_f\}, I_t, \{q_f\}, \delta_t)$ where $I_t = \{\{(q_0, \bot, \epsilon) \mid q_0 \in I\}\}$. To deal with the last symbol \$, we have to characterize the sets of d-configurations reached after reading words in $Dom(T)$. T being functional, each of these sets of d-configurations C comes with a single output word v:

$$\mathcal{C} = \left\{ (C, v) \in \mathsf{Dconfs}(T) \times \Sigma^* \mid \begin{array}{l} \exists q \in F.\ (q, \bot, v) \in C \text{ and} \\ \forall (q', \bot, v') \in C,\ q' \in F \implies v' = v \end{array} \right\}$$

Rules in δ_t are:

$$\begin{array}{ll} C \xrightarrow{a/\epsilon} \mathsf{update}(C, a) & \text{for } C \in \mathsf{Dconfs}(T) \text{ and } a \in \Sigma \\ C \xrightarrow{\$/v} q_f & \text{for } (C, v) \in \mathcal{C} \end{array}$$

LEMMA 17. *$(u, v) \in [\![T]\!]$ iff $(u\$, v) \in [\![t]\!]$.*

PROOF. It can be checked easily by induction on $|u|$ that for every $u \in \Sigma^*$, the current state of t after reading u is $\bigcup_{q_0 \in I} \{(q, \sigma, v) \mid (q_0, \bot) \xrightarrow{u/v} (q, \sigma)\}$. Let us check whether reading the last symbol \$ leads to a correct state. Let $u \in \Sigma^*$. If $u \notin Dom(T)$, then there is no run of T on u of the form $(q, \bot) \xrightarrow{u/v} (q', \bot)$ with $q \in I$ and $q' \in F$. Hence, the state C reached by t after reading u, if it exists, is such that $(C, v) \notin \mathcal{C}$ for all $v \in \Sigma^*$, so $u \notin Dom(t)$. If $u \in Dom(T)$, then the state of t reached after reading u is $C = \bigcup_{q_0 \in I} \{(q, \bot, v) \mid (q_0, \bot) \xrightarrow{u/v} (q, \bot)\}$. As T is functional, there is a unique v for all $(q, \bot, v) \in C$ such that $q \in F$, and such elements of C exist, so that $(C, v) \in \mathcal{C}$, and $(u\$, v) \in [\![t]\!]$. ∎

As t is deterministic, the algorithm NAIVE only consists in computing the unique run of t on the input word u. Let $\mathsf{out}(u) = \max_{v \in \mathsf{reach}(u)} |v|$, and let $\mathsf{out}^{\max}(u)$ be its maximal value over prefixes: $\mathsf{out}^{\max}(u) = \max_{u' \text{ prefix of } u} \mathsf{out}(u')$.

PROPOSITION 18. *The maximal amount of memory used by* NAIVE *for processing $u \in \Sigma^*$ is in $O(|Q| \cdot |\Gamma|^{h(u)} \cdot \mathsf{out}^{max}(u))$. The preprocessing time, and the time used by* NAIVE *to process each symbol of u are both polynomial in $|Q|$, $|\Gamma|$, $|\delta|$, $h(u)$, $\mathsf{out}^{max}(u)$ and $|\Sigma|$.*

PROOF. As T is functional, there cannot be two distinct co-accessible d-configurations (q, σ, v) and (q, σ, v') in C. This remark proves the space complexity. For time complexity, updating a d-configuration is just a research of rules to apply. Each of them will generate a new d-configuration, and is retrieved in constant time. ∎

A.2 Compact representation of runs of fVPTs: NAIVE$_{\text{COMPACT}}$.

We present the data structure S_u^T representing all d-configurations stored by NAIVE on the VPT T after reading u, i.e., the state of t reached after reading u. This structure is illustrated in Fig. 2. This first improvement avoids the exponential blowup in $h(u)$. The structure S_u^T is a labeled DAG (directed acyclic graph) whose nodes are configurations of T (with an additional root node #, and each node has a depth) and edges are labeled by delays: $nodes(S_u^T) = \{\#\} \uplus Q \times (\Gamma \cup \{\bot\}) \times \mathbb{N}$ and $edges(S_u^T) \subseteq nodes(S_u^T) \times \Sigma^* \times nodes(S_u^T)$. This DAG will have as leaves (nodes without outgoing edges) current configurations. The structure S_u^T is defined inductively on u according to the following algorithms.

$$edges(S_\epsilon^T) = \{\# \overset{\epsilon}{\hookrightarrow} (q, \bot, 0) \mid q \in I\}$$

When a call letter $c \in \Sigma_c$ is read, the structure S_u^T is updated such that, for every leaf of S_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 also the possible new generated leaves (procedure REMOVE_EDGES). Algorithm 1 describes how S_{uc}^T is computed from S_u^T. For a return letter $r \in \Sigma_r$,

Algorithm 1 Updating structure S with a call symbol.

procedure UPDATE_CALL(S, c)
  $newEdges \leftarrow \emptyset$
  $orphans \leftarrow \emptyset$
  for $(q, \gamma, i) \in leaves(S)$ **do**
    if $\exists v, \gamma', q' \mid (q, c, v, \gamma', q') \in \delta$ **then**
      for $(v, \gamma', q') \mid (q, c, v, \gamma', q') \in \delta$ **do**
        $newEdges.add((q, \gamma, i) \overset{v}{\hookrightarrow} (q', \gamma', i+1))$
    else
      $orphans.add((q, \gamma, i))$
  $edges(S) \leftarrow edges(S) \cup newEdges$

procedure REMOVE_EDGES($S, orphans$)
  while $orphans \neq \emptyset$ **do**
    $n \leftarrow orphans.pop()$
    for $m \mid \exists v,\ m \overset{v}{\hookrightarrow} n$ **do**
      $remove(S, m \overset{v}{\hookrightarrow} n)$
      if $\nexists n', v',\ m \overset{v'}{\hookrightarrow} n'$ **then** $orphans.add(m)$

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 2. Only edges and reachable nodes need to be stored, so that $|S_u^T| \leq (hc(u)+1) \cdot |Q|^2 \cdot |\Gamma|^2 \cdot \mathsf{out}(u, T)$.

We prove the correctness of this construction using the transition function $\Rightarrow_u$ based on edges of S_u^T, that gathers the stack content and delay. The relation $\Rightarrow_u$ is the smallest relation in $(Q \times \Gamma^* \times \mathbb{N}) \times \Sigma^* \times (Q \times \Gamma^* \times \mathbb{N})$ containing $\hookrightarrow$ such that: if $(q_0, \sigma_0, i) \overset{v}{\Rightarrow}_u (q_1, \sigma_1\gamma, j)$ and $(q_1, \gamma, j) \overset{v'}{\hookrightarrow} (q_2, \gamma', j+1)$ then $(q_0, \sigma_0, i) \overset{vv'}{\Rightarrow}_u (q_2, \sigma_1\gamma\gamma', j+1)$ (we may have $\sigma = \epsilon$ and $\gamma = \bot$). The set of d-configurations stored in S_u^T is defined by: $C(S_u^T) = \{(q, \sigma, v) \mid \exists i,\ (q, \sigma, i) \in leaves(S_u^T)$ and $\# \overset{v}{\Rightarrow}_u (q, \sigma, i)\}$. The following lemma shows that S_u^T exactly encodes the d-configurations computed by the IST t.

LEMMA 19. *$C(S_u^T)$ is the state of the IST t after reading u.*

PROOF. As mentioned in the proof of Lemma 17, the current state of t after reading u is $\bigcup_{q_i \in I} \{(q, \sigma, v) \mid (q_i, \bot) \xrightarrow{u/v} (q, \sigma)\}$. Hence proving the following invariant is sufficient to prove this lemma:

> For every $0 \leq i \leq hc(u)$, $\# \overset{v}{\Rightarrow}_u (q, \sigma, i)$ iff there exists $q_i \in I$ such that $(q_i, \bot) \xrightarrow{u_1 \cdots u_k / v} (q, \sigma)$ where $k = \max\{j \mid hc(u_1 \cdots u_j) = i\}$.

We prove it by induction on $|u|$. If $|u| = 0$, then $i = 0$ and the equivalence holds, as we can assume ϵ-loops (without output) on initial states.

Algorithm 2 Updating structure S with a return symbol.

```
    procedure UPDATE_RETURN(S, r)
2:      newEdges ← ∅
        orphans ← ∅
4:      for (q_ℓ, γ_ℓ, i) ∈ leaves(S) do
            if ∃v, q | (q_ℓ, r, v, γ_ℓ, q) ∈ δ then
6:              for (v, q) | (q_ℓ, r, v, γ_ℓ, q) ∈ δ do
                    for (q_0, γ_0, v_0) | (q_0, γ_0, i−1) ↪^{v_0} (q_ℓ, γ_ℓ, i) ∈ edges(S) do
8:                      for (n, v_1) | n ↪^{v_1} (q_0, γ_0, i−1) ∈ edges(S) do
                            newEdges.add(n ↪^{v_1 v_0 v} (q, γ_0, i−1))
10:         else
                orphans.add((q_ℓ, γ_ℓ, i))
12:     remove_edges(S, orphans)
        remove_leaves(S)
14:     edges(S) ← edges(S) ∪ newEdges

16: procedure REMOVE_LEAVES(S)
        for n ∈ leaves(S) do
18:         for (m, v) | m ↪^{v} n ∈ edges(S) do
                remove(S, m ↪^{v} n)
```

Assume that the property holds for a given u, we prove that if also holds for the well-nested prefix uc. Let $orphans$ be the set of leaves collected by the outermost for-loop of UPDATE_CALL. These are the leaves $(q, \gamma, hc(u))$ of S_u^T such that no rule (q, c, v, γ', q') exists in δ. Hence, corresponding configurations are blocked, and can be removed. The procedure REMOVE_EDGES propagates these deletions, so that after the call to this procedure, the structure exactly contains configurations that can be updated by c. Hence, by induction hypothesis, the equivalence holds for $0 \leq i \leq hc(u)$. For $i = hc(uc) = hc(u) + 1$, let $k = \max\{j \mid hc(u_1 \cdots u_j) = i\}$. We have:

$$
\begin{aligned}
&\# \overset{vv'}{\Longrightarrow}_{uc} (q, \sigma\gamma\gamma', i) \\
&\quad \text{iff} \quad \exists q_1,\ \# \overset{v}{\Rightarrow}_{uc} (q_1, \sigma\gamma, i-1) \quad \text{and} \quad (q_1, \gamma, i-1) \overset{v'}{\hookrightarrow} (q, \gamma', i) && (1)\\
&\quad \text{iff} \quad \exists q_1,\ \exists q_i \in I,\ (q_i, \bot) \xrightarrow{u/v} (q_1, \sigma\gamma) \quad \text{and} \quad (q_1, \gamma, i-1) \overset{v'}{\hookrightarrow} (q, \gamma', i) && (2)\\
&\quad \text{iff} \quad \exists q_1,\ \exists q_i \in I,\ (q_i, \bot) \xrightarrow{u/v} (q_1, \sigma\gamma) \quad \text{and} \quad (q_1, c, v', \gamma', q) \in \delta && (3)\\
&\quad \text{iff} \quad \exists q_i \in I,\ (q_i, \bot) \xrightarrow{uc/vv'} (q, \sigma\gamma\gamma')
\end{aligned}
$$

(1) is by definition of $\Rightarrow_{uc}$. (2) holds because, as mentioned above, REMOVE_EDGES removes non-accessible configurations, and by induction hypothesis. Here, we also have $k - 1 = \max\{j \mid hc(u_1 \cdots u_j) = i - 1\}$, as uc ends with a call symbol: so $u_1 \cdots u_{k-1} = u$ and $u_1 \cdots u_k = uc$. (3) is due to the way UPDATE_CALL operates: it adds children to leaves according to rules of δ.

