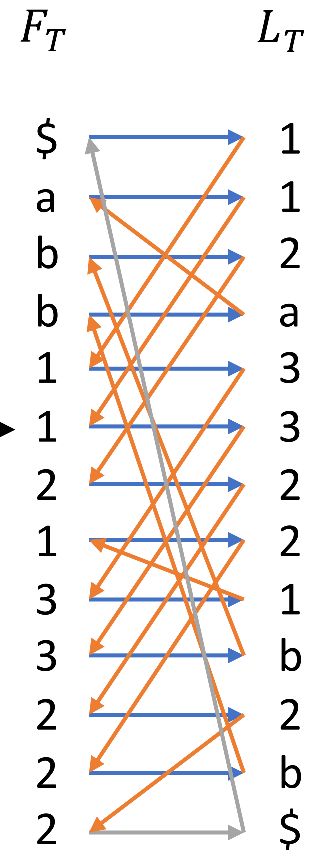
Extending the Parameterized Burrows-Wheeler Transform

Eric M. Osterkamp (University of Münster)

Dominik Köppl (University of Yamanashi)

T = ACAbCAabABBA$



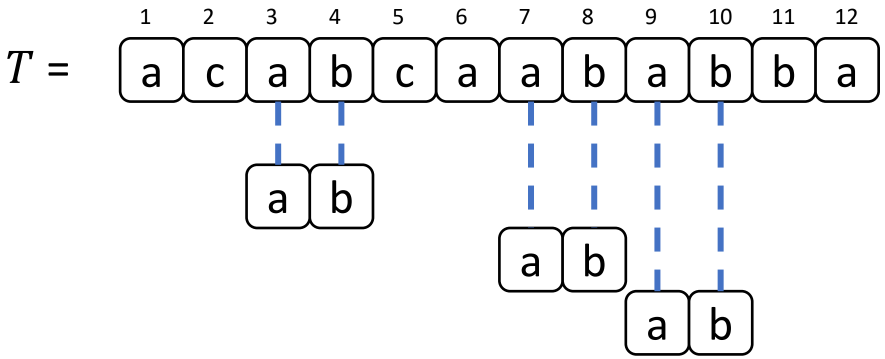
based on the slides for the final presentation of the Bachelor thesis of the first author

DCC ‘24

Pattern Matching

- alphabet Σ
- text $T \in \Sigma^*$, pattern $P \in \Sigma^*$
- occurrence of P in T :
  substring of T that equals P
- PM: count all occurrences of P in T
  write as $T.count(P)$
- goal: index T for efficient PM

- $\Sigma =$ {a,b,c}
- $T =$ acabcaababba
- $P =$ ab
- occurrences of P in T
  at positions 3, 7 and 9
- $T.count(P) = 3$



Parameterized Strings

- alphabet Σ_s of static symbols (s-symbols)
- alphabet Σ_p of parameterized symbols (p-symbols)
- $\Sigma_s \cap \Sigma_p = \emptyset$
- string over $\Sigma := \Sigma_s \cup \Sigma_p$ is a parameterized string (p-string)
- character in Σ called symbol, σ := |Σ| size of alphabet

example

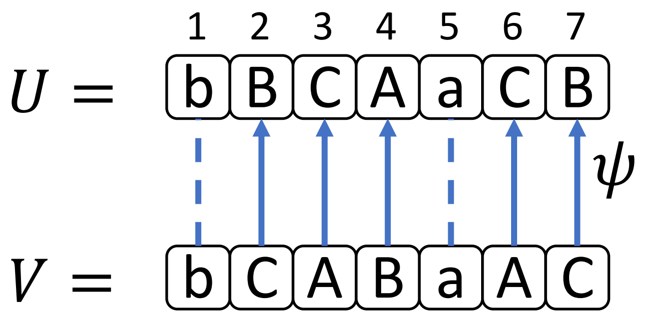
- $\Sigma_s = \{a, b\}$, $\Sigma_p = \{A, B, C\}$
- $T = \text{ACAbCAabABBA}$

Parameterized Matching (p-Matching)

