

Bounded-delay enumeration of regular languages

Antoine Amarilli, Mikaël Monet

March 5, 2024

Inria



Who

Joint work with **Mikaël Monet** – thanks to him for preparing these slides :)



<https://arxiv.org/abs/2209.14878>

Presented at **STACS'23**

Introduction

Main results

Proof of the lower bound

Proof (sketch) of the upper bound

Conclusion

Introduction

Gray code for n -bit words

- Gray code over n -bit words: a permutation

$$w_1, w_2, \dots, w_{2^n}$$

of $(a + b)^n$ such that w_i, w_{i+1} differ by exactly one bit.

Example: build the Reflected Binary Code (RBC) by induction:

Gray code for n -bit words

- Gray code over n -bit words: a permutation

$$w_1, w_2, \dots, w_{2^n}$$

of $(a + b)^n$ such that w_i, w_{i+1} differ by exactly one bit.

Example: build the Reflected Binary Code (RBC) by induction:

- for $n = 0$, simply ϵ

Gray code for n -bit words

- Gray code over n -bit words: a permutation

$$w_1, w_2, \dots, w_{2^n}$$

of $(a + b)^n$ such that w_i, w_{i+1} differ by exactly one bit.

Example: build the Reflected Binary Code (RBC) by induction:

- for $n = 0$, simply ϵ
- given the RBC w_1, \dots, w_{2^n} for n -bit words, we build the RBC $w'_1, \dots, w'_{2^{n+1}}$ for $(n + 1)$ -bit words:

Gray code for n -bit words

- Gray code over n -bit words: a permutation

$$w_1, w_2, \dots, w_{2^n}$$

of $(a + b)^n$ such that w_i, w_{i+1} differ by exactly one bit.

Example: build the Reflected Binary Code (RBC) by induction:

- for $n = 0$, simply ϵ
- given the RBC w_1, \dots, w_{2^n} for n -bit words, we build the RBC $w'_1, \dots, w'_{2^{n+1}}$ for $(n + 1)$ -bit words:

$$\begin{array}{c} w_1 \\ \vdots \\ w_{2^n} \end{array}$$

Gray code for n -bit words

- Gray code over n -bit words: a permutation

$$w_1, w_2, \dots, w_{2^n}$$

of $(a + b)^n$ such that w_i, w_{i+1} differ by exactly one bit.

Example: build the Reflected Binary Code (RBC) by induction:

- for $n = 0$, simply ϵ
- given the RBC w_1, \dots, w_{2^n} for n -bit words, we build the RBC $w'_1, \dots, w'_{2^{n+1}}$ for $(n + 1)$ -bit words:

$$\begin{array}{c} w_1 \\ \vdots \\ w_{2^n} \\ \hline w_{2^n} \\ \vdots \\ w_1 \end{array}$$

Gray code for n -bit words

- Gray code over n -bit words: a permutation

$$w_1, w_2, \dots, w_{2^n}$$

of $(a + b)^n$ such that w_i, w_{i+1} differ by exactly one bit.

Example: build the Reflected Binary Code (RBC) by induction:

- for $n = 0$, simply ϵ
- given the RBC w_1, \dots, w_{2^n} for n -bit words, we build the RBC $w'_1, \dots, w'_{2^{n+1}}$ for $(n + 1)$ -bit words:

$$\begin{array}{c} a w_1 \\ \vdots \\ a w_{2^n} \\ \hline w_{2^n} \\ \vdots \\ w_1 \end{array}$$

Gray code for n -bit words

- Gray code over n -bit words: a permutation

$$w_1, w_2, \dots, w_{2^n}$$

of $(a + b)^n$ such that w_i, w_{i+1} differ by exactly one bit.

Example: build the **Reflected Binary Code** (RBC) by induction:

- for $n = 0$, simply ϵ
- given the RBC w_1, \dots, w_{2^n} for n -bit words, we build the RBC $w'_1, \dots, w'_{2^{n+1}}$ for $(n + 1)$ -bit words:

$$\begin{array}{c} a w_1 \\ \vdots \\ a w_{2^n} \\ \hline b w_{2^n} \\ \vdots \\ b w_1 \end{array}$$

Gray code for n -bit words

- Gray code over n -bit words: a permutation

$$w_1, w_2, \dots, w_{2^n}$$

of $(a + b)^n$ such that w_i, w_{i+1} differ by exactly one bit.

