**Noname manuscript No.**
(will be inserted by the editor)

# Polylog space compression, pushdown compression, and Lempel-Ziv are incomparable

**Elvira Mayordomo** ⋆ · **Philippe Moser** ⋆ ·
**Sylvain Perifel**

Received: date / Accepted: date

**Abstract** The pressing need for efficient compression schemes for XML documents has recently been focused on stack computation [11,17], and in particular calls for a formulation of information-lossless stack or pushdown compressors that allows a formal analysis of their performance and a more ambitious use of the stack in XML compression, where so far it is mainly connected to parsing mechanisms. In this paper we introduce the model of pushdown compressor, based on pushdown transducers that compute a single injective function while keeping the widest generality regarding stack computation.

We also consider online compression algorithms that use at most polylogarithmic space (plogon). These algorithms correspond to compressors in the data stream model.

We compare the performance of these two families of compressors with each other and with the general purpose Lempel-Ziv algorithm. This comparison is made without any a priori assumption on the data's source and considering the asymptotic compression ratio for infinite sequences. We prove that in all cases they are incomparable.

**Keywords** compression algorithms · plogon · computational complexity · data stream algorithms · Lempel-Ziv algorithm · pushdown compression

⋆ Research supported in part by Spanish Government MEC and the European Regional Development Fund (ERDF) under Projects TIN2005-08832-C03-02 and TIN2008-06582-C03-02.

Elvira Mayordomo
Departamento de Informática e Ingeniería de Sistemas, Instituto de Investigación en Ingeniería de Aragón (I3A), María de Luna 1, Universidad de Zaragoza, 50018 Zaragoza, SPAIN.
E-mail: elvira@unizar.es

Philippe Moser
Department of Computer Science, National University of Ireland Maynooth, Maynooth, Co. Kildare, Ireland.
E-mail: pmoser@cs.nuim.ie

Sylvain Perifel
LIAFA (Université Paris Diderot – Paris 7, CNRS), Paris, France.
E-mail: sylvain.perifel(at)liafa.jussieu.fr

## 1 Introduction

The compression algorithms that are required for today massive data applications necessarily fall under very limited resource restrictions. For instance, in the case of the data stream setting, the algorithm receives a stream of elements one-by-one and can only store a brief summary of them, in fact the amount of available memory is far below linear [3,14]; and in the context of XML data bases (XML stands for EXtensible Markup Language, it specifies a standard way to add structure to data) the main limiting factor being document size renders the use of syntax directed compression particularly appropriate, i.e. compression centered on the grammar-based generation of XML-texts and performed with stack memory [11,17].

In this paper we introduce and formalize useful compression mechanisms that can be implemented within low resource-bounds, namely pushdown compressors and poly-logarithmic space online compression algorithms. In order to study the strengths and the limitations of these two compression mechanisms, we compare their compression ratios with each other. In particular, we address the following questions. While a stack mechanism seems particularly useful on structured documents like XML, it can also be used on other kinds of texts: how then does it perform? How does it compare with low-resource online compression (which at first sight could be thought of as less specific)? The first has a particular way to handle memory, but it does not have the limitation on memory size of the second.

It is also interesting to compare their performances with the widely-used general-purpose compression algorithm of Lempel and Ziv [18]. This is a particular algorithm, not a family of algorithms like our two models, but it is not low-resource. How does it perform in comparison with our two models? This question is especially important since Lempel-Ziv algorithm was already compared with another kind of low-resource compression. More specifically, finite state compressors (an even more low-resource compression mechanism) were extensively used and studied before the celebrated result of Lempel and Ziv [18] that their algorithm is asymptotically better than any finite-state compressor. However, until recently the natural extension of finite-state to pushdown compressors has received much less attention, a situation that has changed due to new specialized compressors for XML. It is therefore natural to ask if the result of Lempel and Ziv transposes to pushdown compressors, or if there are situations where pushdown compression performs better.

The work done on stack transducers has been basic and very connected to parsing mechanisms. Transducers were initially considered by Ginsburg and Rose in [9] for language generation, further corrected in [10], and summarized in [5]. For these models the role of nondeterminism is specially useful in the concept of $\lambda$-rule, that is a transition in which a symbol is popped from the stack without reading any input symbol.

We introduce here the concept of pushdown compressor as the most general stack transducer that is compatible with information-lossless compression. We allow the use of $\lambda$-rules while having a deterministic (unambiguous) model. The existence of end-markers is also allowed, since it allows the compressor to move away from mere prefix extension. A more feasible model will also be considered where the pushdown compressor is required to be invertible by a pushdown transducer (see Section 3.1). As mentioned before, stack compression is especially adequate for XML-texts and has been extensively used [11,17]. In the context of XML compression, it is standard to consider *visibly* pushdown automata [4,15] (where the stack behavior is determined by

the *type* of the input symbol; see section 3.1 for a definition) which is an even more restrictive computation model, which we will also analyze.

Polylogarithmic space online compressors (plogon) are compression algorithms that use at most polylogarithmic memory while accessing the input only once. This type of algorithms models the compression that can actually be performed in the setting of data streams, where sublinear space bounds and online input access are assumed, with constant and polylogarithm being the main bounds [3, 14].

For the comparison of different compression mechanisms we consider asymptotic compression ratio for infinite sequences, and without any a priori assumption on the data's source. Notice that this excludes results that assume a certain probability distribution on the data, for instance the fact that under an ergodic source, the Lempel-Ziv compression coincides exactly with the entropy of the source with high probability on finite inputs [18]. This last result is useful when the data source is known, but it is not informative for arbitrary inputs, i.e. when the data source is unknown (notice that an infinite sequence is Lempel-Ziv incompressible with probability one). Therefore for the comparison of compression algorithms on general sequences, either an experimental or a formal approach is needed, such as that used in [16]. In this paper we follow [16] using a worst case approach, that is, we consider asymptotic performance on every infinite sequence.

We prove that the performance of plogon compressors, pushdown compressors and Lempel-Ziv's compression scheme is incomparable in the strongest sense. For each two of these three mechanisms we construct a sequence that is compressed optimally in one scheme but is not in the other, and vice-versa. In all cases the separation is the strongest possible, i.e. optimal compressibility is achieved in the worst case (i.e. almost all prefixes of the sequence are optimally compressible), whereas incompressibility is present even in the best case (i.e. only finitely many prefixes of the sequence are compressible).

For the comparison of pushdown transducers with both plogon and Lempel-Ziv, we use the most general pushdown model (where the pushdown compressor need not be invertible by a pushdown transducer) for incompressibility and the more restrictive (where the pushdown compressor is required to be invertible by a pushdown transducer) for compressibility, thus obtaining the tightest results.

The proofs are interesting by themselves, since the witnesses of each of the separations proved show the strengths and drawbacks of each of the compression mechanisms. For instance pushdown compressors cannot take advantage of patterns, while Lempel-Ziv algorithm compresses well even non correlative repetitions, and plogon machines require extra information to compress this kind of data.

This paper contains a revised version of the results in [2] and [21].

The paper is organized as follows. Section 2 contains some preliminaries. In section 3, we present pushdown compressors and plogon compressor along with some basic properties and notations, as well as a review of the Lempel-Ziv (LZ78) algorithm. In section 4 we present our main results. We end with a brief final remark on connections and consequences of these results for effective dimension and prediction algorithms.

## 2 Preliminaries

Let us fix some notation for strings and languages. Let $\Sigma$ be finite alphabet with at least two symbols. W.lo.g. we assume that $0, 1 \in \Sigma$. A *string* is an element of $\Sigma^n$ for some integer $n$ and a *sequence* is an element of $\Sigma^\infty$. For a string $x$, its length is denoted

by $|x|$. If $x, y$ are strings, we write $x \leq y$ (called lexicographic order) if $|x| < |y|$ or $|x| = |y|$ and $x$ precedes $y$ in alphabetical order. The empty string is denoted by $\lambda$. For $S \in \Sigma^\infty$ and $i, j \in \mathbb{N}$, we write $S[i..j]$ for the string consisting of the $i^{\text{th}}$ through $j^{\text{th}}$ symbols of $S$, with the convention that $S[i..j] = \lambda$ if $i > j$, and $S[1]$ is the leftmost symbol of $S$. We say string $y$ is a prefix of string (sequence) $x$, denoted $y \sqsubset x$, if there exists a string (sequence) $a$ such that $x = ya$. For a string $x$, $x^{-1}$ denotes $x$ written in reverse order. For a function $f : A \to B$, $f(x) = \perp$ means $f$ is not defined on input $x$. For a sum $\sum_{j=1}^{n} a_j$ let term$(k)$ denote $a_k$. For a function $f$, $f^{(2)}$ denotes $f \circ f$.

Given a sequence $S$ and a function $T : \Sigma^* \to \Sigma^*$, the $T$- upper and lower compression ratios of $S$ are given by

$$\rho_T(S) = \liminf_{n\to\infty} \frac{|T(S[1 \ldots n])|}{n}, \text{ and}$$
$$\mathrm{P}_T(S) = \limsup_{n\to\infty} \frac{|T(S[1 \ldots n])|}{n}.$$

We use $K(w)$ to denote the standard (plain) Kolmogorov complexity, that is, fix a universal Turing Machine $U$. Then for each string $w \in \Sigma^*$,

$$K(w) = \min\{|p| \,|\, p \in \{0,1\}^*, U(p) = w\}$$

i.e., $K(w)$ is the size of the shortest binary program that makes $U$ output $w$. Although some authors use $C(w)$ to denote (plain) Kolmogorov complexity, we reserve this notation to denote a particular compression algorithm $C$ on input $w$.

## 3 Compressors with low resource-bounds

In this section we consider several families of lossless compression methods that use very low computing resources. We introduce a detailed definition of stack-computable compressors together with some variants and review poly-logarithmic space computable compressors and the celebrated Lempel-Ziv algorithm.

### 3.1 Pushdown compressors

We discuss next different formalizations of information lossless compressors that are equipped with stack memory. The most general ones are allowed to use a bounded number of lambda-rules, that is, stack movements that don't consume an input symbol. The most restricted pushdown compressors we consider here are visibly pushdown automata that are suitable for XML compression.

There are several natural variants for the model of pushdown transducer [5], both allowing different degrees of nondeterminism and computing partial (multi)functions by requiring final state or empty stack termination conditions. But our purpose here is to compute a total and well-defined (single valued) function (we call such a function a compressor), therefore nondeterminism should be very limited and natural termination conditions are equivalent.

The main variants that will influence the computing power of a pushdown compressor while remaining information lossless are the presence of lambda-rules, the possible restrictions of stack movements, and the use of an endmarker, that is an extra symbol signaling the end of the finite input.

We will introduce here pushdown compressors, invertible pushdown compressors, and visibly pushdown compressors (this last one defined in [4,15]). The definitions below are adapted from those in [2,21].

Intuitively a pushdown compressor (PDC) is a stack equipped automata, that after reading both the next input symbol and the topmost stack symbol, enters a new state, outputs a symbol, and pushes (resp. pops) a symbol to (resp. from) the stack. There is a special type of computation step (called $\lambda$-transition) where the PDC does not read the next input symbol, and only performs a computation with its stack. To avoid cases where the compressor would take too much time computing only using its stack and not reading its input, we bound the number of allowed $\lambda$-transition per input symbol, we refer to this as a bounded pushdown compressor (BPDC). Here is a definition.

**Definition 1** A *bounded pushdown compressor* (BPDC) is an 8-tuple

$$C = (Q, \Sigma, \Gamma, \delta, \nu, q_0, z_0, c)$$

where

- $Q$ is a finite set of states
- $\Sigma$ is the finite input/output alphabet
- $\Gamma$ is the finite stack alphabet
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \to Q \times \Gamma^*$ is the transition function
- $\nu : Q \times \Sigma \times \Gamma \to \Sigma^*$ is the output function
- $q_0 \in Q$ is the initial state
- $z_0 \in \Gamma$ is the start stack symbol
- $c \in \mathbb{N}$ is an upper bound on the number of $\lambda$-rules per input symbol.

We use $\delta_Q$ and $\delta_{\Gamma^*}$ for the projections of function $\delta$. We restrict $\delta$ so that $z_0$ cannot be removed from the stack bottom, that is, for every $q \in Q$, $b \in \Sigma \cup \{\lambda\}$, either $\delta(q, b, z_0) = \perp$, or $\delta(q, b, z_0) = (q', vz_0)$, where $q' \in Q$ and $v \in \Gamma^*$.

Note that the transition function $\delta$ accepts $\lambda$ as an input character in addition to elements of $\Sigma$, which means that $C$ has the option of not reading an input character while altering the stack, such a movement is called a *$\lambda$-rule*. In this case $\delta(q, \lambda, a) = (q', \lambda)$, that is, we pop the top symbol of the stack. To enforce determinism, we require that at least one of the following hold for all $q \in Q$ and $a \in \Gamma$:

- $\delta(q, \lambda, a) = \perp$,
- $\delta(q, b, a) = \perp$ for all $b \in \Sigma$.

We *restrict the number of $\lambda$-rules* that can be applied as follows: between the input symbols in positions $n$ and $n + 1$ a maximum of $c$ $\lambda$-rules can be applied.