- U, V p-strings
- U p-matches V $:\Leftrightarrow$ if $|U| = |V|$ and $\exists$ a bijection $\psi: \Sigma_p \to \Sigma_p$ with
  - $U[i] = V[i]$ if $V[i] \in \Sigma_s$
  - $U[i] = \psi\,(V[i])$ otherwise
- write $U =_p V$ iff U and V p-match

example

- U = bBCAaCB
- V = bCABaAC
- ψ(A) = C, ψ(B) = A, ψ(C) = B

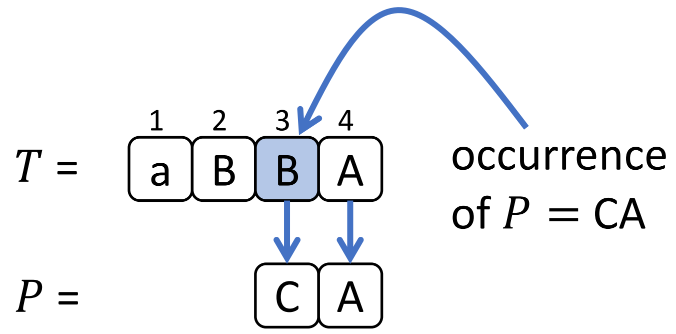


$\Rightarrow U =_p V$

Parameterized Pattern Matching (PPM)

[Baker '93]

- T : text p-string
- P : pattern p-string
- occurrence of P in T : substring of T that p-matches P
- PPM: count all occurrences of P in T, written as $T.count(P)$
- goal: index text T for efficient PPM



Indexes for PPM

data structure	time for PPM	reference
suffix tree	O(m log σ)	[Baker ‘93]
suffix array	O(m + log n)	[Deguchi + ‘08]
position heap	O(m log σ + m σ_p)	[Diptarama+ ‘17]
suffix tray	O(m + log σ)	[Fujisato+ ‘21]
DAWG	O(m log σ)	[Nakashima+ ‘22]

- σ :=$|\Sigma|$ alphabet size
- σ_p := $|\Sigma_p|$
- n := $|T|$, text size
- m : $|P|$, pattern length

All data structures need O(n log n) bits

PPM in small memory

- parameterized Burrows-Wheeler transform (pBWT) [Ganguly+ ’17]
  - n lg σ + O(n) bits
  - computes $T.count(P)$ in O(m log σ) time
- simplified pBWT [Kim, Cho ‘21]
  - 2n lg σ + O(n) bits
  - same time complexities

- both approaches use space linear in the number of bits of the input!

Applications for PPM

many use cases

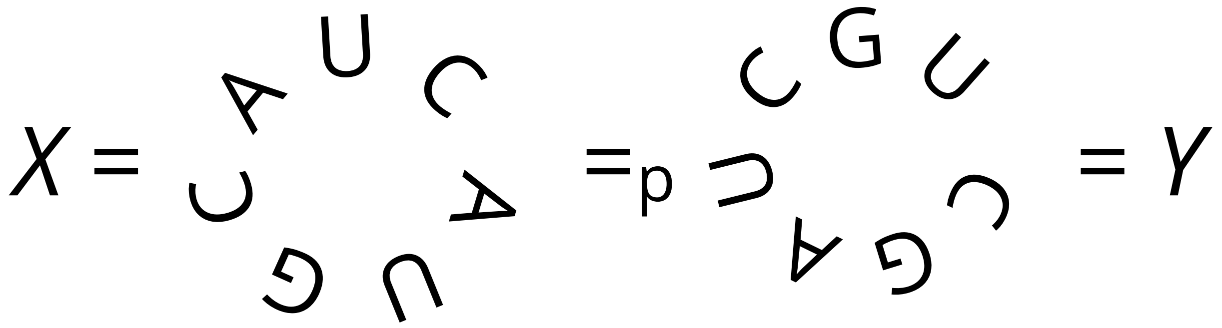
- software maintenance [Baker ‘97]
- plagiarism detection
- analyzing genetic data [Shibuya ’04]

