# Dictionary Compression and Repetitiveness Measures

**Hideo Bannai (hdbn.dsc@tmd.ac.jp)**

**M&D Data Science Center, Institute of Integrated Research, Institute of Science Tokyo**

Institute of SCIENCE TOKYO

June 12-13, 2025 CPM Summer School @ Milano

# Data Compression

- A process of obtaining a smaller representation of the data by discovering and exploiting regularities in the data.
- Important for reducing communication, storage, etc.



# Compression of Individual Strings

- Kolmogorov Complexity – ultimate compression
  - The length $C(x)$ of the shortest program (for a fixed UTM) that generates $x$.
  - Amount of information/repetitiveness in a string.
    - Constant difference between choice of (Turing complete) programming language.
  - There is no algorithm that can compress all strings.
    - For any $n$, there exists an length-$n$ string $x_n$ with $C(x_n) \geq n$.
      - For the $2^n$ strings of length $n$, there are only $2^0 + 2^{1} + \cdots 2^{n-1} = 2^n - 1$ shorter descriptions.
  - Not computable.
    - If computable, could enumerate all strings to find for any $n$, the first $x_n$ s.t. $C(x_n) \geq n$.
    - For large enough $n$, $C(x_n)$ would be larger than the program for $x_n$ described above.

Turing machines are perhaps too powerful for modeling redundancies.
➔ Dictionary Compression!

# Dictionary Compression

- A family of representations (or algorithms to compute them) that are based on copy and paste operations.
- Computable.
- Good compression performance for *highly repetitive strings*, i.e., versioned documents, multiple genomes of same species, student reports.
- Many different representations offering various trade-offs between
  - compression time
  - compressed size
  - ease of processing in compressed space
    - access, pattern matching, indexing, etc.

# Contents

- Introduce various families of dictionary compression, related measures, and their relations
  - Run-length Encoding (RLE), Bidirectional Macro Scheme (BMS), Lempel-Ziv (LZ) parsings, grammar based compression (straight line programs SLP), run-length SLP (RLSLP), run-length Burrows-Wheeler transform (RLBWT), string attractors, length-normalized substring complexity.
- Focus on (asymptotic) representation sizes rather than running times of algorithms.
- Selection of results I personally find interesting / important (and biased towards results with (relatively) simple proofs).
  - ~~Hopefully not too technical…~~
    May be a bit technical but ideas are simple
  - Assume basic CS algorithms knowledge. Knowledge of suffix data structures (suffix trees, suffix arrays, LCP arrays) may help in some places.

# Related CPM talks next week

## Thursday June 19th

**09:00 - 10:00 Keynote Talk 3** (Chair: Gianluca Della Vedova)

Inge Li Gørtz.
Locality Sensitive Hashing and Compressed Computation

**10:00 - 10:20: Coffee break**

**10:20 - 12:00 Contributed talks 5 [space optimization]** (Chair: NN)

| | |
|---|---|
| 10:20 - 10:40 | Itai Boneh, Shay Golan.<br>Covers in Optimal Space |
| 10:40 - 11:00 | Hideo Bannai, Tomohiro I, Yuto Nakashima.<br>On the compressiveness of the Burrows-Wheeler transform |
| 11:00 - 11:20 | Takuya Mieno, Shunsuke Inenaga.<br>Space-Efficient Online Computation of String Net Occurrences |
| 11:20 - 11:40 | Jarno N. Alanko, Ruben Becker, Davide Cenzato, Travis Gagie, Sung-Hwan Kim, Bojana Kodric, Nicola Prezza.<br>The Trie Measure, Revisited |
| 11:40 - 12:00 | Yoshifumi Sakai.<br>Linear-space LCS enumeration for two strings |

**12:00 - 13:30: Lunch break**

**13:30 - 14:00: Highlight Talk 2** (Chair: NN)

| | |
|---|---|
| 13:30 - 14:00 | David Koslicki.<br>Prokrustean Graph: A substring index for rapid k-mer size analysis (RECOMB 2025) |

**14:00 - 15:00: Contributed talks 6 [run-length compression]** (Chair: NN)

| | |
|---|---|
| 14:00 - 14:20 | Gonzalo Navarro, Alejandro Pacheco.<br>Counting on General Run-Length Grammars |
| 14:20 - 14:40 | Yuto Iguchi, Ryo Yoshinaka, Ayumi Shinohara.<br>Pattern Matching on Run-Length Grammar-Compressed Strings in Linear Time |
| 14:40 - 15:00 | Philip Bille, Inge Li Gørtz, Simon Puglisi, Simon R. Tarnow.<br>Compressed Dictionary Matching on Run-Length Encoded Strings |

# Run-Length Encoding, Bidirectional Macro Schemes, and Lempel-Ziv parsings

# Run-Length-Encoding (RLE) $\rho$

For any string T, the **run-length-encoding** RLE(T) of T is a sequence of $\rho$(T) phrases s.t. each phrase is a maximal same-symbol run, and each phrase is encoded by its symbol and length.

Simplest (?) form of dictionary compression.

Example

$$\texttt{CCCCCCCCCCCCCCCCCCCCPPMMMMM} \rightarrow \texttt{C}^{20}\texttt{P}^{2}\texttt{M}^{5}$$

May not be very good even when the string is repetitive:

$$\texttt{CPMCPMCPMCPMCPMCPMCPMCPM} \rightarrow \texttt{C}^{1}\texttt{P}^{1}\texttt{M}^{1}\texttt{C}^{1}\texttt{P}^{1}\texttt{M}^{1} \ldots$$

# Bidirectional Macro Schemes (BMS)

[Storer&Szymanski 1982]

Most general form of dictionary compression.

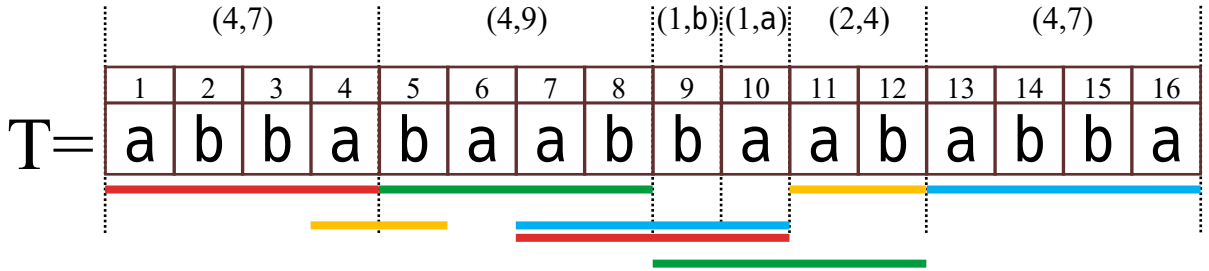
A BMS of a string T is a partitioning of T into phrases s.t.:

- Phrases of length ≧ 2 have a reference occurrence elsewhere in T.
- The referencing does not induce a cycle.

Each phrase is encoded as a pair:

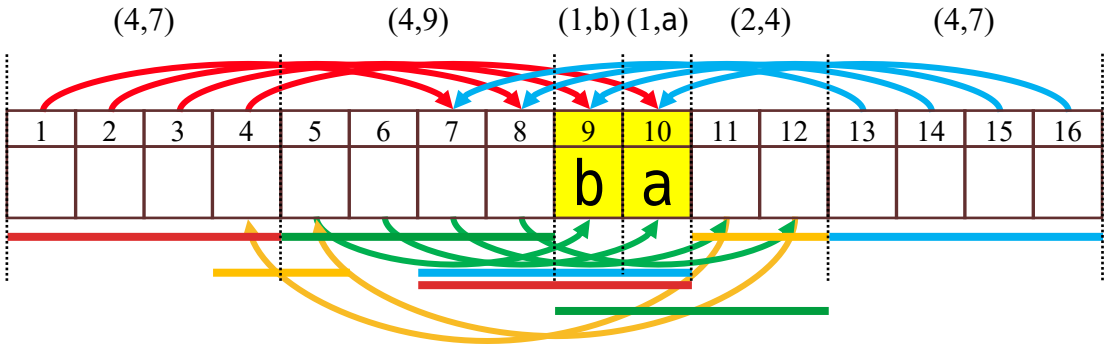
(length ≧ 2, reference position) or (1, symbol)

*b*(T): size of smallest BMS for T. (NP-hard to compute [Gallant 1982])



# Decompressing a BMS

Given a sequence of encoded BMS phrases:

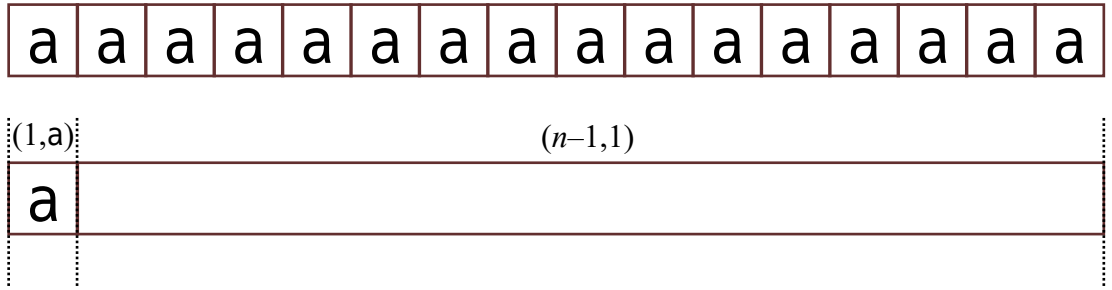



- It is easy to compute the length of the string.
- The reference of each phrase induces a reference for each position.
- The string can be reconstructed from the phrase encoding, iff the induced referencing graph is acyclic
  - For any position, traversing the reference will eventually lead to a ground phrase $(1, c)$ for some $c \in \Sigma$.

# RLE as a BMS and separation between $b$ and $\rho$

For any string T, $b(\mathrm{T}) \leq 2\rho(\mathrm{T})$.

Any run of same symbols can be represented by two BMS phrases.




There exists an infinite family of strings s.t.

$$b(\mathrm{T}) = \mathrm{O}(1) \text{ and } \rho(\mathrm{T}) = \Omega(n).$$

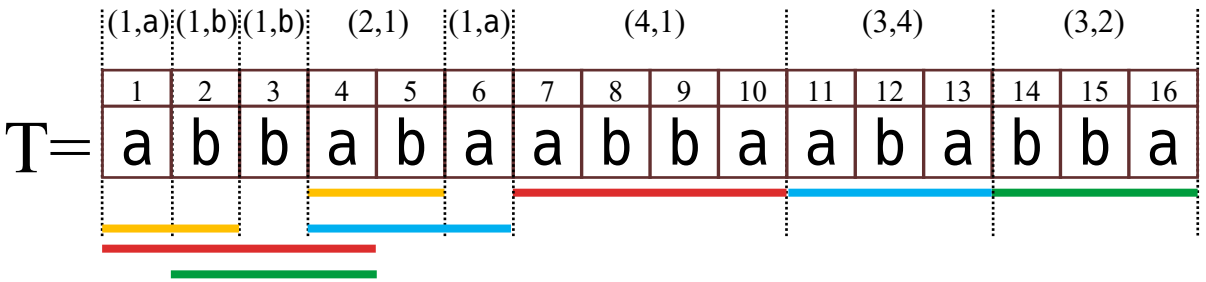
e.g.: $\mathrm{T} = (\mathtt{CPM})^{n/3}$

# Lempel-Ziv (like) parsings $z$

An **LZ-like parsing** is a BMS where references always point to previous positions. $z(T)$: size of smallest LZ-like parsing for T.

For any string T, the size $b(T) \leq z(T)$.

Proof: Straightforward from definition.



# Lempel-Ziv (like) parsings $z$

An **LZ-like parsing** is a BMS where references always point to previous positions. $z(T)$: size of smallest LZ-like parsing for T.

For any string T, the size $b(T) \leq z(T)$.

Proof: Straightforward from definition.

For any string T, the greedy LZ-like parsing (aka LZ77) is of size $z(T)$.

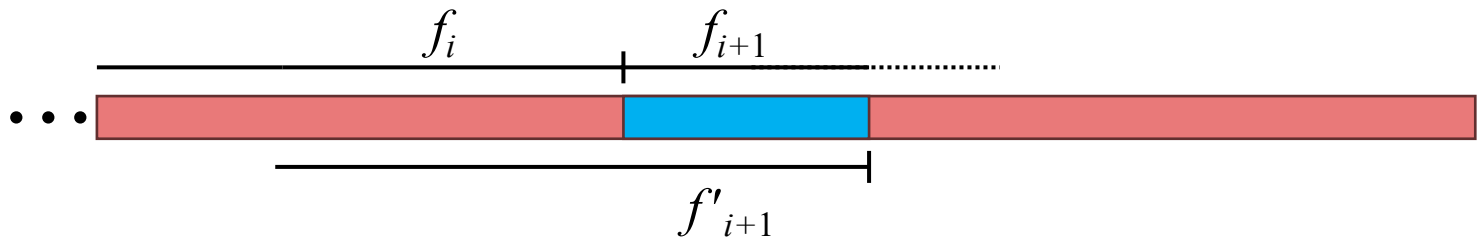
$z$ is one of the smallest efficiently computable compressed representation.
The idea of LZ77 [Ziv&Lempel 76,77] is used in many compressed file formats (png, pdf, zip, xz, 7z, gz, etc.) and web browsers.

# Optimality LZ-like parsing: LZ77

For any string T, the greedy LZ-like parsing (aka LZ77) is of size $z$(T).

Proof

- Let $f_1 \ldots f_z$ be the greedy LZ-parse and $f'_1 \ldots f'_{z'}$ an arbitrary LZ-like parse. Claim: for any $k$, $|f_1 \ldots f_k| \geq |f'_1 \ldots f'_k|$ holds, and thus $z \leq z'$.
  - For $i = 1, f_1 = f'_1 = \mathrm{T}[1]$. Suppose it holds for all $k \leq i$.
  - If $|f_1 \ldots f_i| \geq |f'_1 \ldots f'_{i+1}|$, clearly $|f_1 \ldots f_{i+1}| \geq |f'_1 \ldots f'_{i+1}|$.
  - If $|f_1 \ldots f_i| < |f'_1 \ldots f'_{i+1}|$
    - since the suffix of $f'_{i+1}$ that starts after $f_i$ has a previous occurrence, it is a prefix of $f_{i+1}$ and therefore $|f_1 \ldots f_{i+1}| \geq |f'_1 \ldots f'_{i+1}|$.



# Computing LZ77 efficiently

$z$(T) can be computed in linear time.

There are a wide variety of algorithms

- Suffix-tree/Suffix-array based
  - byproduct of Ukkonen's linear time online construction (O($n$logσ) time)
  - O($n$) time for linearly sortable alphabets using suffix tree or suffix array and LCP/LPF arrays
- More recently, sub-linear time and space:
  - O($n$ log$\sigma$/log$^{1/2}n$) time and O($n$/log$_\sigma n$) space [Kempa&Kociumaka 2024]

# Separation between $b$ and $z$ (1)

There exist an infinite family of strings s.t.

$$b(\mathrm{T}) = \mathrm{O}(1) \text{ and } z(\mathrm{T}) = \Omega(\log n)$$

Proof: Consider Fibonacci words.

