Prague Stringology Conference 2014

Computing Abelian Covers and Runs

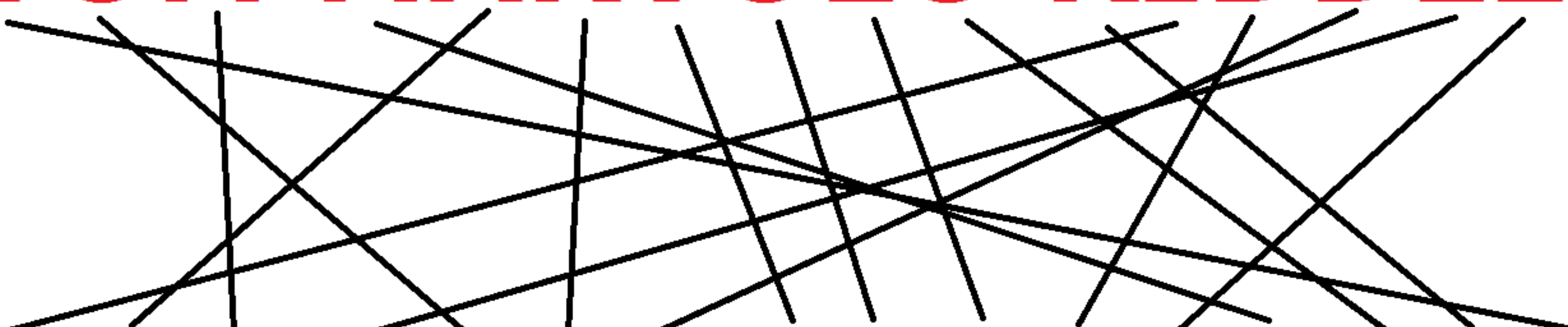
Shohei Matsuda, Shunsuke Inenaga,
Hideo Bannai, and Masayuki Takeda
Kyushu University

Background

The study of *Abelian equivalence* of strings dates back to at least the early 60's, as seen in the paper by Erdös.

Two strings u, v are said to be *Abelian equivalent* if u is a permutation of the characters appearing in v.

TOM MARVOLO RIDDLE



I AM LORD VOLDEMORT

[J.K.Rowling,1997]

Background

The study of *Abelian equivalence* of strings dates back to at least the early 60's, as seen in the paper by Erdös.

Two strings *u*, *v* are said to be *Abelian equivalent* if *u* is a permutation of the characters appearing in *v*.

Example

TOM MARVOLO RIDDLE and **I AM LORD VOLDEMORT** are *Abelian equivalent.*

Abelian equivalence of strings has attracted much attention and has been studied extensively in several contexts.

Our Contributions

✧ Two new regularities on strings with respect to *Abelian equivalence*,

- *Abelian covers* and
- *Abelian runs*

of strings, which are generalizations of

- covers [Apostolico et al., 1991] and
- runs [Kolpakov and Kucherov,1999]

of strings, respectively.

✧ Non-trivial algorithms to compute these new string regularities.

Parikh vector

$\Sigma = \{a_1,\ldots,a_m\}$: integer *alphabet*

$w \in \Sigma^*$: *string*

$P_w[k]$: num. of occurrences of k-th character in w

$P_w = \langle P_w[1], \ldots, P_w[m] \rangle$: Parikh vector of w

Example

$$w = \text{a a b a b}$$

$$P_w = \langle 3, 2 \rangle$$

Partial order on Parikh vectors

$1 \le k \le m,\ P_x[k] \ge P_y[k]$ and $|x| > |y| \Leftrightarrow P_x > P_y$

Example

$x = \text{a b a b b a a}$ $\quad y = \text{a a b a b}$

$P_x = \langle 4, 3 \rangle$ $\quad P_y = \langle 3, 2 \rangle$

$$P_x > P_y$$

Abelian equivalence

$P_x = P_y \Leftrightarrow$ x and y are *Abelian equivalent.*

Example

$$x = \mathrm{aabab},\ y = \mathrm{baaba}$$

$$P_x = P_y = \langle 3, 2 \rangle$$

x and y are *Abelian equivalent.*

Abelian covers

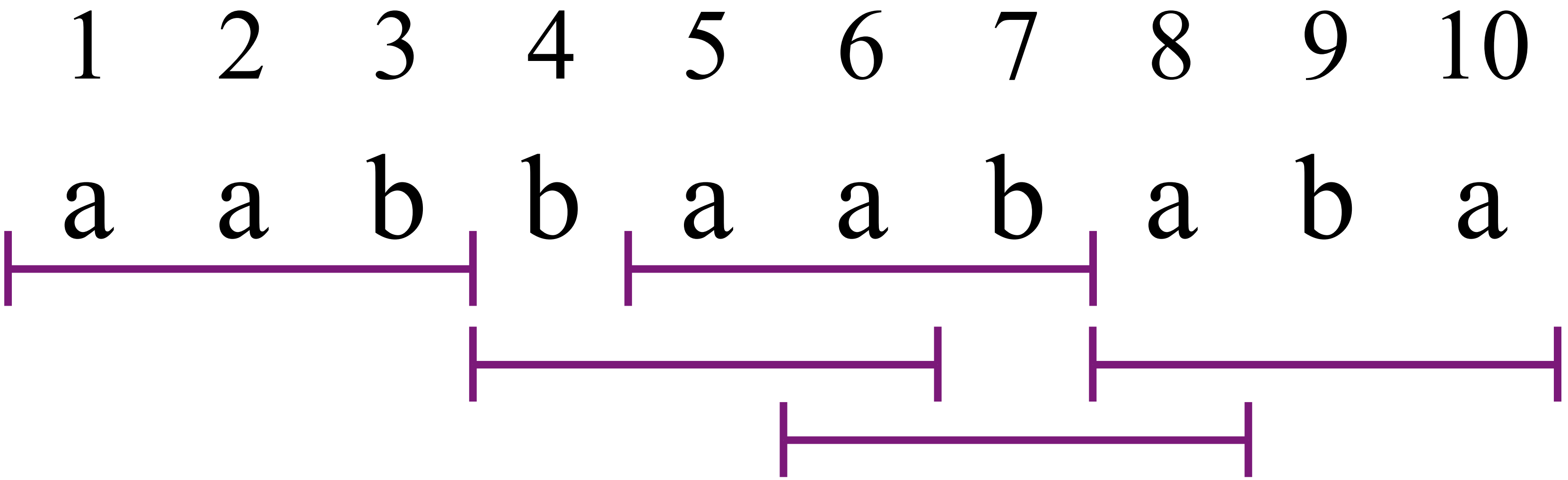
Definition

For a string w of length $n \geq 2$, a set $I = \{[b_k, e_k] : 1 \leq b_k \leq e_k \leq n, 1 \leq k \leq |I|\}$ of intervals is an *Abelian cover* of w, if for every $1 \leq k \leq |I|$,

- $[b_k, e_k] \neq [1, n]$,
- $\bigcup_{1 \leq k \leq |I|} [b_k, e_k] = [1, n]$, and
- $P_{w[b_1, e_1]} = P_{w[b_k, e_k]}$.