RNA matching

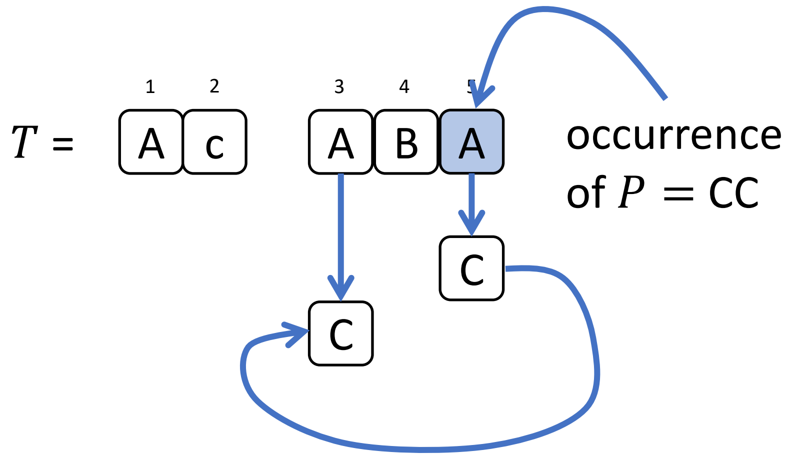
[Shibuya' 04]

- matching RNA base pair
- X = AUGCAUC
- Y = CGAUCGU
- $\psi(X) = Y$
- but some RNA structures are cyclic, so there is a need for cyclic pattern matching ⇒ circular parameterized pattern matching (CPPM)

ψ :
$A \mapsto C$,
$U \mapsto G$,
$C \mapsto U$,
$G \mapsto A$



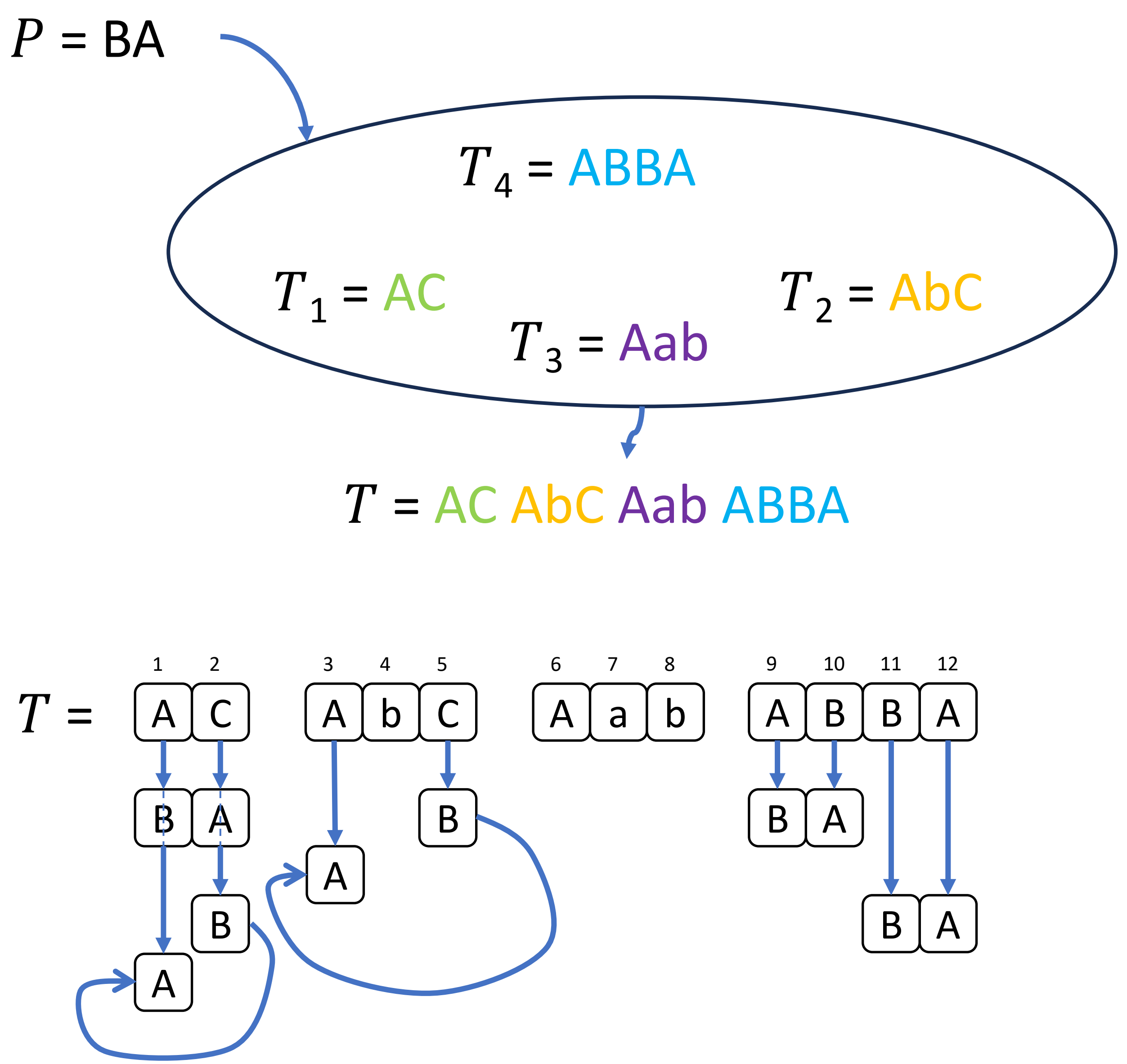
Circular PPM (CPPM)