Now we show that the property holds for the well-nested prefix ur, if it holds for u. Let $h = hc(u)$. Procedure UPDATE_RETURN checks, for each leaf, whether a rule can be applied. If not, the leaf is removed, and orphaned edges too, as explained for call symbols. Then, the hth level is

removed and the $(h-1)$th updated, according to rules of T. Hence the property remains true for $0 \leq i \leq h-2$. We have:

$$
\begin{aligned}
& \# \overset{v}{\Rightarrow}_{ur} (q, \sigma\gamma\gamma_0, h-1) \\
& \quad \text{iff} \quad \exists q_0, q_1, q_\ell, \gamma_\ell, v', v_0, v_1,\ \# \overset{v'}{\Rightarrow}_u (q_1, \sigma\gamma, h-2) \text{ and} \\
& \qquad (q_1, \gamma, h-2) \overset{v_1}{\hookrightarrow} (q_0, \gamma_0, h-1) \in \mathit{edges}(S_u^T) \text{ and} \\
& \qquad (q_0, \gamma_0, h-1) \overset{v_0}{\hookrightarrow} (q_\ell, \gamma_\ell, h) \in \mathit{edges}(S_u^T) \text{ and} \\
& \qquad (q_\ell, r, v'', \gamma_\ell, q) \in \delta \text{ and } v = v' v_1 v_0 v'' \qquad (1) \\
& \quad \text{iff} \quad \exists q_0, q_\ell, \gamma_\ell, v', v_0,\ \# \overset{v'}{\Rightarrow}_u (q_0, \sigma\gamma\gamma_0, h-1) \text{ and} \\
& \qquad (q_0, \gamma_0, h-1) \overset{v_0}{\hookrightarrow} (q_\ell, \gamma_\ell, h) \in \mathit{edges}(S_u^T) \text{ and} \\
& \qquad (q_\ell, r, v'', \gamma_\ell, q) \in \delta \text{ and } v = v' v_0 v'' \qquad (2) \\
& \quad \text{iff} \quad \exists q_\ell, \gamma_\ell, v',\ \# \overset{v'}{\Rightarrow}_u (q_\ell, \sigma\gamma\gamma_0\gamma_\ell, h) \in \mathit{edges}(S_u^T) \text{ and} \\
& \qquad (q_\ell, r, v'', \gamma_\ell, q) \in \delta \text{ and } v = v' v'' \qquad (3) \\
& \quad \text{iff} \quad \exists q_\ell, \gamma_\ell, v', q_i \in I\ (q_i, \bot) \xrightarrow{u/v'} (q_\ell, \sigma\gamma\gamma_0\gamma_\ell) \text{ and} \\
& \qquad (q_\ell, r, v'', \gamma_\ell, q) \in \delta \text{ and } v = v' v'' \qquad (4) \\
& \quad \text{iff} \quad \exists q_i \in I,\ (q_i, \bot) \xrightarrow{ur/v} (q, \sigma\gamma\gamma_0)
\end{aligned}
$$

Equivalence (1) reflects how UPDATE_RETURN generates the new leaves. (2) and (3) come from the definition of $\Rightarrow_u$. (4) is obtained by induction hypothesis and the fact that, if $k = \max\{j \mid hc(u_1 \cdots u_j) = h\}$, then $u_1 \cdots u_k = u$. ∎

The depth of the DAG obtained after reading u is the current height of u plus 1, each level has at most $|Q| \cdot |\Gamma|$ nodes, and each edge is labelled with a word of length less than $\mathsf{out}(u)$.

PROPOSITION 20. *The maximal amount of memory used by* NAIVE$_{\text{COMPACT}}$ *on* $u \in \Sigma^*$ *is in* $O(|Q|^2 \cdot |\Gamma|^2 \cdot (h(u)+1) \cdot \mathsf{out}^{max}(u))$. *The preprocessing time, and the time used by* NAIVE *to process each symbol of* u *are both polynomial in* $|Q|$, $|\Gamma|$, $|\delta|$, $h(u)$, $\mathsf{out}^{max}(u)$ *and* $|\Sigma|$.

A.3 LCPIN algorithm

We extend the definition of the largest common prefix to sets of d-configurations: if $C \subseteq \mathsf{Dconfs}(T)$, then $\mathsf{lcp}(C) = \mathsf{lcp}(\{v \mid (q, \sigma, v) \in C\})$. Let $\mathsf{rem\_lcp}$ be the function that removes the largest common prefix to a set of d-configurations: $\mathsf{rem\_lcp}(C) = \{(q, \sigma, v') \in \mathsf{Dconfs}(T) \mid (q, \sigma, \mathsf{lcp}(C) \cdot v') \in C\}$. From an fVPT T, we define the IST $\tau = (\mathsf{Dconfs}(T) \uplus \{q_f\}, I_\tau, \{q_f\}, \delta_\tau)$ where $I_\tau = \{\{(q_0, \bot, \epsilon) \mid q_0 \in I\}\}$. We keep the same definition of $\mathcal{C}$ as in NAIVE, and rules of δ_τ are:

$$
\begin{aligned}
& C \xrightarrow{a/\mathsf{lcp}(\mathsf{update}(C,a))} \mathsf{rem\_lcp}(\mathsf{update}(C,a)) && \text{for } C \in \mathsf{Dconfs}(T) \text{ and } a \in \Sigma \\
& C \xrightarrow{\$/v} q_f && \text{for } (C, v) \in \mathcal{C}
\end{aligned}
$$

We start by proving the correctness of the definition of the IST τ. This definition is illustrated in Fig. 5.

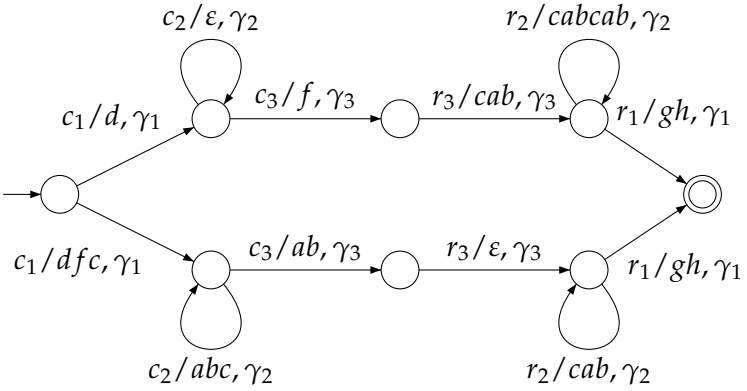


Figure 4: A functional VPT on $\Sigma_c = \{c_1, c_2, c_3\}$ and $\Sigma_r = \{r_1, r_2, r_3\}$.

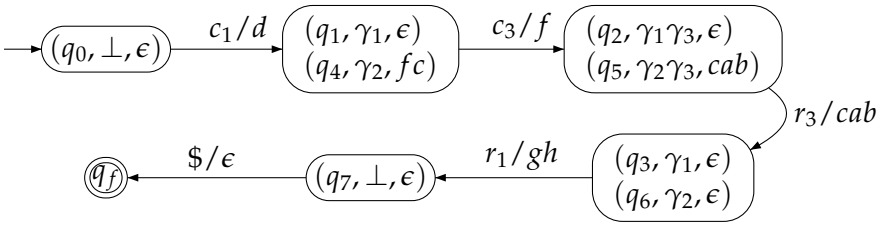


Figure 5: Part of the IST τ corresponding to the VPT in Fig. 4, computed by LCPIN on input $c_1c_3r_3r_1$.

LEMMA 21. *$(u, v) \in [\![T]\!]$ iff $(u\$, v) \in [\![\tau]\!]$.*

PROOF. By an induction on $|u|$, it can be checked that $C_0 \xrightarrow{u/\mathsf{lcp}_{\mathsf{in}}(u,T)}_{\tau} C$ where C_0 is the only element in I_τ and C is obtained from the run of t on u: $C = \mathsf{rem\_lcp}(C')$ with $C_0 \xrightarrow{u/\epsilon}_t C'$. The remainder of the proof is similar to the proof of Lemma 17. ∎

We now provide algorithms for the second step of the computation performed by LCPIN on an input symbol a. Recall that the first step is the same as in $\text{NAIVE}_{\text{COMPACT}}$, i.e. Algorithm 1 if a is a call symbol, and Algorithm 2 if it is a return symbol, which transforms S_u^T to a new structure S'. The second step is the computation of $\mathsf{lcp}(C(S'))$, which is output, and removed from every branch of S', using Algorithm 3.

Algorithm 3 starts with the procedure *factorize*, that processes every nodes in a bottom-up manner (from leaves to the root #). For every node, the lcp of all outgoing edges is moved to all incoming edges. This is illustrated in Fig. 6.

For every node $n \in \mathit{nodes}(S_u^T)$, let S_n be the structure obtained from S_u^T just after returning from $\mathit{factorize}(S, n, \mathit{done})$, and let $\Rightarrow_n$ be the relation defined like $\Rightarrow_u$, but on S_n. Note that the

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

Figure 6: Changes performed by *factorize* on a node.

Algorithm 3 Compute $\mathsf{lcp}(C(S))$, and update S to the structure encoding $\mathsf{rem\_lcp}(C(S))$.

```
    procedure OUTPUT_LCP(S)
2:      factorize(S, #, ∅)
        ℓ ← lcp({v | ∃n, # -v-> n})
4:      output ℓ
        for m, v | # -v-> m do
6:          let p be such that v = ℓ · p
            replace # -v-> m by # -p-> m in S
8:
    function FACTORIZE(S, n, done)
10:     if n ∉ leaves(S) then
            for m, v | n -v-> m and m ∉ done do
12:             done ← factorize(S, m, done)
            if n = # then return done ∪ {n}
14:         factor ← lcp({v | ∃m, n -v-> m})
            for m, v | n -v-> m do
16:             let p be such that v = factor · p
                replace n -v-> m by n -p-> m in S
18:         for p, v | p -v-> n do
                replace p -v-> n by p -v·factor-> n in S
20:     return done ∪ {n}
```

structure has the same set of nodes and edges after being processed by Algorithm 3, only the labels of edges are updated. Let B_n be the set of branches from node n to a leaf: $B_{n_0} = \{(n_0, n_1, \ldots, n_k) \mid \forall 0 \leq i < k,\ \exists v_i,\ n_i \xhookrightarrow{v_i} n_{i+1} \text{ and } n_k \in \mathit{leaves}(S_u^T)\}$. For a branch b, we write $V_n(b)$ for its output in the structure S_n, and $V(b)$ for its output in S_u^T: $V_n((n_0, \ldots, n_k)) = v_0 \cdots v_{k-1}$ where $n_i \xhookrightarrow{v_i} n_{i+1}$ for all $0 \leq i < k$ in S_n. We extend this definition to sets of branches: $V_n(B) = \{V_n(b) \mid b \in B\}$. Note also that each node is processed once using *factorize*, and in a bottom-up way: when processing node n, all its descendants have been updated before (cf lines 11 and 12). The following property is the main invariant proving the correctness of Algorithm 3.