Abelian covers

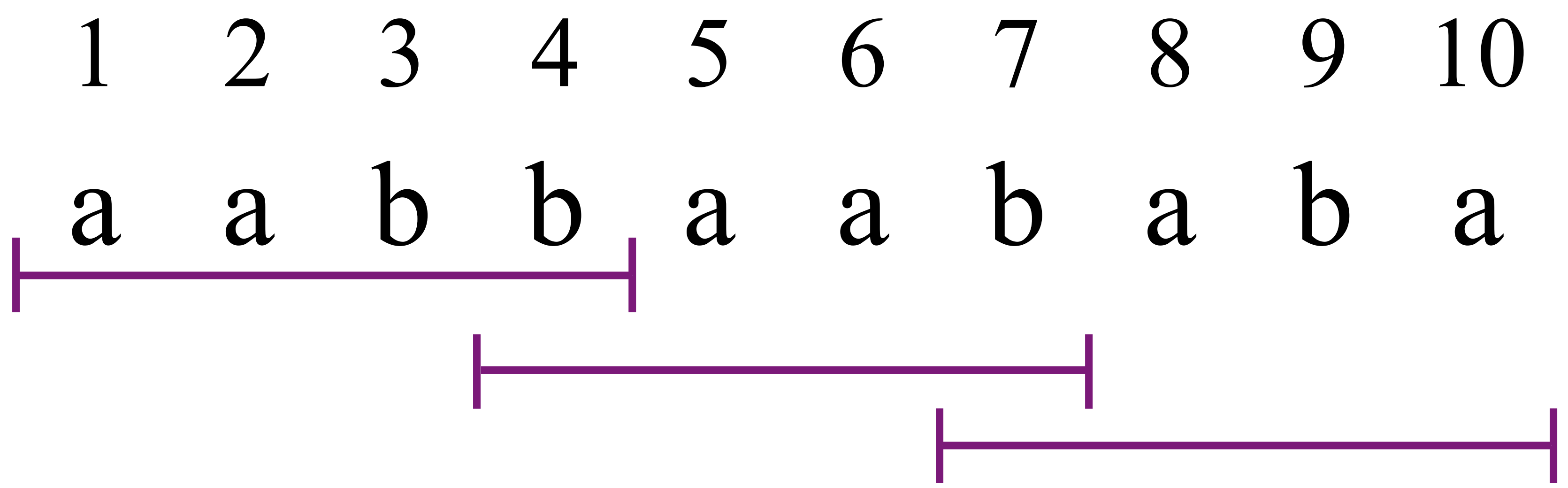
Example



Set {[1, 3], [4, 6], [5, 7], [6, 8], [8, 10]} of intervals is an *Abelian cover* of this string.

Abelian covers

Example



Set {[1, 4], [4, 7], [7, 10]} of intervals is also an *Abelian cover* of this string.

Abelian covers

Problem 1 (*Abelian cover* existence)

Given a string w, determine whether or not w has an *Abelian cover*.

Abelian covers

Lemma 1 (*Abelian covers*)

String w of length n has an *Abelian cover*
$\Leftrightarrow P_{w[1,\, i]} = P_{w[n-i+1,\, n]}$ for some $1 \leq i < n$.

Lemma 1 (*Abelian covers*)

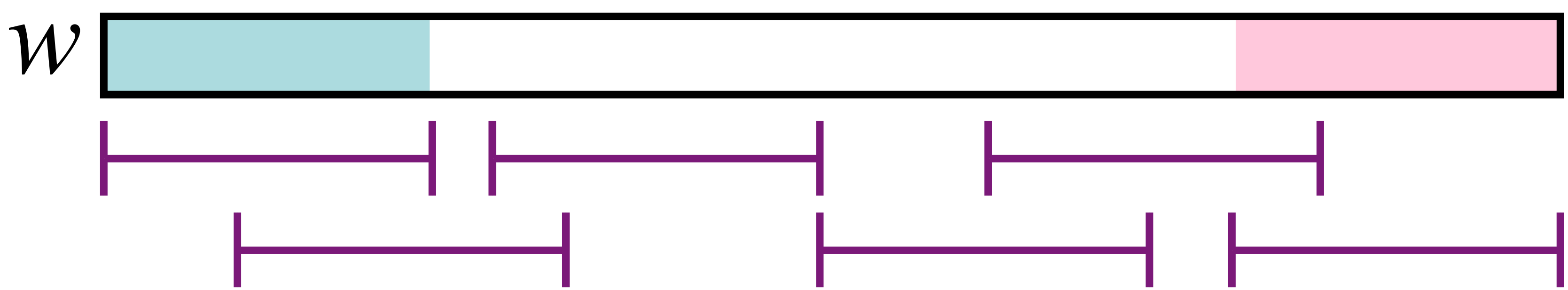
String w of length n has an *Abelian cover*

$\Leftrightarrow P_{w[1,\,i]} = P_{w[n-i+1,\,n]}$ for some $1 \le i < n$.

Proof (➔)

If w has an *Abelian cover* $\{[b_1, e_1], \ldots, [b_{|I|}, e_{|I|}]\}$,

then $P_{w[b_1,\,e_1]} = P_{w[b_{|I|},\,e_{|I|}]}$.



Lemma 1 (*Abelian covers*)

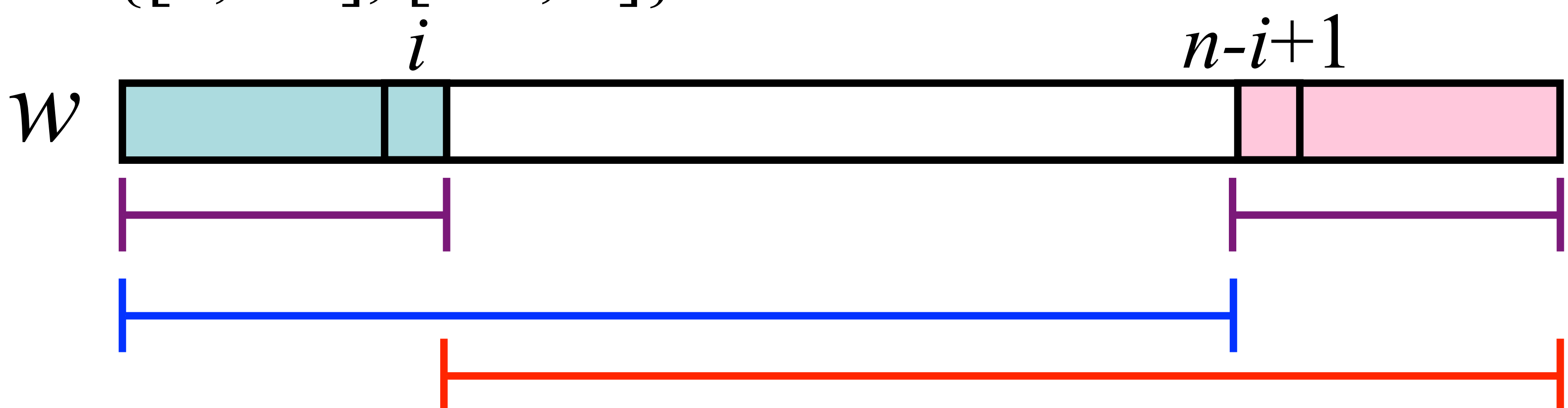
String w of length n has an *Abelian cover* $\Leftrightarrow P_{w[1,\, i]} = P_{w[n-i+1,\, n]}$ for some $1 \leq i < n$.

Proof (←)

If, for some $1 \leq i \leq n/2$, $P_{w[1,\, i]} = P_{w[n-i+1,\, n]}$,

then $P_{w[1,\, n-i]} = P_{w[i+1,\, n]}$ and

$I = \{[1, n-i], [i+1, n]\}$ is an *Abelian cover* of w.

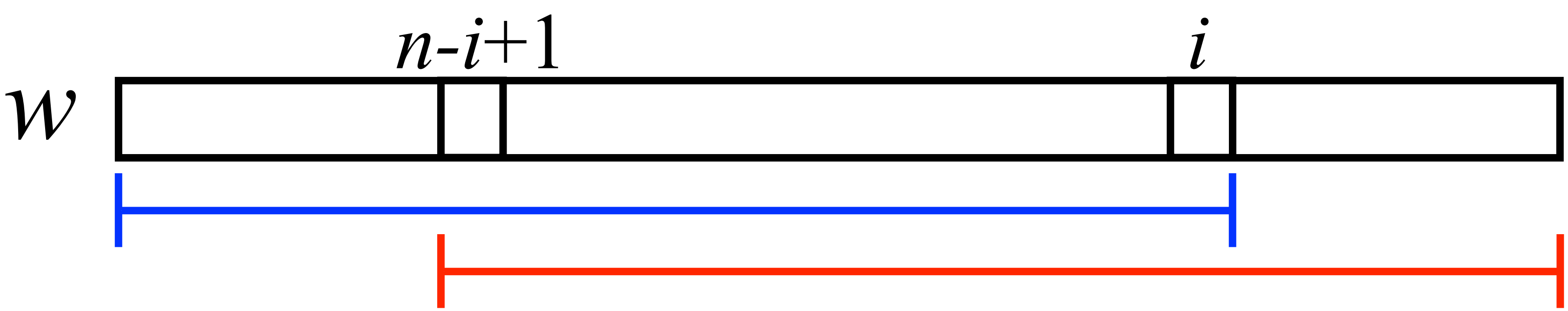


Lemma 1 (*Abelian covers*)

String w of length n has an *Abelian cover*
$\Leftrightarrow P_{w[1,\,i]} = P_{w[n-i+1,\,n]}$ for some $1 \le i < n$.