- text p-strings $T = \{T_1, \ldots, T_d\}$
- pattern p-string P
- occurrence of P in T refers to the starting position of a substring of $T_1, \ldots, T_d$ that p-matches P
- all text p-strings are viewed circularly
- CPPM: count all occurrences of P in T
- goal: index texts $T_1, \ldots, T_d$ for efficient CCPM

Example: CPPM

- $T = \{\text{AC, AbC, Aab, ABBA}\}$
- $P = \text{BA}$
- occurrences of P in T at positions 1, 2, 4, 9 and 11
- $T.count(P) = 5$



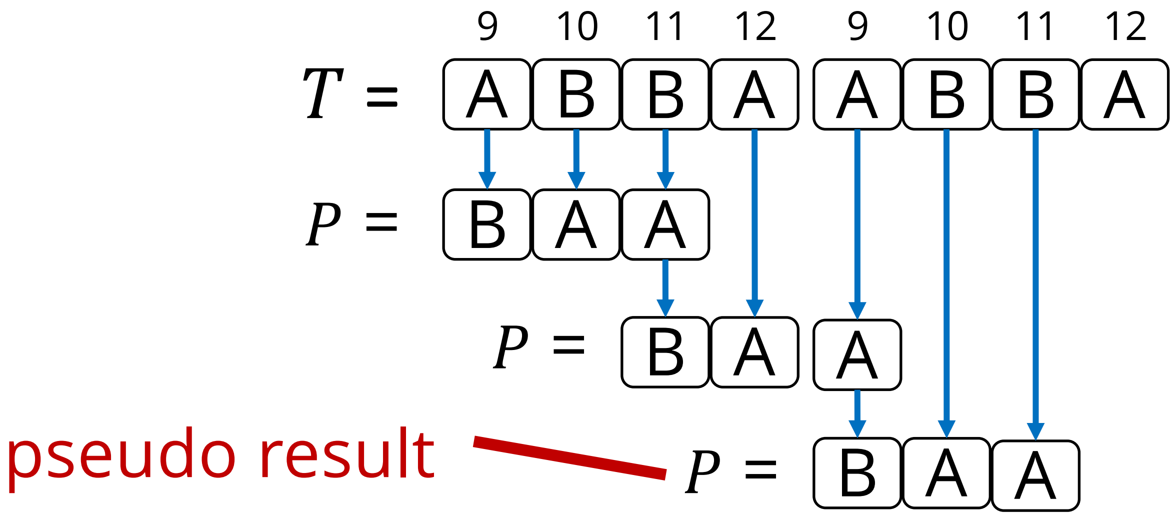
Simple idea for CPPM

- general naive approach for matching P in T circularly:
- perform classic matching of P in $T \cdot T$
- may generate pseudo results in the second part
- discard pseudo results in postprocessing

Example:

T = ABBA

P = BAA



From pBWT to epBWT

define epBWT based on two encodings

- prev-encoding $\langle V \rangle$ [Baker '93]
- Hashimoto-encoding «V», [Hashimoto+ '22]

motivation is explained with a review of pBWT

BWT (Burrows-Wheeler transform):

