

# Constant-Time Dynamic Enumeration of Word Infixes in a Regular Language

Antoine Amarilli  
Inria & Univ. Lille

Sven Dziadek  
Télécom SudParis

Luc Segoufin  
Inria & ENS Paris

# Enumeration Algorithms

query: factor matching  $a\Sigma^*a$

data: word  $w$

evaluation

results

abeeeebaebeabeebeaee

( 0, 6)

( 0, 10)

( 0, 17)

( 6, 10)

( 6, 17)

(10, 17)

Drawback:

Possibly:

quadratic number of results

but quadratic complexity easy

# Enumeration Algorithms

query: factor matching  $a\Sigma^*a$

data: word  $w$

abeeeebaebeabeebeaee

evaluation

results

Drawback:

Possibly:

quadratic number of results

but quadratic complexity easy

enumeration algorithms (more fine-grained):

- delay (between two consecutive outputs)

# Enumeration Algorithms

query: factor matching  $a\Sigma^*a$

data: word  $w$

abeeeebaebeabeebeaee

evaluation

results

( 0, 10)

Drawback:

Possibly:

quadratic number of results

but quadratic complexity easy

enumeration algorithms (more fine-grained):

- delay (between two consecutive outputs)

# Enumeration Algorithms

query: factor matching  $a\Sigma^*a$

data: word  $w$

evaluation

results

abeeeebaebeabeebeaee

( 0, 10)  
(10, 17)

Drawback:

Possibly:

quadratic number of results

but quadratic complexity easy

enumeration algorithms (more fine-grained):

- delay (between two consecutive outputs)

# Enumeration Algorithms

query: factor matching  $a\Sigma^*a$

data: word  $w$

evaluation

results

abeeeebaebeabeebeaee

( 0, 10)  
(10, 17)  
( 6, 10)

Drawback:

Possibly:

quadratic number of results  
but quadratic complexity easy

enumeration algorithms (more fine-grained):

- delay (between two consecutive outputs)

# Enumeration Algorithms

query: factor matching  $a\Sigma^*a$

data: word  $w$

evaluation

abeeeebaebeabeebeaee

results

( 0, 10)

(10, 17)

( 6, 10)

( 0, 17)

( 0, 6)

( 6, 17)

Drawback:

Possibly:

quadratic number of results

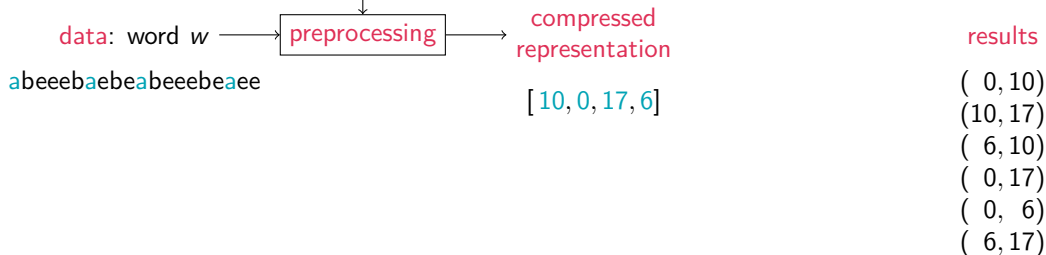
but quadratic complexity easy

enumeration algorithms (more fine-grained):

- delay (between two consecutive outputs)

# Enumeration Algorithms

query: factor matching  $a\Sigma^*a$



## Drawback:

Possibly:

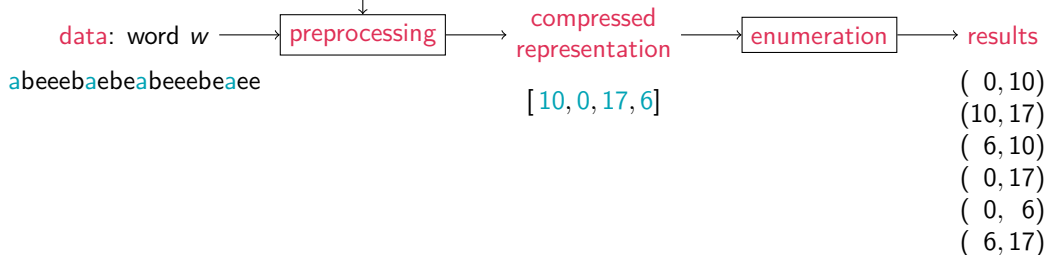
quadratic number of results  
but quadratic complexity easy

enumeration algorithms (more fine-grained):

- delay (between two consecutive outputs)
- preprocessing (time to produce compressed representation)

# Enumeration Algorithms

query: factor matching  $a\Sigma^*a$



## Drawback:

Possibly:

quadratic number of results  
but quadratic complexity easy

enumeration algorithms (more fine-grained):

- delay (between two consecutive outputs)
- preprocessing (time to produce compressed representation)

# Enumeration Algorithms

query: factor matching  $a\Sigma^*a$

data: word  $w$   
abeeeebae**b**aebeeebeae

update: change  
letter at position 10 to  $b$

preprocessing

compressed  
representation  
[ 10, 0, 17, 6 ]

enumeration

results

( 0, 10)  
(10, 17)  
( 6, 10)  
( 0, 17)  
( 0, 6)  
( 6, 17)