$F_0 = \mathtt{b}$

$F_1 = \mathtt{a}$

$F_2 = \mathtt{ab}$

$F_3 = \mathtt{aba}$

$F_4 = \mathtt{abaab}$

…

$F_i = F_{i-1}F_{i-2}$

Can also be defined by a morphism: $\varphi(\mathtt{a}) = \mathtt{ab}$, $\varphi(\mathtt{b}) = \mathtt{a}$, $F_i = \varphi^i(\mathtt{b})$

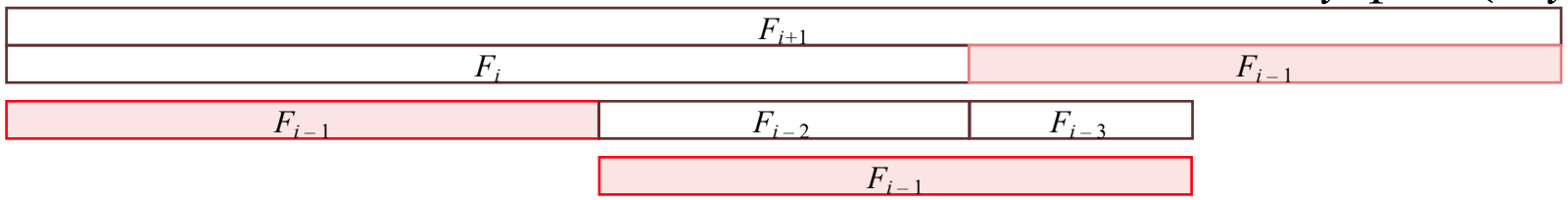
# Separation between $b$ and $z$ (1)

There exist an infinite family of strings s.t.

$$b(\mathrm{T}) = \mathrm{O}(1) \text{ and } z(\mathrm{T}) = \Omega(\log n)$$

Proof: Consider Fibonacci words.

- For $k \geq 4$, $z(F_k) = k$. True for $k = 4$: abaab
- Suppose it is true for $4 \leq k \leq i$ and consider $F_{i+1} = F_i F_{i-1}$
  - However, no two occurrences of $F_{i-1}$ are preceded by the same symbol.
    - $F_{i-1}$ occurs in $F_{i+1}$ exactly 3 times: a prefix square and suffix.
      - any other occurrence implies periodicity ➔ occurrence of $x^4$ for some $x$, which is impossible.
    - The last two symbols of: $F_{2j}$ is ab and $F_{2j+1}$ is ba.
- Therefore, at least one new factor starts in last $F_{i-1}$: $z(F_{i+1}) = i + 1$.



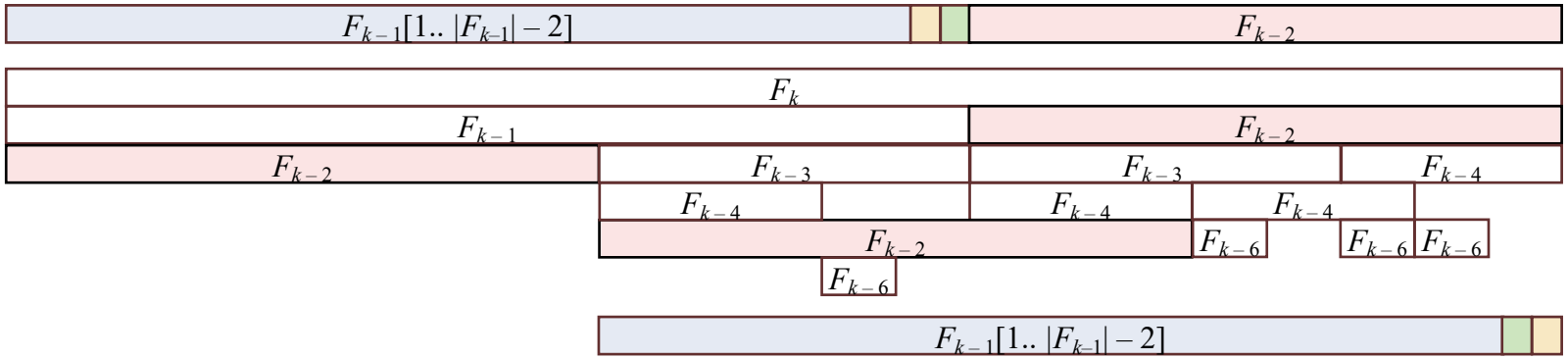
# Separation between $b$ and $z$ (2)

There exist an infinite family of strings s.t.

$$b(\mathrm{T}) = \mathrm{O}(1) \text{ and } z(\mathrm{T}) = \Omega(\log n)$$

Proof: $F_k\,(k \geq 4)$ can be partitioned into 4 BMS phrases.

- $(F_{k-1} - 2,\ F_{k-2}+1)$, (1, b/a), (1, a/b), $(F_{k-2},\ 1)$ or



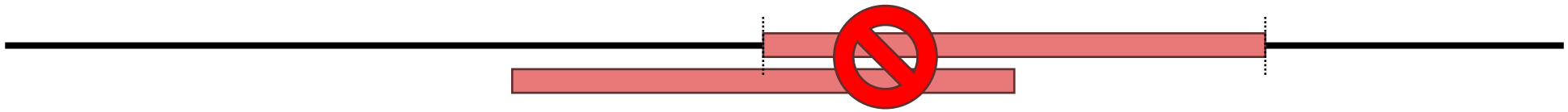

- $F_{k-1} = F_{k-2}\,F_{k-3} = F_{k-2}\,F_{k-4}\,F_{k-5} = F_{k-2}\,F_{k-4}\,F_{k-6}\,F_{k-7}\,\ldots =$
- Referencing is acyclic – all references are unidirectional in lex order of rotation.



# Other Lempel-Ziv (like) parsings $z$

Variants:

- LZ77 $z$
- LZ77 without self-referencing $z_{no}$ reference cannot overlap with phrase



For unary strings: $z(\mathrm{T}) = \mathrm{O}(1)$, $z_{no}(\mathrm{T}) = \Theta(\log n)$

- Greedy LZEnd [Kreft&Navarro 2010] $z_e$ reference must end at a phrase end
- Optimal LZEnd $z_{end}$
- LZD [Goto et al. 2015] $z_d$ phrase is pair of previous phrases
- LZ78 $z_{78}$ phrase is previous phrase + 1 symbol

$z \leq z_{no} \leq z_e \leq z_{end} \leq z_d \leq z_{78}$

Institute of SCIENCE TOKYO

# Grammar-based Compression

# Expressions with more structure

# Grammars / SLP *g*

A **Straight-Line Program (SLP)** is a context free grammar in Chomsky normal form, deriving a single string.

$X \rightarrow YZ$ or $X \rightarrow c \in \Sigma$ Size of SLP = # of rules

*g*(T): size of smallest SLP for T

Example

$X_7 \rightarrow X_6X_5$

$X_6 \rightarrow X_4X_5$

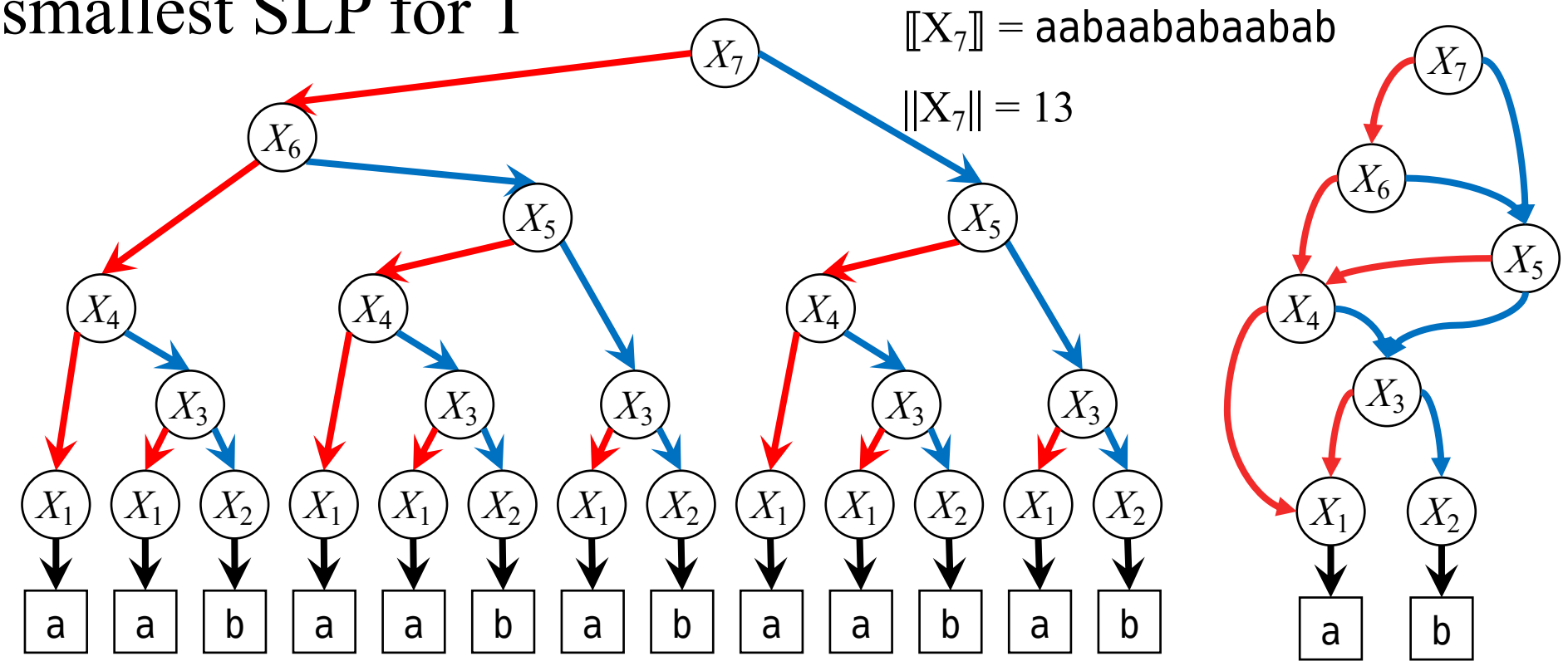
$X_5 \rightarrow X_4X_3$

$X_4 \rightarrow X_1X_3$

$X_3 \rightarrow X_1X_2$

$X_2 \rightarrow$ b

$X_1 \rightarrow$ a

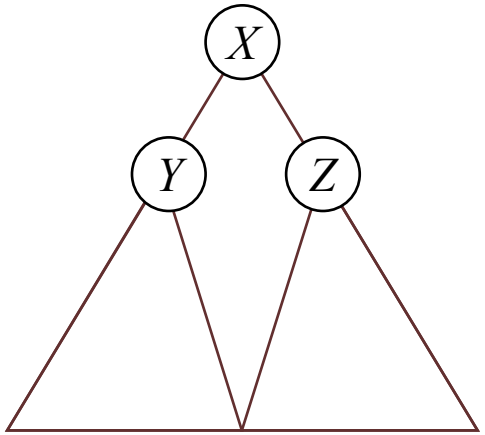


# Access on SLPs

Given SLP $S$ of size $x$ for a string T of length $n$, there exists an $O(x)$ size data structure that can be computed in $O(|x|)$ time, and can answer access queries (return T[$i$] for query $i$) in $O(height(S))$ time.

Proof

- For each rule $X \rightarrow YZ$, compute length $||X||$ of string that $X$ derives.
  - Simple bottom-up DP in topological order $||X|| = ||Y||+||Z||$
- Access to position $i$ in $[\![X]\!]$ :
  - If $i \leq ||Y||$, position $i$ in $[\![Y]\!]$
  - Otherwise, position $i - ||Y||$ in $[\![Z]\!]$



# SLP induces an LZ-like parsing

Partial parse tree: Only expand leftmost occurrence of each variable.

Leaves of partial parse tree ➔ phrase of $LZ_{no}$-like parsing

#(internal nodes) = #(rules) – $\sigma$ = $g$ – $\sigma$ = #(leaves) – 1

➔ $g \geq g - \sigma + 1 = \#(\text{leaves}) = z_g \geq z_{no} \geq z$

Example

$X_7 \rightarrow X_6 X_5$

$X_6 \rightarrow X_4 X_5$

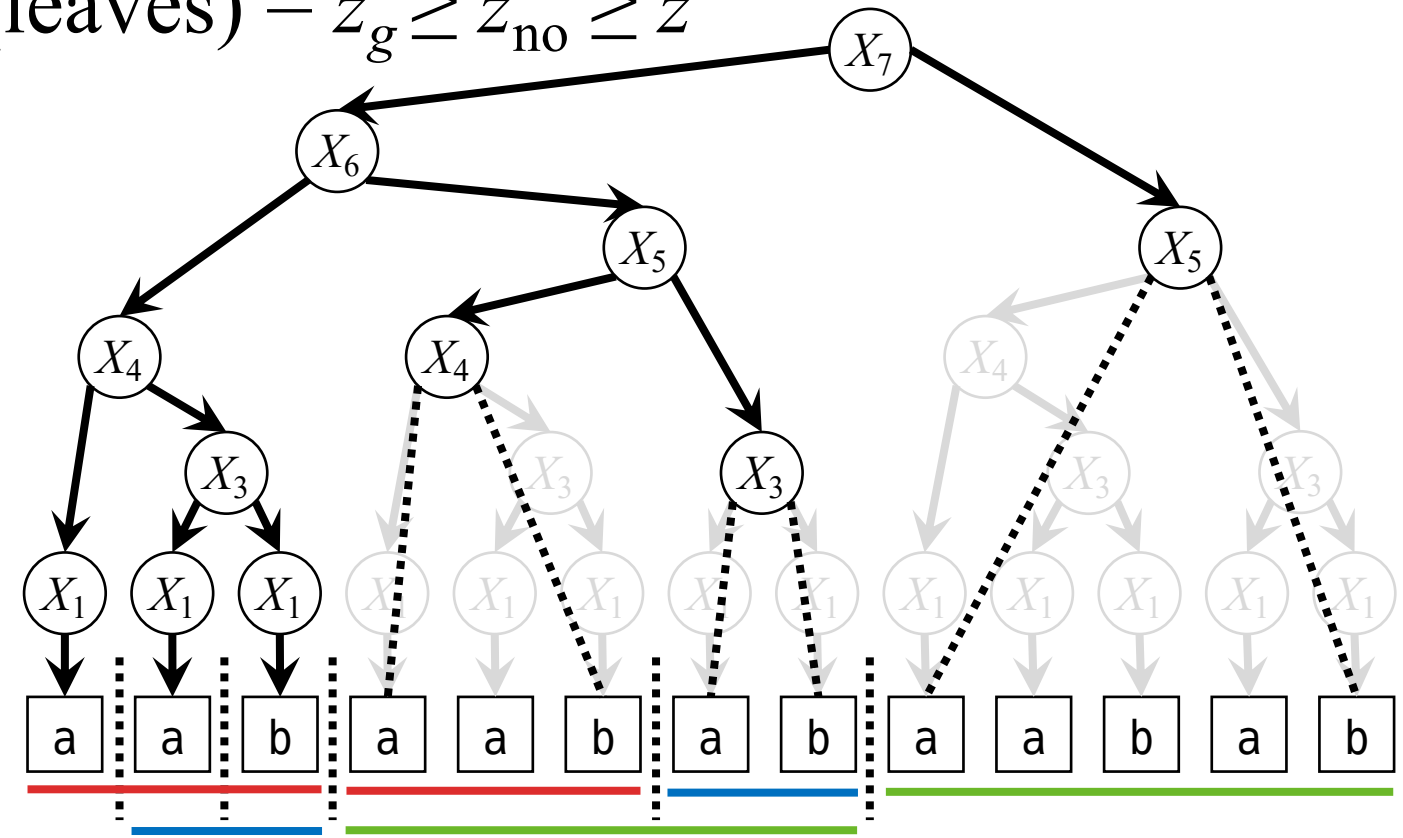
$X_5 \rightarrow X_4 X_3$

$X_4 \rightarrow X_1 X_3$

$X_3 \rightarrow X_1 X_2$

$X_2 \rightarrow$ b

$X_1 \rightarrow$ a



# Run-Length SLP (RLSLP) $g_{rl}$

SLP + run-length rules: $X_i \rightarrow X_j^k$ $g_{rl}$(T): size of smallest RLSLP for T

Partial parse tree: split run-length rules as $X_j$ and $X_j^{k-1}$.