Proof (←)

If, for some $n/2 < i < n$, $P_{w[1,\,i]} = P_{w[n-i+1,\,n]}$,
$I = \{[1, i], [n-i+1, n]\}$ is an *Abelian cover* of w.



Algorithm (*Abelian covers*)

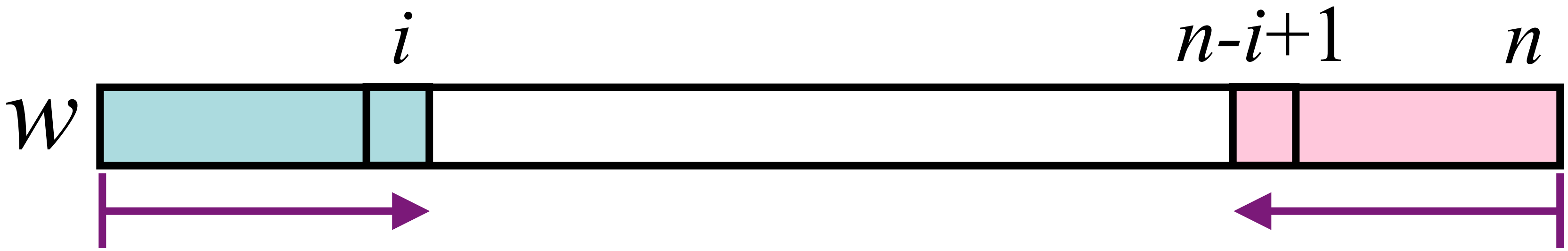
Lemma 1 (Abelian covers)

String w of length n has an *Abelian cover*

$\Leftrightarrow P_{w[1, i]} = P_{w[n-i+1, n]}$ for some $1 \leq i < n$.

We look for an *Abelian border* of w.

Algorithm (*Abelian covers*)



We scan w from left and from right, and check if $w[1, i]$ and $w[n\text{-}i+1, n]$ are *Abelian equivalent* for some i.

Algorithm (*Abelian covers*)

w = a b a c b a c a a b c

	prefix	suffix
a	0	0
b	0	0
c	0	0

$counter = 3$ — # of matching elements of the Parikh vectors.

Algorithm (*Abelian covers*)

w = a b a c b a c a a b c

	prefix	suffix
a	1	0
b	0	0
c	0	0

$counter = 2$

# of matching elements of the Parikh vectors.

Algorithm (*Abelian covers*)

w = a b a c b a c a a b c

	prefix	suffix
a	1	0
b	0	0
c	0	1

$counter = 1$

# of matching elements of the Parikh vectors.

Algorithm (*Abelian covers*)

w = a b a c b a c a a b c

	prefix	suffix
a	1	0
b	1	0
c	0	1

$counter = 0$

# of matching elements of the Parikh vectors.

Algorithm (*Abelian covers*)

w = a b a c b a c a a b c

	prefix	suffix
a	1	0
b	1	1
c	0	1

$counter = 1$

\# of matching elements of the Parikh vectors.

Algorithm (*Abelian covers*)

$w =$ a b a c b a c a a b c

	prefix	suffix
a	2	0
b	1	1
c	0	1

$counter = 1$

# of matching elements of the Parikh vectors.

Algorithm (*Abelian covers*)

$w =$ a b a c b a c a a b c

	prefix	suffix
a	2	1
b	1	1
c	0	1

$counter = 1$

# of matching elements of the Parikh vectors.

Algorithm (*Abelian covers*)

$w = $ a b a c b a c a a b c

	prefix	suffix
a	2	1
b	1	1
c	1	1

$counter = 2$

# of matching elements of the Parikh vectors.

Algorithm (*Abelian covers*)

$w =$ a b a c b a c a a b c

	prefix	suffix
a	2	2
b	1	1
c	1	1

counter = 3

String *w* has an *Abelian border*.

Time and Space (*Abelian covers*)

Theorem 1

Given a string w of length n, we can determine whether or not w has an *Abelian cover* in $O(n)$ time with $O(|\Sigma|)$ working space.

- The Parikh vectors of all prefixes and suffixes can be computed and compared in $O(n)$ time.
- We maintain two Parikh vectors requiring $O(|\sum|)$ space.

Abelian runs

Definition

Substring $w[i, j]$ of string w is an *Abelian run* of w, if

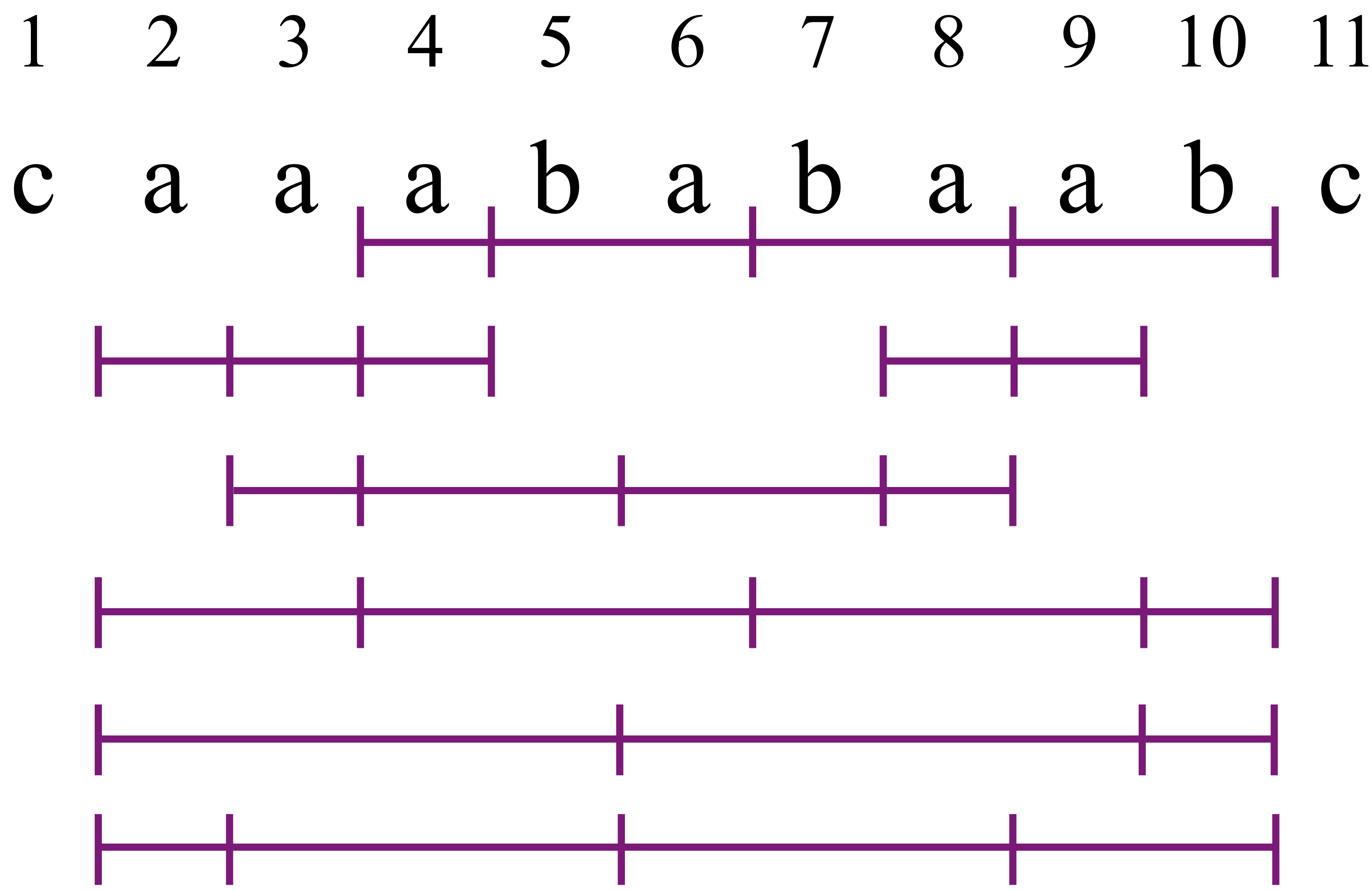
- $w[i, j] = u' u_1 \cdots u_r u''$ with $r \geq 2$,
- $P_{u'} < P_{u_1} = \cdots = P_{u_r} > P_{u''}$,
- $P_{w[i-1]u'} \nleq P_{u_1}$ and
- $P_{u''w[j+1]} \nleq P_{u_1}$,

and is represented by 5-tuple $(i, |u'|, |u_1|, |u''|, r)$.