Example: build the **Reflected Binary Code** (RBC) by induction:

- for $n = 0$, simply ϵ
- given the RBC w_1, \dots, w_{2^n} for n -bit words, we build the RBC $w'_1, \dots, w'_{2^{n+1}}$ for $(n + 1)$ -bit words:

$$\begin{array}{rcl} w'_1 & = & a w_1 \\ \vdots & & \vdots \vdots \\ & & a w_{2^n} \\ & & \text{-----} \\ & & b w_{2^n} \\ \vdots & & \vdots \vdots \\ w'_{2^{n+1}} & = & b w_1 \end{array}$$

Gray code for languages

- Concatenate Gray codes for $n = 0, 1, 2, \dots$: we obtain a permutation w_1, w_2, \dots of $(a + b)^*$ where consecutive words are at **Levenshtein distance** one.

Gray code for languages

- Concatenate Gray codes for $n = 0, 1, 2, \dots$: we obtain a permutation w_1, w_2, \dots of $(a + b)^*$ where consecutive words are at **Levenshtein distance** one.
- In general, let $L \subseteq \Sigma^*$ be any language over some alphabet Σ . We say that L is **1-orderable for the Levenshtein distance** if there exists a permutation w_1, w_2, \dots of L such that consecutive words are at Levenshtein distance 1.

Gray code for languages

- Concatenate Gray codes for $n = 0, 1, 2, \dots$: we obtain a permutation w_1, w_2, \dots of $(a + b)^*$ where consecutive words are at **Levenshtein distance** one.
- In general, let $L \subseteq \Sigma^*$ be any language over some alphabet Σ . We say that L is **1-orderable for the Levenshtein distance** if there exists a permutation w_1, w_2, \dots of L such that consecutive words are at Levenshtein distance 1.
- **Examples:** Are these languages 1-orderable for the Levenshtein distance?
 - a^*

Gray code for languages

- Concatenate Gray codes for $n = 0, 1, 2, \dots$: we obtain a permutation w_1, w_2, \dots of $(a + b)^*$ where consecutive words are at **Levenshtein distance** one.
- In general, let $L \subseteq \Sigma^*$ be any language over some alphabet Σ . We say that L is **1-orderable for the Levenshtein distance** if there exists a permutation w_1, w_2, \dots of L such that consecutive words are at Levenshtein distance 1.
- **Examples:** Are these languages 1-orderable for the Levenshtein distance?
 - a^* **yes**

Gray code for languages

- Concatenate Gray codes for $n = 0, 1, 2, \dots$: we obtain a permutation w_1, w_2, \dots of $(a + b)^*$ where consecutive words are at **Levenshtein distance** one.
- In general, let $L \subseteq \Sigma^*$ be any language over some alphabet Σ . We say that L is **1-orderable for the Levenshtein distance** if there exists a permutation w_1, w_2, \dots of L such that consecutive words are at Levenshtein distance 1.
- **Examples:** Are these languages 1-orderable for the Levenshtein distance?
 - a^* **yes**
 - a^*b^*

Gray code for languages

- Concatenate Gray codes for $n = 0, 1, 2, \dots$: we obtain a permutation w_1, w_2, \dots of $(a + b)^*$ where consecutive words are at **Levenshtein distance** one.
- In general, let $L \subseteq \Sigma^*$ be any language over some alphabet Σ . We say that L is **1-orderable for the Levenshtein distance** if there exists a permutation w_1, w_2, \dots of L such that consecutive words are at Levenshtein distance 1.
- **Examples:** Are these languages 1-orderable for the Levenshtein distance?
 - a^* **yes**
 - a^*b^* **yes (BLACKBOARD)**

Gray code for languages

- Concatenate Gray codes for $n = 0, 1, 2, \dots$: we obtain a permutation w_1, w_2, \dots of $(a + b)^*$ where consecutive words are at **Levenshtein distance** one.
- In general, let $L \subseteq \Sigma^*$ be any language over some alphabet Σ . We say that L is **1-orderable for the Levenshtein distance** if there exists a permutation w_1, w_2, \dots of L such that consecutive words are at Levenshtein distance 1.
- **Examples:** Are these languages 1-orderable for the Levenshtein distance?
 - a^* **yes**
 - a^*b^* **yes (BLACKBOARD)**
 - $(aa)^*$

Gray code for languages

- Concatenate Gray codes for $n = 0, 1, 2, \dots$: we obtain a permutation w_1, w_2, \dots of $(a + b)^*$ where consecutive words are at **Levenshtein distance** one.
- In general, let $L \subseteq \Sigma^*$ be any language over some alphabet Σ . We say that L is **1-orderable for the Levenshtein distance** if there exists a permutation w_1, w_2, \dots of L such that consecutive words are at Levenshtein distance 1.
- **Examples:** Are these languages 1-orderable for the Levenshtein distance?
 - a^* **yes**
 - a^*b^* **yes (BLACKBOARD)**
 - $(aa)^*$ **no**

Orderability for the Levenshtein distance

Definition

We say that $L \subseteq \Sigma^*$ is *d -orderable for the Levenshtein distance* if there exists a permutation w_1, w_2, \dots of L such that any two consecutive words are at Levenshtein distance at most d .