- last character of all cyclic rotations sorted in lexicographic order
- $T.count(P)$ via length of reported range of backward search
- how to use that technique with p-matching?

pBWT

review of the simplified pBWT [Kim, Cho '21] for PPM

Comparing p-Strings

- consider conjugates of $V = \text{AABB}$

A A B B	sort lexicographically →	A A B B
B A A B		A B B A
B B A A		B A A B
A B B A		B B A A

- $\text{AABB} =_p \text{BBAA} \neq_p \text{ABBA} =_p \text{BAAB}$, but $\text{AABB} < \text{ABBA} < \text{BAAB} < \text{BBAA}$
- cannot use original p-string to sort or p-match!

prev-Encoding

given p-string V, compute prev-encoding $\langle V \rangle$ of V as follows:

- replace leftmost occurrence of any p-symbol in V by ∞
- replace each other by distance to its previous occurrence

for every p-string U: $\langle \mathrm{V} \rangle = \langle U \rangle \Leftrightarrow V =_p U$ [Baker '93]

T	$=$	A	C	A	b	C	A	a	b	A	B	B	A
$\langle T \rangle$	$=$	∞	∞	2	b	3	3	a	b	3	∞	1	3

- but unstable under rotation

$\langle T \rangle[2..]\langle T \rangle[1] =$	∞	2	b	3	3	a	b	3	∞	1	3	∞
$T[2..]T[1] =$	C	A	b	C	A	a	b	A	B	B	A	A
$\langle T[2..]T[1] \rangle =$	∞	∞	b	3	3	a	b	3	∞	1	3	1

Hashimoto-Encoding [Hashimoto+ '22]

- view p-string V circularly and replace each occurrence of a p-symbol in V by the number of distinct p-symbols until its next occurrence
- write «V» for the Hashimoto-encoding of V

$T =$	A	C	A	b	C	A	a	b	A	B	B	A
«T» $=$	2	2	2	b	3	1	a	b	2	1	3	1

- for every p-string U: «V» $=$ «U» $\Leftrightarrow V =_p U$ [Hashimoto+ '22]
- encoding is commutative with rotation!

«T»$[2..]$«T»$[1] =$	2	2	b	3	1	a	b	2	1	3	1	2
$T[2..]T[1] =$	C	A	b	C	A	a	b	A	B	B	A	A
«$T[2..]T[1]$» $=$	2	2	b	3	1	a	b	2	1	3	1	2

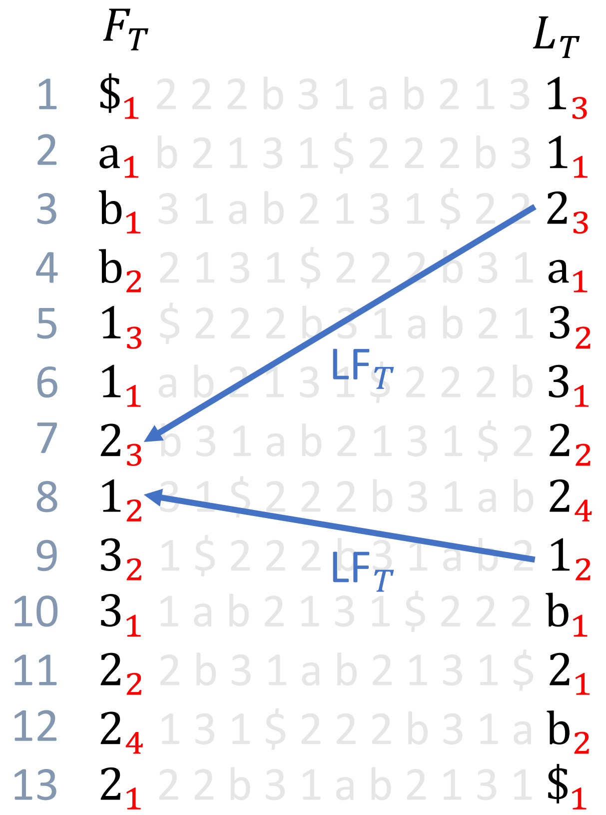
Parameterized BWT (pBWT)