Example

1	2	3	4	5	6	7	8	9	10	11
c	a	a	a	b	a	b	a	a	b	c

Example



Example

1	2	3	4	5	6	7	8	9	10	11
c	a	a	a	b	a	b	a	a	b	c

u' = [4], u_1 = [5, 6], u_2 = [7, 8], u_3 = [9, 10]

$$P_{u_1} = P_{u_2} = P_{u_3}$$

u_1, u_2 and u_3 are called the <u>*cores*</u> of this *Abelian run*.

Example

1	2	3	4	5	6	7	8	9	10	11
c	a	a	a	b	a	b	a	a	b	c

u' (position 4), u_1 (positions 5–6), u_2 (positions 7–8), u_3 (positions 9–10)

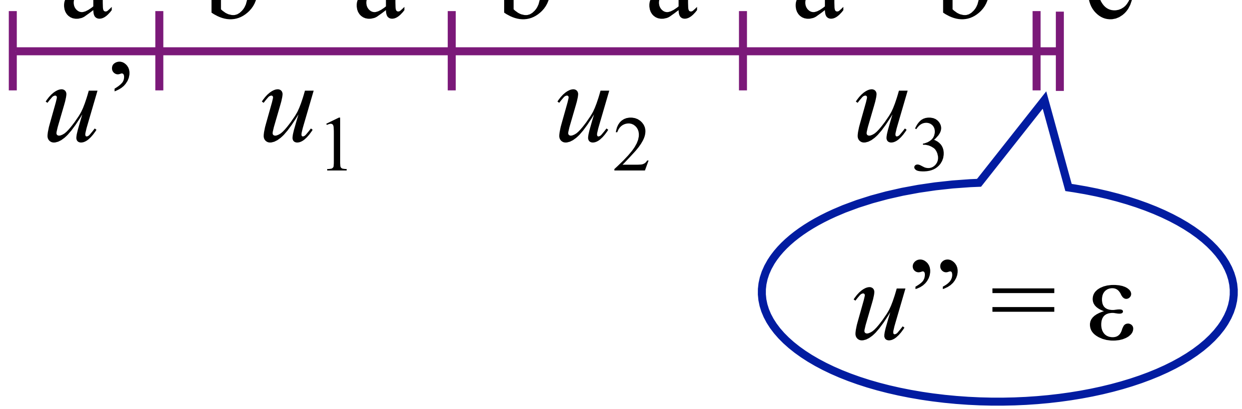
$$P_{u'} < P_{u_1}$$

$$P_{\mathrm{a}u'} \not\leq P_{u_1}$$

u' is the <u>*left arm*</u> of this *Abelian run*.

Example

1	2	3	4	5	6	7	8	9	10	11
c	a	a	a	b	a	b	a	a	b	c



$$P_{u_1} > P_{u''}$$

u'' is the <u>*right arm*</u> of this *Abelian run*.

Problem 2 (All *Abelian runs*)

Given a string *w*, compute all *Abelian runs* in *w*.

Algorithm (All *Abelian runs*)

Our algorithm consists of the following three steps:

① Compute All *Abelian squares*
② Merge *Abelian squares* into *cores* $u_1, \ldots, u_r$
③ Compute *left arms u'* and *right arms u''*

Algorithm (All *Abelian runs*)

Our algorithm consists of the following three steps:

① Compute All *Abelian squares*
② Merge *Abelian squares* into *cores* $u_1, \ldots, u_r$
③ Compute *left arms u'* and *right arms u''*

We construct a table *T* for steps ① and ②.

Algorithm step ① (*Abelian runs*)

Definition

Table T is a $n/2 \times (n\text{-}1)$ table such that for $1 \leq k \leq n\text{-}1$ and $1 \leq d \leq n/2$

- $T[d, k] = 1$ if $P_{w[k-d+1,\, k]} = P_{w[k+1,\, k+d]}$
- $T[d, k] = 0$ otherwise,

and $T[d, k]$ are undefined for $n/2 < d$, $k\text{-}d\text{+}1 < 1$ and $n < k\text{+}d$.

Table T represents all *Abelian squares* of w.

Table *T*

d \ k	1	2	3	4	5	6	7	8	9	10	11
	c	a	a	a	b	a	b	a	a	b	c
1	0	1	1	0	0	0	0	1	0	0	
2		0	0	0	1	1	0	1	0		
3			0	0	1	1	0	0			
4				0	1	0	0				
5					0	0					

Table T

$d \backslash k$	1	2	3	4	5	6	7	8	9	10	11
	c	a	a	a	b	a	b	a	a	b	c
$P_{w[4,6]} = P_{w[7,9]}$											
1	0	1	1	0	0	0	0	1	0	0	
2		0	0	0	1	1	0	1	0		
3			0	0	1	1	0	0			
4				0	1	0	0				
5					0	0					

Table T

d \ k	1	2	3	4	5	6	7	8	9	10	11
	c	a	a	a	b	a	b	a	a	b	c
$P_{w[3,6]} \neq P_{w[7,10]}$											
1	0	1	1	0	0	0	0	1	0	0	
2		0	0	0	1	1	0	1	0		
3			0	0	1	1	0	0			
4				0	1	0	0				
5					0	0					

Table T ⇨ All *Abelian squares* of w

d \ k	1	2	3	4	5	6	7	8	9	10	11
	c	a	a	a	b	a	b	a	a	b	c
					$P_{w[3, 6]} \neq P_{w[7, 10]}$						
1	0	1	1	0	0	0	0	1	0	0	
2		0	0	0	1	1	0	1	0		
3			0	0	1	1	0	0			
4				0	1	0	0				
5					0	0					

Step ① Compute All *Abelian squares*

$d \backslash k$	1	2	3	4	5	6	7	8	9	10	11
	c	a	a	a	b	a	b	a	a	b	c
						$P_{w[6,6]} \neq P_{w[7,7]}$					
1	0	1	1	0	0	0					
2		0	0	0	1						
3			0	0	1						
4				0	1						
5					0						

Step ① Compute All *Abelian squares*

d \ k	1	2	3	4	5	6	7	8	9	10	11
	c	a	a	a	b	a	b	a	a	b	c
1	0	1	1	0	0	0					
2		0	0	0	1	1					
3			0	0	1						
4				0	1						
5					0						

$$P_{w[5,6]} = P_{w[7,8]}$$

Step ① Compute All *Abelian squares*

d \ k	1	2	3	4	5	6	7	8	9	10	11
	c	a	a	a	b	a	b	a	a	b	c
1	0	1	1	0	0	0					
2		0	0	0	1	1					
3			0	0	1	1					
4				0	1	0					
5					0	0					