Orderability for the Levenshtein distance

Definition

We say that $L \subseteq \Sigma^*$ is *d -orderable for the Levenshtein distance* if there exists a permutation w_1, w_2, \dots of L such that any two consecutive words are at Levenshtein distance at most d .

Definition

We say that $L \subseteq \Sigma^*$ is *orderable for the Levenshtein distance* if there exists $d \in \mathbb{N}$ such that L is d -orderable for the Levenshtein distance.

Orderability for the Levenshtein distance

Definition

We say that $L \subseteq \Sigma^*$ is *d -orderable for the Levenshtein distance* if there exists a permutation w_1, w_2, \dots of L such that any two consecutive words are at Levenshtein distance at most d .

Definition

We say that $L \subseteq \Sigma^*$ is *orderable for the Levenshtein distance* if there exists $d \in \mathbb{N}$ such that L is d -orderable for the Levenshtein distance.

Examples: Are these orderable for the Levenshtein distance?

- for $k \in \mathbb{N}$, the language $(a^k)^*$

Orderability for the Levenshtein distance

Definition

We say that $L \subseteq \Sigma^*$ is *d -orderable for the Levenshtein distance* if there exists a permutation w_1, w_2, \dots of L such that any two consecutive words are at Levenshtein distance at most d .

Definition

We say that $L \subseteq \Sigma^*$ is *orderable for the Levenshtein distance* if there exists $d \in \mathbb{N}$ such that L is d -orderable for the Levenshtein distance.

Examples: Are these orderable for the Levenshtein distance?

- for $k \in \mathbb{N}$, the language $(a^k)^*$ *yes*

Orderability for the Levenshtein distance

Definition

We say that $L \subseteq \Sigma^*$ is *d -orderable for the Levenshtein distance* if there exists a permutation w_1, w_2, \dots of L such that any two consecutive words are at Levenshtein distance at most d .

Definition

We say that $L \subseteq \Sigma^*$ is *orderable for the Levenshtein distance* if there exists $d \in \mathbb{N}$ such that L is d -orderable for the Levenshtein distance.

Examples: Are these orderable for the Levenshtein distance?

- for $k \in \mathbb{N}$, the language $(a^k)^*$ *yes*
- $a^* + b^*$

Orderability for the Levenshtein distance

Definition

We say that $L \subseteq \Sigma^*$ is *d -orderable for the Levenshtein distance* if there exists a permutation w_1, w_2, \dots of L such that any two consecutive words are at Levenshtein distance at most d .

Definition

We say that $L \subseteq \Sigma^*$ is *orderable for the Levenshtein distance* if there exists $d \in \mathbb{N}$ such that L is d -orderable for the Levenshtein distance.

Examples: Are these orderable for the Levenshtein distance?

- for $k \in \mathbb{N}$, the language $(a^k)^*$ **yes**
- $a^* + b^*$ **no (BLACKBOARD)**

We extend these definitions to other distances:

- the **push-pop distance**. Defined like the Levenshtein distance, but the basic operations are:
 - $\text{popL}()$ and $\text{popR}()$, to delete the last (resp., the first) letter of the word; and
 - $\text{pushL}(\alpha)$ and $\text{pushR}(\alpha)$ for $\alpha \in \Sigma$, to add the letter α at the beginning (resp., at the end) of the word.

Other distances: definitions

We extend these definitions to other distances:

- the **push-pop distance**. Defined like the Levenshtein distance, but the basic operations are:
 - $\text{popL}()$ and $\text{popR}()$, to delete the last (resp., the first) letter of the word; and
 - $\text{pushL}(\alpha)$ and $\text{pushR}(\alpha)$ for $\alpha \in \Sigma$, to add the letter α at the beginning (resp., at the end) of the word.
- the **push-pop-right distance**. Defined like the push-pop distance, but only allows $\text{popR}()$ and $\text{pushR}(\alpha)$ for $\alpha \in \Sigma$.