We first consider the transition function $\delta$ as having inputs in $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$, meaning that only the top symbol of the stack is relevant. Then we use the extended transition function $\delta^* : Q \times \Sigma^* \times \Gamma^+ \to Q \times \Gamma^*$, defined recursively as follows. For $q \in Q$, $v \in \Gamma^+$, $w \in \Sigma^*$, and $b \in \Sigma$

$$\delta^*(q, \lambda, v) = \begin{cases} \delta^*(\delta_Q(q, \lambda, v), \lambda, \delta_{\Gamma^*}(q, \lambda, v)), & \text{if } \delta(q, \lambda, v) \neq \perp; \\ (q, v), & \text{otherwise.} \end{cases}$$

$$\delta^*(q, wb, v) = \begin{cases} \delta^*(\delta_Q(\delta^*_Q(q, w, v), b, \delta^*_{\Gamma^*}(q, w, v)), \lambda, \delta_{\Gamma^*}(\delta^*_Q(q, w, v), b, \delta^*_{\Gamma^*}(q, w, v))), \\ \qquad \text{if } \delta^*(q, w, v) \neq \perp \text{ and } \delta(\delta^*_Q(q, w, v), b, \delta^*_{\Gamma^*}(q, w, v)) \neq \perp; \\ \perp, \quad \text{otherwise.} \end{cases}$$

That is, $\lambda$-rules are implicit in the definition of $\delta^*$. We abbreviate $\delta^*$ to $\delta$, and $\delta(q_0, w, z_0)$ to $\delta(w)$. We define the *output* from state $q$ on input $w \in \Sigma^*$ with $z \in \Gamma^*$ on the top of the stack by the recursion $\nu(q, \lambda, z) = \lambda$,

$$\nu(q, wb, z) = \nu(q, w, z)\, \nu(\delta_Q(q, w, z), b, \delta_{\Gamma^*}(q, w, z)).$$

The *output* of the compressor $C$ on input $w \in \Sigma^*$ is the string $C(w) = \nu(q_0, w, z_0)$.

The input of an information-lossless compressor can be reconstructed from the output and the final state reached on that input.

**Definition 2** A BPDC $C = (Q, \Sigma, \Gamma, \delta, \nu, q_0, z_0, c)$ is *information-lossless* (IL) if the function

$$\begin{aligned} \Sigma^* &\to \Sigma^* \times Q \\ w &\mapsto (C(w), \delta_Q(w)) \end{aligned}$$

is one-to-one. An *information-lossless bounded pushdown compressor* (ILBPDC) is a BPDC that is IL.

Intuitively, a BPDC *compresses* a string $w$ if $|C(w)|$ is significantly less than $|w|$. Of course, if $C$ is IL, then not all strings can be compressed. Our interest here is in the degree (if any) to which the prefixes of a given sequence $S \in \Sigma^\infty$ can be compressed by an ILBPDC.

We will also consider BPDC that have endmarkers, which enables the BPDC to know when its input ends, a feature that achieve better compression rates in some cases.

**Definition 3** An *information-lossless bounded pushdown compressor with endmarkers* (ILBPDCwE) is a BPDC $C = (Q, \Sigma \cup \{\$\}, \Gamma, \delta, \nu, q_0, z_0, c)$ with input alphabet $\Sigma \cup \{\$\}$ ($\$ \notin \Sigma$) such that the function

$$\begin{aligned} \Sigma^* &\to \Sigma^* \times Q \\ w &\mapsto (C(w\$), \delta_Q(w)) \end{aligned}$$

is one-to-one.

Notice that the use of endmarkers can improve compression. In particular each ILBPDC is a particular case of ILBPDC with endmarkers, but there are ILBPDC with endmarkers that perform better than usual ILBPDC.

We will denote as pushdown compression ratio the concept corresponding to the more general family of pushdown compressors, those that use endmarkers.

The best-case *pushdown compression ratio* of a sequence $S \in \Sigma^\infty$ is $\rho_{PD}(S) = \inf\{\rho_C(S) \mid C \text{ is an ILBPDCwE}\}$.
The worst-case *pushdown compression ratio* of a sequence $S \in \Sigma^\infty$ is $\mathrm{P}_{PD}(S) = \inf\{\mathrm{P}_C(S) \mid C \text{ is an ILBPDCwE}\}$.

Notice that so far we have not required that the computation should be invertible by another pushdown transducer, which is a natural requirement for practical compression schemes. The standard PD compression model does not guarantee the decompression to be feasible and it is currently not known whether the exponential time brute force

inversion can even be improved to polynomial time. To guarantee both decompression and compression to be feasible, we require the existence of a PD machine that given the compressed string (and the final state), outputs the decompressed one. This yields two PD compression schemes, the standard one (PD) and invertible PD. Contrary to Finite State computation, it is not known whether both are equivalent. This is by no means a limitation, since all results in this paper are always stated in the strongest form, i.e. we obtain results of the form "X beats PD" and "invertible PD beats X".

Here is the definition of invertible PD compressors. We want this definition to be the most restrictive one and therefore it is based on regular ILBPDC.

**Definition 4** $(C, D)$ is an invertible PD compressor (denoted invPD) if $C$ is an ILBPDC and $D$ is a PD transducer s.t. $D(C(w), \delta_Q(w)) = w$, i.e. $D$, given both $C(w)$ and the final state, outputs $w$.

The best-case *invertible pushdown compression ratio* of a sequence $S \in \Sigma^\infty$ is $\rho_{\text{invPD}}(S) = \inf\{\rho_C(S) \mid \ C \text{ is an invPD}\}$.
The worst-case *invertible pushdown compression ratio* of a sequence $S \in \Sigma^\infty$ is $\mathrm{P}_{\text{invPD}}(S) = \inf\{\mathrm{P}_C(S) \mid \ C \text{ is an invPD}\}$.

We end this section with the concept of visibly pushdown automata from [4,15] that is extensively used in the compression of XML.

A *visibly pushdown compressor* (visiblyPD) is an information-lossless bounded pushdown compressor for which the input alphabet has three types of symbols, call symbols, return symbols, and internal symbols. The main restriction is that while reading a call, the automaton must push one symbol, while reading a return symbol, it must pop one symbol (if the stack is non-empty), and while reading an internal symbol, it can only update its control state.

Therefore the compression ratio attained by visibly pushdown automata is an upper bound on the compression ratio attained through the pushdown compressors defined above.

### 3.2 Plogon compressors

We introduce the family of compressors that can be computed online with at most polylogarithmic space. Notice that these resource bounds correspond to those of the data stream model [3,14], where the input size is massive in comparison with the available memory, and the input can only be read once.

**Definition 5** (Hartmanis, Immerman, Mahaney [12]) A Turing machine $M$ is a plogon transducer if it has the following properties, for each input string $w$

- the computation of $M(w)$ reads its input from left to right (no turning back),
- $M(w)$ is given $|w|$ written in binary (on a special tape),
- $M(w)$ writes the output from left to right on a write-only output tape,
- $M(w)$ uses memory bounded by $\log(|w|)^c$, for a constant $c$.

We denote with plogon the class of plogon transducers.

Note that contrary to Finite State transducers (and similarly to ILBPDCwE), a plogon transducer is not necessarily a mere extender, i.e., there is a plogon transducer $M$ and strings $w, x$ such that $M(wx) \not\supset M(w)$.

**Definition 6** A plogon transducer $C : \Sigma^* \to \Sigma^*$ is an information lossless compressor (ILplog) if it is 1-1.

The best-case *plogon compression ratio* of a sequence $S \in \Sigma^\infty$ is $\rho_{\text{plogon}}(S) = \inf\{\rho_C(S) \mid C \text{ is an ILplog}\}$.
The worst-case *plogon compression ratio* of a sequence $S \in \Sigma^\infty$ is $\mathrm{P}_{\text{plogon}}(S) = \inf\{\mathrm{P}_C(S) \mid C \text{ is an ILplog}\}$.

### 3.3 Lempel-Ziv compression scheme

Let us give a brief description of the classical LZ78 algorithm [18]. Given an input $x \in \Sigma^+$, LZ parses $x$ into different phrases $x_i$, i.e., $x = x_1 x_2 \dots x_n$ ($x_i \in \Sigma^+$) such that every proper prefix $y \sqsubset x_i$, appears before $x_i$ in the parsing (i.e. there exists $j < i$ s.t. $x_j = y$). Therefore for every $i$, $x_i = x_{l(i)} b_i$ for $l(i) < i$ and $b_i \in \Sigma$. We sometimes denote the number of phrases in the parsing of $x$ as $P(x)$. After step $i$ of the algorithm, the $i$ first phrases $x_1, \dots, x_i$ have been parsed and stored in the so-called *dictionary*. Thus, each step adds one word to the dictionary.

LZ encodes $x_i$ by a prefix free encoding of $l(i)$ and the symbol $b_i$, that is, if $x = x_1 x_2 \dots x_n$ as before, the output of LZ on input $x$ is

$$LZ(x) = c_{l(1)} b_1 c_{l(2)} b_2 \dots c_{l(n)} b_n$$

where $c_i$ is a prefix-free coding of $i$ (and $x_0 = \lambda$).

For a string $z = xy$ we denote by $LZ(y|x)$ the output of LZ on $y$ after having read $x$ already.

LZ is usually restricted to the binary alphabet, but the description above is valid for any alphabet $\Sigma$.

## 4 The performances of the LZ78 algorithm, plogon compressors and pushdown compressors are incomparable

In this section we prove that the two families of compressors we have introduced, pushdown and plogon compressors, and the Lempel-Ziv compression scheme, are all incomparable. That is, for any pair among those three, there are different individual sequences on which one is outperformed by the other and vice versa. In all cases we get low worst-case rate ($\rho$) for one method versus high best-case rate (P) for the other, i.e. the widest possible separation between them.

### 4.1 Lempel-Ziv beats Pushdown compression

Our first result shows that there is a sequence that our most general family of pushdown compressors cannot compress and that is optimally compressible by Lempel-Ziv.

The proof is based on two intuitions, that require a careful analysis. The first one is that from a few Kolmogorov-random strings a much longer pushdown-incompressible string can be constructed. On the other hand, a sequence with enough (and non-consecutive) repeated substrings can be compressed optimally by Lempel-Ziv.

**Theorem 1** *There exists a sequence $S$ such that*

$$\mathrm{P}_{LZ}(S) = 0$$

*and*

$$\rho_{PD}(S) = 1.$$

*Proof* Consider the sequence $S = S_1 S_2 \ldots$ where $S_n$ is constructed as follows. Let $x = x_1 x_2 \ldots x_{n^2}$ ($|x_i| = n$) be a Kolmogorov-random string with $K(x) \geq n^3 \log |\Sigma|$. Let

$$S_n = x_{i_1} \ldots x_{i_l}$$

where $i_j \in \{1, \ldots, n^2\}$ for every $1 \leq j \leq l$ are indexes, defined later on. Let

$$l = \frac{1}{n} \sum_{k=1}^{n} k \min(|\Sigma|^k, n^{\frac{2k}{n}+1})$$

so that

$$|S_n| = nl = \sum_{k=1}^{n} k \min(|\Sigma|^k, n^{\frac{2k}{n}+1}). \tag{1}$$

Let us show that for every $\epsilon > 0$ and for $n$ large enough

$$n^{5-\epsilon} \leq |S_n| \leq n^5. \tag{2}$$

We prove the first inequality.

$$|S_n| = \sum_{k=1}^{n} k \min(|\Sigma|^k, n^{\frac{2k}{n}+1}) \leq n \,\mathrm{term}(n) \leq n \cdot n \cdot n^{\frac{2n}{n}+1} = n^5.$$

For the second inequality we have

$$\begin{aligned} |S_n| &= \sum_{k=1}^{n} k \min(|\Sigma|^k, n^{\frac{2k}{n}+1}) \\ &\geq \sum_{k=(1-\frac{\epsilon}{4})n}^{n} k \min(|\Sigma|^k, n^{\frac{2k}{n}+1}) \\ &\geq \frac{n\epsilon}{4} \,\mathrm{term}((1-\frac{\epsilon}{4})n) \\ &\geq n^{5-\epsilon}. \end{aligned}$$

Let $C_1, C_2, \ldots$ be an enumeration of all ILBPDCwE such that $C_i$ can be encoded in at most $i$ bits and such that a maximum of $\log^{(2)} i$ $\lambda$-rules can be applied per symbol. The following claim shows that there are many $C$-incompressible strings $x_i$.

*Claim* Let $F_n = \{C_1, \ldots, C_{\log n}\}$. Let $w \in \Sigma^*$.

1. Let $C \in F_n$. There are at least $(1 - \frac{1}{2 \log n})n^2$ strings $x_i$ ($1 \leq i \leq n^2$) such that
$$|C(wx_i)| - |C(w)| > n - 2\sqrt{n}.$$
2. There is a string $x_i$ such that for every $C \in F_n$,
$$|C(wx_i)| - |C(w)| > n - 2\sqrt{n}.$$

*Proof* (of Claim 4.1) After having read $w$, $C$ is in state $q$, with stack content $yz$, where $y$ denotes the $n \log^{(2)} n$ topmost symbols of the stack (if the stack is shorter then $y$ is the whole stack). It is clear that while reading an $x_i$, $C$ will not pop the stack below $y$.