$$P_{w[2,6]} = P_{w[7,11]}$$

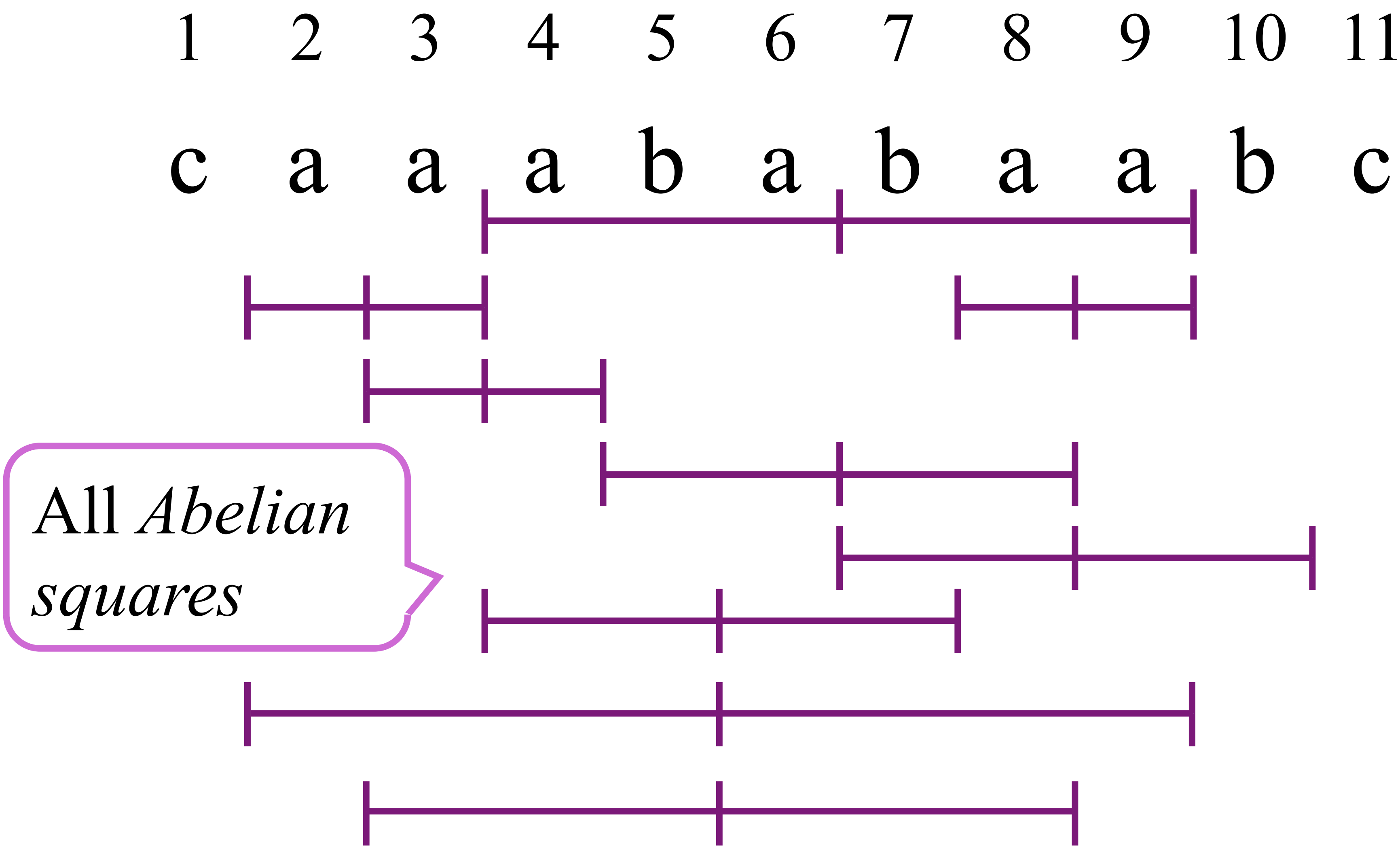
Step ① Compute All *Abelian squares*

Lemma 2

Table T requires $O(n^2)$ space and can be computed in $O(n^2)$ time.

- Each column of T can be computed in $O(n)$ time.
- It takes $O(n^2)$ time for all columns.

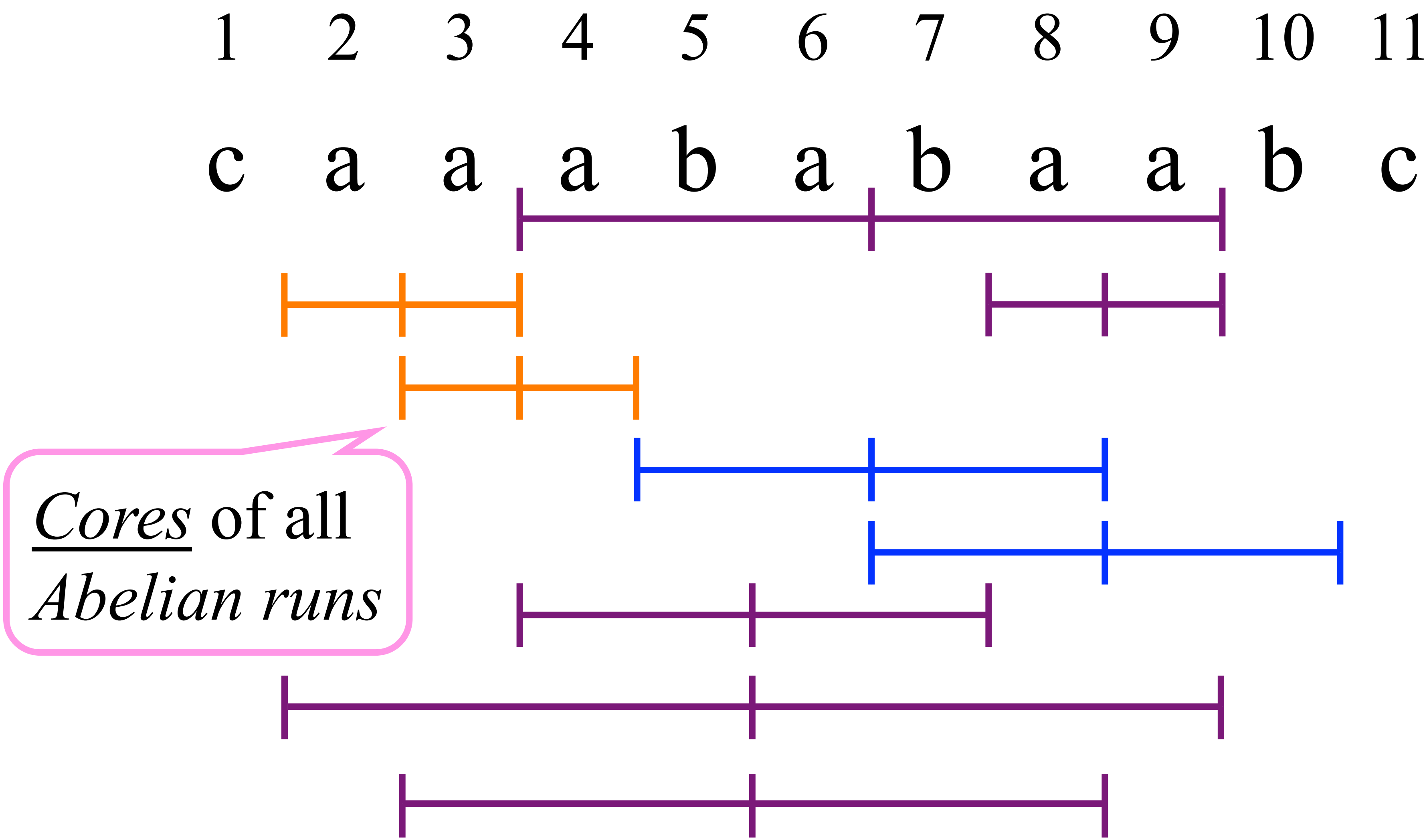
Step ② Merge *Abelian squares* into *cores*