Other distances: first observations

languages orderable for push-pop-right \subseteq languages orderable for push-pop \subseteq languages orderable for Levenshtein.

Other distances: first observations

languages orderable for push-pop-right \subseteq languages orderable for push-pop \subseteq languages orderable for Levenshtein.

Are these inclusions strict?

Other distances: first observations

languages orderable for push-pop-right \subseteq languages orderable for push-pop \subseteq languages orderable for Levenshtein.

Are these inclusions strict?

- $(\epsilon + a)b^*$

Other distances: first observations

languages orderable for push-pop-right \subseteq languages orderable for push-pop \subseteq languages orderable for Levenshtein.

Are these inclusions strict?

- $(\epsilon + a)b^*$ orderable for push-pop (hence for Levenshtein).

Other distances: first observations

languages orderable for push-pop-right \subseteq languages orderable for push-pop \subseteq languages orderable for Levenshtein.

Are these inclusions strict?

- $(\epsilon + a)b^*$ orderable for push-pop (hence for Levenshtein).
For push-pop-right?

Other distances: first observations

languages orderable for push-pop-right $\not\subseteq$ languages orderable for push-pop \subseteq languages orderable for Levenshtein.

Are these inclusions strict?

- $(\epsilon + a)b^*$ orderable for push-pop (hence for Levenshtein).
For push-pop-right? no

Other distances: first observations

languages orderable for push-pop-right $\not\subseteq$ languages orderable for push-pop \subseteq languages orderable for Levenshtein.

Are these inclusions strict?

- $(\epsilon + a)b^*$ orderable for push-pop (hence for Levenshtein).
For push-pop-right? no
- a^*b^* orderable for Levenshtein (prev slides).

Other distances: first observations

languages orderable for push-pop-right $\not\subseteq$ languages orderable for push-pop \subseteq languages orderable for Levenshtein.

Are these inclusions strict?

- $(\epsilon + a)b^*$ orderable for push-pop (hence for Levenshtein).
For push-pop-right? no
- a^*b^* orderable for Levenshtein (prev slides).
For push-pop?

Other distances: first observations

languages orderable for push-pop-right $\not\subseteq$ languages orderable for push-pop \subseteq languages orderable for Levenshtein.

Are these inclusions strict?

- $(\epsilon + a)b^*$ orderable for push-pop (hence for Levenshtein).
For push-pop-right? no
- a^*b^* orderable for Levenshtein (prev slides).
For push-pop? yes (BLACKBOARD)

Other distances: first observations

languages orderable for push-pop-right $\not\subseteq$ languages orderable for push-pop \subseteq languages orderable for Levenshtein.

Are these inclusions strict?

- $(\epsilon + a)b^*$ orderable for push-pop (hence for Levenshtein).
For push-pop-right? no
- a^*b^* orderable for Levenshtein (prev slides).
For push-pop? yes (BLACKBOARD)
- $\{a^n(b+c)a^n \mid n \in \mathbb{N}\}$ orderable for Levenshtein.

Other distances: first observations

languages orderable for push-pop-right $\not\subseteq$ languages orderable for push-pop \subseteq languages orderable for Levenshtein.

Are these inclusions strict?

- $(\epsilon + a)b^*$ orderable for push-pop (hence for Levenshtein).
For push-pop-right? no
- a^*b^* orderable for Levenshtein (prev slides).
For push-pop? yes (BLACKBOARD)
- $\{a^n(b+c)a^n \mid n \in \mathbb{N}\}$ orderable for Levenshtein.
For push-pop?

Other distances: first observations

languages orderable for push-pop-right $\not\subseteq$ languages orderable for push-pop \subseteq languages orderable for Levenshtein.

Are these inclusions strict?

- $(\epsilon + a)b^*$ orderable for push-pop (hence for Levenshtein).
For push-pop-right? no
- a^*b^* orderable for Levenshtein (prev slides).
For push-pop? yes (BLACKBOARD)
- $\{a^n(b+c)a^n \mid n \in \mathbb{N}\}$ orderable for Levenshtein.
For push-pop? no

Questions

We focus on regular languages

We focus on **regular languages**

- What are the regular languages that are orderable:
 - for the Levenshtein distance?
 - for the push-pop distance?
 - for the push-pop-right distance?