## Drawback:

Possibly:

quadratic number of results  
but quadratic complexity easy

enumeration algorithms (more fine-grained):

- delay (between two consecutive outputs)
- preprocessing (time to produce compressed representation)

# Enumeration Algorithms

query: factor matching  $a\Sigma^*a$

data: word  $w$   
abeeeebae**b**aebeeebeae

update: change  
letter at position 10 to  $b$

preprocessing

compressed  
representation

[~~10~~, 0, 17, 6]

enumeration

results

( 0, 10)  
(10, 17)  
( 6, 10)  
( 0, 17)  
( 0, 6)  
( 6, 17)

## Drawback:

Possibly:

quadratic number of results  
but quadratic complexity easy

enumeration algorithms (more fine-grained):

- delay (between two consecutive outputs)
- preprocessing (time to produce compressed representation)
- update (time to update compressed representation)

# Enumeration Algorithms

query: factor matching  $a\Sigma^*a$

data: word  $w$   
abeeeebaebeabeeeebaee

update: change  
letter at position 10 to  $b$

preprocessing

compressed  
representation  
 $[10, 0, 17, 6]$

enumeration

results

~~( 0, 10)~~  
~~(10, 17)~~  
~~( 6, 10)~~  
( 0, 17)  
( 0, 6)  
( 6, 17)

Drawback:

Possibly:

quadratic number of results  
but quadratic complexity easy

enumeration algorithms (more fine-grained):

- delay (between two consecutive outputs)
- preprocessing (time to produce compressed representation)
- update (time to update compressed representation)  
(updates reset enumeration)

# Motivation

## Related Work: Niewerth, Segoufin 2018

Enumeration of regular language infixes over words possible with linear preprocessing, constant delay enumeration and logarithmic-time updates ensuring constant additional memory.

# Motivation

## Related Work: Niewerth, Segoufin 2018

Enumeration of regular language infixes over words possible with linear preprocessing, constant delay enumeration and **logarithmic-time updates** ensuring constant additional memory.

Can we do better than logarithmic-time updates?

# Motivation

## Related Work: Niewerth, Segoufin 2018

Enumeration of regular language infixes over words possible with linear preprocessing, constant delay enumeration and **logarithmic-time updates** ensuring constant additional memory.

Can we do better than logarithmic-time updates?

**Yes, of course:**  $L = a$  can be enumerated with constant-time updates

## Related Work: Niewerth, Segoufin 2018

Enumeration of regular language infixes over words possible with linear preprocessing, constant delay enumeration and **logarithmic-time updates** ensuring constant additional memory.

Can we do better than logarithmic-time updates?

Yes, of course:  $L = a$  can be enumerated with constant-time updates

*Research question:* What class of regular languages can be enumerated with

1. **constant-time updates**,
2. constant delay enumeration,
3. linear preprocessing, and
4. constant additional memory?

# Motivation

## Related Work: Niewerth, Segoufin 2018

Enumeration of regular language infixes over words possible with linear preprocessing, constant delay enumeration and **logarithmic-time updates** ensuring constant additional memory.

Can we do better than logarithmic-time updates?

Yes, of course:  $L = a$  can be enumerated with constant-time updates

*Research question:* What class of regular languages can be enumerated with

1. **constant-time updates**,
2. constant delay enumeration,
3. linear preprocessing, and
4. constant additional memory?

(Unit-cost RAM model with cell size logarithmic in the word length)

# Motivation

## Related Work: Niewerth, Segoufin 2018

Enumeration of regular language infixes over words possible with linear preprocessing, constant delay enumeration and logarithmic-time updates ensuring constant additional memory.

Can we do better than logarithmic-time updates?

Yes, of course:  $L = a$  can be enumerated with constant-time updates

But how exactly?

*Research question:* What class of regular languages can be enumerated with

1. constant-time updates,
2. constant delay enumeration,
3. linear preprocessing, and
4. constant additional memory?

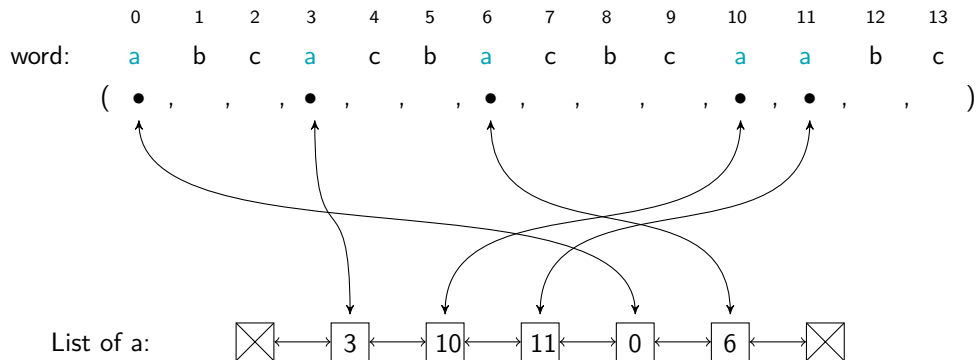
(Unit-cost RAM model with cell size logarithmic in the word length)