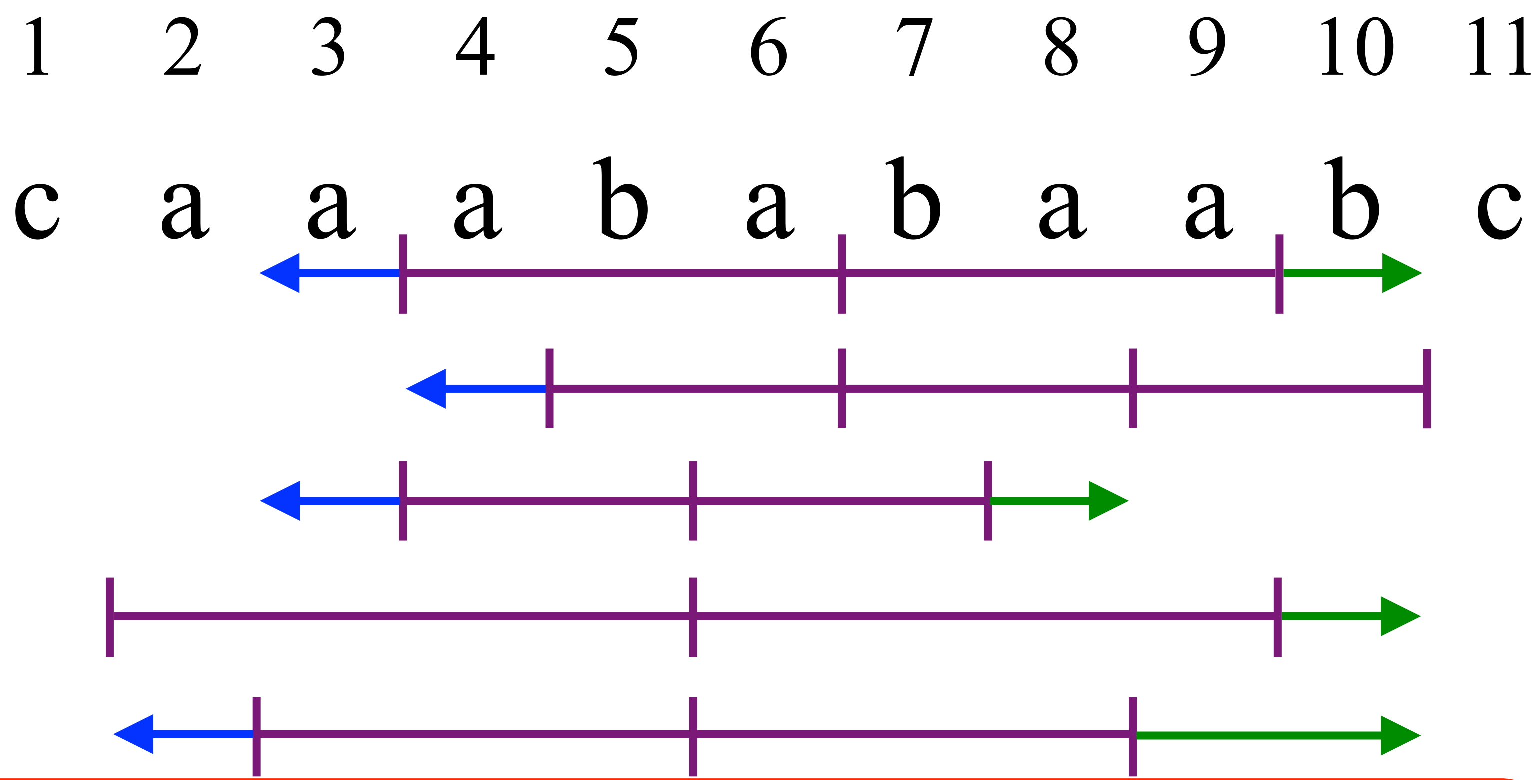
Step ② Merge *Abelian squares* into *cores*

$d \backslash k$	1	2	3	4	5	6	7	8	9	10	11
	c	a	a	a	b	a	b	a	a	b	c
				d=1							
1	0	1	1	0	0	0	0	1	0	0	
2		0	0	0	1	1	0	1	0		
3			0	0	1		0		d=2		
4				0	1	0	0				
5					0	0					

Step ② Merge *Abelian squares* into *cores*



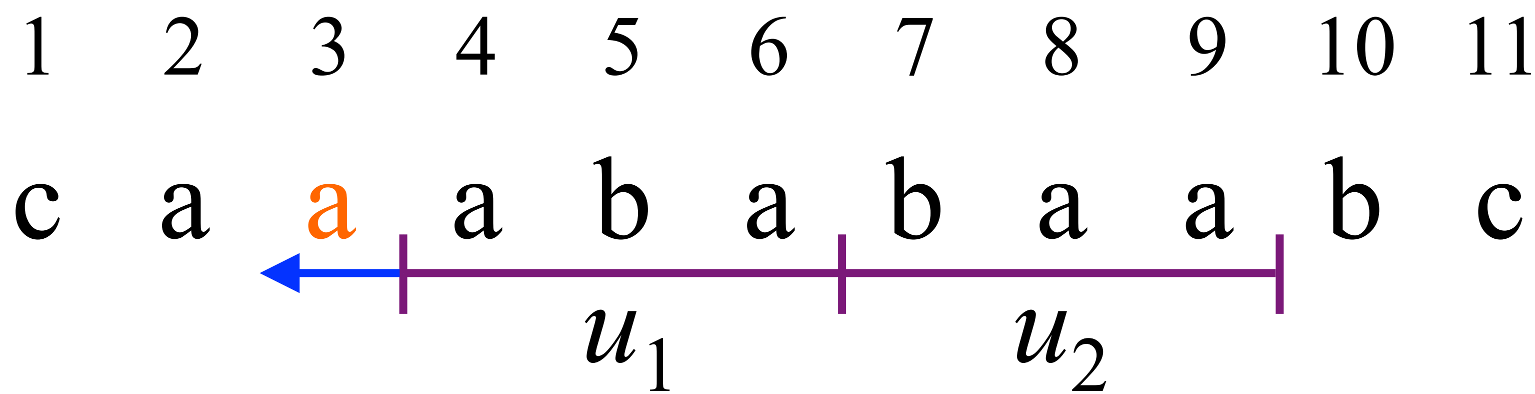
Step ③ *Left arms* and *right arms*



We scan w from u_1 to left and from u_r to right, and check if

- $P_{u'} < P_{u_1} = \cdots = P_{u_r} > P_{u''}$,
- $P_{w[i-1]u'} \leq P_{u_1}$ and
- $P_{u''w[j+1]} \leq P_{u_1}$.