Leaves of partial parse tree ➔ phrase of LZ-like parsing

#(internal nodes) = #(rules) – $\sigma$ = $g_{rl}$ – $\sigma$ = #(leaves) – 1

$g_{rl} \geq g_{rl} - \sigma + 1 = \#(\text{leaves}) = z_{grl} \geq z$

Example

$X_7 \rightarrow X_5X_6$

$X_6 \rightarrow X_3^3$

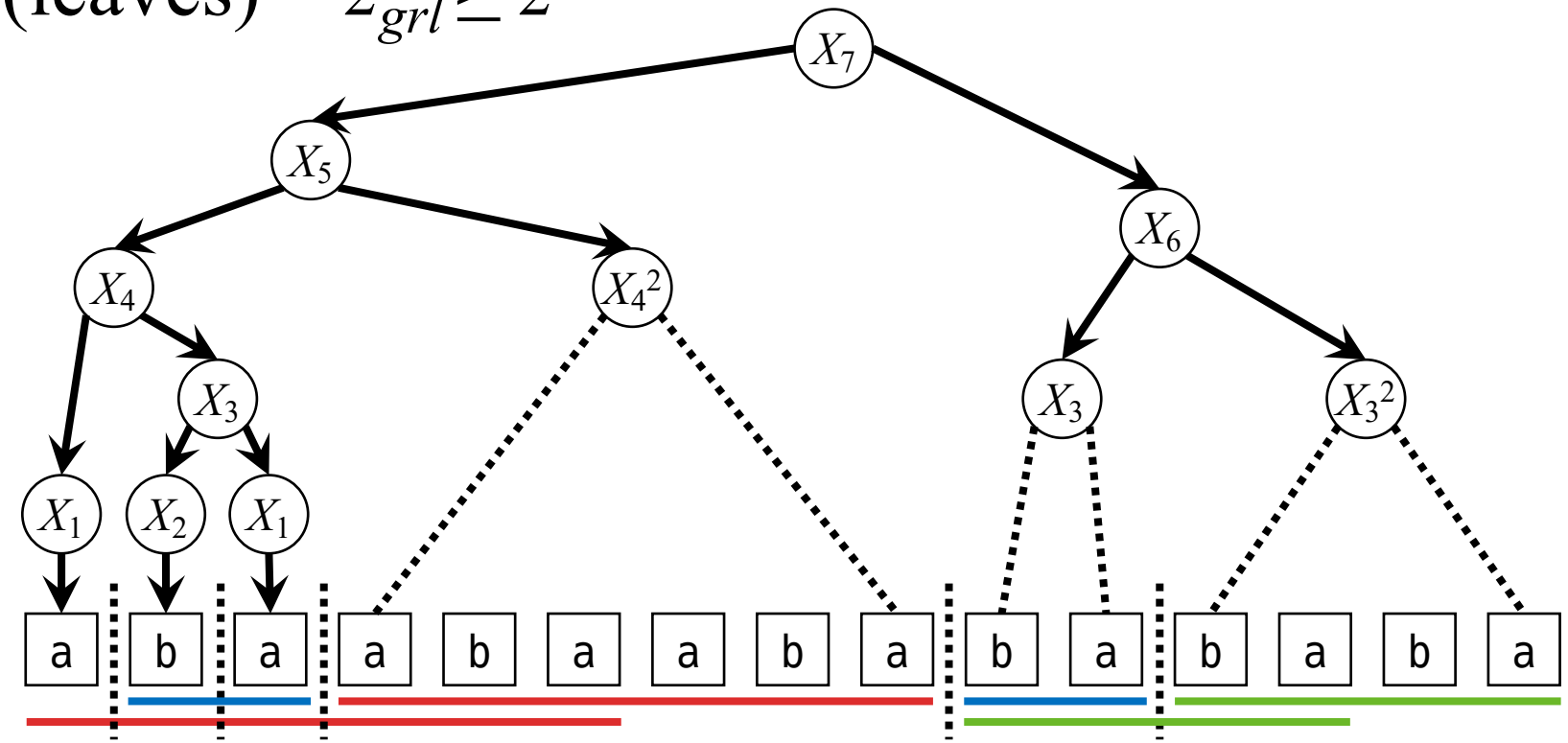
$X_5 \rightarrow X_4^3$

$X_4 \rightarrow X_1X_3$

$X_3 \rightarrow X_2X_1$

$X_2 \rightarrow$ b

$X_1 \rightarrow$ a



# Computing SLP/RLSLP

- Computing the smallest SLP / smallest RLSLP is NP-hard
  [Charikar et al. 2005][Kawamoto et al. 2024]
- No polynomial time algorithm w/ approximation ratio less than 8569/8568 unless P = NP [Charikar et al. 2005]

- Several different ways to compute O(log $n$) approximations. e.g.:
  - Construction from LZ77 [Rytter 2002, 2003][Charikar et al. 2005]
  - Locally consistent parsing [Jeż 2015]

# SLP from LZ77no

[Rytter 2002, 2003]

An SLP of size $O(z_{no}(T)\log n)$ can be computed in $O(n)$ time.
➔ $g(T) = O(z_{no}(T)\log n)$

Proof

Consider self-balancing grammars.

An **AVL grammar** is an SLP such that the parse tree is an **AVL tree** – a balanced binary tree where the difference between the heights of the left and right subtree is at most 1.

# AVL Grammars

[Rytter 2002, 2003]

Given AVL grammars for $S_1, S_2$, an AVL grammar for $S=S_1S_2$ can be obtained by adding $d = O(|height(S_1) - height(S_2)|)$ non-terminals.

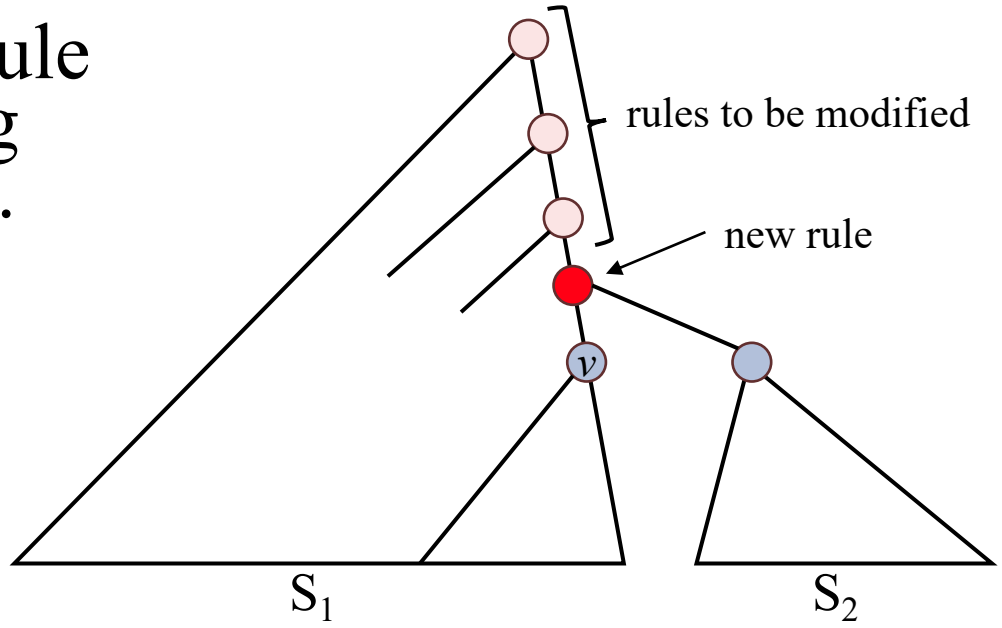
Proof

Assume $height(S_1) \geq height(S_2)$. (other case is symmetric)

- Add new rule as parent of $v$ on the right-most path of $S_1$, where $height(v) = height(S_2)$.
- All rules above $v$ are changed to a fresh rule since they may be used elsewhere, adding $d \leq height(S_1) - height(S_2)$ non-terminals.

The grammar may need re-balancing as root to $v$ path is longer.

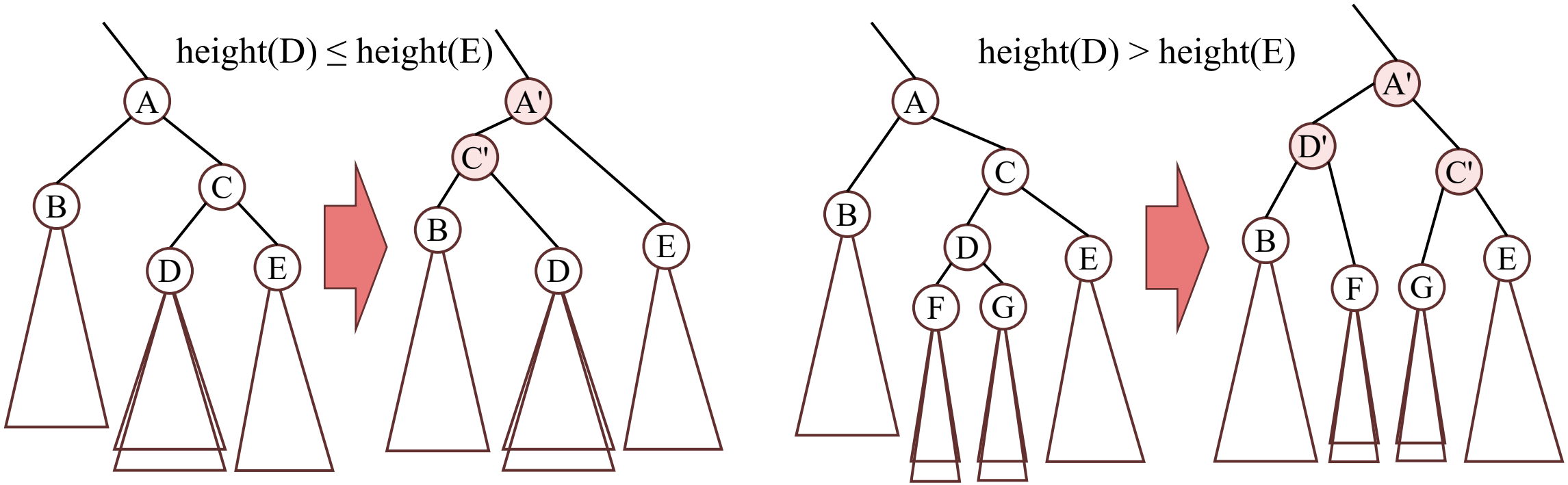
Note: $height(S) \leq \max(height(S_1), height(S_2))+1$



# Re-balancing AVL grammars

[Rytter 2002, 2003]

- Essentially same as AVL trees; by O(*d*) rotations.
- Rotate for rule A→BC if *height*(C) = *height*(B)+2
  - Two cases depending on heights of C → DE




- Only O(1) new rules are required per rotation.
- Continue rebalancing for ancestors.

# SLP from LZ77no (1)

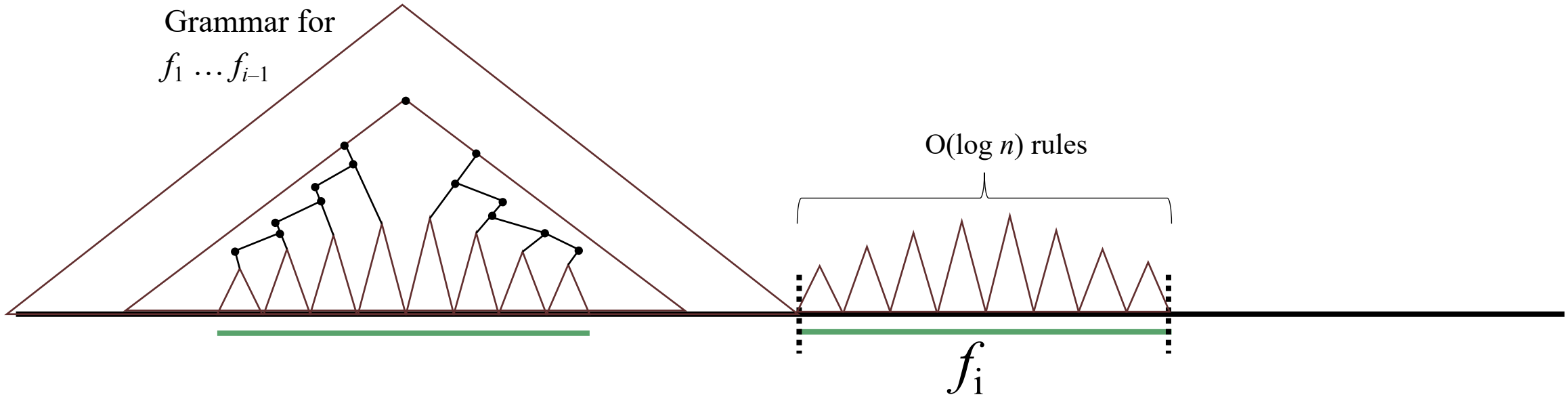
[Rytter 2002, 2003]

Construct AVL grammar for T, given non-self referencing LZ77 parse of T.