## Related Work & Examples

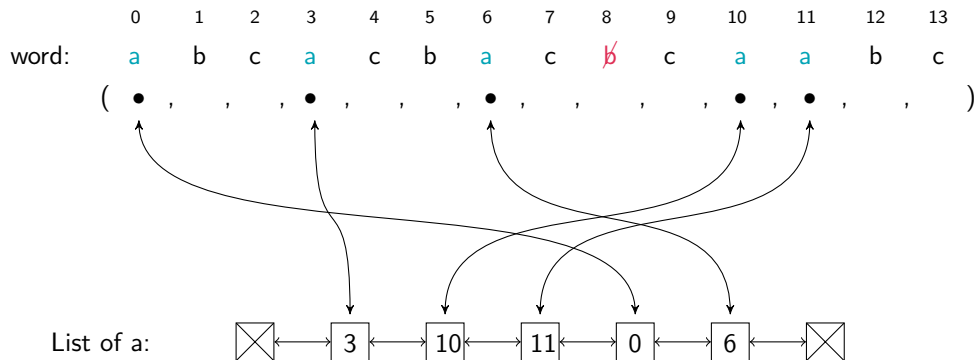
## Data Structure - Occurrence Lists

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| word: | a | b | c | a | c | b | a | c | b | c | a  | a  | b  | c  |