Let $T = (1 - \frac{1}{2 \log n})n^2$, and let $C(q, yz, x_i\$)$ denote the output of $C$ when started in state $q$ on input $x_i\$$ with stack content $yz$. Suppose the claim false, i.e. there exist more than $n^2 - T$ words $x_i$ such that $C(q, yz, x_i\$) = p_i$, ends in state $q_i$, and $|p_i| \leq n - 2\sqrt{n} + O(1)$ (notice that the output on symbol \$ is $O(1)$). Denote by $G$ the set of such strings $x_i$. This yields the following short program for $x$ (coded with alphabet $\Sigma$):

$$p = (n, C, q, y, a_1 t_1 a_2 t_2 \dots a_{n^2} t_{n^2})$$

where each comma costs less than $3 \log |s|$, where $s$ is the element between two commas; $a_i = 1$ implies $t_i = x_i$, $a_i = 0$ implies $x_i \in G$ and $t_i = d(q_i)01d(|p_i|)01p_i$ (where $d(z)$ for any string $z$, is the string written with every symbol doubled), i.e. $|t_i| \leq n - \sqrt{n}$. $p$ is a program for $x$: once $n$ is known, each $a_i t_i$ yields either $x_i$ (if $a_i = 1$) or $(p_i, q_i)$ (if $a_i = 0$). From $(p_i, q_i)$, simulating $C(q, yz, u\$)$ for each $u \in \Sigma^n$ yields the unique $u = x_i$ such that $C(q, yz, u\$) = p_i$ and ends in state $q_i$. The simulations are possible, because $C$ does not read its stack further than $y$, which is given. We have

$$\begin{aligned} |p| &\leq O(\log n) + n \log^{(2)} n + (n+1)T + (n^2 - T)(n - \sqrt{n}) \\ &\leq O(n^2) + n^3 - \frac{n^{2.5}}{2 \log n} \\ &\leq n^3 - \frac{n^{2.5}}{4 \log n} \end{aligned}$$

which contradicts the randomness of $x$, thus proving part 1.

Let $W_j$ be the set of strings $x_i$ that are compressible by $C_j$; by 1., $|W_j| \leq n^2 / 2 \log n$. Let $R = \{x_i\}_{i=1}^{n^2} - \cup_{j=1}^{\log n} W_j$ be the set of strings incompressible by all $C \in F_n$. We have

$$|R| \geq n^2 - \log n \cdot n^2 / 2 \log n = n^2/2 > 1.$$

This proves part 2.

We finish the definition of $S_n$ by picking $x_{i_1}$ to be the first string fulfilling the second part of Claim 4.1 for $w = S_1 S_2 \dots S_{n-1}$. The construction is similar for all strings $\{x_{i_j}\}_{j=2}^{l}$, by taking $w = S_1 S_2 \dots S_{n-1} x_{i_1} \dots x_{i_{j-1}}$, thus ending the construction of $S_n$.

Let us show that $\rho_{PD}(S) = 1$. Let $\epsilon > 0$. Let $C = C_k$ be an ILBPDCwE; then for almost every $n$, and for all $0 \le t \le |S_n|/n$, $0 \le i < n$ we have

$$\begin{aligned}
&\frac{|C(S_1 \dots S_{n-1} S_n[1 \dots tn+i]\$)|}{|S_1 \dots S_{n-1} S_n[1 \dots tn+i]|} \\
&\ge \frac{\sum_{j=k}^{n-1}(j - 2\sqrt{j})|S_j|/j + t(n - 2\sqrt{n}) - O(1)}{\sum_{j=1}^{n-1}|S_j| + (t+1)n} \\
&\ge 1 - \frac{\sum_{j=1}^{k-1}|S_j|}{\sum_{j=1}^{n-1}|S_j| + (t+1)n} - 2\frac{\sum_{j=k}^{n-1}|S_j|/\sqrt{j}}{\sum_{j=1}^{n-1}|S_j| + (t+1)n} - 2\frac{t\sqrt{n} + n/2}{\sum_{j=1}^{n-1}|S_j| + (t+1)n} \\
&\ge 1 - \epsilon/4 - O(1)\frac{\sum_{j=1}^{n-1} j^{4.5}}{\sum_{j=1}^{n-1} j^{5-\delta}} - \epsilon/4 \qquad \text{(by Equation 2)} \\
&\ge 1 - \epsilon/2 - \frac{O(1)(n-1)\,\mathrm{term}(n-1)}{\frac{n}{3}\,\mathrm{term}(\frac{n}{3})} \\
&\ge 1 - \epsilon/2 - \frac{O(1)(n-1)(n-1)^{4.5}}{\frac{n}{3}(\frac{n}{3})^{5-\delta}} \\
&\ge 1 - \epsilon/2 - \epsilon/2 = 1 - \epsilon \qquad \text{(choosing } \delta = 0.1\text{)}
\end{aligned}$$

i.e. $\rho_{PD}(S) = 1$.

We show that $\mathrm{P}_{LZ}(S) = 0$. Suppose LZ has already parsed input $S_1 \dots S_{n-1}$, and has $d_n$ words in its dictionary ($d_n \le n|S_n|$). Let $P$ be the parsing of $S_n$ by LZ, let $t_P$ be the size of the largest string in $P$ and let $1 \le k \le t_P$. Let us compute the maximum number of strings of size $k$ in $P$. Any string $u$ of size $k$ in a parsing of $S_n$ is of the form

$$u = x_{t_1}[t \dots n]x_{t_2} \dots x_{t_{k/n}}$$

i.e. amounts to choose $k/n$ strings $x_{t_i}$ and the position $1 \le t \le n$ where $u$ starts in $x_{t_1}$. Therefore there are at most $\#k = n \cdot (n^2)^{k/n} = n^{1+2k/n}$ such words $u$ of size $k$.

Let $P_w$ be the worst-case parsing of $S_n$, that starts on an empty dictionary and parses all possible strings of size $k$ in $S_n$ (for every $k \le t_w$), where $t_w$ is the size of the largest string in $P_w$ i.e., $\min(|\Sigma|^1, n^{1+2/n})$ strings of size one are parsed, followed by $\min(|\Sigma|^2, n^{1+4/n})$ strings of size 2, ..., followed by $\min(|\Sigma|^k, n^{1+2k/n})$ strings of size $k$, and so on. Because

$$\sum_{k=1}^{n} k \min(|\Sigma|^k, n^{\frac{2k}{n}+1}) = |S_n|$$

we have $t_w \le n$.

Let $p$ (resp. $p_w$) be the number of phrases in $P$ (resp. $P_w$). We have $p \le p_w$, and $|LZ(S_n|S_1 \dots S_{n-1})| \le p \log(p + d_n)$. Since

$$p_w = \sum_{k=1}^{t_w} \min(|\Sigma|^k, n^{\frac{2k}{n}+1}) \le n\,\mathrm{term}(n) = n^4$$

we have

$$|LZ(S_n|S_1 \dots S_{n-1})| \le n^4 \log(n^4 + n|S_n|) \le n^{4+\alpha}$$

where $\alpha > 0$ can be arbitrary small.

Let $0 \leq t \leq |S_n|/n$, $0 \leq i < n$. We have

$$\begin{aligned}
\frac{|LZ(S_1 \dots S_{n-1} S_n[1 \dots tn+i])|}{|S_1 \dots S_{n-1} S_n[1 \dots tn+i]|} &\leq \frac{\sum_{j=1}^{n-1} |LZ(S_j|S_1 \dots S_{j-1})| + |LZ(S_n|S_1 \dots S_{n-1})|}{\sum_{j=1}^{n-1} |S_j|} \\
&\leq \frac{\sum_{j=1}^{n-1} |LZ(S_j|S_1 \dots S_{j-1})|}{\sum_{j=1}^{n-1} |S_j|} + \frac{n^{4+\alpha}}{\sum_{j=1}^{n-1} |S_j|} \\
&\leq \frac{\sum_{j=1}^{n-1} j^{4+\alpha}}{\sum_{j=1}^{n-1} j^{5-\delta}} + \frac{n^{4+\alpha}}{\sum_{j=1}^{n-1} j^{5-\delta}} \\
&\leq \epsilon/2 + \epsilon/2 \leq \epsilon
\end{aligned}$$

i.e. $\mathrm{P}_{LZ}(S) = 0$.

### 4.2 Lempel-Ziv beats plogon compressors

The Lempel-Ziv algorithm can also surpass plogon compressors. Our second comparison detects sequences on which Lempel-Ziv achieves optimal compression whereas a plogon compressor has the worst possible performance. The construction is based on repetition of Kolmogorov random strings. We show that Lempel-Ziv works well on any repeated pattern, whereas in polylogarithmic space big patterns cannot be stored.

**Theorem 2** *There exists a sequence $S$ such that*

$$\mathrm{P}_{LZ}(S) = 0 \quad \textit{and } \rho_{\mathrm{plogon}}(S) = 1.$$

The proof will use the following general property that bounds the output of Lempel-Ziv on strings of the form $w = u^n$.

**Lemma 1** *Let $n \in \mathbb{N}$ and let $u \in \Sigma^*$, where $u \neq \lambda$. Define $l = 1 + |u|$ and $w = u^n$. Consider the execution of Lempel-Ziv on $w$ starting from a dictionary containing $d \geq 0$ phrases. Then we have that*

$$|LZ(w)| \leq \sqrt{2l|w|}\log(d + \sqrt{2l|w|}) \tag{3}$$

*Proof* (of Lemma 1) Let us fix $n$ and consider the execution of Lempel-Ziv algorithm on $w$: as it parses the word, it enlarges its dictionary of phrases. Fix an integer $k$ and let us bound the number of new words of size $k$ in the dictionary. As the algorithm parses $|u|$, the number of different words of size $k$ in $u^n$ is at most $|u|$ (at most one beginning at each symbol of $u$). Therefore we obtain a total of at most $|u|$ different new words of size $k$ in $w$. This total is bounded from above by $l = |u| + 1$.

Therefore at the end of the algorithm and for all $k$, the dictionary contains at most $l$ new words of size $k$. We can now bound from above the size of the compressed image of $w$. Let $p$ be the number of new phrases in the parsing made by Lempel-Ziv algorithm. The size of the compression is then $p\log(p+d)$: indeed, the encoding of each phrase consists in a new symbol and a pointer towards one of the $p+d$ words of the dictionary. The only remaining step is thus to evaluate the number $p$ of new words in the dictionary.

Let us order the words of the dictionary by increasing length and call $t_1$ the total length of the first $l$ words (that is, the $l$ smallest words), $t_2$ the total length of the $l$

following words (that is, words of index between $l+1$ and $2l$ in the order), and so on: $t_k$ is the sum of the size of the words with index between $(k-1)l+1$ and $kl$. Since the sum of the size of all these words is equal to $|w|$, we have

$$|w| = \sum_{k \geq 1} t_k.$$

Furthermore, since for each $k$ there are at most $l$ new words of size $k$, the words taken into account in $t_k$ all have size at least $k$: hence $t_k \geq kl$. Thus we obtain

$$|w| = \sum_{k \geq 1} t_k \geq \sum_{k=1}^{p/l} kl \geq \frac{p^2}{2l}.$$

Hence $p$ satisfies

$$\frac{p^2}{2l} \leq |w|, \text{ that is, } p \leq \sqrt{2l|w|}.$$

The size of the compression of $w$ is $p \log(p+d) \leq \sqrt{2l|w|} \log(d + \sqrt{2l|w|})$, which ends the proof of Lemma 1.

*Proof* (of Theorem 2) Let $A, c \in \mathbb{N}$ with $c \geq 7$. For each $i \in \mathbb{N}$, let $R_i$ be a Kolmogorov random string with $|R_i| = i$ (i.e. $K(R_i) > i \log |\Sigma| - A$ for $A$ the constant just fixed). Let

$$S_n = R_1 R_2^{2^c} R_3^{3^c} \dots R_n^{n^c}$$

($R_n^{n^c}$ means $n^c$ copies of $R_n$) and let $S$ be the infinite sequence having all $S_n$ as prefixes.

The following three lemmas will analyze the performance of Lempel-Ziv on all prefixes of $S$.

**Lemma 2**

$$\frac{|LZ(S_n)|}{|S_n|} \leq \frac{n^{\frac{c+6}{2}}}{n^{c+1}}$$

*for $n$ large enough.*

*Proof* (of Lemma 2) Denote by $LZ(i|i-1)$ the output of LZ on $R_i^{i^c}$, after having parsed $S_{i-1}$ already.

Using the notation of Lemma 1, let $w = R_i^{i^c}$; thus $l = 1 + |R_i| = 1 + i$, and $d \leq |S_{i-1}| \leq (i-1)^{c+2}$. Thus

$$|LZ(i|i-1)| \leq \sqrt{2(i+1)i^{c+1}} \log((i-1)^{c+2} + \sqrt{2(i+1)i^{c+1}}) < i^{(c+3)/2}$$

for $i$ large enough ($i \geq N_0$). Thus for $n$ sufficiently large

$$\begin{aligned} |LZ(S_n)| &= \sum_{j=1}^{n} |LZ(j|j-1)| \\ &= \sum_{j=1}^{N_0-1} |LZ(j|j-1)| + \sum_{j=N_0}^{n} |LZ(j|j-1)| \\ &\leq n + n \cdot n^{(c+3)/2} \leq n^{(c+6)/2} \end{aligned}$$

for $n$ large enough, which ends the proof of Lemma 2.