LEMMA 22. *For every node $n \neq \#$, let $\ell = \mathsf{lcp}(V(B_n))$. Then,*

1. *for every branch $b \in B_n$, $V(b) = \ell \cdot V_n(b)$*
2. *for every p, v such that $p \xhookrightarrow{v} n$ in S_n, $v = v'\ell$ with $p \xhookrightarrow{v'} n$ in S_u^T*

PROOF. We prove the following property by bottom-up induction on the structure. This property is true on leaves, as *factorize* does not modify their incoming edges. Assume that the property holds for all descendants of a node n. Let $\ell = \mathsf{lcp}(V(B_n))$, and consider a branch $b \in B_n$. The function *factorize* applied at n computes the lcp of edges outgoing from n and removes it on every branch. Hence, if n' be the node processed by *factorize* before n, then:

$$V_{n'}(b) = \mathsf{lcp}(\{v \mid \exists p,\ n \xhookrightarrow{v} p \text{ in } S_{n'}\}) \cdot V_n(b) \tag{1}$$

Let p be the second node in branch $b = (n, p, \ldots)$. As p and all its descendants are processed before n' and not modified until the call of *factorize* on p, we have $V_{n'}(b) = V_p(b)$. Let us decompose $V_p(b)$ according to p: $V_p(b) = v_0 \cdot V_p(b_p)$ where $n \overset{v_0}{\hookrightarrow} p$ in S_p and b_p is the branch obtained from b by removing n. Using the induction hypothesis applied at p, we get $v_0 = v_1\ell'$ with $\ell' = \mathsf{lcp}(V(B_p))$, $n \overset{v_1}{\hookrightarrow} p$ in S_u^T, and $V(b_p) = \ell' \cdot V_p(b_p)$. Thus $V_p(b) = v_1 \cdot \ell' \cdot V_p(b_p) = v_1 \cdot V(b_p) = V(b)$. Equation (1) becomes:

$$V(b) = \mathsf{lcp}(\{v \mid \exists p,\ n \overset{v}{\hookrightarrow} p \text{ in } S_{n'}\}) \cdot V_n(b) \tag{2}$$

Let us write $\ell'' = \mathsf{lcp}(\{v \mid \exists p,\ n \overset{v}{\hookrightarrow} p \text{ in } S_{n'}\})$. It remains to prove that $\ell'' = \ell$. Notice that this will prove both parts of the lemma. We have $\ell'' = \mathsf{lcp}(\{v \mid \exists p,\ n \overset{v}{\hookrightarrow} p \text{ in } S_p\})$ because p and its descendants are unchanged between calls of *factorize* on p and n'. Using the induction hypothesis on each p, we obtain:

$$\ell'' = \mathsf{lcp}(\{v \cdot \mathsf{lcp}(V(B_p)) \mid n \overset{v}{\hookrightarrow} p \text{ in } S_u^T\}) = \mathsf{lcp}(V(B_n)) = \ell$$

This concludes the proof. ∎

The next lemma ensures that *factorize* preserves the semantic of the structure, i.e. the set of encoded configurations.

LEMMA 23. $C(F(S_u^T)) = C(S_u^T)$.

PROOF.

$$\begin{aligned}
&(q,\sigma,v) \in C(S_u^T)\\
&\quad \text{iff } \exists i,\ (q,\sigma,i) \in \mathit{leaves}(S_u^T) \text{ and } \# \overset{v}{\Rightarrow}_u (q,\sigma,i)\\
&\quad \text{iff } \exists i,\ (q,\sigma,i) \in \mathit{leaves}(S_u^T) \text{ and } \exists p,\ \# \overset{v_0}{\hookrightarrow} p \text{ and } p \overset{v_1}{\Rightarrow}_u (q,\sigma,i) \text{ with } v = v_0v_1\\
&\quad \text{iff } \exists i,\ (q,\sigma,i) \in \mathit{leaves}(S_u^T) \text{ and } \exists p,\ \# \overset{v_0}{\hookrightarrow} p \text{ and } v = v_0V(b)\\
&\qquad \text{where } b \text{ is a branch } p \Rightarrow_u (q,\sigma,i)\\
&\quad \text{iff } \exists i,\ (q,\sigma,i) \in \mathit{leaves}(S_u^T) \text{ and } \exists p,\ \# \overset{v_0}{\hookrightarrow} p \text{ and } v = v_0 \cdot \mathsf{lcp}(V(B_p)) \cdot V_n(b)\\
&\qquad \text{where } b \text{ is a branch } p \Rightarrow_u (q,\sigma,i) \qquad (1)\\
&\quad \text{iff } (q,\sigma,v) \in C(F(S_u^T))
\end{aligned}$$

Equivalence (1) is by Lemma 22. ∎

PROOF. [Proposition 3] Correctness of Algorithm 3 is ensured by Lemma 22, and the fact that at the root node, it uses the same technique to factorize and output the lcp. In terms of memory requirement, the number of nodes of S_u^T remains bounded as before, while words on edges have length at most $\mathsf{out}_{\neq}^{\max}(u)$, as each of them participate in a d-configuration in $C(S_u^T)$, as proved by Lemma 23 and Lemma 19. All procedures are in polynomial time. ∎

B Bounded Memory Evaluation Problems

B.1 Bounded Memory functional FSTs

PROPOSITION 24. *Let T be a functional* FST.
1. *$[\![T]\!]$ is BM iff T is subsequentializable;*
2. *It is decidable in* PTIME *if $[\![T]\!]$ is BM [30].*

PROOF. Statement 2 is proved in [30] (it is proved that subsequentializability is decidable in PTIME). We prove statement 1. Clearly, if $[\![T]\!]$ is definable by a subsequential 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 $[\![T]\!]$ is BM, there exists $K \in \mathbb{N}$ and a TT M that transforms any input word u into $[\![T]\!](u)$ 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 subsequential 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). ∎

B.2 Bounded Memory pushdown transductions

PROPOSITION 25. *It is undecidable whether a pushdown transduction is BM.*

PROOF. We reduce the problem of deciding whether the language of a pushdown automaton P over an alphabet A is regular to BM. Any letter of A is seen as an internal symbol. We associate with P a pushdown transducer I_P which defines the identity on $L(P)$. Clearly, if $L(P)$ is regular, it is defined by a finite automaton which can easily be turned into a Turing transducer defining $[\![I_P]\!]$ and which uses a memory that depends on the size of the automaton only. Conversely, if $[\![I_P]\!]$ is BM, there exists a function f and a TT M equivalent to I_P and which uses at most $f(1, |M|)$ bits of memory, i.e. an amount of memory which depends on the size of M only. The machine M can easily be turned into a finite automaton which defines P, whose states are the configurations of the working tape of M. ∎

B.3 Proof of Proposition 5

PROOF. If $[\![T]\!]$ is BM, there exist K and a TT M such that M evaluates any input word in space at most K. We can easily extract from M a finite automaton that defines $Dom(T)$, 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 $Dom(T)$ have a height bounded by m. If the height of the words in $Dom(T)$ is bounded, then their height is bounded by n^2. Indeed, assume that there exists a word $u \in Dom(T)$ whose height is strictly larger than n^2. Then there exists a run of T on u of the following form:

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

such that $u = u_1u_2u_3u_4u_5$, σ' is not empty, and i (resp. f) is an initial (resp. final) state of T. The existence of this decomposition follows from the consideration of the set of pairs of positions in u corresponding to matching calls and returns of well-nested subwords of u. Then one can iterate the matching loops around q and p to generate words in $Dom(T)$ with arbitrarily large heights, yielding a contradiction. Therefore $\mathsf{FST}(T, n^2)$ is equivalent to T. As in the proof of Proposition 24, we can regard M as a subsequential FST T_M whose set of states are configurations of the machine. The FST T_M is equivalent to T, and therefore to $\mathsf{FST}(T, n^2)$. Since T_M is subsequential, $\mathsf{FST}(T, n^2)$ is subsequentializable and therefore by Proposition 24, $\mathsf{FST}(T, n^2)$ is BM. The converse is obvious.

Therefore to check whether $[\![T]\!]$ 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 counts the height of the word. Then we check in NPTIME whether evaluating T can be done in constant memory if we fix both the transducer and the height of the word (Theorem 10). Since here the height is bounded by n^2, it is equivalent to checking bounded memory. ∎

C On deciding height bounded memory for VPTs

Before going into the proofs of the results of this section, we first prove that our horizontal twinning property, when restricted to FSTs, is equivalent to the twinning property for FSTs defined in [24].

C.1 Twinning properties for FSTs

Since we use results on the twinning property for FSTs in this paper, we clarify the definition of twinning property for FSTs. In the core of the paper, it is said that restricting the horizontal twinning property to FSTs correspond to the usual twinning property of FSTs. By "usual" we mean the following definition, taken from [24].

DEFINITION 26. *[Twinning property for FSTs of [24]]*
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 either $v_2 = w_2 = \epsilon$, or the following holds:
(i) $|v_2| = |w_2|$
(ii) $v_1(v_2)^\omega = w_1(w_2)^\omega$

Our twinning property for FSTs is obtained by restricting the horizontal twinning property of VPTs to FSTs. By restricting we mean the following:

DEFINITION 27. *[Twinning property for FSTs of this paper] 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 $\Delta(v_1, w_1) = \Delta(v_1v_2, w_1w_2)$.

The two definitions are equivalent, as shown by the next lemma:

LEMMA 28. *Definitions 26 and 27 are equivalent.*

PROOF. First suppose that Definition 26 holds. If $v_2 = w_2 = \epsilon$, then clearly $\Delta(v_1, w_1) = \Delta(v_1v_2, w_1w_2)$. Otherwise $|v_2| = |w_2|$ and $v_1(v_2)^\omega = w_1(w_2)^\omega$. Then necessarily $v_1 \preceq w_1$ (i.e. v_1 is a prefix of w_1) or $w_1 \preceq v_1$. Wlog suppose that $v_1 \preceq w_1$, i.e. $w_1 = v_1w_1'$ for some w_1'. Therefore $\Delta(v_1, w_1) = (\epsilon, w_1')$, and $\Delta(v_1v_2, w_1w_2) = \Delta(v_2, w_1'w_2)$.

We now prove that $\Delta(v_2, w_1'w_2) = (\epsilon, w_1')$. Since $v_1(v_2)^\omega = w_1(w_2)^\omega$, we have $(v_2)^\omega = w_1'(w_2)^\omega$. Therefore $v_2w_1'(w_2)^\omega = v_2^\omega = w_1'(w_2)^\omega$. Since $|v_2| = |w_2|$, we get $v_2w_1' = w_1'w_2$, from which we have $\Delta(v_2, w_1'w_2) = \Delta(v_2, v_2w_1') = (\epsilon, w_1')$.