## Data Structure - Occurrence Lists

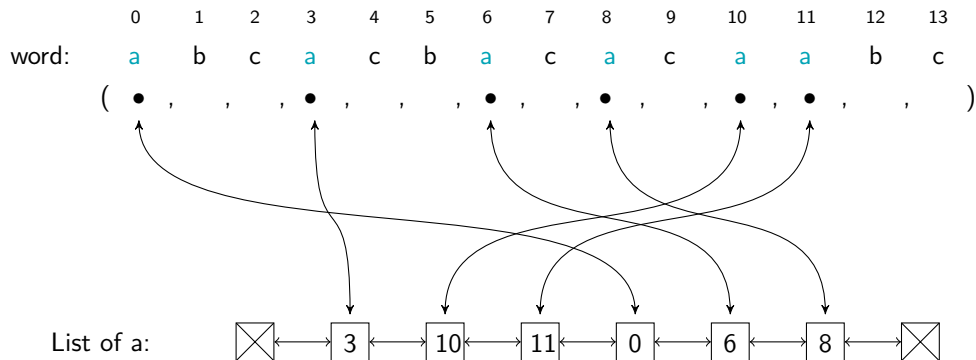


## Data Structure - Occurrence Lists



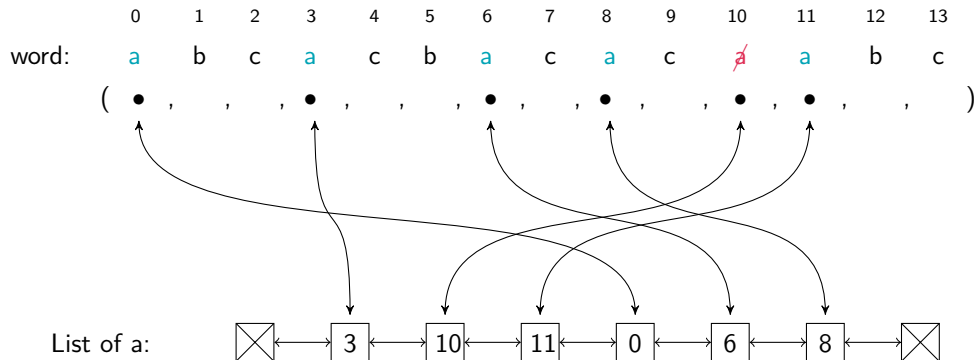
add item: constant time

## Data Structure - Occurrence Lists



add item: constant time

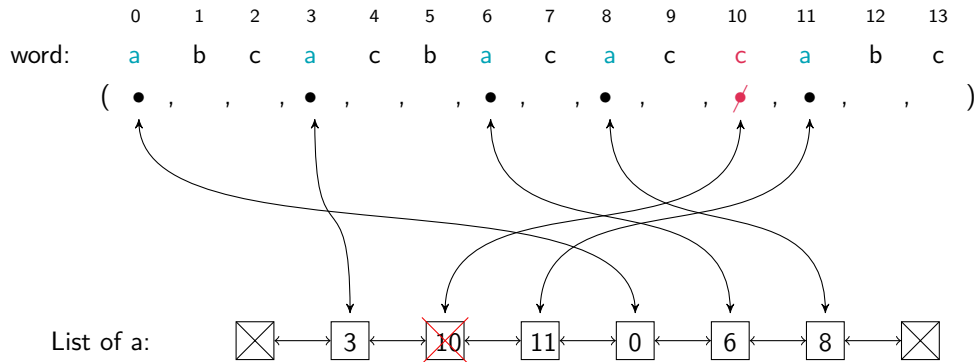
## Data Structure - Occurrence Lists



add item: constant time

delete item: constant time

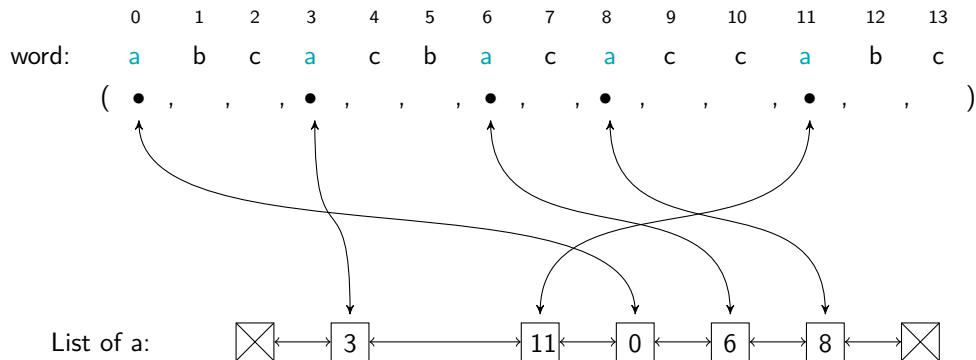
## Data Structure - Occurrence Lists



add item: constant time

delete item: constant time

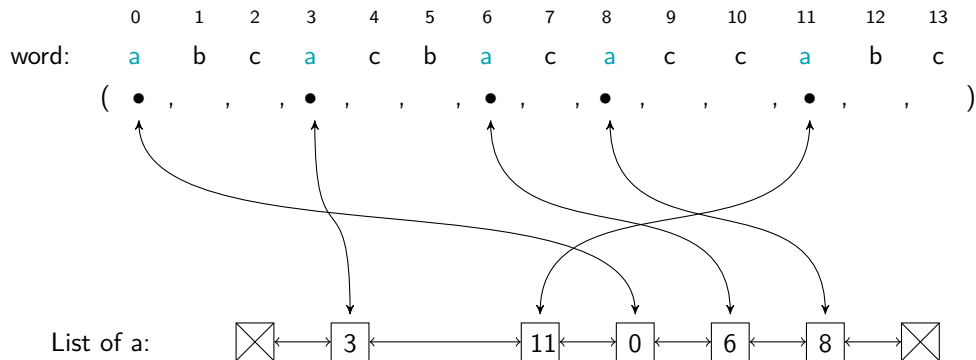
## Data Structure - Occurrence Lists



add item: constant time

delete item: constant time

## Data Structure - Occurrence Lists

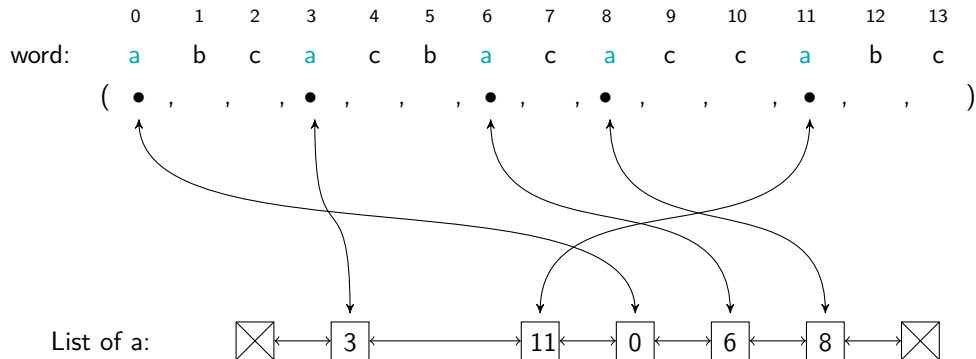


add item: constant time

delete item: constant time

retrieve items: constant delay

## Data Structure - Occurrence Lists



add item: constant time

delete item: constant time

retrieve items: constant delay

Folklore:

similar to LinkedHashSet in Java

## Examples

Consider languages on alphabet  $\Sigma = \{a, b, c\}$ .

Can they be enumerated with constant-time updates and constant delay?

- $a$

# Examples

Consider languages on alphabet  $\Sigma = \{a, b, c\}$ .

Can they be enumerated with constant-time updates and constant delay?

–  $a$  ✓

# Examples

Consider languages on alphabet  $\Sigma = \{a, b, c\}$ .

Can they be enumerated with constant-time updates and constant delay?

- $a$  ✓
- $a\Sigma^*a$

# Examples

Consider languages on alphabet  $\Sigma = \{a, b, c\}$ .

Can they be enumerated with constant-time updates and constant delay?

- $a$  ✓
- $a\Sigma^*a$  ✓

# Examples

Consider languages on alphabet  $\Sigma = \{a, b, c\}$ .

Can they be enumerated with constant-time updates and constant delay?

- $a$  ✓
- $a\Sigma^*a$  ✓
- $a\Sigma^*b$

## Examples

Consider languages on alphabet  $\Sigma = \{a, b, c\}$ .

Can they be enumerated with constant-time updates and constant delay?