**Lemma 3** *Let* $S_{n,t} = R_1 R_2^{2^c} R_3^{3^c} \dots R_n^{n^c} R_{n+1}^t$ *where* $1 \leq t < (n+1)^c$. *Then*

$$\frac{|LZ(S_{n,t})|}{|S_{n,t}|} \leq \frac{n^{(c+7)/2}}{n^{c+1}}$$

*for* $n$ *large enough.*

*Proof* (of Lemma 3)
Using Lemma 2 we have

$$\begin{aligned} |LZ(S_{n,t})| &= |LZ(S_n)| + |LZ(R_{n+1}^t|S_n)| \\ &\leq n^{(c+6)/2} + |LZ(R_{n+1}^t|S_n)| \end{aligned}$$

Applying Lemma 1 with $w = R_{n+1}^t$, $d \leq |S_n| \leq n^{c+2}$, $l = n+2$, $|w| = t(n+1)$ yields (for $n$ large enough)

$$\begin{aligned} |LZ(R_{n+1}^t|S_n)| &\leq \sqrt{2t(n+1)(n+2)}\log(n^{c+2} + \sqrt{2t(n+1)(n+2)}) \\ &\leq n^{3/2}\sqrt{t} \leq n^{(c+5)/2}. \end{aligned}$$

Whence

$$\frac{|LZ(S_{n,t})|}{|S_{n,t}|} \leq \frac{n^{(c+6)/2} + n^{(c+5)/2}}{n^{c+1}} \leq \frac{n^{(c+7)/2}}{n^{c+1}}$$

which ends the proof of Lemma 3.

**Lemma 4** *For almost every* $k$, $\frac{|LZ(S[1\dots k]|)}{k} \leq k^{(-1+9/(c+3))/2}$ *i.e., for any* $c \geq 7$, $\mathrm{P}_{LZ}(S) = 0$.

*Proof* (of Lemma 4) Let $k \in \mathbb{N}$ and let $n, t, l$ $(0 \leq l \leq n,\ 0 \leq t < (n+1)^c)$ be such that $S[1\dots k] = S_n R_{n+1}^t R_{n+1}[1\dots l]$. On $R_{n+1}[1\dots l]$, LZ outputs at most $l\log(S[1\dots k]) = O(n\log n)$ symbols. Since $k \leq (n+1)^{c+2} < n^{c+3}$, Lemma 3 yields

$$\frac{|LZ(S[1\dots k])|}{k} \leq \frac{n^{(c+7)/2} + O(n\log n)}{n^{c+1}} \leq n^{(-c+6)/2} \leq k^{(-1+9/(c+3))/2}\ .$$

Let us show that the sequence $S$ is not compressible by ILplogs. For this we show that each large substring $x$ of the input that is a Kolmogorov random word cannot be compressed by a plogon transducer, independently of the computation performed before processing $x$.

Let $C$ be an ILplog. For strings $z, \alpha, \beta, x$ with $z = \alpha x \beta$ and $|z| = m$, denote by $C(s, x, m)$ the output of $C$ starting in configuration $s$ and reading $x$ out of an input of length $m$. A valid configuration, is a configuration $s$ such that there exists a string $c$ such that $C(s_0, c, m)$ ends in configuration $s$, where $s_0$ is the start configuration of $C$. For example if $s$ is the configuration of $C$ after reading $a$, then $C(s, x, m)$ is the output of $C$ while reading part $x$ of input $z = axb$. Note that $|s| \leq \log(m)^{O(1)}$.

**Lemma 5** *Let* $C$ *be an ILplog, running in space* $\log^a m$, *and let* $0 < T \leq 1$. *Then for every* $d \in \mathbb{N}$ *and almost every* $r \in \mathbb{N}$, *for every random string* $x \in \Sigma^r$ *(with* $K(x) \geq T|x|\log|\Sigma| - A$ *for some fixed constant A), for every* $M$ *with* $|x| \leq M \leq |x|^d$ *and for every valid configuration* $s$ *(*$|s| \leq \log^a M$*)*

$$|C(s, x, M)| \geq T|x| - \log^{2a}|x|.$$

*Proof* (of Lemma 5) Suppose by contradiction that $C(s,x,M)=p$, with $|p|<Tr-\log^{2a} r$; denote by $s^x$ the configuration of $C$ after having read $x$ starting in $s$. Then $p'=(s^x,s,M,r,p)$ ($p'$ is encoded by doubling all symbols in $s^x,s,M,r$, separated by the delimiter 01 followed by $p$) yields a program for $x$ (coded with alphabet $\Sigma$):
"Find $y$ with $|y|=r$ such that $C(s,y,M)=p$, and $C$ ends in configuration $s^x$ after reading $y$."

$y$ is unique because otherwise suppose there are two strings $y,y'$ ($|y|=|y'|$) such that $C(s,y,M)=C(s,y',M)$, and $C$ ends in the same configuration on $y$ and $y'$. Let $b$ be a string that brings $C$ into configuration $s$. Then for $z=1^{M-|by|}$ we have $C(byz)=C(by'z)$ which contradicts $C$ being 1-1. Therefore $y$ is unique, i.e. $y=x$. Thus for $r$ sufficiently large

$$|p'|\le 2(|s^x|+|s|+|M|+|r|)+|p|\le 2(\log^a r^d+\log^a r^d+\log r^d+\log r)+Tr-\log^{2a} r$$
$$\le Tr-\frac{\log^{2a} r}{2}$$

which contradicts the randomness of $x$.

**Lemma 6** *Let $C$ be an ILplog, running in space $\log^a m$. Then for every $\epsilon>0$ and for almost every $m$, $\frac{C(S[1\cdots m])}{m}>1-\epsilon$ i.e., $\rho_{\text{plogon}}(S)=1$.*

*Proof* of Lemma 6 Let $\epsilon>0$ and let $\epsilon'=\frac{\epsilon}{4\cdot 3^{c+2}}$. Let $n,t,l$ ($0\le l\le n$, $0\le t<n^c$) be such that $S[1\dots m]=S_{n-1}R_n^tR_n[1\dots l]$.

The idea is to apply Lemma 5 to $R_{\epsilon' n}^{(\epsilon' n)^c}\dots R_{n-1}^{(n-1)^c}R_n^tR_n[1\dots l]$. Let $d$ be such that $(\epsilon' n)^d\ge n^{c+2}$ (for all $n\ge 2$), i.e. $(\epsilon' n)^d\ge m$. By Lemma 5, $C$ on input $S[1\dots m]$, will output at least $j-\log^{2a} j$ symbols on each $R_j$ ($\epsilon' n\le j\le n$). Therefore

$$|C(S[1\dots m])|\ge\sum_{j=\epsilon' n}^{n-1}(j-\log^{2a} j)j^c+t(n-\log^{2a} n)$$

whence

$$\begin{aligned}\frac{|C(S[1\dots m])|}{m}&\ge\frac{\sum_{j=\epsilon' n}^{n-1}j^c(j-\log^{2a} j)+t(n-\log^{2a} n)}{\sum_{j=1}^{n-1}j^{c+1}+(t+1)n}\\&\ge\frac{\sum_{j=\epsilon' n}^{n-1}j^c(j-\alpha j)+t(n-\log^{2a} n)}{\sum_{j=1}^{n-1}j^{c+1}+(t+1)n}\\&\ge\frac{(1-\alpha)(\sum_{j=\epsilon' n}^{n-1}j^{c+1}+(t+1)n)}{(1+\alpha')(\sum_{j=1}^{n-1}j^{c+1}+(t+1)n)}-\frac{(1-\alpha)n}{\sum_{j=1}^{n-1}j^{c+1}+(t+1)n}\end{aligned}$$

where $\alpha,\alpha'>0$ can be chosen arbitrarily small (for $n$ large enough). Let $\alpha,\alpha'>0$ be such that $\frac{1-\alpha}{1+\alpha'}>1-\epsilon/2$. Thus

$$\begin{aligned}\frac{|C(S[1\dots m])|}{m}&\ge\frac{1-\alpha}{1+\alpha'}-\frac{1-\alpha}{1+\alpha'}\cdot\frac{\sum_{j=1}^{\epsilon' n-1}j^{c+1}}{\sum_{j=1}^{n-1}j^{c+1}}-\epsilon/4\ge\frac{1-\alpha}{1+\alpha'}-\frac{\epsilon' n^{c+2}}{n/3(n/3)^{c+1}}-\epsilon/4\\&=\frac{1-\alpha}{1-\alpha'}-\epsilon' 3^{c+2}-\epsilon/4>1-\epsilon/2-\epsilon/4-\epsilon/4\\&>1-\epsilon\ .\end{aligned}$$

Since $\epsilon$ is arbitrary, $\rho_{\text{plogon}}(S)=1$.

This finishes the proof of Theorem 2.

4.3 Invertible pushdown beats plogon compressors

In this section we take the most restrictive classes of pushdown compressors, namely invertible pushdown automata and visibly pushdown automata, and show that they both outperform plogon compressors.

The proof is based on using a list of Kolmogorov random strings together with their reverses to construct the sequence witnessing the separation. A careful choice of the length of these random strings makes the result incompressible by plogon devices.

**Theorem 3** *For each $\epsilon > 0$ there exists a sequence $S$ such that*

$$\mathrm{P}_{\mathrm{invPD}}(S) \leq 1/2 \quad \textit{and } \rho_{\mathrm{plogon}}(S) \geq 1 - \epsilon.$$

*Proof* Let $\epsilon_1, \epsilon_2 > 0$ and let $k \in \mathbb{N}$ to be determined later (as $k > 4/\epsilon_2$).

We first notice that for each $m \in \mathbb{N}$ there is a string $y \in \Sigma^*$ with $|y| = km$ and such that $y[ik+1..(i+1)k] \neq 1^k$ for every $i$ and $K(y) \geq \frac{k-1}{k}|y|\log|\Sigma|$. This can be proved by a simple counting argument.

Let $t_n = k^{\lceil \frac{\log n}{\log k} \rceil}$, so that

$$n \leq t_n \leq nk. \tag{4}$$

For each $n \in \mathbb{N}$ let $y_n \in \Sigma^{kt_n}$ be as above ($y_n[ik+1..(i+1)k] \neq 1^k$ for every $i$ and $K(y_n) \geq \frac{k-1}{k}|y_n|\log|\Sigma|$).

Consider the sequence $S = y_1 1^k y_1^{-1} y_2 1^k y_2^{-1} \dots y_n 1^k y_n^{-1} \dots$. We will refer to the $1^k$ separators as flags. Consider the following invertible pushdown compressor $(C, D)$. Informally on both $y_j$ and flag zones, $C$ outputs the input. On a $y_j^{-1}$ zone, $C$ outputs a zero for every $1/\epsilon_1$ symbols, and checks using the stack that the input is indeed $y_j^{-1}$. If the test fails, $C$ outputs an error flag, enters an error state, and from then on it outputs the input.

The complete definitions of $C$ and $D$ are given for the sake of completeness. Let $A \geq 1/\epsilon_1$ with $A = k^a$ for some $a \in \mathbb{N}$, i.e. guaranteeing that $A|\,|y_n|$ for almost every $n$. The set of states $Q$ is:

- the start state $q_0^s$
- the counting states $q_1^s, \dots, q_b^s$ and $q_0$, with $b = k\sum_{j=1}^{2^{\lceil a \log k \rceil}}(2t_j + 1)$
- the flag checking states $q_1^{f_1}, \dots, q_k^{f_1}$ and $q_1^{f_0}, \dots, q_k^{f_0}$
- the pop flag states $q_0^r, \dots, q_k^r$
- the compress states $q_1^c, \dots, q_{A+1}^c$
- the error state $q^e$.

We now describe the transition function $\delta : Q \times \Sigma^* \times \Sigma^* \to Q \times \Sigma^*$. At first $C$ counts from $q_0^s$ to $q_b^s$. This guarantees that for later $y_j$, $A|\,|y_j|$. For $0 \leq i \leq b-1$ let

$$\delta(q_i^s, x, y) = (q_{i+1}^s, y)$$

and

$$\delta(q_b^s, \lambda, y) = (q_0, y).$$

After counting has taken place, a new $y$ zone starts; the input is pushed to the stack, and it is checked for the flag, by groups of $k$ symbols.

$$\delta(q_0, x, y) = \begin{cases} (q_1^{f_1}, xy) & \text{if } x = 1 \\ (q_1^{f_0}, xy) & \text{if } x \neq 1 \end{cases}$$

and for $1 \leq i \leq k-1$

$$\delta(q_i^{f_0}, x, y) = (q_{i+1}^{f_0}, xy)$$

$$\delta(q_i^{f_1}, x, y) = \begin{cases} (q_{i+1}^{f_1}, xy) & \text{if } x = 1 \\ (q_{i+1}^{f_0}, xy) & \text{if } x \neq 1 \end{cases}$$

If the flag has not been detected after $k$ symbols, the test starts again.

$$\delta(q_k^{f_0}, \lambda, y) = (q_0, y).$$

If the flag has been detected the pop flag state is entered

$$\delta(q_k^{f_1}, \lambda, y) = (q_0^r, y).$$

Since the flag has been pushed to the stack it has to be removed, thus for $0 \leq i \leq k-1$

$$\delta(q_i^r, \lambda, y) = (q_{i+1}^r, \lambda)$$

$$\delta(q_k^r, \lambda, y) = (q_1^c, y).$$

$C$ then checks using the stack that the input is indeed $y_j^{-1}$, counting modulo $A$. If the test fails, an error state is entered, thus for $1 \leq i \leq A$

$$\delta(q_i^c, x, y) = \begin{cases} (q_{i+1}^c, \lambda) & \text{if } x = y \\ (q^e, y) & \text{if } x \neq y \text{ and } y \neq z_0 \\ (q_1^{f_1}, xz_0) & \text{if } x = 1,\ y = z_0 \\ (q_1^{f_0}, xz_0) & \text{if } \neq 1,\ y = z_0 \end{cases}$$

Once $A$ symbols have been checked, the test starts again

$$\delta(q_{A+1}^c, \lambda, y) = (q_1^c, y).$$

The error state is a loop, $\delta(q^e, x, y) = (q^e, y)$.