We focus on **regular languages**

- What are the regular languages that are orderable:
 - for the Levenshtein distance?
 - for the push-pop distance?
 - for the push-pop-right distance?
- Can we recognize them? (e.g., given a DFA)

We focus on **regular languages**

- What are the regular languages that are orderable:
 - for the Levenshtein distance?
 - for the push-pop distance?
 - for the push-pop-right distance?
- Can we recognize them? (e.g., given a DFA)
- Can we always partition a regular language into a finite number of orderable languages? (as in $a^* + b^*$)

We focus on **regular languages**

- What are the regular languages that are orderable:
 - for the Levenshtein distance?
 - for the push-pop distance?
 - for the push-pop-right distance?
- Can we recognize them? (e.g., given a DFA)
- Can we always partition a regular language into a finite number of orderable languages? (as in $a^* + b^*$)
- When L is orderable, can we design an **enumeration algorithm** for it? With what **delay**? (poly, constant?)

Main results

Main results (Levenshtein and push-pop)

Let L be regular. We show:

- There exists $t \in \mathbb{N}$ and regular languages L_1, \dots, L_t such that

$$L = L_1 \sqcup \dots \sqcup L_t$$

and each L_i is orderable for the **push-pop** distance

Main results (Levenshtein and push-pop)

Let L be regular. We show:

- There exists $t \in \mathbb{N}$ and regular languages L_1, \dots, L_t such that

$$L = L_1 \sqcup \dots \sqcup L_t$$

and each L_i is orderable for the **push-pop** distance

- This t is optimal, even for the Levenshtein distance: L cannot be partitioned into less than t orderable languages for the **Levenshtein** distance.

Main results (Levenshtein and push-pop)

Let L be regular. We show:

- There exists $t \in \mathbb{N}$ and regular languages L_1, \dots, L_t such that

$$L = L_1 \sqcup \dots \sqcup L_t$$

and each L_i is orderable for the **push-pop** distance

- This t is optimal, even for the Levenshtein distance: L cannot be partitioned into less than t orderable languages for the **Levenshtein** distance.
 - This shows L is orderable for Levenshtein **iff** it is for push-pop!

Main results (Levenshtein and push-pop)

Let L be regular. We show:

- There exists $t \in \mathbb{N}$ and regular languages L_1, \dots, L_t such that

$$L = L_1 \sqcup \dots \sqcup L_t$$

and each L_i is orderable for the **push-pop** distance

- This t is optimal, even for the Levenshtein distance: L cannot be partitioned into less than t orderable languages for the **Levenshtein** distance.
 - This shows L is orderable for Levenshtein **iff** it is for push-pop!
- When L is orderable for push-pop then, in a suitable pointer machine model, we have an algorithm that outputs **push-pop edit scripts** to enumerate L , with **bounded delay** (i.e., independent from the current word length)

Enumeration algorithms with push-pop edit scripts

Let L regular, e.g., $(\epsilon + a)b^*$. **GOAL**: enumerate L with a delay that is independent from the length of the current word.

Enumeration algorithms with push-pop edit scripts

Let L regular, e.g., $(\epsilon + a)b^*$. **GOAL**: enumerate L (in a certain sense) with a delay that is independent from the length of the current word.

Enumeration algorithms with push-pop edit scripts

Let L regular, e.g., $(\epsilon + a)b^*$. **GOAL**: enumerate L (in a certain sense) with a delay that is independent from the length of the current word. Example of a **push-pop** program for $(\epsilon + a)b^*$:

```
int main{
    output();
    while (true) {
        pushR(b); output();
        pushL(a); output();
        popL();
    }
}
```

The current word w_i is maintained on a (doubly-ended) queue (**BLACKBOARD**)

Enumeration algorithms with push-pop edit scripts

Let L regular, e.g., $(\epsilon + a)b^*$. **GOAL**: enumerate L (in a certain sense) with a delay that is independent from the length of the current word. Example of a **push-pop** program for $(\epsilon + a)b^*$:

```
int main{  
    output();  
    while (true) {  
        pushR(b); output();  
        pushL(a); output();  
        popL();  
    }  
}
```

The current word w_i is maintained on a (doubly-ended) queue (**BLACKBOARD**)

An **edit script** is a sequence of push or pop operations executed between two `output()` instructions. This push-pop program **enumerates** $(\epsilon + a)b^*$ with bounded delay.

Proof of the lower bound

Theorem

For a regular language L , there exist regular L_1, \dots, L_t such that

$$L = L_1 \sqcup \dots \sqcup L_t$$

and each L_i is orderable for the push-pop distance. Moreover L cannot be partitioned into less than t orderable languages for the Levenshtein distance.