Conversely, suppose that Definition 27 holds and $v_2w_2 \neq \epsilon$. Since $\Delta(v_1, w_1) = \Delta(v_1v_2, w_1w_2)$, we have either $v_1 \preceq w_1$ or $w_1 \preceq v_1$. Wlog suppose that $v_1 \preceq w_1$, i.e. $w_1 = v_1w_1'$ for some w_1'. Therefore we have:

$$\Delta(v_1, w_1) = (\epsilon, w_1') = \Delta(v_1v_2, w_1w_2) = \Delta(v_2, w_1'w_2)$$

Consequently, $v_2 \preceq w_1'w_2$, and in particular, $w_1'w_2 = v_2w_1'$, which gives us the following series of equalities:

$$w_1(w_2)^\omega = v_1w_1'(w_2)^\omega = v_1v_2w_1'(w_2)^\omega = v_1(v_2)^2w_1'(w_2)^\omega = \cdots = v_1(v_2)^\omega$$

∎

C.2 HTP is decidable

LEMMA 29. *Let T be an* fVPT*, T does not satisfy the HTP if and only if there exist $q_0, q_0' \in I$, $q, q' \in Q$, $\sigma, \sigma' \in \Gamma^*$ and $u_1, v_1, w_1, u_2, v_2, w_2 \in \Sigma^*$, with either $v_2 \neq \epsilon$ or $w_2 \neq \epsilon$ and:*

$$(q_0, \bot) \xrightarrow{u_1/v_1} (q, \sigma) \xrightarrow{u_2/v_2} (q, \sigma) \qquad (q_0', \bot) \xrightarrow{u_1/w_1} (q', \sigma') \xrightarrow{u_2/w_2} (q', \sigma')$$

and either (i) $|v_2| \neq |w_2|$ or (ii) $|v_2| = |w_2|$, $|v_1| \leq |w_1|$ and $v_1v_2 \not\preceq w_1w_2$.

PROOF. Let suppose that there exist $q_0, q_0' \in I$, $q, q' \in Q$, $\sigma, \sigma' \in \Gamma^*$ and $u_1, v_1, w_1, u_2, v_2, w_2 \in \Sigma^*$, with either $v_2 \neq \epsilon$ or $w_2 \neq \epsilon$ and :

$$(q_0, \bot) \xrightarrow{u_1/v_1} (q, \sigma) \xrightarrow{u_2/v_2} (q, \sigma) \qquad (q_0', \bot) \xrightarrow{u_1/w_1} (q', \sigma') \xrightarrow{u_2/w_2} (q', \sigma')$$

And let show that if (i) or (ii) hold so does $\Delta(v_1, w_1) \neq \Delta(v_1v_2, w_1w_2)$.

First suppose that (i) holds. Let $x = v_1 \wedge w_1$ and $z = v_1v_2 \wedge w_1w_2$, clearly there exists $y \in \Sigma^*$ such that $z = xy$. By definition, we have $\Delta(v_1, w_1) = (x^{-1}v_1, x^{-1}w_1)$ and $\Delta(v_1v_2, w_1w_2) = ((xy)^{-1}v_1v_2, (xy)^{-1}w_1w_2)$. Therefore if (i) holds then either $|x^{-1}v_1| \neq |(xy)^{-1}v_1v_2|$ or $|x^{-1}w_1| \neq |(xy)^{-1}w_1w_2)|$. Indeed, suppose the first inequality does not hold, we have $|x^{-1}v_1| = |v_1| - |x| = |(xy)^{-1}v_1v_2| = |v_1| + |v_2| - |x| - |y|$, that is $0 = |y| - |v_2|$. Therefore one can check that the second inequality must hold. We have shown that (i) implies $\Delta(v_1, w_1) \neq \Delta(v_1v_2, w_1w_2)$.

Now, suppose (ii) holds and let us show that we also have $\Delta(v_1, w_1) \neq \Delta(v_1v_2, w_1w_2)$. We have (ii) $|v_2| = |w_2|$, $|v_1| \leq |w_1|$ and $v_1v_2 \not\preceq w_1w_2$. Note that for any $u, v, x, y \in \Sigma^*$ with $\Delta(u, v) = (x, y)$ we have $u \not\preceq v$ if and only if $x \neq \epsilon$ and $y \neq \epsilon$. Therefore, if we pose $(y_1, y_2) = \Delta(v_1v_2, w_1w_2)$, we have, by hypothesis, $v_1v_2 \not\preceq w_1w_2$ and so $y_1 \neq \epsilon$ and $y_2 \neq \epsilon$. Let pose $(x_1, x_2) = \Delta(v_1, w_1)$, if $(x_1, x_2) = (y_1, y_2)$ then $x_1 \neq \epsilon$ and $x_2 \neq \epsilon$, that is $v_1 \not\preceq w_1$, but then $\Delta(v_1v_2, w_1w_2) = \Delta(v_1, w_1) \cdot (v_2, w_2)$ which cannot be equal to $\Delta(v_1, w_1)$ (because, by hypothesis, one of v_2 and w_2 is not the empty word).

Now let suppose the HTP does not hold and let show that (i) or (ii) is satisfied for some words (as above). So there exist $q_0, q_0' \in I$, $q, q' \in Q$, $\sigma, \sigma' \in \Gamma^*$ and $u_1, v_1, w_1, u_2, v_2, w_2 \in \Sigma^*$ with :

$$(q_0, \bot) \xrightarrow{u_1/v_1} (q, \sigma) \xrightarrow{u_2/v_2} (q, \sigma) \qquad (q_0', \bot) \xrightarrow{u_1/w_1} (q', \sigma') \xrightarrow{u_2/w_2} (q', \sigma')$$

such that $\Delta(v_1, w_1) \neq \Delta(v_1v_2, w_1w_2)$. Moreover, let us suppose, by contradiction, that for all $k \in \mathbb{N}$, if we replace u_2 by ${u_2}^k$ (and thus v_2 and w_2 are replaced by ${v_2}^k$ and ${w_2}^k$ respectively), we get a system such that neither (i) nor (ii) do hold, that is for all $k \in \mathbb{N}$ we have $|{v_2}^k| = |{w_2}^k|$, $|v_1| \leq |w_1|$ and $v_1{v_2}^k \preceq w_1{w_2}^k$. We now prove that this implies $\Delta(v_1, w_1) = \Delta(v_1v_2, w_1w_2)$, which is a contradiction with the hypothesis. On the one hand, if for all k we have $v_1{v_2}^k \preceq w_1{w_2}^k$ then we have $w_1 = v_1{v_2}^a v_2'$ for some $a \in \mathbb{N}$ and some $v_2' \preceq v_2$. So, $\Delta(v_1, w_1) = \Delta(v_1, v_1{v_2}^a v_2') = (\epsilon, {v_2}^a v_2')$. On the other hand, $v_1{v_2}^k \preceq w_1{w_2}^k$ and $|v_2| = |w_2|$ implies that we have $w_1w_2 = v_1{v_2}^{a+1}v_2'$. So we have $\Delta(v_1v_2, w_1w_2) = \Delta(v_1v_2, v_1{v_2}^{a+1}v_2') = (\epsilon, {v_2}^a v_2')$. Therefore $\Delta(v_1, w_1) = \Delta(v_1v_2, w_1w_2)$. This concludes the proof. ∎

C.3 Proof of Theorem 10

Let T be an fVPT. We show that $[\![T]\!]$ is HBM iff T satisfies the HTP.

If $[\![T]\!]$ is HBM, then the HTP holds for T by Lemma 30 (proved in this section). Conversely, if T satisfies the HTP, then we apply Lemma 31 (proved this section) which bounds the maximal difference between outputs of T, and then Proposition 3 gives the complexity the evaluation algorithm.

LEMMA 30. *Let T be an fVPT. If $[\![T]\!]$ is HBM, then the HTP holds for T.*

PROOF. By definition of HBM and BM, if $[\![T]\!]$ is HBM and there exists $K \in \mathbb{N}$ such that for all $u \in Dom(T)$, $h(u) \leq K$, then $[\![T]\!]$ is BM.

Now 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 σ, σ' and states $q, q' \in Q$, $q_0, q_0' \in I$ and $q_f, q_f' \in F$ such that:

$$\begin{cases} (q_0, \bot) \xrightarrow{u_1/v_1} (q, \sigma) \xrightarrow{u_2/v_2} (q, \sigma) \xrightarrow{u_3/v_3} (q_f, \bot) \\ (q_0', \bot) \xrightarrow{u_1/w_1} (q', \sigma') \xrightarrow{u_2/w_2} (q', \sigma') \xrightarrow{u_3'/w_3} (q_f', \bot) \end{cases}$$

and $\Delta(v_1, w_1) \neq \Delta(v_1v_2, w_1w_2)$. Let $K = \max(h(u_1u_2u_3), h(u_1'u_2'u_3'))$. By definition of $\mathsf{FST}(T, K)$ (states are configurations of T) and of the twinning property for FSTs, the twinning property for FSTs does not hold for $\mathsf{FST}(T, K)$. Therefore $\mathsf{FST}(T, K)$ is not subsequentializable [25] and by Proposition 24 $[\![\mathsf{FST}(T, K)]\!]$ is not BM. Therefore $[\![T]\!]$ is not HBM, otherwise $[\![T]\!]$ could be evaluated in space complexity $f(h(u))$ on any input word u, for some function f. That corresponds to bounded memory if we fix the height of the words to K at most. ∎

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. We first prove that this maximal delay is exponentially bounded by the height of the word.

LEMMA 31. *Let T be an fVPT. If the HTP holds for T, then for all $s \in \Sigma^*$ we have $\mathsf{out}_{\neq}^{max}(s) \leq (|Q| \cdot |\Gamma|^{h(s)})^2 M$, where $M = \max\{|t| \mid (q, a, t, \gamma, q') \in \delta\}$.*

PROOF. Let $s \in \Sigma^*$. We consider two cases. We first assume that $s \in Dom(T)$. Let $u \in \Sigma^*$ be a prefix of s, we will prove that $\mathsf{out}_{\neq}(u) \leq (|Q| \cdot |\Gamma|^{h(u)})^2 M$. Note that there exist $N = |Q| \cdot |\Gamma|^{h(u)}$ configurations reachable by words of height less than $h(u)$. The proof is similar to that of [24] for FST. It proceeds by induction on the length of u. If $|u| \leq N^2$, then the result is trivial. Otherwise, assume that $|u| > N^2$ and let $(q, \sigma, w), (q', \sigma', w') \in Q \times \Gamma^* \times \Sigma^*$ such that there exist runs $\rho : (i, \bot) \xrightarrow{u|v} (q, \sigma)$ and $\rho' : (i', \bot) \xrightarrow{u|v'} (q', \sigma')$, with $i, i' \in I$, $v = \mathsf{lcp}_{\mathsf{in}}(u, T) \cdot w$,

$v' = \mathsf{lcp}_{\mathsf{in}}(u,T)\cdot w'$, and such that $\mathsf{out}_{\neq}(u) = |w|$. As $|u| > N^2$, we can decompose these two runs as follows:

$$\begin{cases} \rho : (i,\bot) & \xrightarrow{u_1/v_1} & (q_1,\sigma_1) & \xrightarrow{u_2/v_2} & (q_1,\sigma_1) & \xrightarrow{u_3/v_3} & (q,\sigma) \\ \rho' : (i',\bot) & \xrightarrow{u_1/v'_1} & (q'_1,\sigma'_1) & \xrightarrow{u_2/v'_2} & (q'_1,\sigma'_1) & \xrightarrow{u_3/v'_3} & (q',\sigma') \end{cases}$$

In addition, we have $u = u_1u_2u_3$, $u_2 \neq \varepsilon$, $v = \mathsf{lcp}_{\mathsf{in}}(u,T)\cdot w = v_1v_2v_3$, and $v' = \mathsf{lcp}_{\mathsf{in}}(u,T)\cdot w' = v'_1v'_2v'_3$. Indeed, by the choice of N, there must exist a pair of configurations that occurs twice. By the HTP property, we obtain $\Delta(v_1v_2, v'_1v'_2) = \Delta(v_1, v'_1)$. By Lemma 33 (see Appendix D.2), this entails the equality $\Delta(v_1v_2v_3, v'_1v'_2v'_3) = \Delta(\Delta(v_1v_2, v'_1v'_2)\cdot(v_3, v'_3)) = \Delta(\Delta(v_1, v'_1)\cdot(v_3, v'_3)) = \Delta(v_1v_3, v'_1v'_3)$. Thus, we obtain $\Delta(w,w') = \Delta(v,v') = \Delta(v_1v_2v_3, v'_1v'_2v'_3) = \Delta(v_1v_3, v'_1v'_3)$. As v_1v_3 and $v'_1v'_3$ are possible output words for the input word u_1u_3, whose length is strictly smaller than $|u|$, we obtain $|w| \leq \mathsf{out}_{\neq}(u_1u_3)$ and the result holds by induction.