We next describe the output function $\nu : Q \times \Sigma^* \times \Sigma^* \to \Sigma^*$. First on the counting states, the input is output, i.e., for $0 \leq i \leq b-1$

$$\nu(q_i^s, x, y) = x.$$

On the flag states the input is output, thus for $1 \leq i \leq k-1$, $a \in \{0, 1\}$

$$\nu(q_i^{f_a}, x, y) = x.$$

There is no output on popping states $q_0^r, \ldots, q_k^r$ and on compressing states $q_1^c, \ldots, q_{A+1}^c$ except after $A$ symbols have been checked i.e.

$$\nu(q_A^c, x, y) = 0 \text{ if } x = y$$

On error, $1^i 0x$ is output, i.e. for $1 \leq i \leq A$

$$\nu(q_i^c, x, y) = 1^i 0x \text{ if } x \neq y \text{ and } y \neq z_0.$$

On the error state, the input is output, that is, $\nu(q^e, x, y) = x$.

Let us verify $C$ is IL, that is, the input can be recovered from the output and the final state. If the final state is not an error state, then both all $y_j$'s and all flags are output as in the input. If the final state is $q_i^c$ then the number $t$ of zeroes after the last

flag (in the output), together with the final state $q_i^c$ determines that the last $y_j^{-1}$ zone is $tA + i - 1$ symbols long.

If the final state is an error state, then the output is of the form (suppose the error happened in the $y_j^{-1}$ zone)

$$ay_j 1^k 0^t 1^i 0 b$$

with $a, b \in \Sigma^*$. The input is uniquely determined to be the input corresponding to output $ay_j 1^k 0^t$ with final state $q_1^c$ followed by

$$y_j^{-1}[tA + 1..tA + i - 1]b.$$

We give the definition of the inverter $D$. The set of states $Q'$ is:

- the start state $q_0^s$
- the counting states $q_1^s, \ldots, q_b^s, q_0$, with $b = k \sum_{j=1}^{2^{\lceil a \log k \rceil}} (2t_j + 1)$
- the flag checking states $q_1^{f_1}, \ldots, q_k^{f_1}$ and $q_1^{f_0}, \ldots, q_k^{f_0}$
- the pop flag states $q_0^r, \ldots, q_k^r$
- the decompress states $q_u^d$ for $u \in \Sigma^{\leq A}$
- the copy states $q_u^w$ for $u \in \Sigma^{\leq A}$
- the output state $q^o$

$D$ receives as input a string followed by a state $q_f \in Q$. Let us describe the transition function $\delta' : Q' \times \Sigma^* \times \Sigma^* \to Q' \times \Sigma^*$ and the output function $\nu' : Q' \times \Sigma^* \times \Sigma^* \to \Sigma^*$ in parallel. At first $D$ counts from $q_0^s$ to $q_b^s$, i.e., for $0 \leq i \leq b - 1$ let

$$\delta'(q_i^s, x, y) = (q_{i+1}^s, y)$$

and

$$\delta'(q_b^s, \lambda, y) = (q_0, y).$$

On the counting states, the input is output, i.e., for $0 \leq i \leq b - 1$

$$\nu'(q_i^s, x, y) = x.$$

At first the input is pushed to the stack, and it is checked for the flag, by groups of $k$ symbols.

$$\delta'(q_0, x, y) = \begin{cases} (q_1^{f_1}, xy) & \text{if } x = 1 \\ (q_1^{f_0}, xy) & \text{if } x \neq 1 \end{cases}$$

and for $1 \leq i \leq k - 1$

$$\delta'(q_i^{f_0}, x, y) = (q_{i+1}^{f_0}, xy)$$

$$\delta'(q_i^{f_1}, x, y) = \begin{cases} (q_{i+1}^{f_1}, xy) & \text{if } x = 1 \\ (q_{i+1}^{f_0}, xy) & \text{if } \neq 1 \end{cases}$$

If the flag has not been detected after $k$ symbols, the test starts again.

$$\delta'(q_k^{f_0}, \lambda, y) = (q_0, y).$$

If the flag has been detected the pop flag state is entered

$$\delta'(q_k^{f_1}, \lambda, y) = (q_0^r, y).$$

Since the flag has been pushed to the stack it has to be removed, thus for $0 \leq i \leq k-1$

$$\delta'(q_i^r, \lambda, y) = (q_{i+1}^r, \lambda)$$

$$\delta'(q_k^r, \lambda, y) = (q_\lambda^d, y)$$

On the flag states the input is output, i.e. for $1 \leq i \leq k-1$, $a \in \{0,1\}$

$$\nu'(q_i^{f_a}, x, y) = x,$$

$$\nu'(q_0, x, y) = x.$$

There is no output on popping states $q_0^r, \ldots, q_k^r$.

The decompressing states pop and memorize $A$ symbols of the stack

$$\delta'(q_u^d, \lambda, y) = (q_{uy}^d, \lambda) \text{ for } |u| < A.$$

If $|u| = A$ then, depending on the next symbol, $u^{-1}$ should be output

$$\delta'(q_u^d, 0, y) = (q_\lambda^d, y) \text{ if } y \neq z_0.$$

$$\delta'(q_u^d, 0, z_0) = (q_0, z_0).$$

$$\nu'(q_u^d, 0, y) = u^{-1}.$$

If 1 is found then there is an error

$$\delta'(q_u^d, 1, y) = (q_u^w, y).$$

$$\delta'(q_{bu}^w, 1, y) = (q_u^w, y).$$

$$\nu'(q_{bu}^w, 1, y) = b.$$

$$\delta'(q_{bu}^w, 0, y) = (q^o, y).$$

If the next symbol is a state then the $y^{-1}$ zone was not complete

$$\nu'(q_u^d, q_i^c, y) = u^{-1}[1..i-1].$$

Once the error has been passed, $D$ stays in the output state. $\delta'(q^o, x, y) = (q^o, y)$, $\nu'(q^o, x, y) = x$.

This ends the description of $(C, D)$.

Let us compute the compression ratio of $C$. For $n$ large enough and since the counting part on the first $b$ symbols of $S$ is of constant size, it is negligible for computing the compression ratio, therefore we can assume wlog that $C$ starts compressing immediately, i.e. $b = 0$; moreover the ratio is largest just after a flag $1^k$ whence

$$\begin{aligned}
\frac{|C(y_1 1^k y_1^{-1} y_2 1^k y_2^{-1} \ldots y_n 1^k)|}{|y_1 1^k y_1^{-1} y_2 1^k y_2^{-1} \ldots y_n 1^k|} &\leq \frac{k(1+\epsilon_1)\sum_{j=1}^n t_j + nk - \epsilon_1 k t_n}{2k\sum_{j=1}^n t_j + nk - kt_n} \\
&\leq \frac{1+\epsilon_1}{2} + \frac{n/2}{\sum_{j=1}^{n-1} t_j} + \frac{t_n/2}{\sum_{j=1}^{n-1} t_j} \\
&\leq 1/2 + \epsilon_1/2 + \frac{n}{n(n-1)} + \frac{nk}{n(n-1)} \\
&< 1/2 + \epsilon_1/2 + \epsilon_1/4 + \epsilon_1/4 = 1/2 + \epsilon_1
\end{aligned}$$

for $n$ sufficiently large. Since $\epsilon_1$ is arbitrary

$$\mathrm{P}_{\mathrm{invPD}}(S) \leq 1/2.$$

We now compute the compression ratio of a plogon compressor on $S$. Let $m \in \mathbb{N}$ and let $n \in \mathbb{N}$ be such that

$$S[1 \dots m] = y_1 1^k y_1^{-1} y_2 1^k y_2^{-1} \dots (y_n 1^k y_n^{-1})[1 \dots i]$$

with $1 \leq i \leq k(1 + 2t_n)$. Let $C$ be an ILplog, running in space $\log^a m$. Let $\epsilon' = \epsilon_2/8k$. Applying Lemma 5 with $d = 3$ and $r$ ranging $\epsilon' n \leq r \leq n$ (such that $r \leq m \leq r^3$ for $n$ sufficiently large), we have that for every $j \in \{\epsilon' n, \dots, n\}$

$$|C(s, y_j^\delta, m)| \geq T|y_j| - \log^{2a}(|y_j|)$$

where $\delta = \pm 1$. Letting $s_j$ (resp. $s'_j$) ($j \in \{\epsilon' n, \dots, n\}$) denote the configuration of $C$ reached on input $S[1 \dots m]$ just before reading the first symbol of $y_j$ (resp. $y_j^{-1}$), we have

$$\begin{aligned}
|C(S[1 \dots m])| &\geq \sum_{j=\epsilon' n}^{n-1} |C(s_j, y_j, m)| + \sum_{j=\epsilon' n}^{n-1} |C(s'_j, y_j^{-1}, m)| \\
&\geq 2 \sum_{j=\epsilon' n}^{n-1} (T|y_j| - \log^{2a} |y_j|) \\
&> 2 \sum_{j=\epsilon' n}^{n-1} (T|y_j| - \gamma|y_j|) \\
&= 2(T - \gamma) \sum_{j=\epsilon' n}^{n-1} |y_j|
\end{aligned}$$

with $\gamma > 0$ arbitrary close to 0, for $n$ large enough. Choosing $\gamma$ and $T = \frac{k-1}{k}$ such that $T - \gamma > 1 - \epsilon_2/4$ (taking $k > 4/\epsilon_2$) yields

$$\begin{aligned}
\frac{|C(S[1 \dots m])|}{|S[1 \dots m]|} &\geq \frac{2(T - \gamma) \sum_{j=\epsilon' n}^{n-1} k t_j}{nk + 2\sum_{j=1}^{n} k t_j} \\
&\geq (T - \gamma) - (T - \gamma)[\frac{n/2}{\sum_{j=1}^{n} t_j} + \frac{t_n}{\sum_{j=1}^{n} t_j} + \frac{\sum_{j=1}^{\epsilon' n - 1} t_j}{\sum_{j=1}^{n} t_j}] \\
&\geq (T - \gamma) - (T - \gamma)[\frac{n}{n(n-1)} + \frac{2kn}{n(n-1)} + \frac{k\epsilon' n(\epsilon' n - 1)}{n(n-1)}] \\
&\geq 1 - \epsilon_2/4 - \epsilon_2/4 - \epsilon_2/4 - \epsilon_2/4 \\
&> 1 - \epsilon_2
\end{aligned}$$

for $n$ sufficiently large, and

$$\rho_{\mathrm{plogon}}(S) \geq 1 - \epsilon_2.$$

Even visibly pushdown automata, extensively used in the compression of XML, can beat plogon compressors. The definition of visibly pushdown automata can be found in section 3.1.

**Theorem 4** *There exists a sequence $S$ such that*

$$\mathrm{P}_{\mathrm{visiblyPD}}(S) \leq 1/2 \quad \textit{and } \rho_{\mathrm{plogon}}(S) \geq 1 - \frac{1}{\log|\Sigma|}.$$

*Proof* The proof is a variation of the proof of Theorem 3. If the alphabet $\Sigma$ has $2t$ symbols, this time the sequence used is $S = y_1 Y_1^{-1} y_2 Y_2^{-1} \dots y_n Y_n^{-1} \dots$, where $y_i$ are Kolmogorov random strings over the first $t$ symbols of the alphabet, and $Y_i$ is the string obtained from $y_i$ by changing each symbol $a$ by symbol $a+t$, that is, $Y_i$ contains only the last $t$ symbols of the alphabet.

### 4.4 Lempel-Ziv is not universal for Pushdown compressors

It is well known that LZ [18] yields a lower bound on the finite-state compression of a sequence [18], i.e., LZ is universal for finite-state compressors.

The following result shows that this is not true for pushdown compression, in a strong sense: we construct a sequence $S$ that is infinitely often incompressible by LZ, but that has almost everywhere pushdown compression ratio less than $\frac{1}{2}$.

**Theorem 5** *For every $\epsilon > 0$, there is a sequence $S$ such that*

$$\mathrm{P}_{\mathrm{invPD}}(S) \leq \frac{1}{2}$$

*and*

$$\rho_{LZ}(S) > 1 - \epsilon.$$

*Proof* Let $\epsilon > 0$, and let $k = k(\epsilon), v = v(\epsilon), v' = v'(\epsilon)$ be integers to be determined later. For any integer $n$, let $T_n$ denote the set of strings $x$ of size $n$ such that $1^j$ does not appear in $x$, for every $j \geq k$. Since $T_n$ contains $\Sigma^{k-1} \times \{0\} \times \Sigma^{k-1} \times \{0\} \dots$ (i.e. the set of strings whose every $k$th symbol is zero), it follows that $|T_n| \geq |\Sigma|^{an}$, where $a = 1 - 1/k$.