For each phrase of non-self-referencing LZ77:

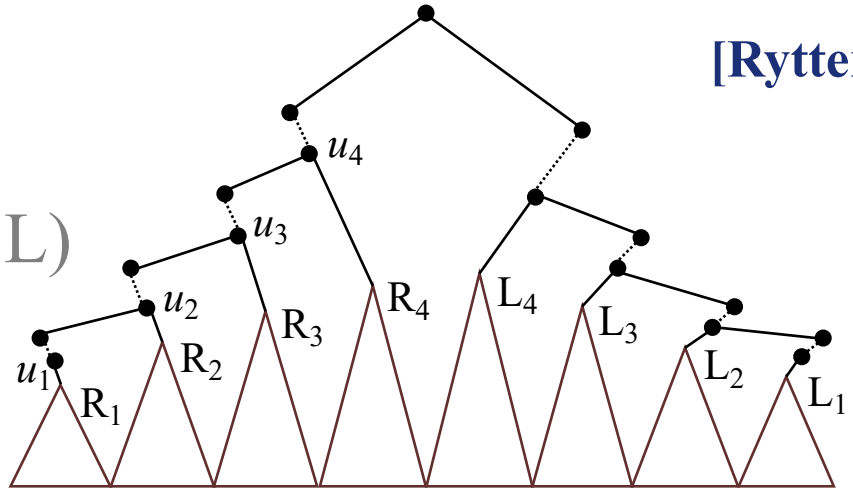
1. Find sequence of O(log $n$) rules whose concatenation is phrase
2. Concatenate the rules

Each concatenation can add O(log $n$) rules ➔ O($z_{no}\log^2 n$) size



# SLP from LZ77no (2)

[Rytter 2002, 2003]



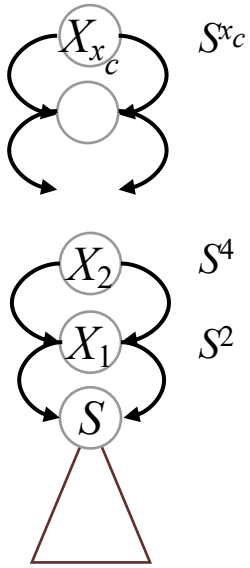

A more precise analysis:

- Let $G_k = \odot_{i=1..k} R_i$ (symmetric for L)
- Claim: $height(G_i) \le height(u_i)$
  - True for $i = 1$
  - For $i+1$:
    - If $height(G_i) \ge height(R_{i+1})$ : $height(G_{i+1}) \le height(G_i)+1 \le height(u_i)+1 \le height(u_{i+1})$
    - If $height(G_i) \le height(R_{i+1})$ : $height(G_{i+1}) \le height(R_{i+1})+1 \le height(u_{i+1})+1$
- # of rules added:
  - If $height(G_i) \ge height(R_{i+1})$
    - $height(G_i) - height(R_{i+1}) \le height(u_i) - height(R_{i+1}) = 1$
  - If $height(G_i) \le height(R_{i+1})$
    - $height(R_{i+1}) - height(G_i) \le height(R_{i+1}) - height(R_i)$
- Total
  - $\sum_i O(|height(R_{i+1}) - height(G_i)|) \le O(\sum_i (height(R_{i+1}) - height(R_i))) + O(\sum_i 1) = O(\log n)$

# SLP from LZ77

An $O(z(T) \log n)$ size grammar can be constructed as follows:
(c.f. [Gawrychowski 2011])

- A self referencing LZ phrase $f$ implies periodicity in the phrase
  There exists $S$ s.t. $f = S^k S'$, where $S'$ is empty or proper prefix of $S$.
- Grammar rules for $S$ and $S'$ add $O(\log n)$ rules using same approach.
- Construct grammar rules for $S^k$ w/ another $O(\log n)$ rules:
  - Let $k = 2^{x_1}+2^{x_2}+\ldots+2^{x_c}$ where $0 \le x_1 < \ldots < x_c \le \lfloor \log k \rfloor$ are chosen appropriately. (binary representation of $k$)
  - Construct balanced non-terminal $X_i$ of height: $i + height(S)$ for each $2^i$
    - Can be done by adding $O(x_c)$ rules.
  - Let $G_i = \bigodot_{j=1..i} X_{x_i}$.
    - $height(G_i) \le max\{ height(G_{i-1}), height(X_i) \} + 1 = height(X_i) + 1$
    - Then, #new rules for $G_c$:
      $\sum_i O(|height(X_{i+1}) - height(G_i)|) = O(\sum_i (x_{i+1} - x_i)) = O(x_c) = O(\log k) = O(\log n)$.



# Separation between $g$ and $z_{no}$

[Charikar et al. 2005]

There exists an infinite family of strings such that
$g/z_{no} = \Omega(\log n/\log \log n)$

To construct this family, utilize relation between SLPs and notion of **addition chains**.

# Addition chains and SLP

An **addition chain** for positive integer $n$ is a strictly increasing sequence of integers: $a_0, a_1, \ldots, a_l$ where $a_0 = 1$, $a_l = n$, and for all $1 \leq i \leq l$, $a_i = a_j + a_k$ for some $1 \leq j, k < i$.

(Shortest) addition chain for $n$ ⇔ (smallest) SLP for $\mathsf{a}^n$

$a_0$ ⇔ $X_0 \rightarrow \mathsf{a}$

$a_i = a_j + a_k$ ⇔ $X_i \rightarrow X_j X_k$

- No polynomial-time algorithm for finding the shortest addition chain is known.
  - It is unknown whether it is NP-complete.
- The decision version of the shortest addition chain for a set of numbers has been shown to be NP-complete. [Downey et al. 1981]

Finding the smallest SLP of **unary strings** already seems difficult.

# Separation between $g$ and $z_{no}$ (1)

[Charikar et al. 2005]

There exists a set of log $k$ integers in [1,$k$] s.t. the length of the shortest addition chain that includes the integers is at least $\Omega(\log^2 k/\log\log k)$. (Simplified version of more general result [Pippenger 1980])

Proof (Counting argument)

- # of such sets = $\binom{k}{\log k} \geq \left(\frac{k}{\log k}\right)^{\log k} = 2^{(\log k)\log(\frac{k}{\log k})} = 2^{\Omega(\log^2 k)}$
- # of addition chains:
  - of length $L \leq \prod_{i=1..L} i^2 = (L!)^2$
  - of length at most $L$

    $= \sum_{i=1..L} (i!)^2 = O((L!)^2) = O(2^{2\log(L!)}) = O(2^{2L\log L}) = 2^{O(L \log L)}$
- # sets covered by a chain is at most $\binom{L}{\log k} = 2^{O(\log k \log L)}$. Total is still $2^{O(L \log L)}$.
  - If $L = o(\log^2 k/\log\log k)$, then $2^{O(L \log L)} = 2^{o(\log^2 k)}$ leading to contradiction.

# Separation between $g$ and $z_{no}$ (2)

[Charikar et al. 2005]

There exists an infinite family of strings such that $g/z_{no} = \Omega(\log n/\log\log n)$

Proof

- Let $T = a^{k_1}\$_1 a^{k_2}\$_2 \cdots \$_{m-1} a^{k_m}$, where $k_1 = \max_i k_i$, $m = \log k_1$, and $g(T) = \Omega(\log^2 k_1/\log\log k_1)$. (existence guaranteed by previous claim)
- $z_{no}(T) = O(\log k_1)$.
- $n = |T| = O(k_1 \log k_1)$
- $g(T)/z_{no}(T) = \Omega(\log k_1/\log\log k_1) = \Omega(\log n/\log\log n)$

# Separation between $g_{rl}$ and $z$

[Bille et al. 2018]

There exists an infinite family of strings such that
$g_{rl}/z = \Omega(\log n/\log\log n)$

Proof

- Let $T = a^{k_1}\$_1 a^{k_2}\$_2 \cdots \$_{m-1} a^{k_m}$, where $k_1 = \max_i k_i$, $m = \log k_1$, and $g(T) = \Omega(\log^2 k_1/\log\log k_1)$.
- Let $T' = t[1..k_1]\$_1 \cdots \$_{m-1} t[1..k_m]^k$ $t$: infinite Thue-Morse word. Since Thue-Morse words are cube free, $g_{rl}(T')=g(T')$.
- Grammar of size $g$ for $T'$ induces grammar of size $g - 1$ for $T$.
  ➔ $g(T') - 1 \geq g(T) = \Omega(\log^2 k_1/\log\log k_1)$.
- $z(T') = O(\log n)$

T is unary

Institute of SCIENCE TOKYO

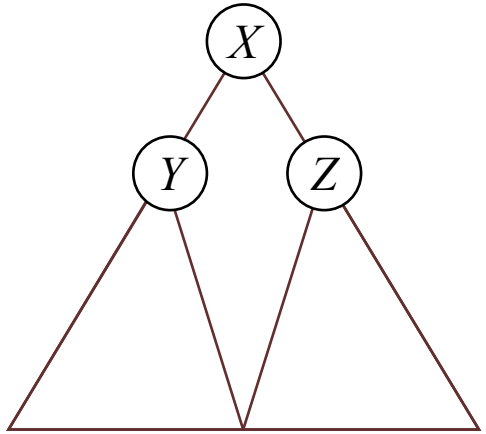
# Balancing grammars

# Access on SLPs

Given SLP $S$ of size $x$ for a string T of length $n$, there exists an $O(x)$ size data structure that can be computed in $O(|x|)$ time, and can answer access queries (return T[$i$] for query $i$) in $O(height(S))$ time.

Proof

- For each rule $X \rightarrow YZ$, compute length $||X||$ of string that $X$ derives.
  - Simple bottom-up DP in topological order $||X|| = ||Y||+||Z||$
- Access to position $i$ in $[\![X]\!]$ :
  - If $i \leq ||Y||$, position $i$ in $[\![Y]\!]$
  - Otherwise, position $i - ||Y||$ in $[\![Z]\!]$




AVL grammar has $O(\log n)$ height ➔ $O(\log n)$ time access.
Is there an $O(\log n)$ height grammar without increasing asymptotic size?

# Access and Balancing Grammars

[Ganardi et al. 2019, 2021]

Given SLP of size $x$ for a string T of length $n$, there exists an $O(x)$ size data structure that can be computed in $O(|x|)$ time, and can answer access queries (return T[$i$] for query $i$) in $O(\log n)$ time.

First solution by [Bille et al. 2011, 2015].
A more general solution by [Ganardi et al. 2019, 2021]:

Given SLP of size $x$ for a string T of length $n$, an $O(x)$ size SLP for T with height $O(\log n)$ can be computed in $O(|x|)$ time.

Proof: a conceptually simple rebalancing technique based on a new decomposition of DAGs, similar to heavy-path decomposition

# Symmetric Centroid Decomposition

[Ganardi et al. 2019, 2021]

Given DAG $G=(V, E)$ with single root $r$, and sink set $W$,
define for each node $v \in V$:

$$\lambda(v) = (\lfloor \log \#(r\text{-}v \text{ paths}) \rfloor, \lfloor \log \#(v\text{-}W \text{ paths}) \rfloor).$$

A **Symmetric Centroid Decomposition** of $G$ is a graph $G'$ where $G' = (V, E')$ where $E' = \{ (u, v) \in E : \lambda(u) = \lambda(v) \}$.

SCD $G'$ partitions G into paths. (SC path)

Proof:
If $v$ has more than 1 incoming edges in $G'$: $(u_1,v)$, $(u_2,v) \in E'$, $\#(r\text{-}v \text{ paths}) \geq \#(r\text{-}u_1 \text{ paths})+\#(r\text{-}u_2 \text{ paths})$.
Assuming wlog $\#(r\text{-}u_1 \text{ paths}) \geq \#(r\text{-}u_2 \text{ paths})$, then, $\#(r\text{-}v \text{ paths}) \geq 2 \cdot \#(r\text{-}u_2 \text{ paths})$.
Therefore $\lfloor \log \#(r\text{-}v \text{ paths}) \rfloor \neq \lfloor \log \#(r\text{-}u_2 \text{ paths}) \rfloor$, a contradiction.

Symmetric arguments for outgoing edges.

# Symmetric Centroid Decomposition [Ganardi et al. 2019, 2021]

Given DAG $G=(V, E)$ with single root $r$, and sink set $W$,
define for each node $v \in V$:

$$\lambda(v) = (\lfloor \log \#(r\text{-}v \text{ paths}) \rfloor, \lfloor \log \#(v\text{-}W \text{ paths}) \rfloor).$$

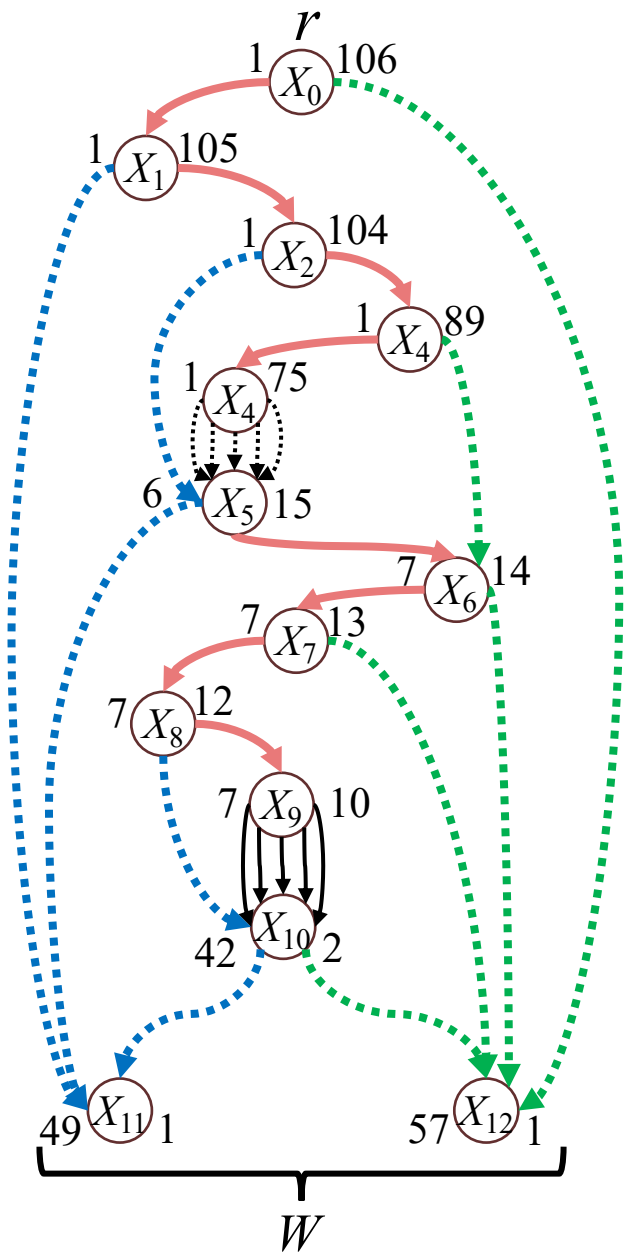
A **Symmetric Centroid Decomposition** of $G$ is a graph $G'$ where $G' = (V, E')$ where $E' = \{ (u, v) \in E : \lambda(u) = \lambda(v) \}$.