- $a$  ✓
- $a\Sigma^*a$  ✓
- $a\Sigma^*b$  ✗ (reduction from Prefix- $U_1$ )

# Examples

Consider languages on alphabet  $\Sigma = \{a, b, c\}$ .

Can they be enumerated with constant-time updates and constant delay?

- $a$  ✓
- $a\Sigma^*a$  ✓
- $a\Sigma^*b$  ✗ (reduction from Prefix- $U_1$ )

## Prefix- $U_1$

Maintain binary word under updates  
Answer if prefix contains a 0

*Conjecture:*

No  $O(1)$  data structure  
for this problem

(by Amarilli, Jachiet, Paperman 2021)

# Examples

Consider languages on alphabet  $\Sigma = \{a, b, c\}$ .

Can they be enumerated with constant-time updates and constant delay?

- $a$  ✓
- $a\Sigma^*a$  ✓
- $a\Sigma^*b$  ✗ (reduction from Prefix- $U_1$ )

With neutral letter  $e$  ( $ww' \in L$  iff  $wew' \in L$ ):

## Prefix- $U_1$

Maintain binary word under updates  
Answer if prefix contains a 0

*Conjecture:*

No  $O(1)$  data structure  
for this problem

(by Amarilli, Jachiet, Paperman 2021)

# Examples

Consider languages on alphabet  $\Sigma = \{a, b, c\}$ .

Can they be enumerated with constant-time updates and constant delay?

- $a$  ✓
- $a\Sigma^*a$  ✓
- $a\Sigma^*b$  ✗ (reduction from Prefix- $U_1$ )

With neutral letter  $e$  ( $ww' \in L$  iff  $wew' \in L$ ):

- $e^*ae^*(ae^*ae^*)^*$  (odd number of  $a$ )

## Prefix- $U_1$

Maintain binary word under updates  
Answer if prefix contains a 0

*Conjecture:*

No  $O(1)$  data structure  
for this problem

(by Amarilli, Jachiet, Paperman 2021)

# Examples

Consider languages on alphabet  $\Sigma = \{a, b, c\}$ .

Can they be enumerated with constant-time updates and constant delay?

- $a$  ✓
- $a\Sigma^*a$  ✓
- $a\Sigma^*b$  ✗ (reduction from Prefix- $U_1$ )

With neutral letter  $e$  ( $ww' \in L$  iff  $wew' \in L$ ):

- $e^*ae^*(ae^*ae^*)^*$  (odd number of  $a$ ) ✓

## Prefix- $U_1$

Maintain binary word under updates  
Answer if prefix contains a 0

*Conjecture:*

No  $O(1)$  data structure  
for this problem

(by Amarilli, Jachiet, Paperman 2021)

# Examples

Consider languages on alphabet  $\Sigma = \{a, b, c\}$ .

Can they be enumerated with constant-time updates and constant delay?

- $a$  ✓
- $a\Sigma^*a$  ✓
- $a\Sigma^*b$  ✗ (reduction from Prefix- $U_1$ )

With neutral letter  $e$  ( $ww' \in L$  iff  $wew' \in L$ ):

- $e^*ae^*(ae^*ae^*)^*$  (odd number of  $a$ ) ✓
- $e^*(ae^*ae^*)^+$  (even number but at least  $2a$ )

## Prefix- $U_1$

Maintain binary word under updates  
Answer if prefix contains a 0

*Conjecture:*

No  $O(1)$  data structure  
for this problem

(by Amarilli, Jachiet, Paperman 2021)

# Examples

Consider languages on alphabet  $\Sigma = \{a, b, c\}$ .

Can they be enumerated with constant-time updates and constant delay?

- $a$  ✓
- $a\Sigma^*a$  ✓
- $a\Sigma^*b$  ✗ (reduction from Prefix- $U_1$ )

With neutral letter  $e$  ( $ww' \in L$  iff  $wew' \in L$ ):

- $e^*ae^*(ae^*ae^*)^*$  (odd number of  $a$ ) ✓
- $e^*(ae^*ae^*)^+$  (even number but at least  $2a$ ) ?

## Prefix- $U_1$

Maintain binary word under updates  
Answer if prefix contains a 0

*Conjecture:*

No  $O(1)$  data structure  
for this problem

(by Amarilli, Jachiet, Paperman 2021)

# Examples

Consider languages on alphabet  $\Sigma = \{a, b, c\}$ .

Can they be enumerated with constant-time updates and constant delay?

- $a$  ✓
- $a\Sigma^*a$  ✓
- $a\Sigma^*b$  ✗ (reduction from Prefix- $U_1$ )

With neutral letter  $e$  ( $ww' \in L$  iff  $wew' \in L$ ):

- $e^*ae^*(ae^*ae^*)^*$  (odd number of  $a$ ) ✓
- $e^*(ae^*ae^*)^+$  (even number but at least  $2a$ ) ?
- $e^*ae^*ae^*$

## Prefix- $U_1$

Maintain binary word under updates  
Answer if prefix contains a 0

*Conjecture:*

No  $O(1)$  data structure  
for this problem

(by Amarilli, Jachiet, Paperman 2021)

# Examples

Consider languages on alphabet  $\Sigma = \{a, b, c\}$ .