We now consider the second case: $s \notin \mathit{Dom}(T)$. Let s' be the longest prefix of s such that there exists s'' such that $s's'' \in \mathit{Dom}(T)$. Since T is reduced (w.l.o.g., as explained in preliminaries), s' correspond to the longest prefix of s on which there exist a run of T. Therefore we have $\mathsf{out}^{\max}_{\neq}(s) \leq \mathsf{out}^{\max}_{\neq}(s')$ and we can apply the proof of the first case (as s' is a prefix of a word that belongs to $\mathit{Dom}(T)$) and we get $\mathsf{out}^{\max}_{\neq}(s') \leq (|Q|\cdot|\Gamma|^{h(s')})^2M$. Moreover, $h(s') \leq h(s)$, therefore $\mathsf{out}^{\max}_{\neq}(s) \leq \mathsf{out}^{\max}_{\neq}(s') \leq (|Q|\cdot|\Gamma|^{h(s)})^2M$ and we are done. ∎

D Quadratic Height Bounded Memory Evaluation

D.1 Decidability of the twinning property

PROOF. [Lemma 13] Let T be a fVPT. We construct in polynomial time a pushdown automaton with s counters † (one reversal) that accepts any word $u = u_1u_2u_3u_4$ that satisfies the premise of the TP 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 TP is not verified). Therefore the TP holds if and only if no word is accepted by the automaton. This can be checked in NPTIME [27].

The automaton simulates any two runs, and guesses the decomposition $u_1u_2u_3u_4$ (it checks that the decomposition is correct by verifying 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). With two counters it can check that $|v_2v_4| = |w_2w_4|$, if it is not the case then it accepts u (indeed the TP is therefore not verified). The automaton checks that $\Delta(v_1v_3, w_1w_3) \neq \Delta(v_1v_2v_3v_4, w_1w_2w_3w_4)$ with the hypothesis (checked in parallel) that $|v_2v_4| = |w_2w_4|$. Let A, B, C, D be four words with $A = a_1 \dots a_l, B = b_1 \dots b_m, C = c_1 \dots c_n, D = d_1 \dots d_p$, $l \geq m$. We show in Lemma 32 below, that $\Delta(A,B) \neq \Delta(C,D)$ holds if, and only if, at least one out of four simple conditions is true. For example, the first condition states that there exists k such that $a_{l-k} \neq b_{l-k}$ and either (i) $k \geq |C|$, or (ii) $c_{n-k} = d_{n-k}$. The automaton guesses which condition holds and verifies it with the help of counters.

We detail how to check the first condition, the others can be checked with the same technique. The automaton guesses the value k and with two counters verifies that $a_{l-k} \neq b_{l-k}$ as follows. First it initializes both counters to $l-k$ (e.g. with an epsilon loop that increments both counters). Then it counts the letters of both words A and B up to $l-k$ and records a_{l-k} and b_{l-k} and verifies that

† The number s does not depend on the transducer T.

they are not equal. Finally it must verify that either (i) or (ii) is satisfied. The verification of (i) is easy. To verify (ii), i.e. $c_{n-k} = d_{n-k}$, it uses two additional counters and proceeds similarly as for checking $a_{l-k} \neq b_{l-k}$, but checks equality instead of inequality. ∎

LEMMA 32. *Let $A, B, C, D \in \Sigma^*$, such that $A = a_1 \dots a_l, B = b_1 \dots b_m, C = c_1 \dots c_n, D = d_1 \dots d_p$, and $l - m = n - p \geq 0$. We have that $\Delta(A,B) \neq \Delta(C,D)$ if and only if one of the following conditions is satisfied:*

1. *there exists k such that $a_{l-k} \neq b_{l-k}$ and either (i) $k \geq |C|$, or (ii) $c_{n-k} = d_{n-k}$;*
2. *there exists k such that $c_{n-k} \neq d_{n-k}$ and either (i) $k \geq |A|$, or (ii) $a_{l-k} = b_{l-k}$;*
3. *there exists k such that $a_{l-k} \neq c_{n-k}$ and either (i) $k < l - m$, or (ii) there exists k' with $a_{k'} \neq b_{k'}$ and $k + k' \leq l$;*
4. *there exist k, k' such that $b_{m-k} \neq d_{p-k}$ and $a_{k'} \neq b_{k'}$ and $k + k' \leq m$.*

PROOF.

Let us define $E = A \wedge B$ and $F = C \wedge D$, and also $A', B', C', D' \in \Sigma^*$ such that $A = EA', B = EB', C = FC'$ and $D = FD'$, i.e. $(A', B') = \Delta(A,B)$ and $(C', D') = \Delta(C,D)$.

We first prove that each condition implies $\Delta(A,B) \neq \Delta(C,D)$, i.e. that $(A', B') \neq (C', D')$. If $|A'| \neq |C'|$ or $|B'| \neq |D'|$ then the result is immediate. Now, assume that $|A'| = |C'|$ and $|B'| = |D'|$.

1. By hypothesis we have $a_{l-k} \neq b_{l-k}$ therefore $|A'| \geq k+1$ (as $a_{l-k} \in A'$) and $|B'| \geq k+1-(l-m)$ (as $b_{l-k} \in B'$). Thus $|C'| \geq k+1$ and $|D'| \geq k+m-l+1$. In particular, this implies that we are in case (ii) as $|C| \geq |C'| \geq k+1$. We consider two cases: (a) if $a_{l-k} \neq c_{n-k}$ we have $A' \neq C'$ (because $a_{l-k} \in A'$ and $c_{n-k} \in C'$ and are at the same position in A' and C' as $|A'| = |C'|$), or (b) if $a_{l-k} = c_{n-k}$ this means that $b_{l-k} \neq d_{n-k}$ (because $a_{l-k} \neq b_{l-k}$ and $c_{n-k} = d_{n-k}$), and so $B' \neq D'$ (because $b_{l-k} \in B'$ and $d_{n-k} \in D'$ because $|B'| = |D'|$).
2. Similar to proof of 1.
3. We prove that condition (i) implies $\Delta(A,B) \neq \Delta(C,D)$. Condition (ii) is proved similarly. We know that $|A'| \geq |A| - |B| = l - m > k$ therefore $a_{l-k} \in A'$. Similarly, $|C'| = |A'| > k$, so $c_{n-k} \in C'$. By hypothesis $a_{l-k} \neq c_{n-k}$, therefore, as $|A'| = |C'|$, A' and C' differ on their kth letter from the right.
4. Similar to proof of 3.

Now suppose that $\Delta(A,B) \neq \Delta(C,D)$ and let us show that one of the conditions is satisfied. First note that if $B' = \varepsilon$, then B is a prefix of A, and thus $|A'| = l - m$. In particular, we obtain $|A'| \leq |C'| \leq |C|$. We call this property $(\dagger_B)$. Similarly, $D' = \varepsilon$ entails $|C'| \leq |A'| \leq |A|$, what we denote by $(\dagger_D)$.

We prove the property by considering several cases:

- $|A'| > |C|$: take $k = |A'| - 1$, we have $a_{l-k} \neq b_{l-k}$. Note that b_{l-k} does not exist iff $l - k > m$, *i.e.* $B' = \varepsilon$. By property $(\dagger_B)$, this can not occur. In addition, by definition of k, we have $k \geq |C|$ and thus condition $1.(i)$ is satisfied.
- $|C'| > |A|$: take $k = |C'| - 1$, we have $c_{n-k} \neq d_{n-k}$ (as above, $(\dagger_D)$ ensures that d_{l-k} is well defined) and $k \geq |A|$: condition $2.(i)$ is satisfied.
- $|A'| \leq |C|$ and $|C'| \leq |A|$:
  - $|A'| > |C'|$ (this implies $|B'| > |D'|$): take $k = |A'| - 1$, we have $a_{l-k} \neq b_{l-k}$ and $c_{n-k} = d_{n-k}$ (existence of b_{l-k} and d_{n-k} is ensured by $(\dagger_B)$ and $(\dagger_D)$): condition $1.(ii)$ is satisfied.

- $|C'| > |A'|$ (this implies $|D'| > |B'|$): take $k = |C'| - 1$, then condition 2.(ii) is satisfied.
- $|A'| = |C'|$ (this implies $|B'| = |D'|$), we suppose $A' \neq C'$ and prove that condition 3 holds (the case $B' \neq D'$ is similar with condition 4 holding). Because $A' \neq C'$ and they have the same size, there must be a $k < |A'|$ with $a_{l-k} \neq c_{n-k}$, we consider two cases:
  - $|A'| \leq |A| - |B|$, this implies that $k < |A| - |B| = l - m$ therefore condition 3.(i) is satisfied.
  - $|A'| > |A| - |B|$, take $k' = l - |A'| + 1$ (the first position of A' in A). Then we have $a_{k'} \neq b_{k'}$ and thus condition 3.(ii) is satisfied.

∎

D.2 TP is preserved by equivalent transducers

In this section we prove Theorem 14. We extend the concatenation to pairs of words and denote it by $\cdot$, i.e. $(u, v) \cdot (u', v') = (uu', vv')$.