SCD $G'$ partitions G into paths. (SC path)

Any root to sink path contains at most $2\log n$ non-SC edges, where $n = \#(r\text{-}W \text{ paths})$.

Proof: Traversing a non-SC path either increases the first element or decreases the second element of $\lambda(v)$.

# SCD Example




Red path = SC path

Example taken from Fig. 1 [Navarro et al. 2025]

#(*r*-*v* paths): left of node
#(*v*-*W* paths): right of node

Blue dotted lines:
left branching edges from SC path

Green dotted lines:
right branching edges from SC path

Note: branching edges cannot rejoin same SC path

# Main Lemma for Balancing

[Ganardi et al. 2019, 2021]

For any string T of length $n$ of **non-terminals**, of a given SLP, an SLP $G$ with the following properties can be constructed:

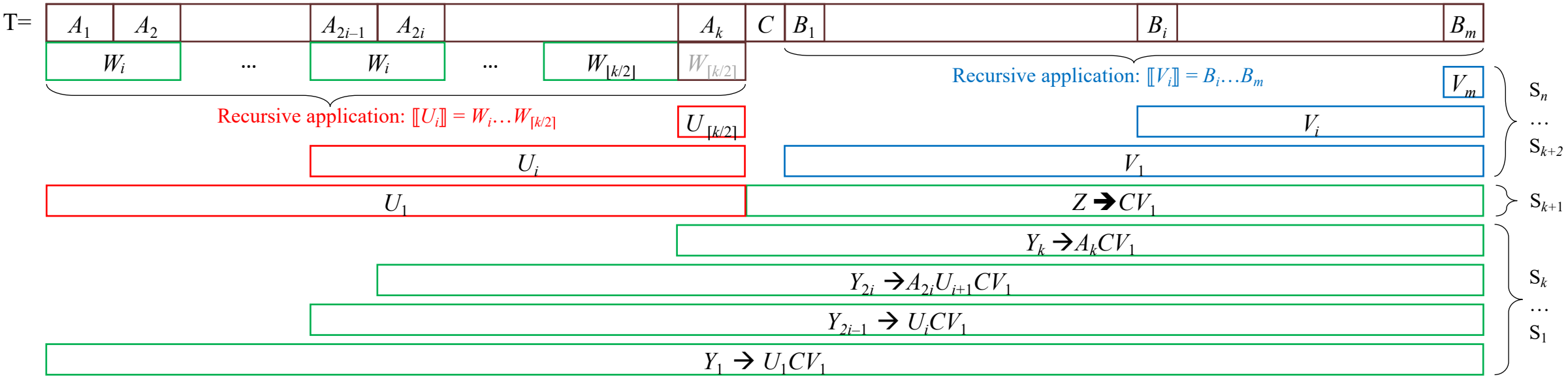
- #non-terminals $\leq 3n$
- |RHS of non-terminal| $\leq 4$ (slight extension of SLP – can be fixed later)
- includes non-terminals $S_1, \ldots S_n$ that derive all non-empty suffixes of T
- Ever path $S_i \dashrightarrow A$ in $G$, where $A$ is a symbol in T, has length at most: $\mathbf{1 + 2(\lceil \log \|S_i\| \rceil - \log \|A\|)} \leq 3 + 2(\log \|S_i\| - \log \|A\|)$

※ Here, $\|X\|$ represents the length of *terminal* sequence derived from non-terminal $X$.

※ Holds symmetrically for prefixes of T as well.

# Main Lemma Proof: Construction Overview

Let T = $A_1..A_kCB_1..B_m$ where $CB_1..B_m$ is shortest suffix of T s.t. $\lceil \log \|CB_1..B_m\| \rceil = \lceil \log \|T\| \rceil$.
Then, $\lceil \log \|A_i \ldots A_kCB_1..B_m\| \rceil = \lceil \log \|CB_1..B_m\| \rceil \geq \lceil \log \|B_1..B_m\| \rceil + 1$ and
$\lceil \log \|A_i \ldots A_kCB_1..B_m\| \rceil = \lceil \log \|CB_1..B_m\| \rceil \geq \lceil \log \|A_1..A_k\| \rceil + 1$



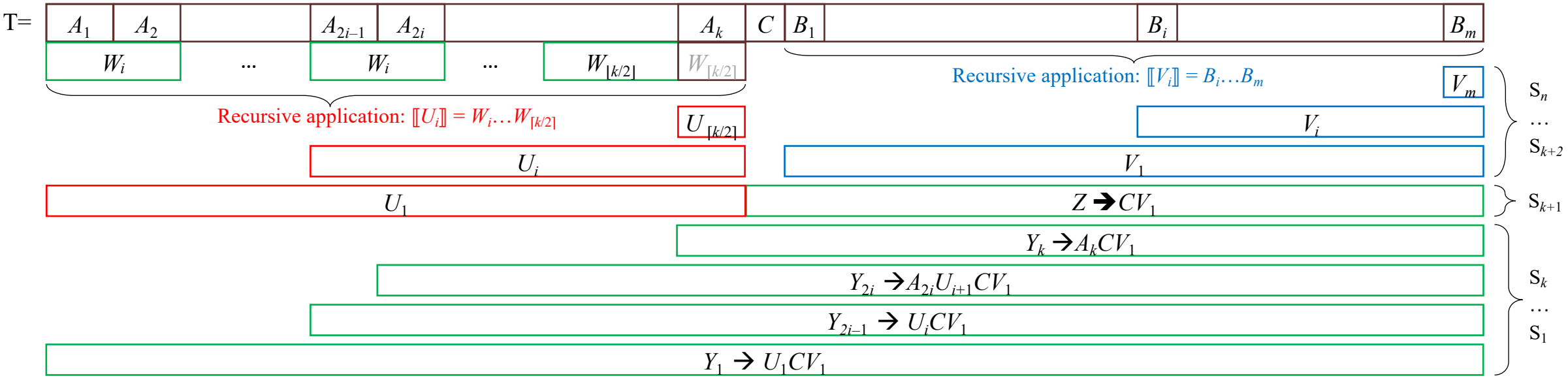

Total # non-terminals:

- New: $W_1 \ldots W_{\lfloor k/2 \rfloor}$, $Y_1..Y_k$, $Z$
- From recursive calls: on $W_1 \ldots W_{\lfloor k/2 \rfloor} \leq 3\lceil k/2 \rceil$, on $B_1 \ldots B_m \leq 3m = 3(n-k+1)$

$\rightarrow \leq \lfloor k/2 \rfloor + k + 1 + 3\lceil k/2 \rceil + 3(n-k-1) < 3n$

# Main Lemma Proof: Checking Path Lengths

Let T = $A_1..A_kCB_1..B_m$ where $CB_1..B_m$ is shortest suffix of T s.t. $\lceil \log \|CB_1..B_m\| \rceil = \lceil \log \|T\| \rceil$.
Then, $\lceil \log \|A_i \ldots A_kCB_1..B_m\| \rceil = \lceil \log \|CB_1..B_m\| \rceil \geq \lceil \log \|B_1..B_m\| \rceil + 1$ and
$\lceil \log \|A_i \ldots A_kCB_1..B_m\| \rceil = \lceil \log \|CB_1..B_m\| \rceil \geq \lceil \log \|A_1..A_k\| \rceil + 1$




$|U_i \rightsquigarrow W_j| \leq 1 + 2(\lceil \log \|U_i\| \rceil - \log \|W_j\|)$
$|U_i \rightsquigarrow A_j| = |U_i \rightsquigarrow W_{\lfloor j/2 \rfloor}| + 1 \leq 2 + 2(\lceil \log \|U_i\| \rceil - \log \|A_j\|)$

$|V_i \rightsquigarrow B_j| \leq 1 + 2(\lceil \log \|V_i\| \rceil - \log \|B_j\|)$

$|Y_i \rightsquigarrow C| = |Z \rightsquigarrow C| = 1$

$|Y_i \rightsquigarrow A_j| \leq 1 + |U_l \rightsquigarrow A_j| \leq 3 + 2(\lceil \log \|U_l\| \rceil - \log \|A_j\|)$
$\leq 3 + 2(\lceil \log \|U_1\| \rceil - \log \|A_j\|)$
$= 3 + 2(\lceil \log \|A_{1\ldots}A_k\| \rceil - \log \|A_j\|)$
$= 1 + 2(\lceil \log \|Y_i\| \rceil - \log \|A_j\|)$

$|Y_i \rightsquigarrow B_j| \leq 1 + |V_1 \rightsquigarrow B_j| \leq 2 + 2(\lceil \log \|V_1\| \rceil - \log \|B_j\|)$
$\leq 2(\lceil \log \|Y_i\| \rceil - \log \|B_j\|)$

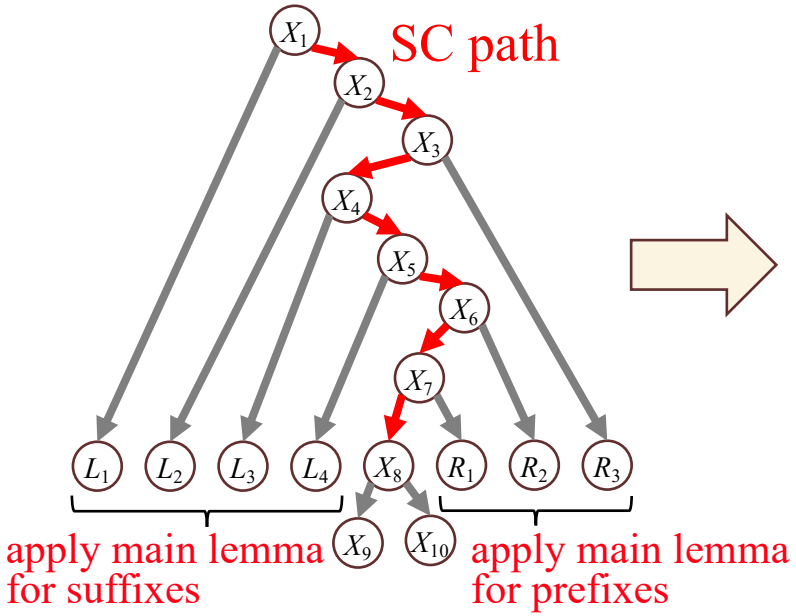
$|Z \rightsquigarrow B_j| = 1 + |V_1 \rightsquigarrow B_j| \leq 2 + 2(\lceil \log \|V_1\| \rceil - \log \|B_j\|)$
$\leq 2(\lceil \log \|Z\| \rceil - \log \|B_j\|)$

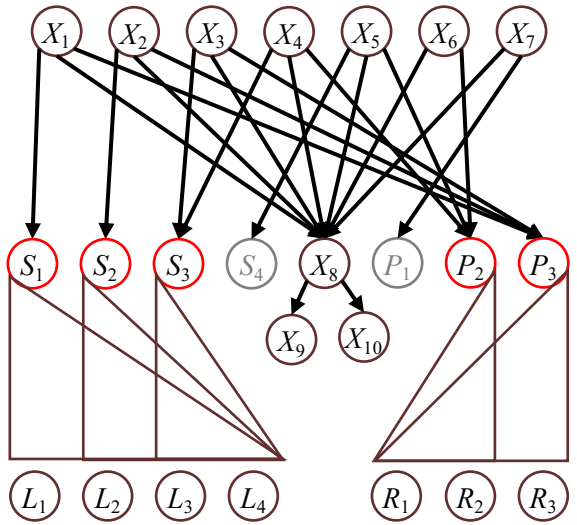
# Rebalancing SC paths with main lemma

For any SC path: $X_1 \ldots X_p$

$L_1,\ldots, L_s$ : left children branching from SC path, $R_1,\ldots, R_t$ : right children branching from SC path

- For any $X_i$ $(1 \le i \le p)$ in path, there exists:
  - suffix $L_j \ldots L_s$ of $L_1 \ldots L_s$ and prefix $R_1 \ldots R_k$ of $R_1 \ldots R_t$ s.t. $[\![X_i]\!] = [\![L_j \ldots L_s X_p R_1 \ldots R_k]\!]$
- Apply main lemma to get
  - rules $S_1,\ldots, S_s$ for suffixes of $L_1 \ldots L_s$ (w/ at most $3s = O(p)$ non-terminals)
  - rules $P_1,\ldots, P_t$ for prefixes of $R_1 \ldots R_t$ (w/ at most $3t = O(p)$ non-terminals)
- Replace RHS of non-terminal $X_i$ with $X_i \rightarrow S_j X_p P_k$ (where $S_j$ or $P_k$ may be empty)



Any root ⇝ sink path in orig SLP is:
$X_0 \rightsquigarrow X_1 \rightsquigarrow \ldots \rightsquigarrow X_k$
where each $X_i \rightsquigarrow X_{i+1}$ is a sub-SC path except last edge leading to $X_{i+1}$
Then, the total path length in new SLP
$\le \sum_i (4 + 2(\log|X_i| - \log|X_{i+1}|))$
$= 4k + 2\log|X_0| = O(\log n)$

$|X_i \rightsquigarrow L_j| = 1 + |S_i \rightsquigarrow L_j| \le 4 + 2(\log|X_i| - \log|L_j|)$
$|X_i \rightsquigarrow R_j| = 1 + |P_i \rightsquigarrow R_j| \le 4 + 2(\log|X_i| - \log|R_j|)$

$|S_i \rightsquigarrow L_j| \le 3 + 2(\log|S_i| - \log|L_j|)$
$|P_i \rightsquigarrow R_j| \le 3 + 2(\log|P_i| - \log|R_j|)$

# Extension to RLSLP

[Navarro et al. 2022,2025]

- The same procedure can be applied to SLPs with run-length rules.
  - $X \rightarrow Y^k$ : By definition, $X$ must end of SC path and will not be part of rebalancing.

# Repetitiveness measures based on substring properties

# String Attractors $\gamma$

[Kempa&Prezza 2018]

A ***string attractor*** of T is a set of positions $I$ of $T$ such that any substring of T has an occurrence that contains a position in $I$.

## Example

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | b | b | a | b | a | a | b | b | a | a | b | a | b | b | a |

The size of the smallest string attractor of T is denoted by $\gamma$(T).

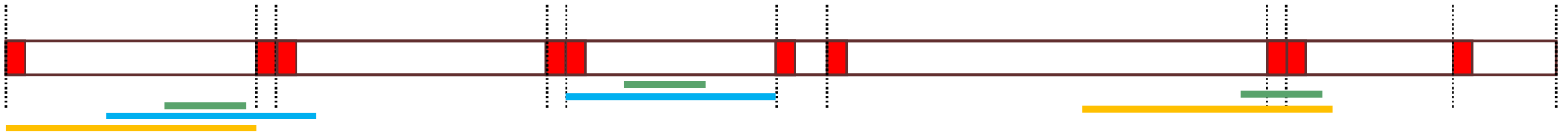
Computing $\gamma$(T) is NP-hard.