*Remark 1* For every string $x \in T_n$ there is a string $y \in T_{n-1}$ and a symbol $b$ such that $yb = x$.

Let $A_n = \{a_1, \dots a_u\}$ be the set of palindromes in $T_n$. Since fixing the $n/2$ first symbols of a palindrome (wlog $n$ is even) completely determines it, it follows that $|A_n| \leq |\Sigma|^{\frac{n}{2}}$. Let us separate the remaining strings in $T_n - A_n$ into $v$ pairs of sets $X_{n,i} = \{x_{i,1}, \dots x_{i,t}\}$ and $Y_{n,i} = \{y_{i,1}, \dots y_{i,t}\}$ with $t = \frac{|T_n - A_n|}{2v}$, $(x_{i,j})^{-1} = y_{i,j}$ for every $1 \leq j \leq t$ and $1 \leq i \leq v$, $x_{i,1}, y_{i,t}$ start with a zero. For convenience we write $X_i$ for $X_{n,i}$.

We construct $S$ in stages. Let $f(k) = 2k$ and $f(n+1) = f(n) + v + 1$. Clearly

$$n^2 > f(n) > n.$$

For $n \leq k-1$, $S_n$ is an enumeration of all strings of size $n$ in lexicographical order. For $n \geq k$,

$$S_n = a_1 \dots a_u \; 1^{f(n)} \; x_{1,1} \dots x_{1,t} \; 1^{f(n)+1} \; y_{1,t} \dots y_{1,1} \; x_{2,1} \dots x_{2,t} \; 1^{f(n)+2} \; y_{2,t} \dots y_{2,1} \dots$$
$$\dots x_{v,1} \dots x_{v,t} 1^{f(n)+v} y_{v,t} \dots y_{v,1}$$

i.e. a concatenation of all strings in $A_n$ (the $A$ zone of $S_n$) followed by a flag of $f(n)$ ones, followed by the concatenations of all strings in the $X_i$ zones and $Y_i$ zones, separated by flags of increasing length. Note that the $Y_i$ zone is exactly the $X_i$ zone written in reverse order. Let

$$S = S_1 S_2 \dots S_{k-1}\ 1^k\ 1^{k+1}\ \dots 1^{2k-1}\ S_k S_{k+1} \dots$$

i.e. the concatenation of the $S_j$'s with some extra flags between $S_{k-1}$ and $S_k$. We claim that the parsing of $S_n$ $(n \geq k)$ by LZ, is as follows:
$a_1, \dots, a_u,\ 1^{f(n)},\ x_{1,1}, \dots, x_{1,t},\ 1^{f(n)+1},\ y_{1,t}, \dots, y_{1,1}, \dots, x_{v,1}, \dots, x_{v,t}, 1^{f(n)+v}, y_{v,t}, \dots, y_{v,1}$. Indeed after $S_1, \dots S_{k-1}\ 1^k\ 1^{k+1}\ \dots 1^{2k-1}$, LZ has parsed every string of size $\leq k-1$ and the flags $1^k\ 1^{k+1}\ \dots 1^{2k-1}$. Together with Remark 1, this guarantees that LZ parses $S_n$ into phrases that are exactly all the strings in $T_n$ and the $v+1$ flags $1^{f(n)}, \dots, 1^{f(n)+v}$.

Let us compute the compression ratio $\rho_{LZ}(S)$. Let $n, i$ be integers. By construction of $S$, LZ encodes every phrase in $S_i$ (except flags), by a phrase in $S_{i-1}$ plus one symbol. Indexing a phrase in $S_{i-1}$ requires a codeword of length at least logarithmic in the number of phrases parsed before, i.e. $\log(P(S_1 S_2 \dots S_{i-2}))$. Since $P(S_i) \geq |T_i| \geq |\Sigma|^{ai}$, it follows that for almost every $i$

$$P(S_1 \dots S_{i-2}) \geq \sum_{j=1}^{i-2} |\Sigma|^{aj} = \frac{|\Sigma|^{a(i-1)} - |\Sigma|^a}{|\Sigma|^a - 1} \geq b|\Sigma|^{a(i-1)}$$

where the inequality holds because $a < 1$ (hence the denominator is less than 1). Letting $t_i = |T_i|$, the number of symbols output by LZ on $S_i$ is at least

$$\begin{aligned} P(S_i) \log P(S_1 \dots S_{i-2}) &\geq t_i \log b|\Sigma|^{a(i-1)} \\ &\geq c t_i (i-1) \end{aligned}$$

where $c = c(a)$ can be made arbitrarily close to 1, by choosing $a$ accordingly. Therefore

$$|LZ(S_1 \dots S_n)| \geq \sum_{j=1}^{n} c t_j (j-1)$$

Since

$$|S_1 \dots S_n| = |S_1 \dots S_{k-1} 1 \dots 1| + |S_k \dots S_n| \leq |\Sigma|^{3k} + \sum_{j=k}^{n} (j t_j + (v+1)(f(j)+v))$$

and $|LZ(S_1 \dots S_n)| \geq \sum_{j=k}^{n} c t_j (j-1)$, the compression ratio is given by

$$\begin{aligned} \rho_{LZ}(S_1 \dots S_n) &\geq c \frac{\sum_{j=k}^{n} t_j (j-1)}{|\Sigma|^{3k} + \sum_{j=k}^{n} (j t_j + (v+1)(f(j)+v))} \\ &= c - c \frac{|\Sigma|^{3k} + \sum_{j=k}^{n} (j t_j + (v+1)(f(j)+v) - t_j (j-1))}{|\Sigma|^{3k} + \sum_{j=k}^{n} (j t_j + (v+1)(f(j)+v))} \\ &= c - c \frac{|\Sigma|^{3k} + \sum_{j=k}^{n} (t_j + (v+1)(f(j)+v))}{|\Sigma|^{3k} + \sum_{j=k}^{n} (j t_j + (v+1)(f(j)+v))} \end{aligned}$$

The second term in this equation can be made arbitrarily small for $n$ large enough: Let $k < M \leq n/3$, we have

$$\begin{aligned}\sum_{j=k}^{n} jt_j &\geq \sum_{j=k}^{M} jt_j + (M+1)\sum_{j=M+1}^{n} t_j \\ &= \sum_{j=k}^{M} jt_j + M\sum_{j=M+1}^{n} t_j + \sum_{j=M+1}^{n} t_j \\ &\geq \sum_{j=k}^{M} jt_j + M\sum_{j=M+1}^{n} t_j + \sum_{j=M+1}^{n} |\Sigma|^{aj} \\ &\geq \sum_{j=k}^{M} jt_j + M\sum_{j=M+1}^{n} t_j + |\Sigma|^{an}\end{aligned}$$

We have

$$|\Sigma|^{an} \geq M[|\Sigma|^{3k} + \sum_{j=k}^{M} t_j + (v+1)\sum_{j=k}^{n}(f(j)+v)]$$

for $n$ large enough, because $f(j) < j^2$. Hence

$$\begin{aligned}&c\frac{|\Sigma|^{3k} + \sum_{j=k}^{n}(t_j + (v+1)(f(j)+v))}{|\Sigma|^{3k} + \sum_{j=k}^{n}(jt_j + (v+1)(f(j)+v))} \\ \leq &c\frac{|\Sigma|^{3k} + \sum_{j=k}^{n}(t_j + (v+1)f(j)+v)}{M[|\Sigma|^{3k} + \sum_{j=k}^{n}(t_j + (v+1)(f(j)+v))]} = \frac{c}{M}\end{aligned}$$

i.e.

$$\rho_{LZ}(S_1 \dots S_n) \geq c - \frac{c}{M}$$

which by definition of $c, M$ can be made arbitrarily close to 1 by choosing $k$ accordingly, i.e

$$\rho_{LZ}(S_1 \dots S_n) \geq 1 - \epsilon.$$

Let us show that $\mathrm{P}_{PD}(S) \leq \frac{1}{2}$. Consider the following ILPD compressor $C$. First $C$ outputs its input until it reaches zone $S_k$. Then on any of the zones $A, X_i$ and the flags, $C$ outputs them symbol by symbol; on $Y_i$ zones, $C$ outputs one zero for every $v'$ symbols of input. To recognize a flag: as soon as $C$ has read $k$ ones, it knows it has reached a flag. For the stack: $C$ on $S_n$ cruises through the $A$ zone up to the first flag, then starts pushing the whole $X_1$ zone onto its stack until it hits the second flag. On $Y_1$, $C$ outputs a 0 for every $v'$ symbols of input, pops one symbol from the stack for every symbol of input, and cruises through $v'$ counting states, until the stack is empty (i.e. $X_2$ starts). $C$ keeps doing the same for each pair $X_i, Y_i$ for every $2 \leq i \leq v$. Therefore at any time, the number of symbols of $Y_i$ read so far is equal to $v'$ times the number of symbols output on the $Y_i$ zone plus the index of the current counting state. On the $Y_i$ zones, $C$ checks that every symbol of $Y_i$ is equal to the symbol it pops from the stack; if the test fails, $C$ enters an error state, outputs an error flag and thereafter outputs every symbol it reads (this guarantees IL on sequences different from $S$). This together with the fact that the $Y_i$ zone is exactly the $X_i$ zone written in reverse order,

guarantees that $C$ is IL. Before giving a detailed construction of $C$, we compute the upper bound it yields on $\mathrm{P}_{PD}(S)$.

*Remark 2* For any $j \in \mathbb{N}$, let $p_j = C(S[1 \dots j])$ be the output of $C$ after reading $j$ symbols of $S$. Is it easy to see that the ratio $\frac{|p_j|}{|S[1 \dots j]|}$ is maximal at the end of a flag following an $X_i$ zone, since the flag is followed by a $Y_i$ zone, on which $C$ outputs one symbol for every $v'$ input symbols.

Let $0 \leq I < v$. We compute the ratio $\frac{|p_j|}{|S[1 \dots j]|}$ inside zone $S_n$ on the last symbol of the flag following $X_{I+1}$. At this location (denoted $j_0$), $C$ has output

$$\begin{aligned}
|p_{j_0}| &\leq |\Sigma|^{3k} + \sum_{j=k}^{n-1} [j|A_j| + (v+1)(f(j)+v) + \frac{j}{2}|T_j - A_j|(1+\frac{1}{v'})] \\
&\quad + n|A_n| + (v+1)(f(n)+v) + \frac{n}{2v}|T_n - A_n|(I+1+\frac{I}{v'}) \\
&\leq |\Sigma|^{pn} + \sum_{j=k}^{n-1} [\frac{j}{2}|T_j|(1+\frac{1}{v'})] + \frac{n}{2v}|T_n|(I+1+\frac{I}{v'})
\end{aligned}$$

where $p > \frac{1}{2}$ can be made arbitrarily close to $\frac{1}{2}$ for $n$ large enough.

The number of symbols of $S$ at this point is

$$\begin{aligned}
|S[1 \dots j_0]| &\geq \sum_{j=k}^{n-1} j|T_j| + n|A_n| + \frac{n}{v}|T_n - A_n|(I+\frac{1}{2}) \\
&\geq \sum_{j=k}^{n-1} j|T_j| + \frac{n}{v}|T_n|(I+\frac{1}{4})
\end{aligned}$$

Hence by Remark 2

$$\begin{aligned}
&\limsup_{n\to\infty} \frac{|p_n|}{|S[1 \dots n]|} \leq \limsup_{n\to\infty} \frac{|\Sigma|^{pn} + \sum_{j=k}^{n-1} [\frac{j}{2}|T_j|(1+\frac{1}{v'})] + \frac{n}{2v}|T_n|(I+1+\frac{I}{v'})}{\sum_{j=k}^{n-1} j|T_j| + \frac{n}{v}|T_n|(I+\frac{1}{4})} \\
&= \limsup_{n\to\infty} [\frac{|\Sigma|^{pn}}{\sum_{j=k}^{n-1} j|T_j| + \frac{n}{v}|T_n|(I+\frac{1}{4})} + \frac{1}{2}\frac{\sum_{j=k}^{n-1} j|T_j| + \frac{n}{v}|T_n|(I+\frac{1}{4})}{\sum_{j=k}^{n-1} j|T_j| + \frac{n}{v}|T_n|(I+\frac{1}{4})} \\
&+ \frac{1}{2v'}\frac{\sum_{j=k}^{n-1} j|T_j|}{\sum_{j=k}^{n-1} j|T_j| + \frac{n}{v}|T_n|(I+\frac{1}{4})} + \frac{n|T_n|}{2v}\frac{\frac{I}{v'}+\frac{3}{4}}{\sum_{j=k}^{n-1} j|T_j| + \frac{n}{v}|T_n|(I+\frac{1}{4})}]
\end{aligned}$$