- text T = ACAbCAabABBA\$
- «T» = 222b31ab2131\$
- pBWT(T) = (F_T, L_T)
- first and last symbols of Hashimoto-encoded conjugates sorted by their prev-encodings
- similar entries of both strings are sorted by succeeding context! [Iseri+ '23]

	F_T												L_T
1	\$	2	2	2	b	3	1	a	b	2	1	3	1
2	a	b	2	1	3	1	\$	2	2	2	b	3	1
3	b	3	1	a	b	2	1	3	1	\$	2	2	2
4	b	2	1	3	1	\$	2	2	2	b	3	1	a
5	1	\$	2	2	2	b	3	1	a	b	2	1	3
6	1	a	b	2	1	3	1	\$	2	2	2	b	3
7	2	b	3	1	a	b	2	1	3	1	\$	2	2
8	1	3	1	\$	2	2	2	b	3	1	a	b	2
9	3	1	\$	2	2	2	b	3	1	a	b	2	1
10	3	1	a	b	2	1	3	1	\$	2	2	2	b
11	2	2	b	3	1	a	b	2	1	3	1	\$	2
12	2	1	3	1	\$	2	2	2	b	3	1	a	b
13	2	2	2	b	3	1	a	b	2	1	3	1	\$

LF (Mapping) Property

- text T = ACAbCAabABBA$
- «T» $= 2_1 2_2 2_3 \mathrm{b}_1 3_1 1_1 \mathrm{a}_1 \mathrm{b}_2 2_4 1_2 3_2 1_3 \$_1$
- first column $F_T = \$_1 \mathrm{a}_1 \mathrm{b}_1 \mathrm{b}_2 1_3 1_1 2_3 1_2 3_2 3_1 2_2 2_4 2_1$
- last column $L_T \;= 1_3 1_1 2_3 \mathrm{a}_1 3_2 3_1 2_2 2_4 1_2 \mathrm{b}_1 2_1 \mathrm{b}_2 \$_1$
- define permutation LF_T by mapping from ith occurrence of a symbol $x \in \Sigma_s \cup [1..|\Sigma_p|]$ in L_T to ith occurrence of x in F_T
- LF property: maps x_k of L_T to x_k of F_T!



epBWT

from pBWT to epBWT

ω-Order

idea: use the infinite iteration of a conjugate as key for sorting

- V^{ω}: infinite iteration of V
- root(V) := primitive root of V (V = ababab ⇒ root(V) = ab)

- V, U : finite strings
- $V =_{\omega} U$:⇔ root(V) = root(U)
- $V \prec_{\omega} U$:⇔ $\exists i$: $V^{\omega}[..i] = U^{\omega}[..i] \wedge V^{\omega}[i+1] < U^{\omega}[i+1]$

Extending the ω-Order to p-Strings

- V, U : finite p-strings
- $V =_\omega U$:⇔ root(«V») = root(«U»)
- $V \prec_\omega U$:⇔ $\exists i$: $\langle V^\omega \rangle[..i] = \langle U^\omega \rangle[..i] \wedge \langle V^\omega \rangle[i+1] < \langle U^\omega \rangle[i+1]$

- extended ω-order already used when defining the pBWT!
- coincides with prev-order for p-strings of the same length

Example: ω-Order on p-Strings

- T_1 = AB, T_2 = ABA, T_3 = ABAB
- $\langle T_1 \rangle < \langle T_2 \rangle < \langle T_3 \rangle$

- $T_1^\omega[..8]$ = A B A B A B A B
- $T_2^\omega[..8]$ = A B A A B A A B
- $T_3^\omega[..8]$ = A B A B A B A B
- «T_1» = 2 2
- «T_3» = 2 2 2 2

- $\langle T_1 \rangle = \infty\ \infty$
- $\langle T_2 \rangle = \infty\ \infty\ 2$
- $\langle T_3 \rangle = \infty\ \infty\ 2\ \ 2$
- $\langle T_1^\omega \rangle[..8] = \infty\ \infty$ 2 2 2 2 2 2
- $\langle T_2^\omega \rangle[..8] = \infty\ \infty$ 2 1 3 2 1 3
- $\langle T_3^\omega \rangle[..8] = \infty\ \infty$ 2 2 2 2 2 2
- $T_2 \prec_\omega T_1 =_\omega T_3$