# $\gamma$ lower bounds $b$

For any string T, any BMS for T induces a string attractor of T of the same size, implying $\gamma(T) \leq b(T)$.

Proof

The set of starting positions (or ending positions) of each phrase in a BMS is a string attractor of T.



If an occurrence of a substring does not contain the starting position of a phrase, it must be a proper substring of a phrase. Then, it has an occurrence elsewhere. Since this process cannot be repeated indefinitely, at some point there must be an occurrence containing the starting position of a phrase.

# Separation between $\gamma$ and $b$

[Kutsukake et al. 2020][Bannai et al. 2021]

Thue-Morse words $t_k = \mu^k(\mathsf{a})$, where $\mu(\mathsf{a}) = \mathsf{ab}$, $\mu(\mathsf{b}) = \mathsf{ba}$.

$t_1 = \mathsf{ab}$
$t_2 = \mathsf{abba}$
$t_3 = \mathsf{abbabaab}$
$t_4 = \mathsf{abbabaabbaababba}$
…
$|t_n| = 2^n$
If $s_k = \mu^n(\mathsf{b})$, then $t_k = t_{k-1}s_{k-1}$, and $s_k = s_{k-1}t_{k-1}$.

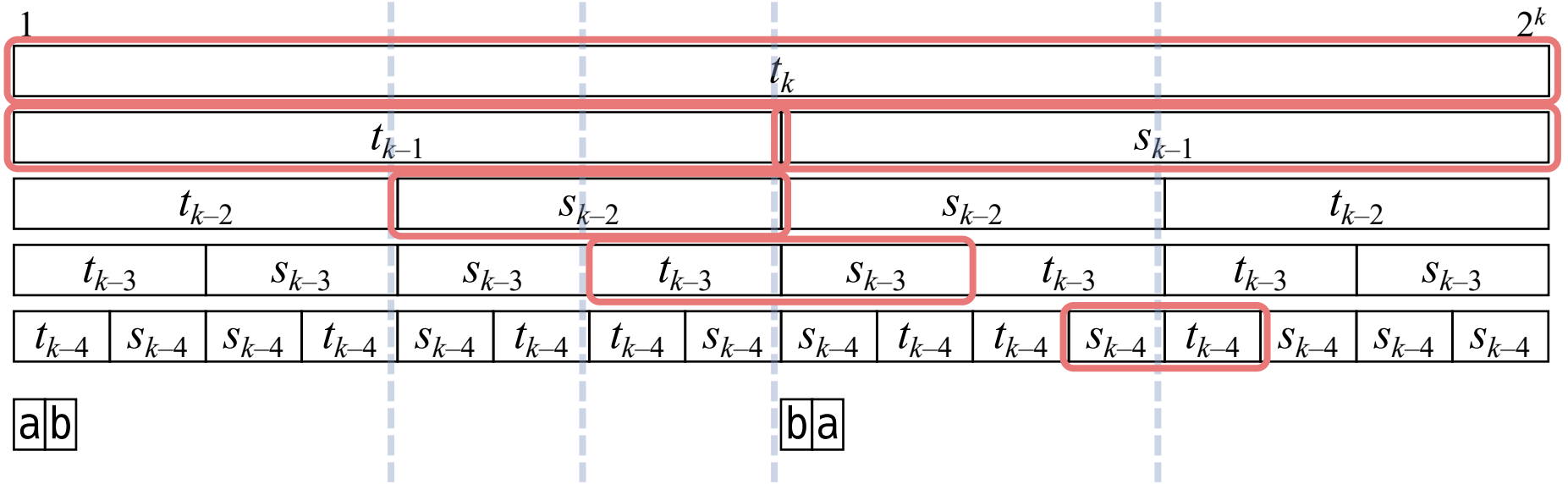
$\gamma(t_k) = 4$ and $b(t_k) = k+2$.

# $\gamma(t_k) = 4$

[Kutsukake et al. 2020]

The set $\{2^{k-2}, 3\cdot 2^{k-3}, 2^{k-1}, 3\cdot 2^{k-2}\}$ is a string attractor of $t_k$.

Proof




- Any node of tree (i.e., $t_i$, $s_i$ for $i \leq k$) has occurrence w/ center in the set.
  - These are centers of occurrences of $t_k$, $t_{k-1}$, $s_{k-1}$, $s_{k-2}$.
  - For any $i$, $t_i$ (resp $s_i$) has occurrence w/ same center as $t_{i+2}$ (resp $s_{i+2}$)
- Any substring includes a center of some $t_i$ or $s_i$ (the LCA of substring)

# $b(t_k) = k+2$

[Bannai et al. 2021]

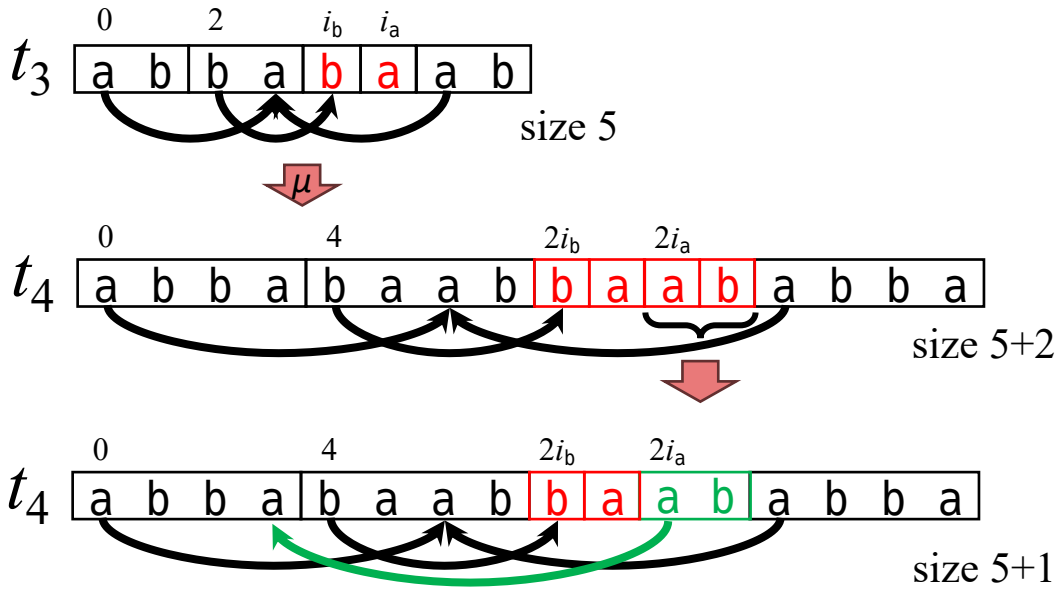
Proof: $b(t_k) \leq 2k$ is simple:

- $b(t_2 = \mathtt{abba}) \leq 4$.
- $t_{k+1}=t_k s_k=t_k s_{k-1} t_{k-1}=(t_{k-1} s_{k-1})(s_{k-1})(t_{k-1})$. Therefore, $b(t_{k+1}) \leq b(t_k)+2$.

$b(t_k) \leq k+2$

Base case $k = 3$.

- Apply $\mu$ (assuming 0-index)
  - Double reference positions.
- Merge pair $\mathtt{ab}(=\mu(\mathtt{a}))$ and reference position 3.
  - New edge does not introduce cycles: this is the only edge w/ different parity between positions.



# $b(t_k) \geq k+2$ (Proof sketch)

Go in opposite direction.

Proof by induction:

- Smallest BMS for $t_2$ has size $2 + 2 = 4$.
- Assume $b(t_k) \geq k+2$ holds for all integers up to some $k \geq 2$.

**Given a BMS of size $s$ for $t_{k+1}$, IF we can construct a BMS of size $s - 1$ for $t_k$, then,**

$s \geq k+3$ must hold since $s - 1 \geq k + 2$.

# $b(t_k) \geq k+2$ (Proof sketch)

Go in opposite direction.

Proof by induction:

- Smallest BMSs for $t_2$, $t_3$, $t_4$ resp. have sizes 4, 5, 6.
- Assume $b(t_k) \geq k+2$ holds for all integers up to some $k \geq 2$.

**Given a BMS of size $s$ for $t_{k+1}$, ~~If~~ we can construct a BMS of size $s - i$ for $t_{k+1-i}$ for some $i \in \{ 1, 2, 3 \}$.**

$s \geq k+3$ must hold since $s - i \geq (k + 1 - i) + 2 = k - i + 3$.

# Length Normalized Substring Complexity $\delta$

[Raskhodnikova et al. 2013]
[Christiansen et al. 2020]
[Kociumaka et al. 2020,2023]

The length normalized substring complexity $\boldsymbol{\delta(T)}$ of T, is:
$\mathbf{max}_k\ |S_k(T)|/k$, where $S_k(T)$ is the set of length-$k$ substrings in T.

Example

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | b | b | a | b | a | a | b | b | a | a | b | a | b | b | a |

| $k$ | $S_k$ | $\lvert S_k\rvert$ | $S_k(T)/k$ |
|---|---|---|---|
| 1 | a,b | 2 | 2 |
| 2 | aa,ab,ba,bb | 4 | 2 |
| 3 | aab,aba,abb,<br>baa,bab,bba | 6 | 2.0 |
| 4 | … | 10 | **2.5** |
| 5 | … | 12 | 2.4 |
| 6 | .. | 11 | 1.8333… |
| … | | | |

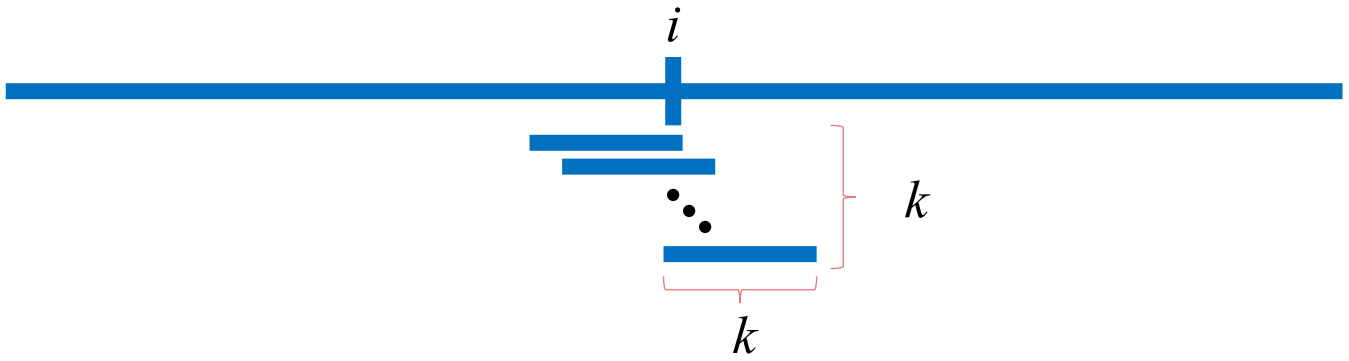
# $\delta$ lowerbounds $\gamma$

For any string T, $\delta(T) \leq \gamma(T)$.

Proof

For any position $i$ in a given string attractor, at most $k$ distinct substrings of length $k$ have an occurrence that contains $i$.

Therefore, $|S_k(T)| \leq k \cdot \gamma(T)$ for any $k$, implying $|S_k|(T)/k \leq \delta(T) \leq \gamma(T)$.



# Separation between $\delta$ and $\gamma$

[Kociumaka et al. 2020,2023]

There is an infinite family of strings s.t. $\delta=O(1)$ and $\gamma=\Omega(\log n)$.

Proof

Let $T_\infty$ be an infinite string s.t.

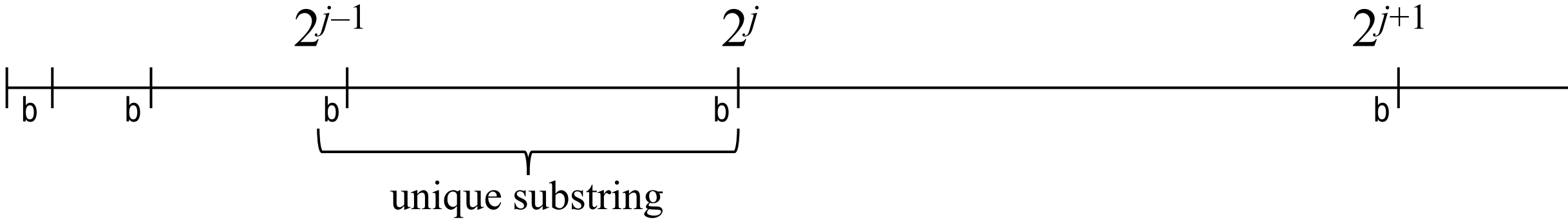
$T_\infty[i] = \mathtt{b}$ if $i = 2^j$ for some $j \geq 0$, and

$T_\infty[i] = \mathtt{a}$ otherwise.

For every $n \geq 1$, $\boldsymbol{\gamma(T_\infty[1..n]) \geq \lfloor \log n \rfloor/2}$ and $\delta(T_\infty[1..n]) \leq 2$.

Every substring starting&ending with $\mathtt{b}$ is unique

➔At least $\lfloor \log n \rfloor/2$ unique substrings whose occurrences are disjoint.



# Separation between $\delta$ and $\gamma$

[Kociumaka et al. 2020,2023]

There is an infinite family of strings s.t. $\delta$=O(1) and $\gamma$=Ω(log $n$).

Proof

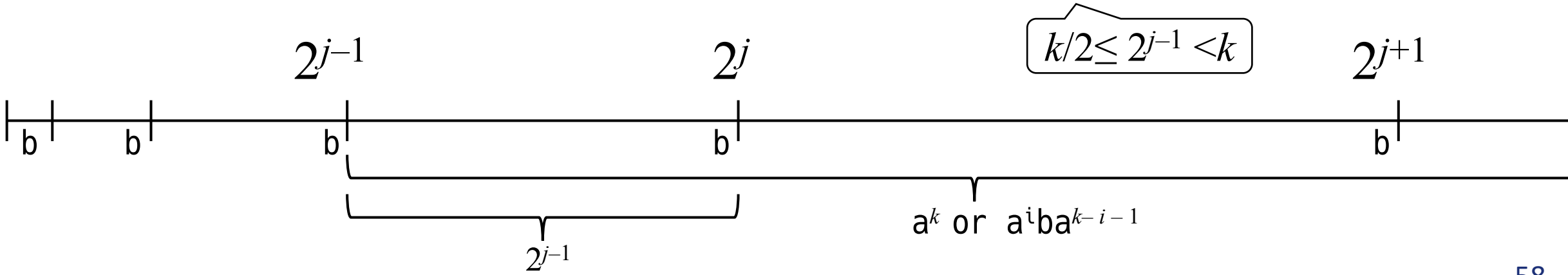
Let $T_\infty$ be an infinite string s.t.