Proof of Theorem 14 PROOF. We assume that T_1 is not twinned and show that T_2 is not twinned either. By definition of the TP there are two runs of the form

$$\begin{cases} (i_1, \bot) \xrightarrow{u_1|v_1} (p_1, \sigma_1) \xrightarrow{u_2|v_2} (p_1, \sigma_1\beta_1) \xrightarrow{u_3|v_3} (q_1, \sigma_1\beta_1) \xrightarrow{u_4|v_4} (q_1, \sigma_1) \\ (i'_1, \bot) \xrightarrow{u_1|v'_1} (p'_1, \sigma'_1) \xrightarrow{u_2|v'_2} (p'_1, \sigma'_1\beta'_1) \xrightarrow{u_3|v'_3} (q'_1, \sigma'_1\beta'_1) \xrightarrow{u_4|v'_4} (q'_1, \sigma'_1) \end{cases}$$

such that (q, σ_1) and (q', σ'_1) are co-accessible and $\Delta(v_1v_3, v'_1v'_3) \neq \Delta(v_1v_2v_3v_4, v'_1v'_2v'_3v'_4)$. We will prove that by pumping the loops on u_2 and u_4 sufficiently many times we will get a similar situation in T_2, proving that T_2 is not twinned. It is easy to show that there exist $k_2 > 0$, $k_1, k_3 \geq 0$, $w_i, w'_i \in \Sigma^*$, $i \in \{1, \ldots, 4\}$, some states $i_2, p_2, q_2, i'_2, p'_2, q'_2$ of T_2 and some stack contents $\sigma_2, \beta_2, \sigma'_2, \gamma'_2$ of T_2 such that we have the following runs in T_2:

$$\begin{cases} (i_2, \bot) \xrightarrow{u_1u_2^{k_1}|w_1} (p_2, \sigma_2) \xrightarrow{u_2^{k_2}|w_2} (p_2, \sigma_2\beta_2) \xrightarrow{u_2^{k_3}u_3u_4^{k_3}|w_3} (q_2, \sigma_2\beta_2) \xrightarrow{u_4^{k_2}|w_4} (q_2, \sigma_2) \\ (i'_2, \bot) \xrightarrow{u_1u_2^{k_1}|w'_1} (p'_2, \sigma'_2) \xrightarrow{u_2^{k_2}|w'_2} (p'_2, \sigma'_2\beta'_2) \xrightarrow{u_2^{k_3}u_3u_4^{k_3}|w'_3} (q'_2, \sigma'_2\beta'_2) \xrightarrow{u_4^{k_2}|w'_4} (q'_2, \sigma'_2) \end{cases}$$

such that (q_1, σ_1) and (q_2, σ_2) are co-accessible with the same input word u_5, and (q'_1, σ'_1) and (q'_2, σ'_2) are co-accessible with the same input word u'_5. Now for all $i \geq 0$, we let

$$\begin{array}{ll} V^{(i)} = v_1(v_2)^{k_1+ik_2+k_3}v_3(v_4)^{k_1+ik_2+k_3} & W^{(i)} = w_1(w_2)^i w_3(w_4)^i \\ V'^{(i)} = v'_1(v'_2)^{k_1+ik_2+k_3}v'_3(v'_4)^{k_1+ik_2+k_3} & W'^{(i)} = w'_1(w'_2)^i w'_3(w'_4)^i \\ D_1(i) = \Delta(V^{(i)}, V'^{(i)}) & D_2(i) = \Delta(W^{(i)}, W'^{(i)}) \end{array}$$

In other words, $D_1(i)$ (resp. $D_2(i)$) is the delay in T_1 (resp. T_2) accumulated on the input word $u_1(u_2)^{k_1+ik_2+k_3}u_3(u_4)^{k_1+ik_2+k_3}$ by the two runs of T_1 (resp. T_2).

There is a relation between the words $V^{(i)}$ and $W^{(i)}$. Indeed, since T_1 and T_2 are equivalent and (q_1, σ_1) and (q_2, σ_2) are both co-accessible by the same input word, for all $i \geq 1$, either $V^{(i)}$ is a prefix of $W^{(i)}$ or $W^{(i)}$ is a prefix of $V^{(i)}$, i.e. there exist $X \in \Sigma^*$ such that: for all $i \geq 1$,

$V^{(i)} = W^{(i)}X$ or for all $i \geq 1$ $V^{(i)}X = W^{(i)}$. Similarly, there exists $X' \in \Sigma^*$ such that for all $i \geq 1$, $V'^{(i)} = W'^{(i)}X'$ or for all $i \geq 1$, $V'^{(i)}X' = W'^{(i)}$.

We now prove the following key result: for all $i, j \geq 1$,

$$D_1(i) \neq D_1(j) \implies D_2(i) \neq D_2(j)$$

We consider two cases (the other ones being symmetric):

- for all $\ell \geq 1$, $V^{(\ell)} = W^{(\ell)}X$ and $V'^{(\ell)} = W'^{(\ell)}X'$. Then we have:

$$\begin{array}{lll} & \Delta(V^{(i)}, V'^{(i)}) & \neq \Delta(V^{(j)}, V'^{(j)}) \\ \Rightarrow & \Delta(W^{(i)}X, W'^{(i)}X') & \neq \Delta(W^{(j)}X, W'^{(j)}X') \\ \Rightarrow & \Delta(\Delta(W^{(i)}, W'^{(i)}) \cdot (X, X')) & \neq \Delta(\Delta(W^{(j)}, W'^{(j)}) \cdot (X, X')) \quad \text{(Lemma 33)} \\ \Rightarrow & \Delta(W^{(i)}, W'^{(i)}) & \neq \Delta(W^{(j)}, W'^{(j)}) \end{array}$$

- for all $\ell \geq 1$, $V^{(\ell)} = W^{(\ell)}X$ and $V'^{(\ell)}X' = W'^{(\ell)}$. Then we have:

$$\begin{array}{lll} & \Delta(V^{(i)}, V'^{(i)}) & \neq \Delta(V^{(j)}, V'^{(j)}) \\ \Rightarrow & \Delta(W^{(i)}X, V'^{(i)}) & \neq \Delta(W^{(j)}X, V'^{(j)}) \\ \Rightarrow & \Delta(\Delta(W^{(i)}, V'^{(i)}) \cdot (X, \epsilon)) & \neq \Delta(\Delta(W^{(j)}, V'^{(j)}) \cdot (X, \epsilon)) \quad \text{(Lemma 33)} \\ \Rightarrow & \Delta(W^{(i)}, V'^{(i)}) & \neq \Delta(W^{(j)}, V'^{(j)}) \\ \Rightarrow & \Delta(\Delta(W^{(i)}, V'^{(i)}) \cdot (\epsilon, X')) & \neq \Delta(\Delta(W^{(j)}, V'^{(j)}) \cdot (\epsilon, X')) \quad \text{(Lemma 34)} \\ \Rightarrow & \Delta(W^{(i)}, V'^{(i)}X') & \neq \Delta(W^{(j)}, V'^{(j)}X') \quad \text{(Lemma 33)} \\ \Rightarrow & \Delta(W^{(i)}, W'^{(i)}) & \neq \Delta(W^{(j)}, W'^{(j)}) \end{array}$$

Now by Lemma 36, since $\Delta(v_1v_3, v'_1v'_3) \neq \Delta(v_1v_2v_3v_4, v'_1v'_2v'_3v'_3)$, there exists $i_0 \geq 1$ such that for all $i, j \geq i_0$, if $i \neq j$ then $\Delta(v_1(v_2)^iv_3(v_4)^i, v'_1(v'_2)^iv'_3(v'_4)^i) \neq \Delta(v_1(v_2)^jv_3(v_4)^j, v'_1(v'_2)^jv'_3(v'_4)^j)$. In particular since $k_2 \geq 1$, we have $D_1(i) \neq D_1(j)$ for all $i, j \geq i_0$ and $i \neq j$. By the last intermediate result, we get $D_2(i_0) \neq D_2(i_0 + 1)$. Therefore the TP does not hold for T_2. ∎

x

LEMMA 33. *For all $u, u', v, v' \in \Sigma^*$, $\Delta(uu', vv') = \Delta(\Delta(u, v) \cdot (u', v'))$.*

PROOF. Let $X = uu' \wedge vv'$ and $Y = u \wedge v$. There exists A, B, C, D such that $A \wedge B = \epsilon$, $C \wedge D = \epsilon$, and:

$$\begin{array}{ll} uu' = XA & u = YC \\ vv' = XB & v = YD \\ \Delta(uu', vv') = (A, B) & \Delta(u, v) = (C, D) \end{array}$$

We have necessarily $|X| \geq |Y|$ since X is the longest common prefix of uu' and vv' and Y is the longest common prefix of u and v. Now we have $YCu' = XA$ and $YDv' = XB$, i.e. $Cu' = Y^{-1}XA$ and $Dv' = Y^{-1}XB$. Since $A \wedge B = \epsilon$, we have $\Delta(Cu', Dv') = (A, B)$, i.e. $\Delta(\Delta(u, v) \cdot (u', v')) = (A, B) = \Delta(uu', vv')$. ∎

LEMMA 34. *For all $u, u', v, v', w, w' \in \Sigma^*$, we have*

$$\Delta(\Delta(u, u') \cdot (w, w')) = \Delta(\Delta(v, v') \cdot (w, w')) \text{ iff } \Delta(u, u') = \Delta(v, v')$$

PROOF. There exists A, B, C, D and X, Y such that:

$$\Delta(u, u') = (A, B) \quad \Delta(v, v') = (C, D) \quad u = XA \quad u' = XB \quad v = YC \quad v' = YD$$

Let also E, F, G, H such that $\Delta(Aw, Bw') = (E, F)$ and $\Delta(Cw, Dw') = (G, H)$. Clearly, if $A = C$ and $B = D$, we have $E = G$ and $F = H$.

Conversely, suppose that $A \neq C$ (the case $B \neq D$ is symmetric). We show that $E \neq G$ or $F \neq H$. By definition of the delay, we know that $A \wedge B = \epsilon$, and $C \wedge D = \epsilon$. Therefore we have the following cases, for some words A', B', C', D' and letters a, b, c, d such that $a \neq b$ and $c \neq d$:

1. $A = aA'$ and $B = bB'$, $C = cC'$ and $D = dD'$ for some A', B', C', D'. Therefore $\Delta(Aw, Bw') = (Aw, Bw') = (E, F)$ and $\Delta(Cw, Dw') = (Cw, Dw') = (G, H)$. Since $A \neq C$, we get $E \neq G$;
2. $A = aA'$ and $B = bB'$, and $D = \epsilon$. Therefore $\Delta(Aw, Bw') = (Aw, Bw') = (E, F)$ and $\Delta(Cw, Dw') = \Delta(Cw, w') = (G, H)$. We have necessarily $|H| \leq |w'|$. Since $B \neq \epsilon$, we have $|F| = |Bw'| > |w'| \geq |H|$. Therefore $F \neq H$;
3. $A = aA'$ and $B = bB'$ and $C = \epsilon$. We can apply the same argument as case 2;
4. $A = \epsilon$ and $C = cC'$ and $D = dD'$. This case is symmetric to case 2;
5. $B = \epsilon$ and $C = cC'$ and $D = dD'$. This case is symmetric to case 2;
6. $A = \epsilon$ and $C = \epsilon$. This case is not possible since we have assumed $A \neq C$;
7. $A = \epsilon$ and $D = \epsilon$ ($C \neq \epsilon$). We have $\Delta(Aw, Bw') = \Delta(w, Bw')$ and $\Delta(Cw, Dw') = \Delta(Cw, w')$. Suppose that $E = G$ and $F = H$. Then there exists Z, Z' such that $w = ZE$, $Bw' = ZF$, $Cw = Z'E$ and $w' = Z'F$. Therefore $Bw' = BZ'F = ZF$, and $BZ' = Z$, so that $w = BZ'E$ and $Cw = CBZ'E = Z'E$, i.e. $CB = \epsilon$, which contradicts $C \neq \epsilon$;
8. $B = \epsilon$ and $C = \epsilon$. This case is symmetric to the previous case;
9. $B = \epsilon$ and $D = \epsilon$. Then we have $\Delta(Aw, Bw') = \Delta(Aw, w')$ and $\Delta(Cw, Dw') = \Delta(Cw, w')$. Again suppose that $E = G$ and $F = H$, therefore there exists Z, Z' such that $Aw = ZE$, $w' = ZF$, $Cw = Z'E$ and $w' = Z'F$. Therefore $Z = Z'$, which implies $Aw = Cw$. This contradicts $A \neq C$.