Since $\sum_{j=k}^{n-1} j|T_j| \geq (n-1)|T_{n-1}| \geq (n-1)\frac{|T_n|}{2}$, we have

$$\begin{aligned}
\sum_{j=k}^{n-1} j|T_j| + \frac{n}{v}|T_n|(I+\frac{1}{4}) &\geq \frac{n-1}{2}|T_n| + \frac{n}{v}|T_n|(I+\frac{1}{4}) \\
&= \frac{n|T_n|}{2v}(v - \frac{v}{n} + 2I + \frac{1}{2}).
\end{aligned}$$

Therefore

$$\limsup_{n\to\infty} \frac{|\Sigma|^{pn}}{\sum_{j=k}^{n-1} j|T_j| + \frac{n}{v}|T_n|(I+\frac{1}{4})} \leq \limsup_{n\to\infty} \frac{|\Sigma|^{pn}}{\frac{(n-1)}{2}|T_n|} \leq \limsup_{n\to\infty} \frac{|\Sigma|^{pn}}{|\Sigma|^{an}} = 0$$

and

$$\frac{1}{2v'} \frac{\sum_{j=k}^{n-1} j|T_j|}{\sum_{j=k}^{n-1} j|T_j| + \frac{n}{v}|T_n|(I+\frac{1}{4})} \leq \frac{1}{2v'}$$

which is arbitrarily small by choosing $v'$ accordingly, and

$$\frac{n|T_n|}{2v} \frac{\frac{I}{v'}+\frac{3}{4}}{\sum_{j=k}^{n-1} j|T_j| + \frac{n}{v}|T_n|(I+\frac{1}{4})} \leq \frac{\frac{I}{v'}+\frac{3}{4}}{v - \frac{v}{n} + 2I + 1}$$

which is arbitrarily small by choosing $v$ accordingly. Thus

$$\mathrm{P}_{PD}(S) = \limsup_{n\to\infty} \frac{|p_n|}{|S[1\dots n]|} \leq \frac{1}{2}.$$

For the sake of completeness we give a detailed description of $C$. Let $Q$ be the following set of states:

- The start state $q_0$, and $q_1, \dots q_w$ the "early" states that will count up to

$$w = |S_1 S_2 \dots S_{k-1}\ 1^k\ 1^{k+1}\ \dots 1^{2k-1}|.$$

- $q_0^A, \dots, q_k^A$ the $A$ zone states that cruise through the $A$ zone up to the first flag.
- $q_j^f$ the $j$th flag state, $(j = 1, \dots, v+1)$
- $q_0^{X_j}, \dots, q_k^{X_j}$ the $X_j$ zone states that cruise through the $X_j$ zone, pushing every symbol on the stack, until the $(j+1)$-th flag is met, $(j = 1, \dots, v)$.
- $q_1^{Y_j}, \dots, q_{v'}^{Y_j}$ the $Y_j$ zone states that cruise through the $Y_j$ zone, popping an symbol from the stack (per input symbol) and comparing it to the input symbol, until the stack is empty, $(j = 1, \dots, v)$.
- $q_0^{r,j}, \dots, q_k^{r,j}$ which after the $j$th flag is detected, pop $k$ symbols from the stack that were erroneously pushed while reading the $j$th flag, $(j = 2, \dots, v+1)$.
- $q_e, q_{e'}$ the error states, if one symbol of $Y_i$ is not equal to the content of the stack.

We next describe the transition function $\delta : Q \times \Sigma^* \times \Sigma^* \to Q \times \Sigma^*$. First $\delta$ counts up to $w$ i.e. for $i = 0, \dots w-1$

$$\delta(q_i, x, y) = (q_{i+1}, y) \quad \text{for any } x, y$$

and after reading $w$ symbols, it enters in the first $A$ zone state, i.e. for any $x, y$

$$\delta(q_w, x, y) = (q_0^A, y).$$

Then $\delta$ skips through $A$ until the string $1^k$ is met, i.e. for $i = 0, \dots k-1$ and any $x, y$

$$\delta(q_i^A, x, y) = \begin{cases} (q_{i+1}^A, y) & \text{if } x = 1 \\ (q_0^A, y) & \text{if } x \neq 1 \end{cases}$$

and

$$\delta(q_k^A, x, y) = (q_1^f, y).$$

Once $1^k$ has been seen, $\delta$ knows the first flag has started, so it skips through the flag until a zero is met, i.e. for every $x, y$

$$\delta(q_1^f, x, y) = \begin{cases} (q_1^f, y) & \text{if } x = 1 \\ (q_0^{X_1}, 0y) & \text{if } x = 0 \end{cases}$$

where state $q_0^{X_1}$ means that the first symbol of the $X_1$ zone (a zero symbol) has been read, therefore $\delta$ pushes a zero. In the $X_1$ zone, delta pushes every symbol it sees until it reads a sequence of $k$ ones, i.e up to the start of the second flag, i.e for $i = 0, \dots k-1$ and any $x, y$

$$\delta(q_i^{X_1}, x, y) = \begin{cases} (q_{i+1}^{X_1}, xy) & \text{if } x = 1 \\ (q_0^{X_1}, xy) & \text{if } x \neq 1 \end{cases}$$

and

$$\delta(q_k^{X_1}, x, y) = (q_0^{r,2}, y).$$

At this point, $\delta$ has pushed all the $X_1$ zone on the stack, followed by $k$ ones. The next step is to pop $k$ ones, i.e for $i = 0, \dots k-1$ and any $x, y$

$$\delta(q_i^{r,2}, x, y) = (q_{i+1}^{r,2}, \lambda)$$

and

$$\delta(q_k^{r,2}, x, y) = (q_2^f, y).$$

At this stage, $\delta$ is still in the second flag (the second flag is always bigger than $2k$) therefore it keeps on reading ones until a zero (the first symbol of the $Y$ zone) is met. For any $x, y$

$$\delta(q_2^f, x, y) = \begin{cases} (q_2^f, y) & \text{if } x = 1 \\ (q_1^{Y_1}, \lambda) & \text{if } x = 0. \end{cases}$$

On the last step, $\delta$ has read the first symbol of the $Y_1$ zone, therefore it pops it. At this stage, the stack exactly contains the $X_1$ zone written in reverse order (except the first symbol), $\delta$ thus uses its stack to check that what follows is really the $Y_1$ zone. If it is not the case, it enters $q_e$. While cruising through $Y_1$, $\delta$ counts with period $v'$. Thus for $i = 1, \dots v' - 1$ and any $x, y$

$$\delta(q_i^{Y_1}, x, y) = \begin{cases} (q_{i+1}^{Y_1}, \lambda) & \text{if } x = y \\ (q_e, \lambda) & \text{otherwise} \end{cases}$$

and

$$\delta(q_{v'}^{Y_1}, x, y) = \begin{cases} (q_1^{Y_1}, \lambda) & \text{if } x = y \\ (q_e, \lambda) & \text{otherwise} \end{cases}$$

Once the stack is empty, the $X_2$ zone begins. Thus, for any $x, y$, $1 \leq i \leq v'$

$$\delta(q_i^{Y_1}, x, z_0) = \begin{cases} (q_1^{X_2}, 1z_0) & \text{if } x = 1 \\ (q_0^{X_2}, 0z_0) & \text{if } x = 0. \end{cases}$$

Then for $2 \leq j \leq v$ the states corresponding to the $X_j$ and $Y_j$ zones behave similarly (that is, states $q_i^{X_j}$, $q_i^{r,j+1}$, $q_{j+1}^f$, and $q_i^{Y_j}$).

At the end of $Y_v$, a new $A$ zone starts, thus for any $1 \leq i \leq v'$

$$\delta(q_i^{Y_v}, x, z_0) = \begin{cases} (q_1^A, z_0) & \text{if } x = 1 \\ (q_0^A, z_0) & \text{if } x = 0. \end{cases}$$

Once in the $q_e$ state the compressor outputs a flag then enters state $q_{e'}$, from that point it simply outputs the input, thus

$$\delta(q_e, \lambda, \lambda) = (q_{e'}, \lambda)$$

and

$$\delta(q_{e'}, x, y) = (q_{e'}, y)$$

The output function outputs the input on every state, except on states $q_1^{Y_j}, \ldots, q_{v'}^{Y_j}$ $(j = 1, \ldots, v)$ where for $1 \leq i < v'$

$$\nu(q_i^{Y_j}, b, y) = \lambda$$

and

$$\nu(q_{v'}^{Y_j}, b, y) = 0$$

and $q_e$ where a flag is output i.e.,

$$\nu(q_e, \lambda, \lambda) = 10.$$

Finally, with a similar construction as in the proof of Theorem 3, the inverse of $C$ can be computed by a pushdown compressor, showing that $C$ is invPD.

### 4.5 Plogon beats Lempel-Ziv

Our next result uses a Copeland-Erdös sequence [6,7] on which Lempel-Ziv has maximal compression ratio, whereas with logspace each prefix of the sequence can be completely reconstructed from its length.

**Theorem 6** *There exists a sequence $S$ such that*

$$\mathrm{P}_{\mathrm{plogon}}(S) = 0 \quad \textit{and } \rho_{LZ}(S) = 1.$$

*Proof* Let $S = E(\Sigma^*)$ be the enumeration of strings over $\Sigma$ in the standard lexicographical order. LZ does not compress $S$ at all, for this algorithm it is the worst possible case, i.e.

$$\rho_{LZ}(S) = 1.$$

For any input $w$, with $|w| = n$, let $m \in \mathbb{N}$, $x \in \Sigma^*$ be such that $w = S[1 \ldots m]x$, and $S[1 \ldots m+1] \not\sqsubset w$. Then we define compressor $C$ as $C(w, |w|) = \mathrm{dbin}(m)01x$, where $\mathrm{dbin}(m)$ is $m$ written in binary with every bit doubled (such that the separator 01 can be recognized). $C$ is clearly 1-1. $C$ is plogon, because on input $(w, n)$, $C$ reads the input online to check that $w$ is a prefix of $S$ (i.e. the standard enumeration of strings over $\Sigma$); the biggest string to check has size $\log n$, therefore the check can be done in plogon. As soon as the check fails, $C$ outputs the length (in binary, with every bit doubled) of the prefix of the input that satisfied the check (at most $2 \log n$ bits) followed by 01 and the rest of the input.

The worst case compression ratio for sequence $S$ is given by

$$\mathrm{P}_{\mathrm{plogon}}(S) = \limsup_{n \to \infty} \frac{|C(S[1 \ldots n], n)|}{n} = \limsup_{n \to \infty} \frac{2 \log n}{n} = 0.$$

4.6 Plogon beats Pushdown compressors

The next result shows that plogon compressors outperform our most general family of pushdown compressors on certain sequences.

The proof is an extension of the intuition in Theorem 1, from a few Kolmogorov-random strings a much longer pushdown-incompressible string can be constructed, even if an identifying index for each string is included. The index can then be used by a polylogarithmic compressor to compress optimally the sequence.

**Theorem 7** *There exists a sequence $S$ such that*

$$\mathrm{P}_{\mathrm{plogon}}(S) = 0 \quad \textit{and } \rho_{\mathrm{PD}}(S) = 1.$$

*Proof* Consider the sequence $S = S_1 S_2 \dots$ where $S_n$ is constructed as follows. Let $x = x_1 x_2 \dots x_{n^2}$ ($|x_i| = n$) be a random string with $K(x) \geq n^3 \log |\Sigma|$. Let

$$S_n = x_1 x_2 \dots x_{n^2} i_1 x_{i_1} i_2 x_{i_2} \dots i_{2^n} x_{i_{2^n}}$$

where $i_j \in \{1, \dots n^2\}$ for every $1 \leq j \leq 2^n$ are indexes coded in $2 \log n$ bits, defined later on.

Let $C_1, C_2, \dots$ be an enumeration of all ILBPDCwE such that $C_i$ can be encoded in at most $i$ bits and such that a maximum of $\log^{(2)} i$ $\lambda$-rules can be applied per symbol.

The following claim shows that there are many $C$-incompressible strings $x_i$.

*Claim* Let $F_n = \{C_1, \dots, C_{\log n}\}$. Let $w \in \Sigma^*$.

1. Let $C \in F_n$. There are at least $(1 - \frac{1}{2 \log n}) n^2$ strings $ix_i$ ($1 \leq i \leq n^2$) such that
$$|C(wix_i)| - |C(w)| > n - 2\sqrt{n}.$$
2. There is a string $x_i$ such that for every $C \in F_n$,
$$|C(wix_i)| - |C(w)| > n - 2\sqrt{n}.$$

*Proof* (of Claim 4.6) After having read $w$, $C$ is in state $q$, with stack content $yz$, where $y$ denotes the $(n + 2 \log n) \log^{(2)} n$ topmost symbols of the stack (if the stack is shorter then $y$ is the whole stack). It is clear that while reading an $ix_i$, $C$ will not pop the stack below $y$.