$T_\infty[i] = \mathtt{b}$ if $i = 2^j$ for some $j \geq 0$, and

$T_\infty[i] = \mathtt{a}$ otherwise.

For every $n \geq 1$, $\gamma(T_\infty[1..n]) \geq \lfloor \log n \rfloor/2$ and $\boldsymbol{\delta(T_\infty[1..n]) \leq 2}$.

For any $2^{j-1} < k \leq 2^j$, # of distinct length-$k$ substrings: start in $[1,2^{j-1}]$ or $\mathtt{a}^k$ or $\mathtt{a}^i\mathtt{b}\mathtt{a}^{k-i-1}$ for $i \in [0..k)$ ➔ $2^{j-1}+k+1$. $\delta = \max_k(2^{j-1}+k+1)/k \leq 2$.



# $\delta$ is not reachable

[Kociumaka et al. 2020,2023]

An $O(\delta(T))$ size representation for all strings T cannot exist.

Proof

There exists an infinite family of strings with $\delta(T) = O(1)$ that requires $\Omega(\log^2 n)$ bits to encode.

- Consider the set of all binary strings s.t.:
  - there is exactly one b in the range $[2\cdot 4^{j-2}+1..4^{j-1}]$, for all $j \geq 2$, and the others are a.
- For any $j$ s.t. $4^{j-1} \leq n$, $2\cdot 4^{j-2}$ positions to choose for $j$th b
  - number of such strings: $\prod_{j=2..} (2\cdot 4^{j-2}) = 2^{\Omega(\log^2 n)}$
- #distinct length-$k$ substrings:
  - At most $4^{j-1}$ distinct substrings starting at position $\leq 4^{j-1}$
  - After position $4^{j-1}$, distance between occurrence of b's $\geq 4^j$
    - For any $k \leq 4^j$, either $a^k$ or $a^i b a^{k-i-1}$ for $0 \leq i < k$
    - $|S_k| \leq 4^{j-1} + k + 1$
  - $\delta = \max_k S_k/k \leq (4^{j-1} + k + 1)/k \leq 2$

# Other known results

- $g_{rl}(\mathrm{T}) = \mathrm{O}(\delta \log(n/\delta))$ [Kociumaka et al. 2021]
- $g(\mathrm{T}) = \mathrm{O}(\gamma \log(n/\gamma))$ [Kociumaka unpublished?]
- Data structures for efficient access and text indices can be implemented in theses spaces.

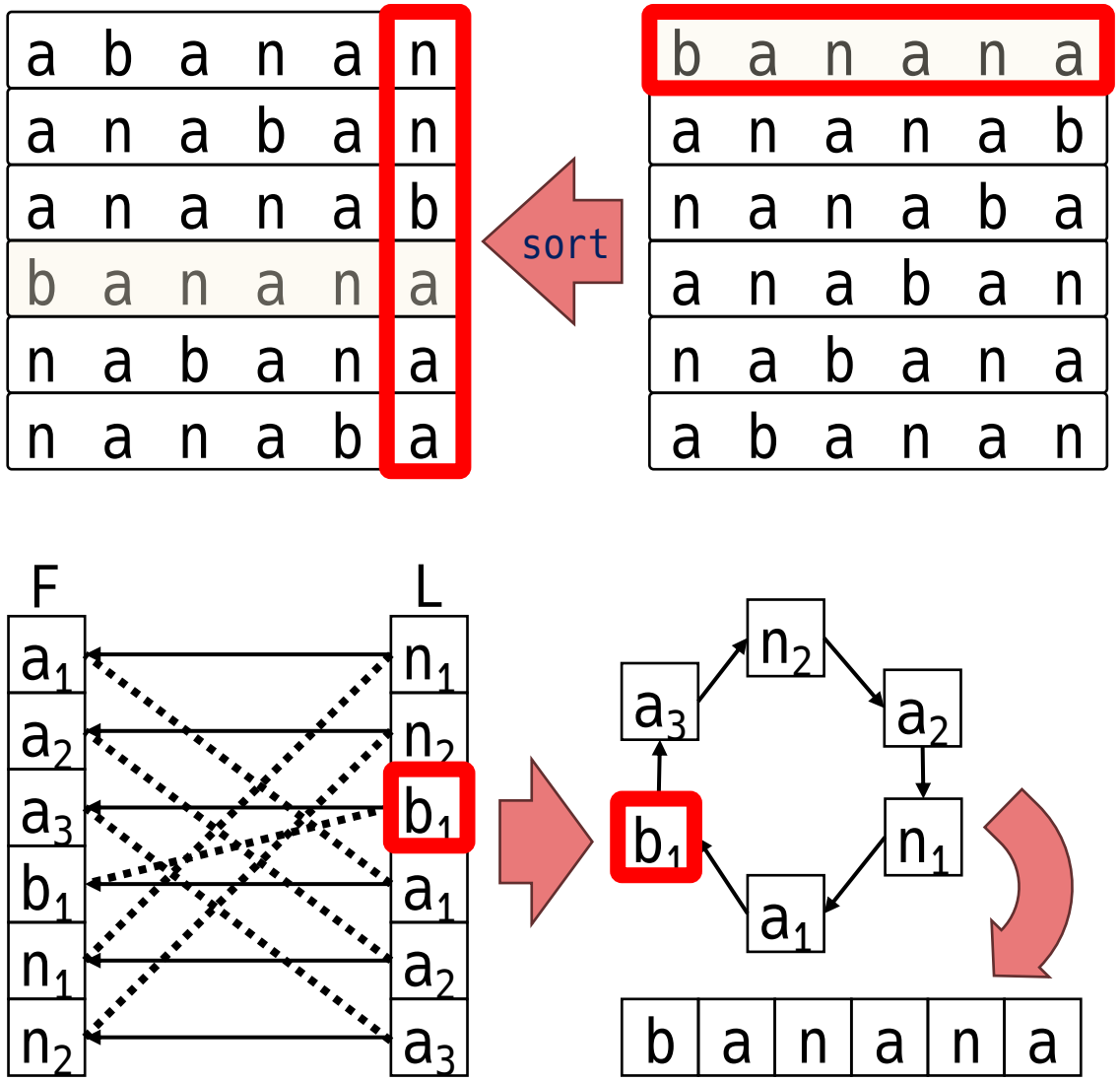
Institute of SCIENCE TOKYO

# Burrows-Wheeler Transform and Dictionary Compression

# Burrows-Wheeler Transform (BWT)

[Burrows&Wheeler 1994]

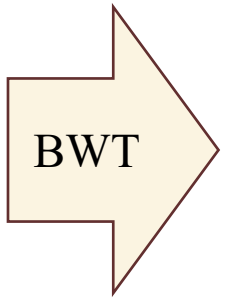
- A "reversible" transform
  - The last symbol of all rotations in lex. order of the rotations
    - BWT(banana) = nnbaaa
  - The inverse transform: via LF-mapping induced by a stable sorting of the symbols
  - BWT is actually a mapping from a *cyclic string* to a string, so we need more information (e.g. the starting position or use end-of-string marker $) to get the original string.
    - $BWT^{-1}$(nnbaaa, 3) = banana




Key component of FM-index and important bioinformatics tools.

# The “clustering effect” of BWT

```
On_31_October_2023,_the_Japanese_cabinet_decid
ed_to_amend_the_National_University_Corporatio
n_Act._The_former_National_University_Corporat
ion_Tokyo_Institute_of_Technology_was_renamed_
the_National_University_Corporation_Institute_
of_Science_Tokyo,_and_the_National_University_
Corporation_Tokyo_Medical_and_Dental_Universit
y_was_dissolved_at_the_same_time._After_being_
passed_by_the_National_Diet_of_Japan,_the_amen
dment_took_effect_on_1_October_2024._The_new_u
niversity's_headquarters_is_located_in_Meguro_
ku,_Tokyo,_where_the_former_Tokyo_Institute_of
_Technology_headquarters_was_located.
```




```
yuo3nodet4223___002_2_____________________.__
_____________nrrnn.yyyydlnoofeoneeere11fff..e,
rnnllllloel,drdetskeesydsosseteeetgse,yddd,ete
dtwysy,ceennnnntcsn__p__pJJuuwwwp_ccrrrrNNNNN_
ooa__iooneeeSAeOOeneeeenene_ie_naamhhhhhchshhht
tthrmhhTTdftvstmdM_MbrimmmDbbmmthttvvvvvvnnino
ooofe__AneoottttttTtTt__wccdccDt_betttttttttt_d
ttttsssssnnnnnno_ooooaaaaaaa__oooiaaaadrraoOo
oioooooooeeaeaeai_iuUUUUUhhIIIeeyyyyytrttll___
_lloTTTTTnns_iiiiiiiiitppppffCCCCaa_rrrrddeeee
eooooe_oouoooooeeeeeeeeeaaa'raiar_esrrrrrrsainnn
ceecannuuuaafrr_________aaaaaaaaasss_cc_iiiiii
iiikqq_gtttliiiiie____tttttgbtgkkkkk
```

Text from: https://en.wikipedia.org/wiki/Institute_of_Science_Tokyo

BWT(T) makes the string “easier to compress”. (used in bzip2 compressor)

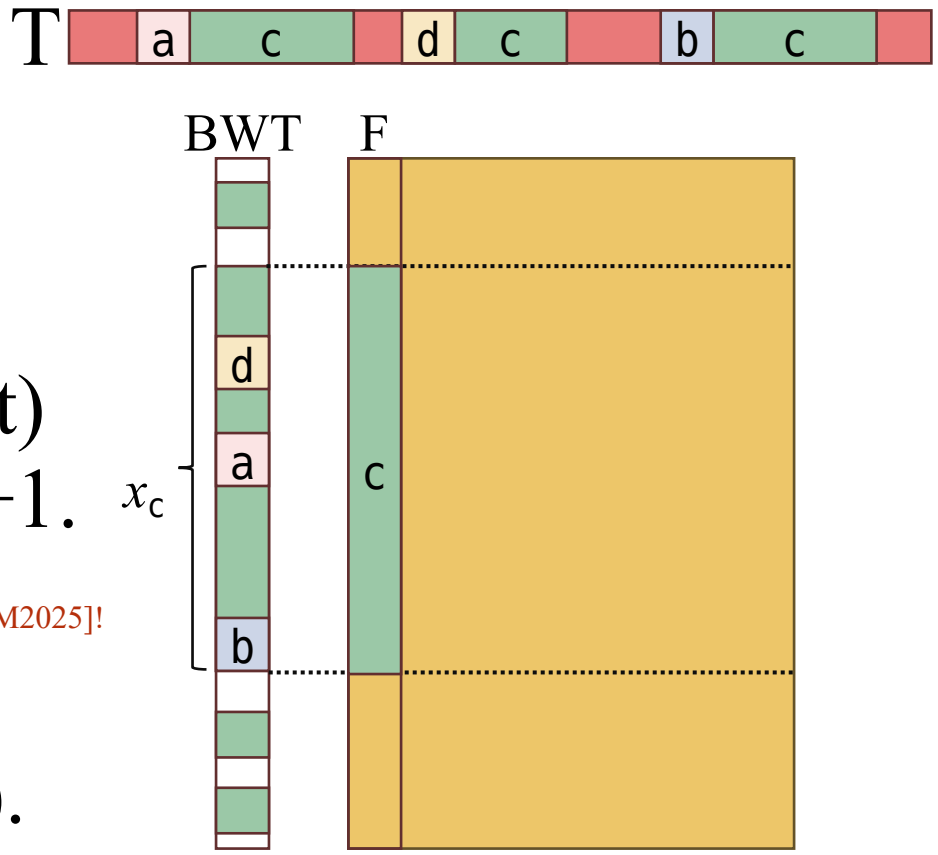
- Rotations with similar context are brought closer together.
- For any string T, $\rho(\text{BWT}(T)) \leq 2 \cdot \rho(T)$.
- There exist infinite string families s.t. $\rho(T) = \Theta(|T|)$ and $\rho(\text{BWT}(T)) = O(1)$.

# Asymmetry of the Clustering Effect of BWT (1)

For any string T, $\rho(\mathrm{BWT}(\mathrm{T})) \leq 2\rho(\mathrm{T})$. [Mantaci et al. 2017]
i.e., BWT increases the run-length encoding by a factor of at most 2.

Proof

- Let $\rho_c(\mathrm{T})$ denote # of runs of $c \in \Sigma$ in T.
- Consider substring $x_c$ of BWT corresponding to rotations starting with $c$.
- Since $c$ is preceded either by $c$, or (at most) $\rho_c(\mathrm{T})$ other non-$c$ symbols, $\rho(x) \leq 2\rho_c(\mathrm{T})+1$.
- However, equality cannot hold. See also [Bannai et al. CPM2025]!
- Therefore, $\rho(x_c) \leq 2\rho_c(\mathrm{T})$, and $\rho(\mathrm{BWT}(\mathrm{T})) \leq \sum_c \rho(x_c) \leq \sum_c 2\rho_c(\mathrm{T}) = 2\rho(\mathrm{T})$.



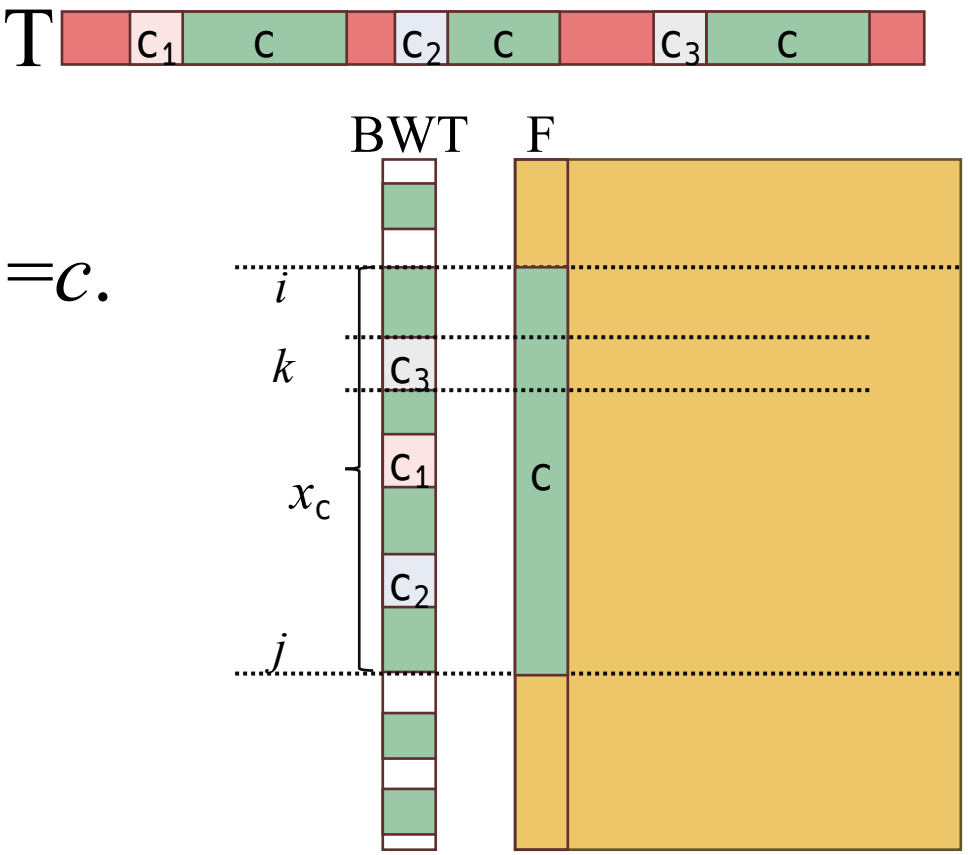
# Asymmetry of the Clustering Effect of BWT (1-2)

$\rho(x) \leq 2\rho_c(\mathrm{T})+1$. However, equality cannot hold.