Can they be enumerated with constant-time updates and constant delay?

- $a$  ✓
- $a\Sigma^*a$  ✓
- $a\Sigma^*b$  ✗ (reduction from Prefix- $U_1$ )

With neutral letter  $e$  ( $ww' \in L$  iff  $wew' \in L$ ):

- $e^*ae^*(ae^*ae^*)^*$  (odd number of  $a$ ) ✓
- $e^*(ae^*ae^*)^+$  (even number but at least  $2a$ ) ?
- $e^*ae^*ae^*$  ?

## Prefix- $U_1$

Maintain binary word under updates  
Answer if prefix contains a 0

*Conjecture:*

No  $O(1)$  data structure  
for this problem

(by Amarilli, Jachiet, Paperman 2021)

## Related Work: Dynamic Membership

test language:  $L = e^* a e^* b e^*$

Letter e neutral ( $ww' \in L$  iff  $wew' \in L$ )

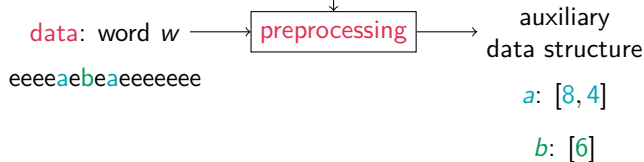
data: word w

eeeeaebaeaaaaaa

## Related Work: Dynamic Membership

test language:  $L = e^* a e^* b e^*$

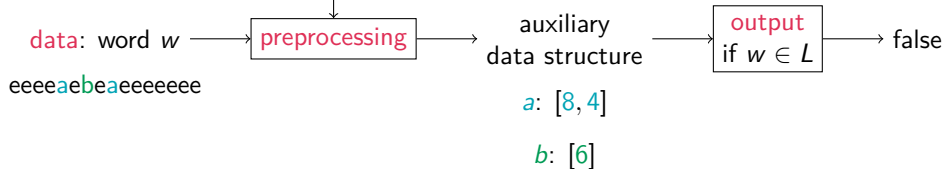
Letter  $e$  neutral ( $ww' \in L$  iff  $wew' \in L$ )



## Related Work: Dynamic Membership

test language:  $L = e^* a e^* b e^*$

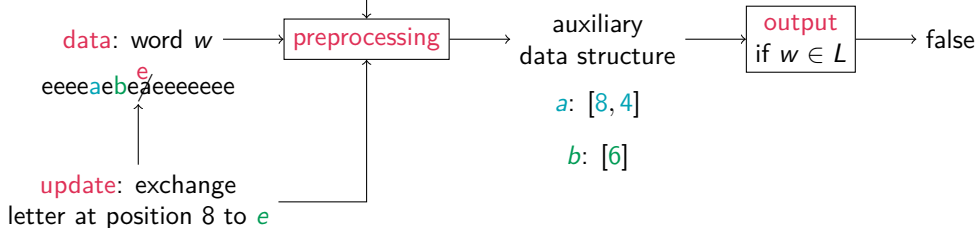
Letter  $e$  neutral ( $ww' \in L$  iff  $wew' \in L$ )



## Related Work: Dynamic Membership

test language:  $L = e^* a e^* b e^*$

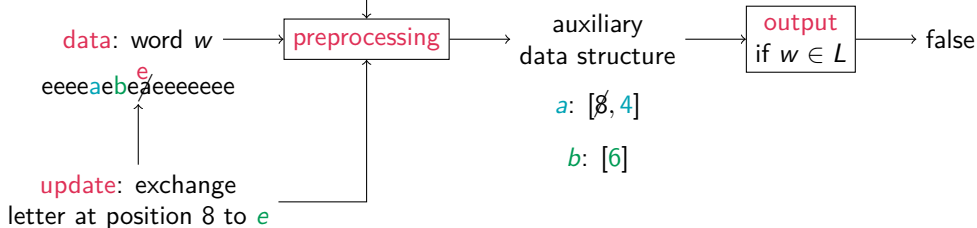
Letter  $e$  neutral ( $ww' \in L$  iff  $wew' \in L$ )



## Related Work: Dynamic Membership

test language:  $L = e^* a e^* b e^*$

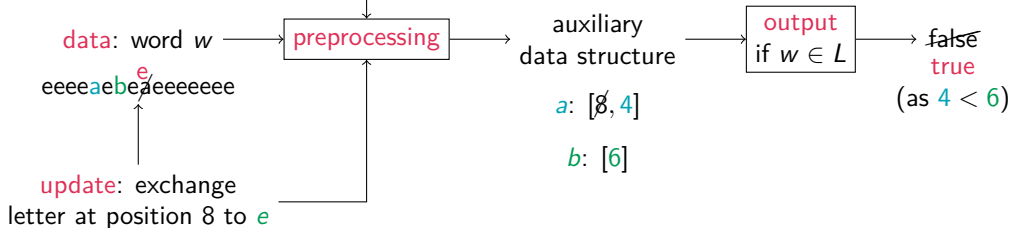
Letter  $e$  neutral ( $ww' \in L$  iff  $wew' \in L$ )