Let $T = (1 - \frac{1}{2 \log n}) n^2$, and let $C(q, yz, ix_i\$)$ denote the output of $C$ when started in state $q$ on input $ix_i\$$ with stack content $yz$. Suppose the claim false, i.e. there exist more than $n^2 - T$ words $ix_i$ such that $C(q, yz, ix_i\$) = p_i$, ends in state $q_i$, and $|p_i| \leq n - 2\sqrt{n} + O(1)$ (notice that the output on symbol \$ is $O(1)$). Denote by $G$ the set of such strings $x_i$. This yields the following short program for $x$ (coded with alphabet $\Sigma$):

$$p = (n, C, q, y, a_1 t_1 a_2 t_2 \dots a_{n^2} t_{n^2})$$

where each comma costs less than $3 \log |s|$, where $s$ is the element between two commas; $a_i = 1$ implies $t_i = x_i$, $a_i = 0$ implies $x_i \in G$ and $t_i = d(q_i)01d(|p_i|)01p_i$ (where $d(z)$ for any string $z$, is the string written with every symbol doubled), i.e. $|t_i| \leq n - \sqrt{n}$. $p$ is a program for $x$: once $n$ is known, each $a_i t_i$ yields either $x_i$ (if $a_i = 1$) or $(p_i, q_i)$ (if $a_i = 0$). From $(p_i, q_i)$, simulating $C(q, yz, u\$)$ for each $u \in \Sigma^{n + 2 \log n}$ yields the unique

$u = ix_i$ such that $C(q, yz, u\$) = p_i$ and ends in state $q_i$. The simulations are possible, because $C$ does not read its stack further than $y$, which is given. We have

$$\begin{aligned}|p| &\leq O(\log n) + (n + 2\log n)\log^{(2)} n + (n+1)T + (n^2 - T)(n - \sqrt{n}) \\ &\leq O(n^2) + n^3 - \frac{n^{2.5}}{2\log n} \\ &\leq n^3 - \frac{n^{2.5}}{4\log n}\end{aligned}$$

which contradicts the randomness of $x$, thus proving part 1.

Let $W_j$ be the set of strings $ix_i$ that are compressible by $C_j$; by 1., $|W_j| \leq n^2/2\log n$. Let $R = \{ix_i\}_{i=1}^{n^2} - \cup_{j=1}^{\log n} W_j$ be the set of strings incompressible by all $C \in F_n$. We have

$$|R| \geq n^2 - \log n \cdot n^2/2\log n = n^2/2 > 1.$$

This proves part 2.

We finish the definition of $S_n$ by picking $i_1 x_{i_1}$ to be the first string fulfilling the second part of Claim 4.6 for $w = S_1 S_2 \dots S_{n-1}$. The construction is similar for all strings $\{x_{i_j}\}_{j=2}^{2^n}$, by taking $w = S_1 S_2 \dots S_{n-1} x_{i_1} \dots x_{i_{j-1}}$, thus ending the construction of $S_n$.

Let us show that $\rho_{\mathrm{PD}}(S) = 1$. Let $\epsilon > 0$. Let $C = C_k$ be an ILBPDCwE; then for almost every $n$, and for all $0 \leq t \leq 2^n$, because $|S_1 \dots S_{n-1}|$ is exponentially larger than the first $n^2$ $x_i$'s of zone $S_n$, it is good enough to compute the compression ratio only after those first $n^2$ $x_i$'s and after each $ix_i$. We have

$$\begin{aligned}&\frac{|C(S_1 \dots S_{n-1} S_n[n^2 + t(n + 2\log n)]\$)|}{|S_1 \dots S_{n-1} S_n[n^2 + t(n + 2\log n)]|} \\ &\geq \frac{\sum_{j=k}^{n-1}(2^j)(j - 2\sqrt{j}) + t(n - 2\sqrt{n})}{\sum_{j=1}^{n-1}(j^2 + 2^j(j + 2\log j)) + n^2 + t(n + 2\log n)} \\ &\geq \frac{(1-\alpha)\sum_{j=1}^{n-1} j2^j + n^2 + tn}{(1+\alpha)\sum_{j=1}^{n-1} j2^j + n^2 + tn} - \frac{2(t+1)\sqrt{n}}{\sum_{j=1}^{n-1} j2^j + n^2 + tn} - \frac{(1-\alpha)\sum_{j=1}^{k} j2^j}{\sum_{j=1}^{n-1} j2^j + n^2 + tn} \\ &\geq 1 - \epsilon/4 - \epsilon/4 - \epsilon/4 \geq 1 - \epsilon\end{aligned}$$

where $\alpha$ can be made arbitrarily small for large enough $n$.

We show that $\mathrm{P}_{\mathrm{plogon}}(S) = 0$. Consider the following plogon compressor $C$, where every output bit is output doubled except commas (coded by 10) and the error flag (coded by 01). First $C$ outputs the length of the input (in binary) followed by a comma. For the $n^2$ first $x_i$'s of zone $S_n$, $C$ outputs them (and stores them). For the remaining $i_j x_{i_j}$'s, only $i_j$ is output, and $C$ checks that what follows $i_j$ is indeed $x_{i_j}$. If at any point in time the test fails, the error mode is entered. In error mode, 01 is output, followed by the rest of the input, starting right after the $i_j$ where the error occurred.

It is easy to check that $C$ is polylog space, since at the beginning of zone $S_n$, the available space is of order $\mathrm{poly}(n)$.

$C$ is IL, because from $C$'s output, we know the length of the input and whether the error mode has been entered or not. If there is no error, all the first $n^2$ $x_i$'s of zone $S_n$ can be recovered, followed by all strings $i_j x_{i_j}$. If the error mode is entered, by the

previous argument the sequence $S_n$ can be reconstructed up to the last $i_j$ before the error. The rest of the output yields the rest of the sequence.

Let us compute the compression ratio. Let $\epsilon > 0$. Let $n \in \mathbb{N}$ and $0 \leq t \leq 2^n$. Because $|S_1 \dots S_{n-1}|$ is exponentially larger than the first $n^2$ $x_i$'s of zone $S_n$, it is good enough to compute the compression ratio only after those first $n^2$ $x_i$'s. We have

$$\begin{aligned}
&\frac{|C(S_1 \dots S_{n-1} S_n[n^3 + t(n + 2\log n)])|}{|S_1 \dots S_{n-1} S_n[n^3 + t(n + 2\log n)]|} \\
&\leq \frac{2[\sum_{j=1}^{n-1} j^3 + 2^j(2\log j) + n^3 + 2t\log n]}{\sum_{j=1}^{n-1} j^3 + 2^j(j + 2\log j) + n^3 + t(n + 2\log n)} \\
&\leq \frac{2[\sum_{j=1}^{n-1} 3 \cdot 2^j \log j + n^3 + 2t\log n]}{\sum_{j=1}^{n-1} j2^j + n^3 + tn} \\
&\leq \frac{6[\sum_{j=1}^{n-1} 2^j \log j]}{\sum_{j=1}^{n-1} j2^j} + \epsilon/4 + \epsilon/4
\end{aligned}$$

Since $\log j < \frac{\epsilon}{24} j$ for all $j > j_0$ we have

$$\begin{aligned}
\frac{|C(S_1 \dots S_{n-1} S_n[n^3 + t(n + 2\log n)])|}{|S_1 \dots S_{n-1} S_n[n^3 + t(n + 2\log n)]|} &\leq \frac{6[\sum_{j=1}^{j_0} 2^j \log j]}{\sum_{j=1}^{n-1} j2^j} + \frac{\epsilon/4[\sum_{j=j_0+1}^{n-1} j2^j]}{\sum_{j=1}^{n-1} j2^j} + \epsilon/2 \\
&\leq \epsilon/4 + \epsilon/4 + \epsilon/2 \leq \epsilon.
\end{aligned}$$

## 5 Final Remark

The increasing use of low-resource compressors designed for massive data compression calls for a theoretical analysis of their performances. We have compared pushdown compression with online polylog-space compression and our proofs show the limitations and the strength of these two models. Furthermore, we have compared them with the general-purpose compression algorithm of Lempel and Ziv: by again showing the weakness of the different compression mechanisms, we proved that the high resources needed by this algorithm do not avoid it to perform more poorly on some instances than our low-resource algorithms.

Let us write a final word on two notions related to the compression ratio: effective dimension and logarithmic loss unpredictability. Effective dimension is an effective version of Hausdorff dimension for infinite sequences $S$: roughly speaking, it can be seen as the most unfair environment in which there is a successful betting strategy on the successive bits of $S$. Logarithmic loss unpredictability is a related notion: intuitively, a sequence $S$ is log-loss predictible if, knowing all the preceding bits of $S$, one can efficiently predict the next one with high probability.

It has been realized that being incompressible, or unpredictible, or having high effective dimension are three facets of the same thing. The equivalence of compression ratio, effective dimension, and log-loss unpredictability has been explored in different settings [8, 13, 20], depending on the compurational resources allowed. It is known that for the cases of finite-state, polynomial-space, recursive, and constructive resource-bounds, natural definitions of compression and dimension coincide, both in the case of

infinitely often compression, related to effective versions of Hausdorff dimension, and that of almost everywhere compression, matched with packing dimension. The general matter of transformation of compressors in predictors and vice versa is widely studied [22].

In this paper we have done a complete comparison of pushdown, plogon compression and LZ-compression. It is straightforward to construct a prediction algorithm based on Lempel-Ziv compressor that uses similar computing resources, and it has been proved in [1] that bounded-pushdown compression and dimension coincide. This leaves us with the natural open question of whether each plogon compressor can be transformed into a plogon prediction algorithm, for which the log-loss unpredictability coincides with the compression ratio of the initial compressor, that is, whether the natural concept of plogon dimension coincides with plogon compressibility. A positive answer would get plogon computation closer to pushdown devices, and a negative one would make it closer to polynomial-time algorithms, for which the answer is likely to be negative [19].

**Acknowledgements** We acknowledge the anonymous reviewers for their comments that enabled to improve the readability of this paper.

## References

1. Albert, P., Mayordomo, E., Moser, P.: Bounded pushdown dimension vs lempel ziv information density. Tech. Rep. TR07-051, ECCC: Electronic Coloquium on Computational Complexity (2007)
2. Albert, P., Mayordomo, E., Moser, P., Perifel, S.: Pushdown compression. In: Proceedings of the 25th Symposium on Theoretical Aspects of Computer Science (STACS 2008), pp. 39–48 (2008)
3. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. Journal of Computer and System Sciences **58**, 137–147 (1999)
4. Alur, R., Madhusudan, P.: Adding nesting structure to words. In: Proceedings of the Tenth International Conference on Developments in Language Theory, *Lecture Notes in Computer Science*, vol. 4036. Springer (2006)
5. Autebert, J., Berstel, J., Boasson, L.: Context-free languages and pushdown automata. In: G. Rozenberg, A. Salomaa (eds.) Handbook of Formal Languages, volume 1, Word, Language, Grammar, pp. 111–174. Springer-Verlag (1997)
6. Champernowne, D.G.: Construction of decimals normal in the scale of ten. J. London Math. Soc. **2**(8), 254–260 (1933)
7. Copeland, A., Erdös, P.: Note on normal numbers. Bulletin of the American Mathematical Society **52**, 857–860 (1946)
8. Dai, J.J., Lathrop, J.I., Lutz, J.H., Mayordomo, E.: Finite-state dimension. Theoretical Computer Science **310**, 1–33 (2004)
9. Ginsburg, S., Rose, G.F.: Preservation of languages by transducers. Information and Control **9**(2), 153–176 (1966)
10. Ginsburg, S., Rose, G.F.: A note on preservation of languages by transducers. Information and Control **12**(5/6), 549–552 (1968)
11. Hariharan, S., Shankar, P.: Evaluating the role of context in syntax directed compression of xml documents. In: Proceedings of the 2006 IEEE Data Compression Conference (DCC 2006), p. 453 (2006)
12. Hartmanis, J., Immerman, N., Mahaney, S.: One-way log-tape reductions. In: Proceedings of the 19th Annual Symposium on Foundations of Computer Science (FOCS'78), pp. 65–72. IEEE Computer Society (1978)
13. Hitchcock, J.M.: Effective fractal dimension: Foundations and applications. Ph.D. thesis, PhD thesis, Iowa State University (2003)

14. Indyk, P., Woodruff, D.: Optimal approximations of the frequency moments of data streams. Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC 2005) pp. 202–208 (2005)
15. Kuma, V., Madhusudan, P., Viswanathan, M.: Visibly pushdown automata for streaming xml. International World Wide Web Conference WWW 2007 pp. 1053–1062 (2007)
16. Lathrop, J.I., Strauss, M.J.: A universal upper bound on the performance of the Lempel-Ziv algorithm on maliciously-constructed data. Compression and Complexity of Sequences '97 pp. 123–135 (1998)
17. League, C., Eng, K.: Type-based compression of xml data. Proceedings of the 2007 IEEE Data Compression Conference (DCC 2007) pp. 272–282 (2007)
18. Lempel, A., Ziv, J.: Compression of individual sequences via variable rate coding. Transaction on Information Theory pp. 530–536 (1978)
19. López-Valdés, M., Mayordomo, E.: Dimension is compression. Proceedings of the 30th International Symposium on Mathematical Foundations of Computer Science pp. 676–685 (2005)
20. Mayordomo., E.: Effective fractal dimension in algorithmic information theory. New Computational Paradigms: Changing Conceptions of What is Computable pp. 259–285 (2008)
21. Mayordomo, E., Moser, P.: polylog space compression is incomparable with Lempel-Ziv and pushdown compression. Proceedings of the 35th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM09) pp. 633–644 (2009)
22. Sculley, D., Brodley, C.E.: Compression and machine learning: A new perspective on feature space vectors. Proceedings of the Data Compression Conference (DCC-2006) pp. 332–341 (2006)