$\rho(x) = 2\rho_c(\mathrm{T})+1$ implies that in $x$, any non-$c$ symbol is preceded and succeeded by $c$.

Then,

- There must exist $k \in [i-1..j-1]$ s.t. $|\mathrm{BWT}[1..k]|_c = |F[1..k]|_c$ and $\mathrm{BWT}[k+1]=c$.
  - This implies that $\mathrm{LF}[k+1] = k+1$, and the LF mapping on the BWT does not form a single cycle.




$|w|_c = |\{\, i : w[i] = c \,\}|$

# Asymmetry of the Clustering Effect of BWT (2)

For any Fibonacci word $F_k$, $\rho(F_k) = \Theta(n)$ and $\rho(\mathrm{BWT}(F_k)) = 2$.

Proof

$F_k$ does not contain repetitions with exponent 4, so $\rho(F_k) = \Theta(n)$.

It is known that $\rho(\mathrm{BWT}(T)) = 2$ for any standard Sturmian word T.

# Run-Length Compressed BWT (RLBWT)

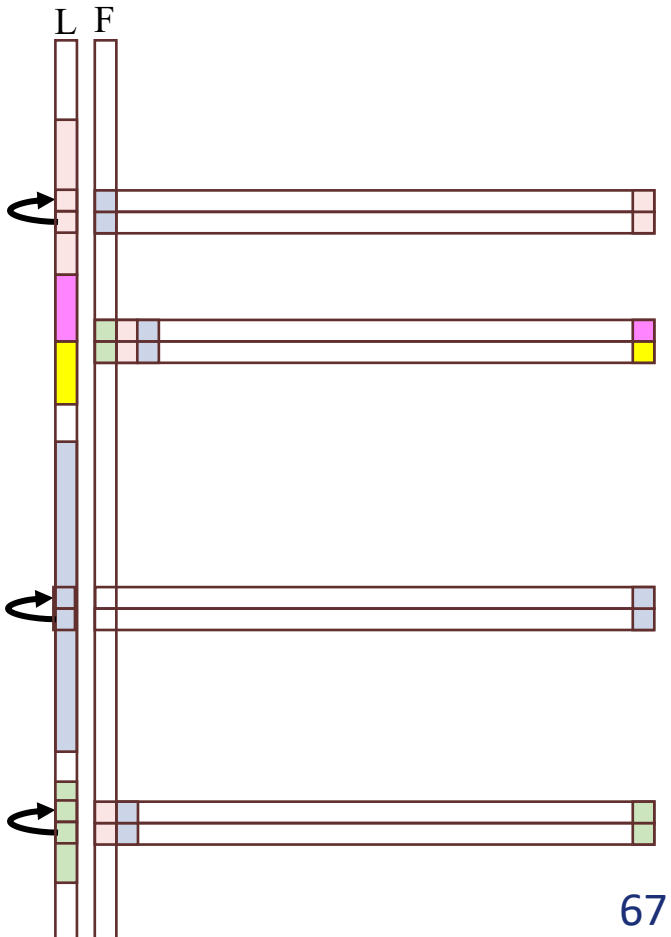
… but why can RLBWT be considered as a dictionary compressor?

RLBWT induces an O($r$) size BMS. For any string T, $b(T) \leq 2r(T)+1$.

Proof:

- In each run of BWT(T), let each symbol (except first in run) reference the preceding symbol and consider which positions can be grouped into the same phrase.
- Let $T_i = T[i..n]T[1..i-1]$ denote $i$th rotation of T.
- If $T_j$ is lex. preceding rot. of $T_i$ and $T[j-1]=T[i-1]$
  - $T_{j-1}$ is lex. preceding rot. of $T_{i-1}$
  - If $T[j] = T[i]$, the offset of the references are equal: $j - i = (j - 1) - (i - 1)$
- This continues until either:
  - $T[j-k] \neq T[i-k]$ ➔ $T_{j-k+1}, T_{i-k+1}$ corresponds to a run boundary or,
  - $T[j-k+1] = 1$ or $T[i-k+1] = 1$

➔ at most $(r-1)+2$ phrases + $r$ ground phrases = $2r+1$

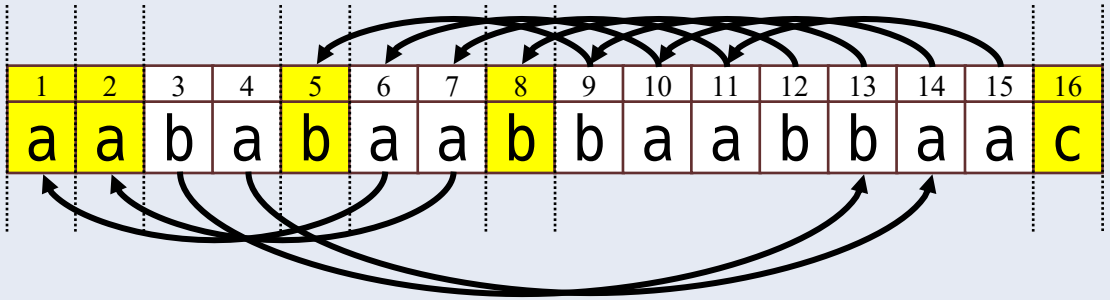


# Run-Length Compressed BWT (RLBWT)

… but why can RLBWT be considered as a dictionary compressor?

RLBWT induces an O($r$) size BMS. For any string T, $b(T) \leq 2r(T)+1$.

| SA[$i$] – 1 | BWT | SA[$i$] | |
|---|---|---|---|
| 16 | c | 1 | aababaabbaabbaac |
| 5 | b | 6 | aabbaabbaacaabab |
| 9 | b | 10 | aabbaacaababaabb |
| 13 | b | 14 | aacaababaabbaabb |
| 3 | b | 4 | abaabbaabbaacaab |
| 1 | a | 2 | ababaabbaabbaaca |
| 6 | a | 7 | abbaabbaacaababa |
| 10 | a | 11 | abbaacaababaabba |
| 14 | a | 15 | acaababaabbaabba |
| 4 | a | 5 | baabbaabbaacaaba |
| 8 | b | 9 | baabbaacaababaab |
| 12 | b | 13 | baacaababaabbaab |
| 2 | a | 3 | babaabbaabbaacaa |
| 7 | a | 8 | bbaabbaacaababaa |
| 11 | a | 12 | bbaacaababaabbaa |
| 15 | a | 16 | caababaabbaabbaa |




$r = 5$ ➔ BMS of size 8.

# Bounding $r$ in terms of $\delta$

[Kempa & Kociumaka 2020]

For any string T, $r(T) = O(\delta(T) \log^2 n)$

Proof

Consider run boundaries with LCP values $\ell \in [k, 2k)$ for $k = 2^0, \ldots, 2^{\lfloor \log n \rfloor}$. If, for each $\ell$, we can bound the number of such run boundaries by $O(\delta(T) \log n)$, the claim follows.

(*cf.*: irreducible LCP values)

$S_{3k}$ : set of length-$3k$ substrings of T (in cyclic sense)

$S_k$ : set of length-$k$ substrings of T (in cyclic sense)

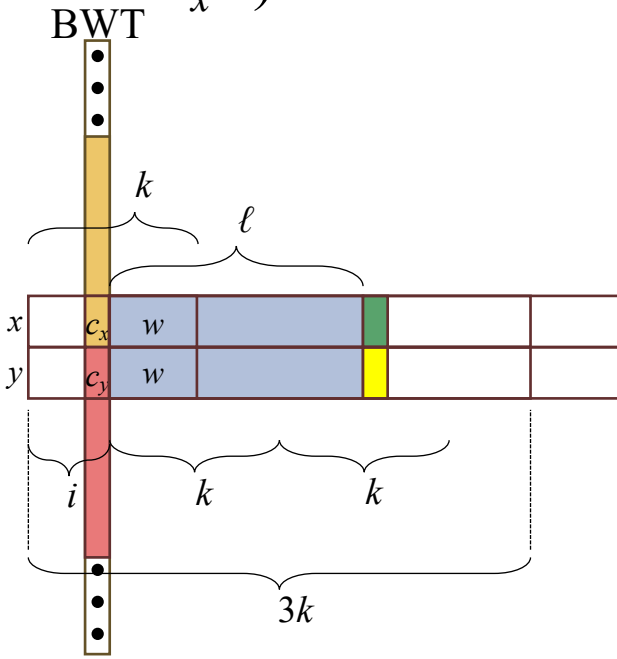
Note that $|S_{3k}|, |S_k| = min\,(n, O(\delta(T)\, k))$.

# Bounding $r$ in terms of $\delta$

[Kempa & Kociumaka 2020]

- Assign cost of $k$ to each run-boundary w/ LCP value $\ell \in [k,2k)$.
- For $1 \le i \le k$, the $i$th unit cost is charged to position $i$ of one of $x, y \in S_{3k}$ s.t. $x[i]$ and $y[i]$ correspond to run-boundary.
  - $x$ if #strings in $S_k$ that end with $c_x w \le$ #strings in $S_k$ that end with $c_y w$
  - $y$ otherwise (#strings in $S_k$ that end with $c_y w <$ #strings in $S_k$ that end with $c_x w$)

1. Any given string $s \in S_{3k}$ can be charged at most $\log n$ times.
   - Charged position $i$ in $x$ implies that #strings in $S_k$ that end with $w \ge 2 \cdot$(#strings in $S_k$ that end with $c_s w$)
2. Any given position $i$ in $s \in S_{3k}$ can be charged at most twice.
   - All rotations of T with same length-$3k$ substring charged at same position $1 \le i \le k$ lie in an interval w/ LCP value $\ge 2k$
     ➔ Can be involved in LCP value $\ell < 2k$ only at beginning or end of this interval.




➔ Total cost $\le |S_{3k}| \cdot 2 \cdot \log n = O(\delta(T) k \log n)$.

➔ # of run-boundaries w/ LCP $\ell \in [k,2k) \le$ (Total cost) / $k = O(\delta(T) \log n)$.

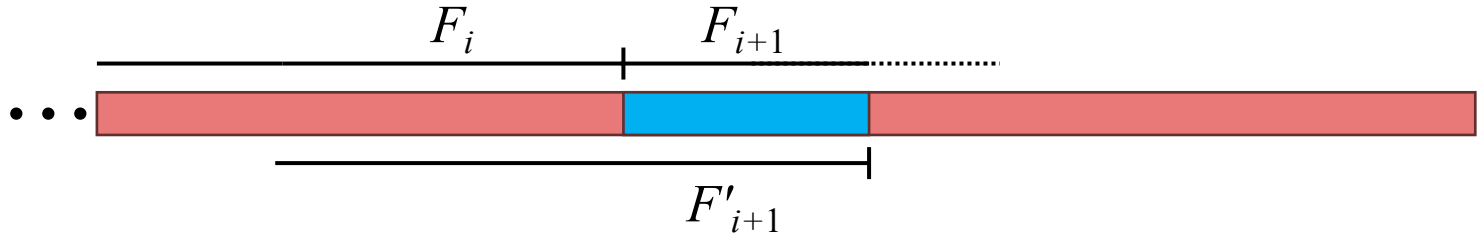
# Lexicographic Parsing $v$

A lexicographic parsing is a BMS where references always point to lexicographically smaller suffix positions.

For any string T, the size $v$(T) of the greedy lexicographic parse is optimal.

Proof

Essentially the same as for $z$(T). The suffix of $F'_{i+1}$ that starts after $F_i$ has (a lexicographically) previous occurrence.



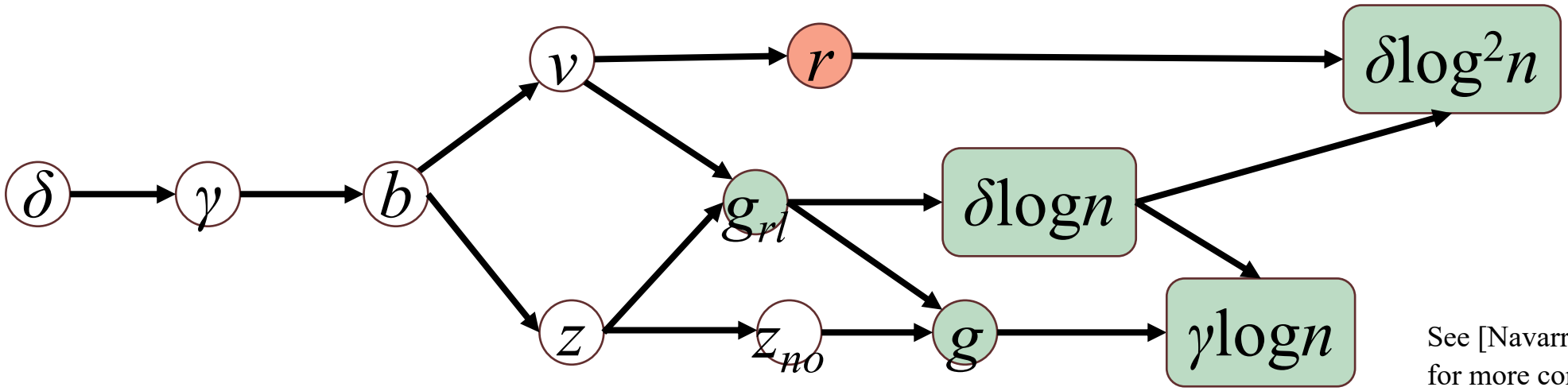

$v(\mathrm{T}) \leq \mathrm{O}(r(\mathrm{T}))$

# Other known relations

- $v \leq g_{rl}$ [Navarro *et al.* 2021]
  - consider partial parse tree keeping the lex. smallest occurrence.
- Separation between $r$ and $v$:
  - $r = \Omega(v \log n)$ on binary de Bruijn sequences
    - $r = \Theta(n)$: string of $\{01,10\}^{n/2}$
    - $v = O(n / \log n)$ (actually, holds for all parsings into distinct substrings)

# Conclusions

Introduced several families of dictionary compressors and related repetitiveness measures and their relations.




See [Navarro 2021] for more complete map.

- Grammars (SLPs) can be balanced to support fast access. Also supports indexing.
- RLBWT supports indexing but unknown if fast access is possible.