## Related Work: Dynamic Membership

test language:  $L = e^*ae^*be^*$

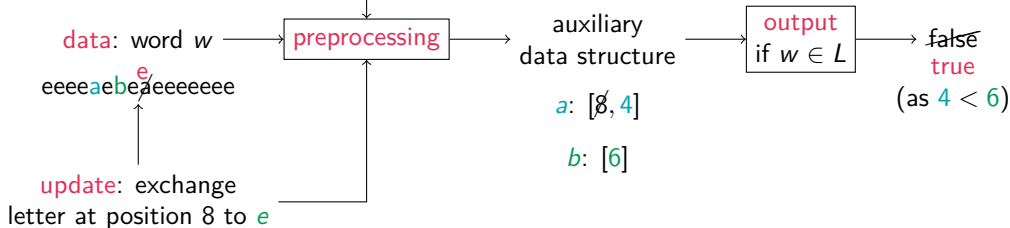
Letter  $e$  neutral ( $ww' \in L$  iff  $wew' \in L$ )



## Related Work: Dynamic Membership

test language:  $L = e^* a e^* b e^*$

Letter  $e$  neutral ( $ww' \in L$  iff  $wew' \in L$ )



## Extensible Language

A language  $L$  is *extensible* iff  $L = \Sigma^* L \Sigma^*$

## Related Work: Dynamic Membership

test language:  $L = e^* a e^* b e^*$

Letter  $e$  neutral ( $ww' \in L$  iff  $wew' \in L$ )

data: word  $w$   
eeeea**e**beeeeeee

update: exchange

letter at position 8 to  $e$

preprocessing

auxiliary  
data structure

$a: [8, 4]$

$b: [6]$

output  
if  $w \in L$

false  
true  
(as  $4 < 6$ )

## Extensible Language

A language  $L$  is *extensible* iff  $L = \Sigma^* L \Sigma^*$

For extensible languages:

dynamic membership on  $L$

is equivalent to

testing if enumeration on  $L$  produces at least one result

## Related Work: Amarilli, Jachiet, Paperman 2021

The dynamic membership problem to  $L$  can be solved in  $O(1)$  iff  $L$  is in ZG  
(conditional to the Prefix- $U_1$  hypothesis)

## Related Work: Amarilli, Jachiet, Paperman 2021

The dynamic membership problem to  $L$  can be solved in  $O(1)$  iff  $L$  is in ZG  
(conditional to the Prefix- $U_1$  hypothesis)

## ZG

Finite monoids satisfying  $yx^{\omega+1} = x^{\omega+1}y$

Intuitively: frequent letters are *central* (they commute with all other letters)

# Contributions

The right side of the slide features a decorative graphic composed of several overlapping triangles. A large teal triangle is at the top, a dark blue triangle is to its right, and a bright pink triangle is at the bottom. These shapes overlap to create a dynamic, abstract composition.

# Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

# Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

## Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| word: | e | a | e | b | e | b | a | e | e | e | b  | a  | e  | a  |



dynamic membership:  $w \in L?$

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| word: | e | a | e | b | e | b | a | e | e | e | b  | a  | e  | a  |



dynamic membership:  $w \in L?$  ✓  $\xRightarrow{\text{even}}$  output  $w$

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| word: | e | a | e | b | e | b | a | e | e | e | b  | a  | e  | e  |



## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| word: | e | a | e | b | e | b | a | e | e | e | b  | a  | e  | e  |



dynamic membership:  $w \in L?$

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| word: | e | a | e | b | e | b | a | e | e | e | b  | a  | e  | e  |



dynamic membership:  $w \in L?$  ✓  $\xRightarrow{\text{odd}}$  don't output

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| word: | e | a | e | b | e | b | a | e | e | e | b  | a  | e  | e  |



Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| word: | e | a | e | b | e | b | a | e | e | e | b  | a  | e  | e  |



dynamic membership:  $w \in L?$

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| word: | e | a | e | b | e | b | a | e | e | e | b  | a  | e  | e  |



dynamic membership:  $w \in L?$  ✓  $\xRightarrow{\text{even}}$  output  $w$

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| word: | e | a | e | b | e | b | a | e | e | e | b  | e  | e  | e  |



dynamic membership:  $w \in L?$  ✓  $\xRightarrow{\text{odd}}$  don't output

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| word: | e | a | e | b | e | b | a | e | e | e | e  | e  | e  | e  |



dynamic membership:  $w \in L?$    $\xRightarrow{\text{even}}$  output  $w$

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| word: | e | a | e | b | e | b | a | e | e | e | e  | e  | e  | e  |



dynamic membership:  $w \in L?$  ✓  $\xRightarrow{\text{odd}}$  don't output

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| word: | e | a | e | b | e | b | a | e | e | e | e  | e  | e  | e  |



dynamic membership:  $w \in L?$  ✓  $\xRightarrow{\text{even}}$  output  $w$

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| word: | e | a | e | b | e | b | a | e | e | e | e  | e  | e  | e  |