Step ③ *Left arms* and *right arms*

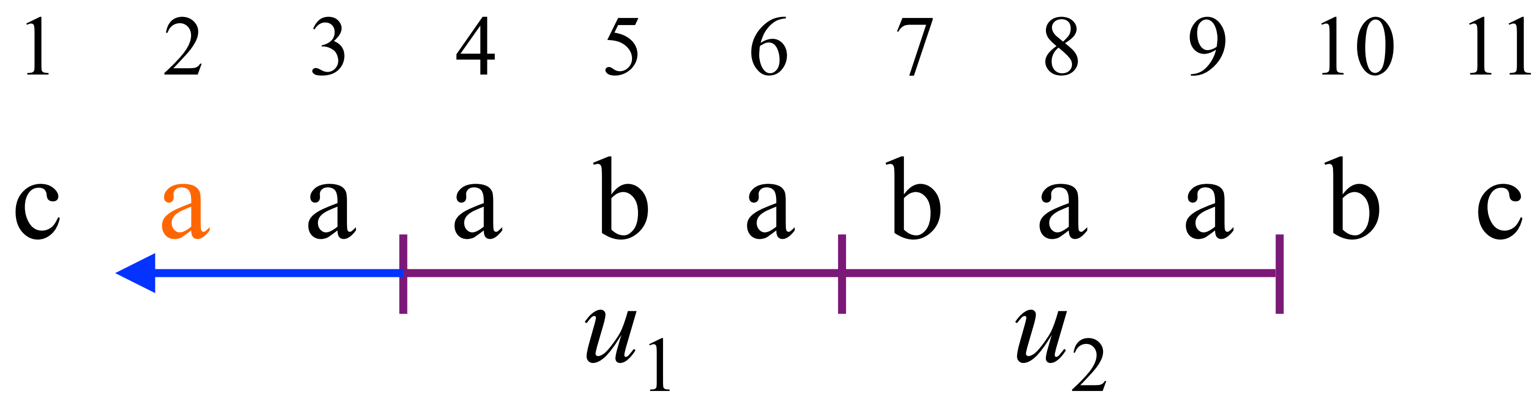


	Left string	u_1
a	1	2
b	0	1
c	0	0

$counter = 0$

of k such that $P_{\text{Left string}}[k] > P_{u_1}[k]$

Step ③ *Left arms* and *right arms*

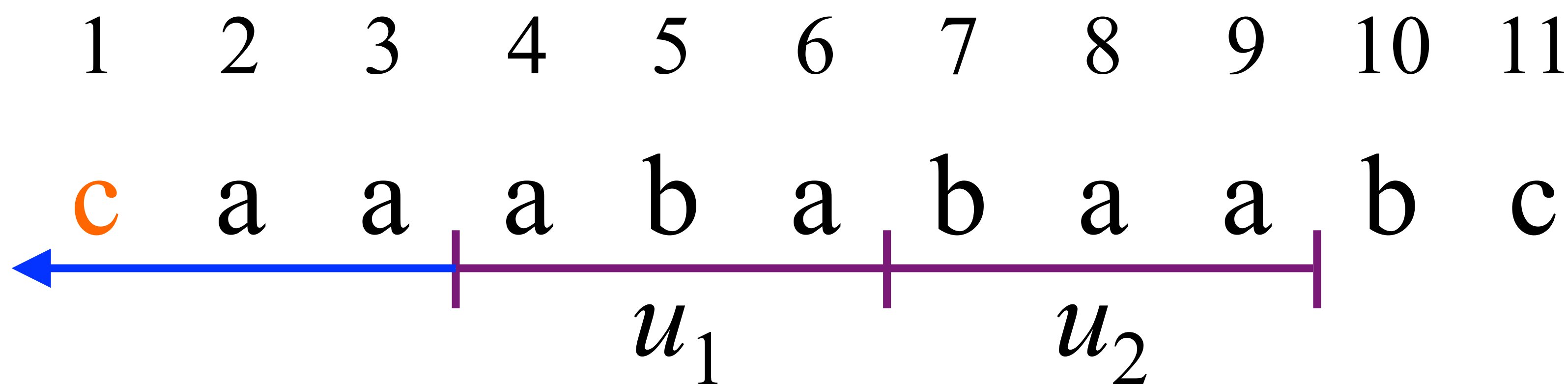


	Left string	u_1
a	2	2
b	0	1
c	0	0

$counter = 0$

of k such that $P_{\text{Left string}}[k] > P_{u_1}[k]$

Step ③ *Left arms* and *right arms*

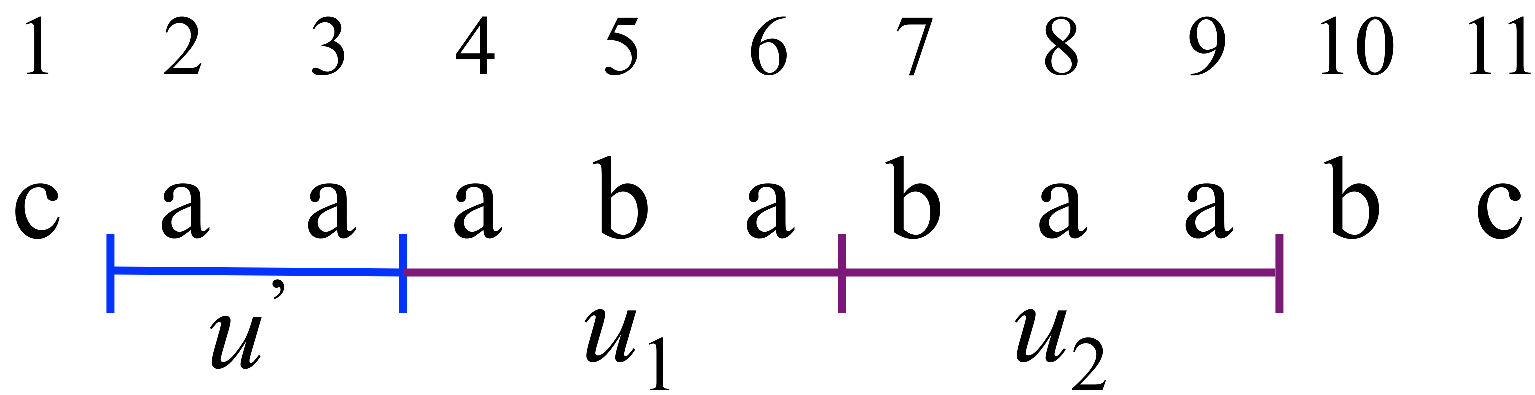


	Left string	u_1
a	2	2
b	0	1
c	1	0

counter =1

of k such that $P_{\text{Left string}}[k] > P_{u_1}[k]$

Step ③ *Left arms* and *right arms*



	Left string	u_1
a	2	2
b	0	1
c	0	0

Maximum number of *Abelian runs*

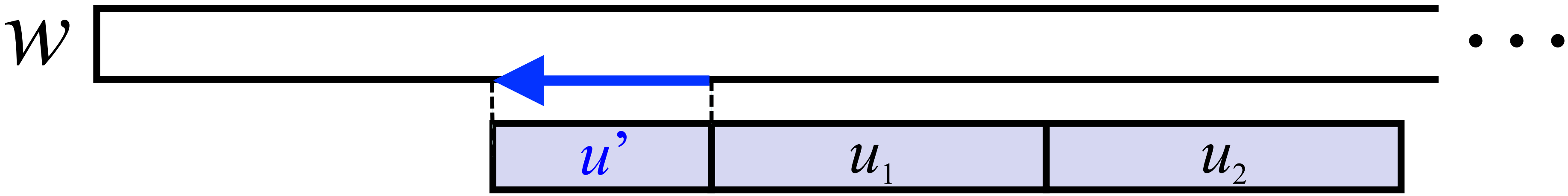
Theorem 2 (*Abelian runs*)

The maximum number of *Abelian runs* in a string w of length n is $\Omega(n^2)$.

- The Cummings–Smyth string $(\mathrm{aababbbab})^n$ of length $8n$ has $\Theta(n^2)$ maximal Abelian runs.
- A naïve algorithm takes $O(n^3)$ time for all *Abelian runs*.

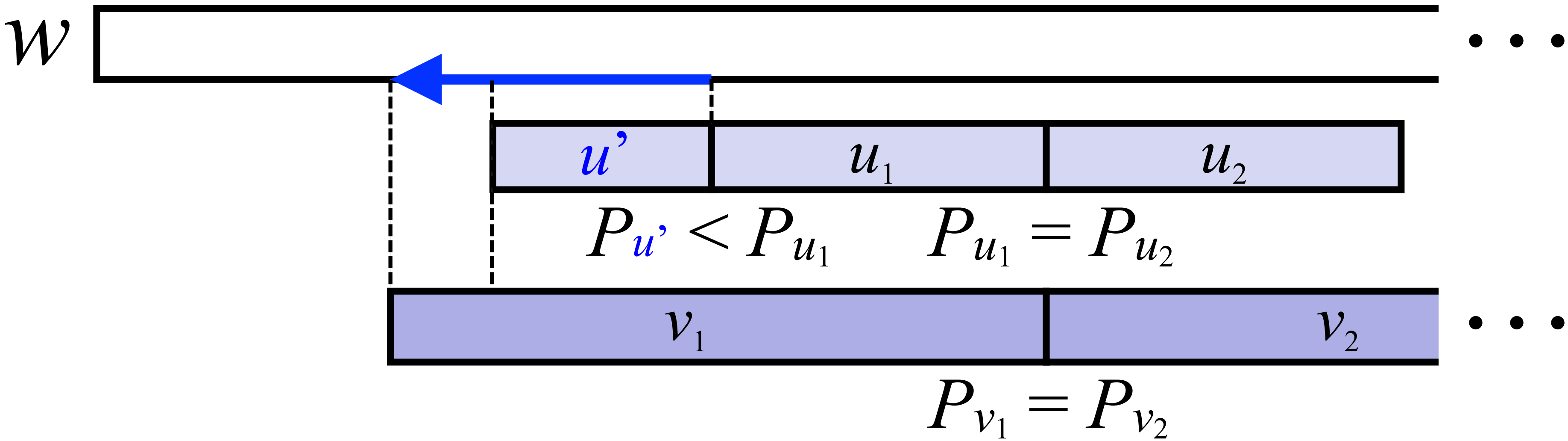
I will explain how to compute the *left* and *right arms* for all *Abelian runs* in a total of $O(n^2)$ time.