∎

LEMMA 35. *Let $v_1, v_2, v_3, v_4, w_1, w_2, w_3, w_4 \in \Sigma^*$ and for all $i \geq 0$, let*

$$V^{(i)} = v_1(v_2)^i v_3(v_4)^i \qquad W^{(i)} = w_1(w_2)^i w_3(w_4)^i.$$

If there exist $K \geq 0$ and $X \in \Sigma^*$ such that for all $i \geq K$, $V^{(i)} = W^{(i)}X$, then for all $i \geq 0$, $V^{(i)} = W^{(i)}X$.

PROOF. First note that we have $|v_2v_4| = |w_2w_4|$, this is a straight consequence from the fact that for all $i \geq K$, $V^{(i)} = W^{(i)}X$. We consider two cases:

- $|v_2| \neq |v_4|$: in that case we can show that the primitive roots of v_2, v_4, w_2, w_4 are conjugate (see for example [26]) and therefore have the same length. Therefore we can apply Theorem 1 of [28] which yields the result.
- $|v_2| = |v_4|$: in that case we also have $|w_2| = |w_4|$, and suppose $|v_1| \geq |w_1|$ (the case $|v_1| < |w_1|$ is similar). There exists $Y \in \Sigma^*$, such that for any $i \geq K$ we have $v_1{v_2}^i = w_1{w_2}^iY$ and $Yv_3{v_4}^i = w_3{w_4}^iX$. Therefore we can apply Theorem 2 of [28] which shows that these equalities hold for any i. For any i we have $V^{(i)} = v_1{v_2}^iv_3{v_4}^i = w_1{w_2}^iYv_3{v_4}^i = w_1{w_2}^iw_3{w_4}^iX = W^{(i)}X$.

∎

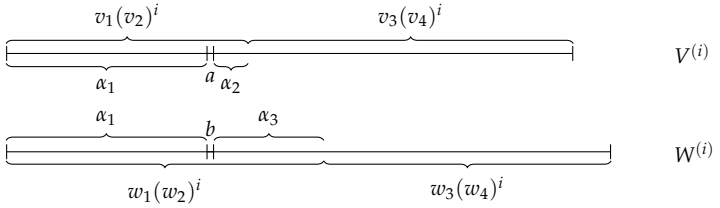
LEMMA 36. *Let $v_1, v_2, v_3, v_4, w_1, w_2, w_3, w_4 \in \Sigma^*$ and for all $i \geq 0$, let*

$$V^{(i)} = v_1(v_2)^i v_3(v_4)^i \qquad W^{(i)} = w_1(w_2)^i w_3(w_4)^i.$$

If $\Delta(V^{(0)}, W^{(0)}) \neq \Delta(V^{(1)}, W^{(1)})$, then there exists $i_0 \geq 1$ such that for all $i, j \geq i_0$, if $i \neq j$ then $\Delta(V^{(i)}, W^{(i)}) \neq \Delta(V^{(j)}, W^{(j)})$.

PROOF. First note that since $\Delta(V^{(0)}, W^{(0)}) \neq \Delta(V^{(1)}, W^{(1)})$, we clearly have $|v_2v_4| \neq \epsilon$ or $|w_2w_4| \neq \epsilon$, We write $u \preceq v$ if u is a prefix of v, and $u||v$ if u and v are incomparable, i.e. $u \wedge v = \epsilon$. We consider several cases:

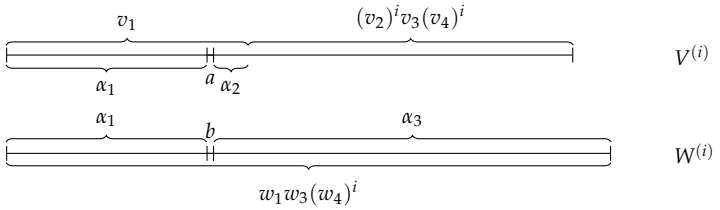
1. there is $K \geq 1$ such that for all $i \geq K$, $V^{(i)} \preceq W^{(i)}$ or $W^{(i)} \preceq V^{(i)}$. Consider the lengths of $V^{(i)}$ and $W^{(i)}$. When K is large enough, one of $V^{(i)}$ and $W^{(i)}$ is always the prefix of the other, for $i \geq K$: $V^{(i)} \preceq W^{(i)}$ if $|v_2v_4| \leq |w_2w_4|$, and $W^{(i)} \preceq V^{(i)}$ otherwise. Let us assume that $|v_2v_4| \leq |w_2w_4|$, i.e., for all $i \geq K$, there exists X_i such that $W^{(i)} = V^{(i)}X_i$. The other case is symmetric. We have $|W^{(i+1)}| - |W^{(i)}| = |w_2w_4| = |V^{(i+1)}| + |X_{i+1}| - |V^{(i)}| - |X_i|$, i.e. $|w_2w_4| = |V^{(i)}| + |v_2v_4| + |X_{i+1}| - |V^{(i)}| - |X_i|$, i.e. $|X_{i+1}| - |X_i| = |w_2w_4| - |v_2v_4|$. We again consider several cases:
   1.1 $|w_2w_4| > |v_2v_4|$. We have $|X_K| < |X_{K+1}| < |X_{K+2}| \ldots$, and by definition of the delay, $\Delta(V^{(i)}, W^{(i)}) = (\epsilon, X_i)$. Therefore the delay always increases in size as i icreases and we get the result;
   1.2 $|w_2w_4| = |v_2v_4|$. We show that this case is not possible. Indeed, it implies that $|X_K| = |X_{K+1}| = |X_{K+2}| \ldots$. Therefore by definition of $V^{(i)}$ and $W^{(i)}$, there exists $K' \geq K$ and $X \in \Sigma^*$ such that $X = X_{K'} = X_{K'+1} = X_{K'+2} \ldots$. By Lemma 35, we get $W^{(0)} = V^{(0)}X$ and $W^{(1)} = V^{(1)}X$, which contradicts $\Delta(W^{(0)}, V^{(0)}) \neq \Delta(W^{(1)}, V^{(1)})$.
2. for all $K \geq 1$, there is $i \geq K$ such that $V^{(i)}||W^{(i)}$. We show in this case that one of the two components of the delay always increases in size when i increases. We consider several cases depending on where the first difference between $V^{(i)}$ and $W^{(i)}$ occurs. The cases we consider also depend on K. In particular, by taking a large K it can reduce the number of cases we have to consider. For some $\alpha_1, \alpha_2, \alpha_3 \in \Sigma^*$ and $a, b \in \Sigma$ such that $a \neq b$, and for a K large enough, one of the following condition holds:
   2.1 there is $i \geq K$ such that $v_1(v_2)^i = \alpha_1 a \alpha_2$ and $w_1(w_2)^i = \alpha_1 b \alpha_3$, as illustrated below.



   Then for all $j \geq i$:

$$\Delta(V^{(j)}, W^{(j)}) = (a\alpha_2(v_2)^{j-i}v_3(v_4)^j, b\alpha_3(w_2)^{j-i}w_3(w_4)^j)$$

   Since $|v_2v_4| \neq \epsilon$ or $|w_2w_4| \neq \epsilon$, some of the two components of the delays is always increasing in size as j icreases, which proves the result;
   2.2 $w_2 = \epsilon$ and $v_1 = \alpha_1 a \alpha_2$ and there is $i \geq K$ such that $w_1w_3(w_4)^i = \alpha_1 b \alpha_3$, as illustrated below.

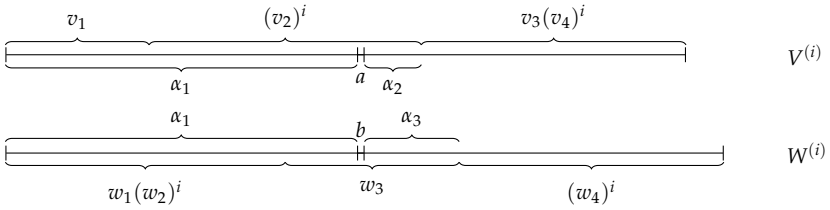


Then for all $j \geq i$:

$$\Delta(V^{(j)}, W^{(j)}) = (a\alpha_1 v_2^j v_3 (v_4)^j, b\alpha_3 (w_4)^{j-i})$$

Since $v_2 v_4 \neq \epsilon$ or $w_4 \neq \epsilon$, one of the two components of the delays is always increaing in size;

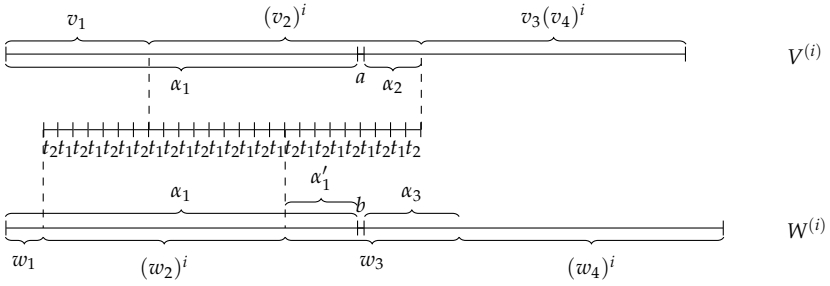
2.3 there is $i \geq K$ such that $v_1 (v_2)^i = \alpha_1 a \alpha_2$ and $w_1 (w_2)^i w_3 = \alpha_1 b \alpha_3$, with $|\alpha_1| \geq |v_1|$ and $|\alpha_1| \geq |w_1 (w_2)^i|$ (otherwise it is case 2.1). We also assume that $w_2 \neq \epsilon$ (otherwise it can be proved similarly as case 2.1). Therefore $v_2 \neq \epsilon$. This case is illustrated below:



We have taken K large enough, so that v_2^i and w_2^i have a common factor of length at least $|v_2| + |w_2|$. The strong theorem of Fine and Wilf [29] implies that the primitive roots of v_2 and w_2 are conjugate: there are $t_1, t_2 \in \Sigma^*$ and $n, p \geq 1$ such that $v_2 = (t_1 t_2)^n$ and $w_2 = (t_2 t_1)^p$. It can be shown (see for instance [26]) that we can choose t_1 and t_2 such that there exist $k_1, k_2 \geq 0$ verifying:

$$v_1 = (v_1 \wedge w_1)(t_2 t_1)^{k_1} t_2 \qquad w_1 = (v_1 \wedge w_1)(t_2 t_1)^{k_2}$$

Let α_1' be such that $\alpha_1 = w_1 (w_2)^i \alpha_1'$. There exist k_3, X and Y such that $\alpha_1' = (t_2 t_1)^{k_3} X$ with $t_2 t_1 = XaY$, and $w_3 = \alpha_1' b \alpha_3$, as illustrated below (we assume that $|w_1| < |v_1|$ on the picture).