dynamic membership:  $w \in L?$  ✓  $\xRightarrow{\text{odd}}$  don't output

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| word: | e | a | e | b | e | b | e | e | e | e | e  | e  | e  | e  |



dynamic membership:  $w \in L?$  ✓  $\xRightarrow{\text{even}}$  output  $w$

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| word: | e | a | e | b | e | e | e | e | e | e | e  | e  | e  | e  |



dynamic membership:  $w \in L?$  ✓  $\xRightarrow{\text{odd}}$  don't output

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| word: | e | a | e | b | e | e | e | e | e | e | e  | e  | e  | e  |



dynamic membership:  $w \in L?$  ✓  $\xRightarrow{\text{even}}$  output  $w$

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| word: | e | a | e | e | e | e | e | e | e | e | e  | e  | e  | e  |



dynamic membership:  $w \in L?$  **X**  $\Rightarrow$  don't output

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| word: | e | a | e | b | e | e | e | e | e | e | e  | e  | e  | e  |



odd length  $\Rightarrow$  output  $w$

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| word: | e | a | e | b | e | b | a | e | e | e | e  | e  | e  | e  |



odd length  $\Rightarrow$  output  $w$

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| word: | e | a | e | b | e | b | a | e | e | e | e  | e  | e  | e  |



odd length  $\Rightarrow$  output  $w$

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| word: | e | a | e | b | e | b | a | e | e | e | b  | e  | e  | e  |



odd length  $\Rightarrow$  output  $w$

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| word: | e | a | e | b | e | b | a | e | e | e | b  | a  | e  | e  |

odd length  $\Rightarrow$  output  $w$

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
| word: | e | a | e | b | e | b | a | e | e | e | b  | a  | e  | a  |

right border reached  $\Rightarrow$  increase left pointer

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| word: | e | a | e | b | e | b | a | e | e | e | b  | a  | e  | a  |

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |   |   |   |   |   |   |   |   |   |    |    |    |    |
|-------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|
|       | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| word: | e | a | e | b | e | b | a | e | e | e | b  | a  | e  | a  |



Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |     |   |   |   |   |   |   |   |   |    |    |     |    |
|-------|---|-----|---|---|---|---|---|---|---|---|----|----|-----|----|
|       | 0 | 1   | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12  | 13 |
| word: | e | a   | e | b | e | b | a | e | e | e | b  | a  | e   | a  |
|       |   | ... |   |   |   |   |   |   |   |   |    |    | ... |    |

Use even-odd technique to decrease right (and increase left) pointers

## Simple Algorithm

Use dynamic membership to solve problem for *extensible ZG languages*

### Example (Simple Algorithm)

Consider language  $L_{ab}$  over  $\Sigma = \{a, b, c\}$  that contains the words  $w$  where an  $a$  is before a  $b$ , or  $|w|_a \geq 2$  or  $|w|_b \geq 2$ .

|       |   |     |   |   |   |   |   |   |   |   |    |    |    |     |
|-------|---|-----|---|---|---|---|---|---|---|---|----|----|----|-----|
|       | 0 | 1   | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13  |
| word: | e | a   | e | b | e | b | a | e | e | e | b  | a  | e  | a   |
|       |   | ... |   |   |   |   |   |   |   |   |    |    |    | ... |

Use even-odd technique to decrease right (and increase left) pointers

*Problems:* non-constant additional memory and not interruptible for updates

# Main Result

## Semi-Extensible ZG Languages

Superclass of above result

Not necessarily extensible for words where non-neutral letters are *rare*

# Main Result

## Semi-Extensible ZG Languages

Superclass of above result

Not necessarily extensible for words where non-neutral letters are *rare*

Algorithm traverses word with two pointers  $l$  and  $r$  using

- initial occurrence lists
- two background traversals (to build occurrence lists of factors)
- data structures
  - ▶ left information (when decreasing the right pointer)
  - ▶ right information (when increasing the left pointer)

## Theorem

*Enumeration of semi-extensible ZG languages can be solved with constant-time updates, constant delay enumeration, linear preprocessing and constant additional memory.*

## *Enumeration Algorithms*

- Allow updates
- Constant time updates/delay
- Occurrence lists
- Dynamic membership
- Semi-extensible ZG languages

## *Enumeration Algorithms*

- Allow updates
- Constant time updates/delay
- Occurrence lists
- Dynamic membership
- Semi-extensible ZG languages

Thank you for your attention!

Backup



Problem:

- maintain binary word under updates
- ask if a prefix contains a 0

*Conjecture:* no  $O(1)$  data structure for this problem

Weaker version of *priority queue problem* or *successor search problem*

### Definition (Semi-extensible ZG language, algebraic def.)

A regular language  $L$  is *semi-extensible ZG* if it is in ZG,  
i.e. its syntactic monoid  $M$  satisfies the ZG equation:

$$\text{for all } x, y \in M, \text{ we have } yx^{\omega+1} = x^{\omega+1}y, \quad (\text{ZG})$$

and further  $L$  is *semi-extensible* in the following sense:

for all words  $x, y, z \in \Sigma^*$  and all non-neutral letters  $a$ , if  $y \in L$  then  $xyza^\omega \in L$ .