Step ③ *Left arms* and *right arms*



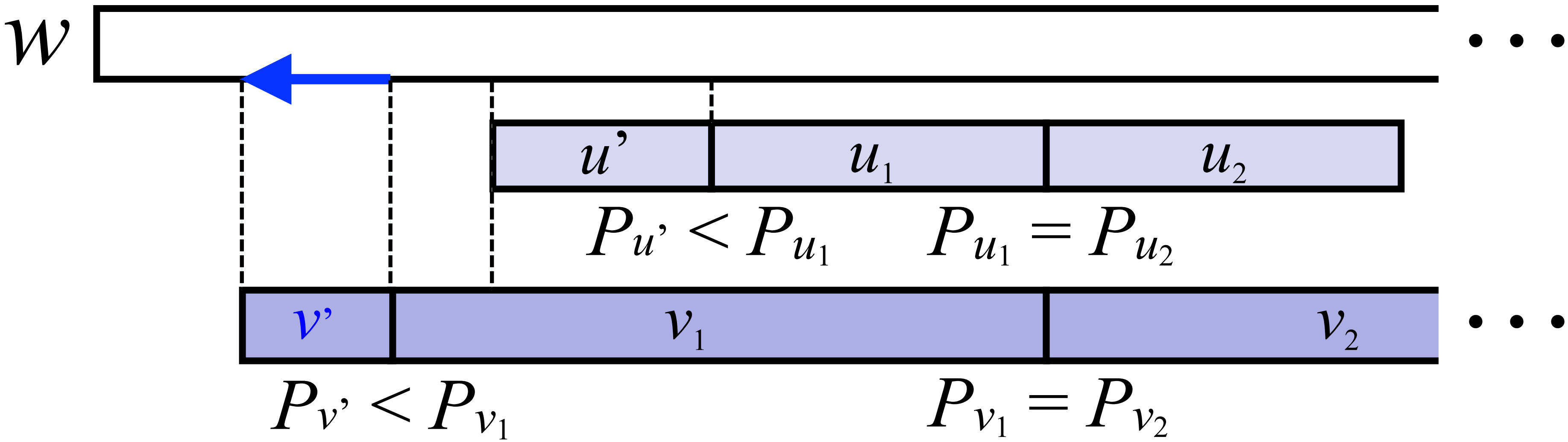
$P_{u'} < P_{u_1}$ $P_{u_1} = P_{u_2}$

Step ③ *Left arms* and *right arms*



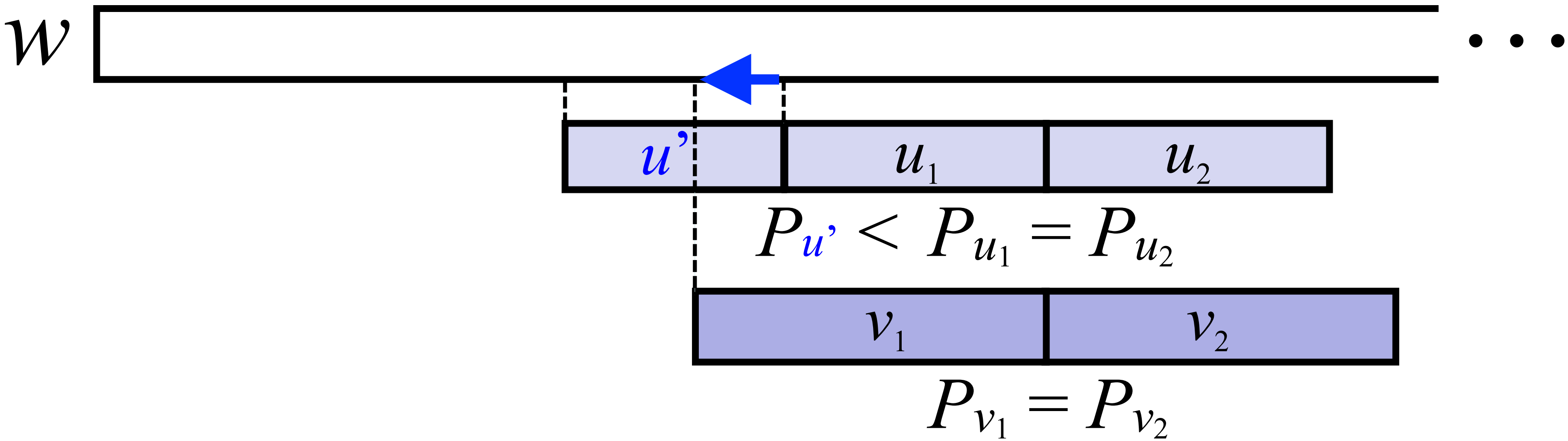
Case 1 ($|u'u_1| < |v_1|$)

Step ③ *Left arms* and *right arms*



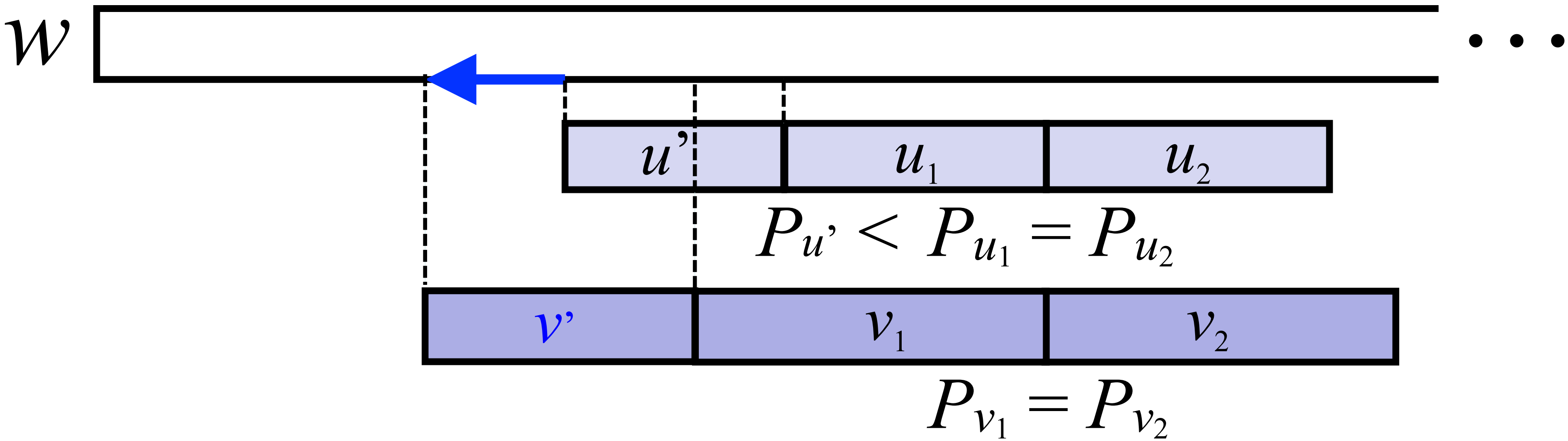
Case 1 ($|u'u_1| < |v_1|$)

Step ③ *Left arms* and *right arms*



Case 2 ($|u'u_1| > |v_1|$)

Step ③ *Left arms* and *right arms*



Case 2 ($|u'u_1| > |v_1|$)

Time and Space (*Abelian runs*)

Theorem 3

Given a string w of length n, we can compute all *Abelian runs* in $O(n^2)$ time with $O(n^2)$ working space.

- Table T requires $O(n^2)$ space and can be computed in $O(n^2)$ time. (Steps ① and ②)
- All *left arms* and *right arms* are computed in $O(n^2)$ time. (Step ③)

Conclusion 1

Problem 1 (*Abelian cover* existence)

- $O(n)$ time with $O(|\Sigma|)$ working space
  - We compute the longest *Abelian cover* of w.

Open problem

Can we compute the shortest *Abelian cover* in faster than $O(n^2)$ time ?

✓ We can compute the shortest *Abelian cover* of w by a naïve algorithm in $O(n^2)$ time.

Conclusion 2

Problem 2 (All *Abelian runs*)

- $O(n^2)$ time with $O(n^2)$ working space
- String w of length n has $\Omega(n^2)$ *Abelian runs*.

Open problem

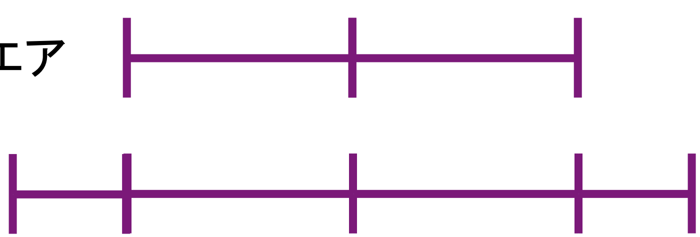
Can we compute all *Abelian runs* in w in $O(n + r)$ time where r is the number of *Abelian runs* in w ?

出力サイズに線形な時間？

本研究では $O(n^2)$ 時間ですべてのアーベル連を求めたが…

L. J. Cummings と W. F. Smyth が
$w = (aababbab)^n$　長さ $8n$
には $\underline{\Omega(n^2)}$ 個の連接するアーベル同値な文字列(アーベルスクエア)が存在することを示した.

アーベルスクエア
アーベル連



アーベル連も
$\Omega(n^2)$ 個