We have for all $j \geq i$:

$$\begin{aligned}
& \Delta(V^{(j)}, W^{(j)}) \\
= \ & \Delta(v_1(v_2)^j v_3(v_4)^j, w_1(w_2)^j w_3(w_4)^j) \\
= \ & \Delta((v_1 \wedge w_1)(t_2t_1)^{k_1} t_2(t_1t_2)^{jn} v_3(v_4)^j, w_1(w_2)^j w_3(w_4)^j) \\
= \ & \Delta((v_1 \wedge w_1) t_2(t_1t_2)^{k_1+jn} v_3(v_4)^j, w_1(w_2)^j w_3(w_4)^j) \\
= \ & \Delta((v_1 \wedge w_1) t_2(t_1t_2)^{k_1+jn} v_3(v_4)^j, (v_1 \wedge w_1)(t_2t_1)^{k_2}(w_2)^j w_3(w_4)^j) \\
= \ & \Delta(t_2(t_1t_2)^{k_1+jn} v_3(v_4)^j, (t_2t_1)^{k_2+jp} w_3(w_4)^j) \\
= \ & \Delta((t_2t_1)^{k_1+jn} t_2 v_3(v_4)^j, (t_2t_1)^{k_2+jp} w_3(w_4)^j)
\end{aligned}$$

We now consider the following subcases:

2.3.1 $|v_2| > |w_2|$. As a consequence, $n > p$, and we can assume that K is big enough, so that $k_1 - k_2 + j(n-p) > 0$. We get:

$$\begin{aligned}
\Delta(V^{(j)}, W^{(j)}) &= \Delta((t_2t_1)^{k_1-k_2+j(n-p)} t_2 v_3(v_4)^j, w_3(w_4)^j) \\
&= \Delta((t_2t_1)^{k_1-k_2+j(n-p)} t_2 v_3(v_4)^j, \alpha_1' b \alpha_3(w_4)^j) \\
&= \Delta((t_2t_1)^{k_1-k_2+j(n-p)} t_2 v_3(v_4)^j, (t_2t_1)^{k_3} X b \alpha_3(w_4)^j)
\end{aligned}$$

We can take j such that $k_1 - k_2 + j(n-p) > k_3$, and therefore we finally have:

$$\begin{aligned}
& \Delta(V^{(j)}, W^{(j)}) \\
= \ & \Delta((t_2t_1)^{k_1-k_2+j(n-p)-k_3} t_2 v_3(v_4)^j, X b \alpha_3(w_4)^j) \\
= \ & \Delta(Xa Y(t_2t_1)^{k_1-k_2+j(n-p)-k_3-1} t_2 v_3(v_4)^j, X b \alpha_3(w_4)^j) \\
= \ & (aY(t_2t_1)^{k_1-k_2+j(n-p)-k_3-1} t_2 v_3(v_4)^j, b \alpha_3(w_4)^j)
\end{aligned}$$

Since $t_2t_1 \neq \epsilon$, we get that the first component of the delay always increases in size when j increases;

2.3.2 $|v_2| < |w_2|$. We can take K large enough such that this case never happens, i.e. $(v_2)^i$ and w_3 do not overlap.

2.3.3 $|v_2| = |w_2|$. Therefore $n = p$, and we have for all $j \geq i$:

$$\Delta(V^{(j)}, W^{(j)}) = \Delta((t_2t_1)^{k_1} t_2 v_3(v_4)^j, (t_2t_1)^{k_2} w_3(w_4)^j)$$

As case 2.3.1, since by hypothesis there is an overlap between $(v_2)^i$ and w_3, we have $k_1 > k_2 + k_3$, and we get:

$$\begin{aligned}
\Delta(V^{(j)}, W^{(j)}) &= \Delta((t_2t_1)^{k_1-k_2} t_2 v_3(v_4)^j, w_3(w_4)^j) \\
&= \Delta((t_2t_1)^{k_1-k_2} t_2 v_3(v_4)^j, (t_2t_1)^{k_3} X b \alpha_3(w_4)^j) \\
&= \Delta((t_2t_1)^{k_1-k_2-k_3} t_2 v_3(v_4)^j, X b \alpha_3(w_4)^j) \\
&= \Delta(Xa Y(t_2t_1)^{k_1-k_2-k_3-1} t_2 v_3(v_4)^j, X b \alpha_3(w_4)^j) \\
&= (aY(t_2t_1)^{k_1-k_2-k_3-1} t_2 v_3(v_4)^j, b \alpha_3(w_4)^j)
\end{aligned}$$

Therefore if $v_4 \neq \epsilon$ and $w_4 \neq \epsilon$, we are done as one of the two components of the delay will increase in size when j increases. If $v_4 = w_4 = \epsilon$ we can explicitly give the form of $\Delta(V^{(0)}, W^{(0)})$ and $\Delta(V^{(1)}, W^{(1)})$:

$$\begin{aligned}
\Delta(V^{(0)}, W^{(0)}) &= \Delta((t_2t_1)^{k_1} t_2 v_3, (t_2t_1)^{k_2+k_3} X b \alpha_3) \\
&= \Delta((t_2t_1)^{k_1-k_2-k_3} t_2 v_3, X b \alpha_3) \\
&= \Delta((t_2t_1)^{k_1} t_2(t_1t_2)^n v_3, (t_2t_1)^{k_2+k_3+n} X b \alpha_3) \\
&= \Delta(V^{(1)}, W^{(1)})
\end{aligned}$$

This is excluded by hypothesis, so this case is not possible.

2.4 the other cases (the first difference occurs between $(v_2)^i$ and $(w_4)^i$, or between v_3 and w_3, or between v_3 and $(w_4)^i$, or between $(v_4)^i$ and $(w_4)^i$) are proved similarly as case 2.3 by decomposing the words as power of their primitive roots. For instance, if the first difference occurs between v_3 and w_3, then either $v_2 = w_2 = \epsilon$ and it is the same as case 2.1, or $v_2 \neq \epsilon$ and $w_2 = \epsilon$ but we can take K large enough so that this case is impossible, or $v_2 \neq \epsilon$ and $w_2 \neq \epsilon$. In this latter case we can take K large enough so that the primitive roots of v_2 and w_2 are conjugate. We have again to distinguish several cases on the relative lengths of v_2 and w_2 (as for 2.3) but the proofs are similar. Similar techniques were already applied to prove that functionality is decidable for VPTs [26].

D.3 Proof of Theorem 16

We can use the same proof as the proof of back direction of Theorem 10, the only difference is the lemma that bounds the maximal difference between outputs of T.

We prove the following lemma, which states that the TP implies that in the evaluation algorithm, the delays stored by the algorithm can be bounded linearly in the height of the input word.

LEMMA 37. *Let T be an* fVPT*. If the TP holds for T, then for any word $s \in \Sigma^*$, we have* $\mathsf{out}^{max}_{\neq}(s) \leq (h(s)+1) \cdot \Big((|Q| \cdot |\Gamma|^{|Q|^4})^2 + 1\Big) \cdot M$*, where* $M = \max\{|t| \mid (q,a,t,\gamma,q') \in \delta\}$.

PROOF. Let $s \in \Sigma^*$. We assume that $s \in Dom(T)$ (we can handle the case $s \notin Dom(T)$ as in the proof of Lemma 31 for the HTP). We use the notion of *current height* $hc(u)$ of a prefix u of s as defined at the beginning of Appendix A. Consider a word u prefix of s. There exists a unique decomposition of u as follows: $u = u_0c_1u_1c_2 \dots u_{n-1}c_nu_n$, where $n = hc(u)$, and for any i, we have $c_i \in \Sigma_c$ and u_i is well-nested. Indeed, as $n = hc(u)$, the word u contains exactly n pending calls, that correspond to c_i's, and other parts of u can be gathered into well-nested words.

If each of the u_i's is such that $|u_i| \leq (|Q| \cdot |\Gamma|^{|Q|^4})^2$, then the property holds as length of word u can be bounded by $(hc(u)+1) \cdot \Big((|Q| \cdot |\Gamma|^{|Q|^4})^2 + 1\Big)$.

Otherwise, we prove that there exists a strictly shorter input word that produces the same delays as u when evaluating the transduction on it. Therefore, consider a word w such that $(q_0, \bot) \xrightarrow{u/w} (q,\sigma)$ for some $q_0 \in I$. Then there exist runs ϱ, ϱ' in T producing respectively as output words v and v', such that $v = (v \wedge v') \cdot w$. Consider the smallest index i such that $|u_i| > (|Q| \cdot |\Gamma|^{|Q|^4})^2$. We distinguish two cases:

1. if $h(u_i) \leq |Q|^4$, then we can reduce the length of u_i using the HTP by exhibiting two configurations occurring twice in runs ϱ and ϱ'. This yields an input word u', strictly shorter than u, that produces the same delays as u (see the proof of Lemma 31).
2. if $h(u_i) > |Q|^4$, then we prove that we can "pump vertically" u_i, and then reduce its length too. Indeed, let k be the first position in word u_i at which height $h(u_i)$ is obtained. As u_i is well-nested, we can define for each $0 \leq j < h(u_i)$ the unique position $left(j)$ (resp. $right(j)$) as the largest index, less than k (resp. the smallest index, larger than k), whose height is j (see Figure 7). As $h(u_i) > |Q|^4$, there exist two heights j and j' such that configurations reached at positions $left(j)$, $left(j')$, $right(j)$ and $right(j')$ in runs ϱ and ϱ' satisfy the premises of the twinning property, considering the prefix $u_0c_1 \dots c_iu_i$ of u. 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 33, this entails that the delays reached after u and u' are the same, proving the result. ∎

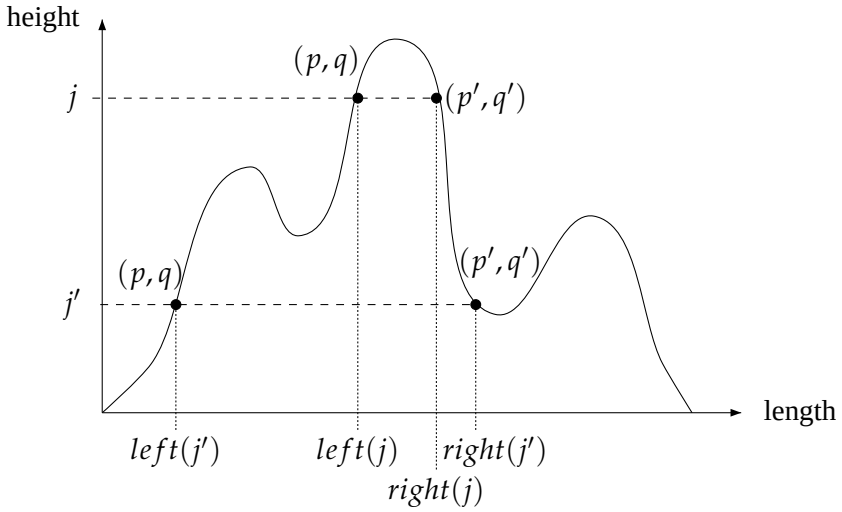


Figure 7: Vertical pumping in a well-nested word

References

[24] M.-P. Béal and O. Carton. Determinization of transducers over finite and infinite words. *Theor. Comput. Sci.*, 289(1):225–251, 2002.

[25] C. Choffrut. Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles. *Theor. Comput. Sci.*, 5(3):325–337, 1977.

[26] E. Filiot, J.-F. Raskin, P.-A. Reynier, F. Servais, and J.-M. Talbot. On functionality of visibly pushdown transducers. *CoRR*, abs/1002.1443, 2010.

[27] E. Filiot, J.-F. Raskin, P.-A. Reynier, F. Servais, and J.-M. Talbot. Properties of visibly pushdown transducers. In *MFCS*, volume 6281 of *LNCS*, pages 355–367. Springer, 2010.

[28] S. Holub and J. Kortelainen. On systems of word equations with simple loop sets. *Theor. Comput. Sci.*, 380(3):363–372, 2007.

[29] M. Lothaire. *Combinatorics on words*. Cambridge University Press, 1997.

[30] A. Weber and R. Klemm. Economy of description for single-valued transducers. *Inf. Comput.*, 118(2):327–340, 1995.