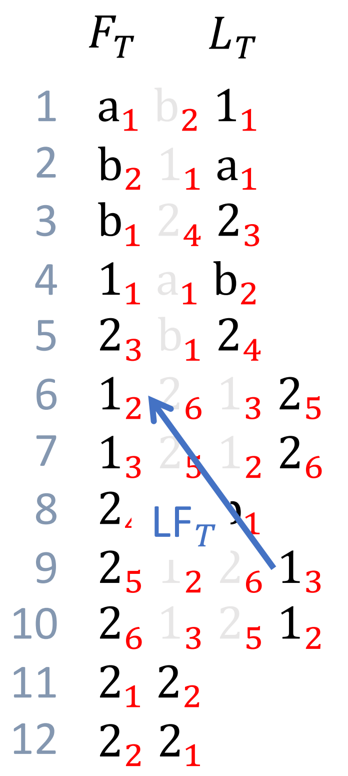
Extended pBWT (epBWT)

- sort conjugates by ω-order
  tie-break:
  - first by index of text string,
  - second by text position
- $T = \{$AC, AbC, Aab, ABBA$\}$
- epBWT$(T) = (F_T, L_T)$
- first and last symbols of Hashimoto-encoded conjugates sorted by their prev-encodings in ω-order

Conjugates	sort by ω-order →	F_T	L_T
2 2 2 2 2 2 2 2	a b 1 a b 1 a b	a	1
2 2 2 2 2 2 2 2	b 1 a b 1 a b 1	b	a
2 b 2 2 b 2 2 b	b 2 2 b 2 2 b 2	b	2
b 2 2 b 2 2 b 2	1 a b 1 a b 1 a	1	b
2 2 b 2 2 b 2 2	2 b 2 2 b 2 2 b	2	2
1 a b 1 a b 1 a	1 2 1 2 1 2 1 2	1	2
a b 1 a b 1 a b	1 2 1 2 1 2 1 2	1	2
b 1 a b 1 a b 1	2 2 b 2 2 b 2 2	2	b
2 1 2 1 2 1 2 1	2 1 2 1 2 1 2 1	2	1
1 2 1 2 1 2 1 2	2 1 2 1 2 1 2 1	2	1
2 1 2 1 2 1 2 1	2 2 2 2 2 2 2 2	2	2
1 2 1 2 1 2 1 2	2 2 2 2 2 2 2 2	2	2

LF (Mapping) Property

- $T = \{$AC, AbC, Aab, ABBA$\}$
- $\{«T_1», «T_2», «T_3», «T_4»\} = \{2_1 2_2, 2_3 b_1 2_4, 1_1 a_1 b_2, 2_5 1_2 2_6 1_3\}$
- first column $F_T = a_1 b_2 b_1 1_1 2_3 1_2 1_3 2_4 2_5 2_6 2_1 2_2$
- last column $L_T = 1_1 a_1 2_3 b_2 2_4 2_5 2_6 b_1 1_3 1_2 2_2 2_1$
- define permutation LF_T by mapping from ith occurrence of a symbol $x \in \Sigma_s \cup [1..|\Sigma_p|]$ in L_T to ith occurrence of x in F_T
- only maps x_k of L_T to x_k of F_T if Hashimoto-encoded texts are primitive! («AC» and «ABBA» are not primitive!)
- remedy : build epBWT on the Hashimoto-encoded roots!



Summary

epBWT is a CPPM index for a set of p-strings

- builds upon pBWT of [Kim, Cho ’21] and eBWT of [Mantaci+ ‘07]
- uses $2n \lg \sigma + O(n)$ bits of space

partially in the paper (full version will follow):

- $T.count(P)$ in $\mathrm{O}(m \lg \sigma)$ time for CPPM
- reconstruction of input up to p-matching equivalence
- construction of index in $O\left(n \frac{\lg^2 n}{\lg \lg n}\right)$ time with $O(n \lg n)$ bits of space
- applications to other matchings such as Cartesian-Tree matching