Theorem

For a regular language L , there exist regular L_1, \dots, L_t such that

$$L = L_1 \sqcup \dots \sqcup L_t$$

and each L_i is orderable for the push-pop distance. Moreover L cannot be partitioned into less than t orderable languages for the Levenshtein distance.

We will now define this number t and show that it is optimal

Connectivity and compatibility of loopable states

Let $A = (Q, \Sigma, q_0, F, \delta)$ be a DFA for L . For $q \in Q$, define A_q to be A where the initial state and final state is q .

Definition: loopable state

A state $q \in Q$ is **loopable** if $L(A_q) \neq \{\epsilon\}$. In other words, when there is a non-empty run that starts and ends at q .

Connectivity and compatibility of loopable states

Let $A = (Q, \Sigma, q_0, F, \delta)$ be a DFA for L . For $q \in Q$, define A_q to be A where the initial state and final state is q .

Definition: loopable state

A state $q \in Q$ is **loopable** if $L(A_q) \neq \{\epsilon\}$. In other words, when there is a non-empty run that starts and ends at q .

Definition: connectivity

Two loopable states $q, q' \in Q$ are **connected** when there is a directed path in A from q to q' , or a directed path in A from q' to q .

Connectivity and compatibility of loopable states

Let $A = (Q, \Sigma, q_0, F, \delta)$ be a DFA for L . For $q \in Q$, define A_q to be A where the initial state and final state is q .

Definition: loopable state

A state $q \in Q$ is **loopable** if $L(A_q) \neq \{\epsilon\}$. In other words, when there is a non-empty run that starts and ends at q .

Definition: connectivity

Two loopable states $q, q' \in Q$ are **connected** when there is a directed path in A from q to q' , or a directed path in A from q' to q .

Definition: compatibility

Two loopable states $q, q' \in Q$ are **compatible** when $L(A_q) \cap L(A_{q'}) \neq \{\epsilon\}$.

Interchangeability of loopable states

Note: The connectivity and compatibility relations of loopable states are reflexive but not transitive

Interchangeability of loopable states

Note: The connectivity and compatibility relations of loopable states are reflexive but not transitive

Definition: interchangeability

Interchangeability is the equivalence relation on loopable states that is defined to be the transitive closure of the union of the connectivity and compatibility relations.

In other words, two loopable states $q, q' \in Q$ are **interchangeable** if there is a sequence $q = q_0, \dots, q_n = q'$ of loopable states such that for all $0 \leq i < n$, the states q_i and q_{i+1} are either connected or compatible.

Interchangeability of loopable states

Note: The connectivity and compatibility relations of loopable states are reflexive but not transitive

Definition: interchangeability

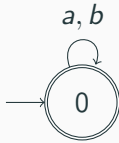
Interchangeability is the equivalence relation on loopable states that is defined to be the transitive closure of the union of the connectivity and compatibility relations.

In other words, two loopable states $q, q' \in Q$ are **interchangeable** if there is a sequence $q = q_0, \dots, q_n = q'$ of loopable states such that for all $0 \leq i < n$, the states q_i and q_{i+1} are either connected or compatible.

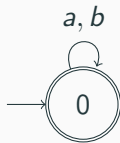
We then **define t to be the number of interchangeable classes**

Some examples follow

Example: $(a + b)^*$

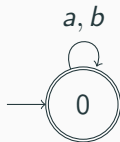


Example: $(a + b)^*$



- Loopable states: 0

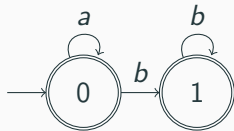
Example: $(a + b)^*$



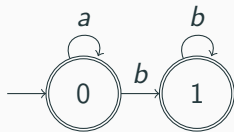
- Loopable states: 0

$\Rightarrow t = 1$

Example: a^*b^*

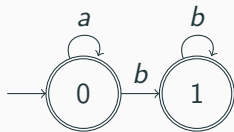


Example: a^*b^*



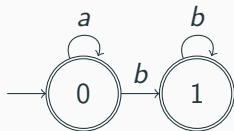
- Loopable states: 0 and 1

Example: a^*b^*



- Loopable states: 0 and 1
- 0 and 1 are connected, hence interchangeable

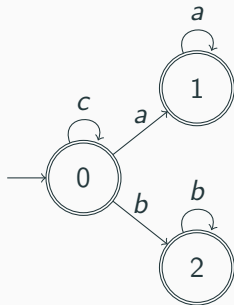
Example: a^*b^*



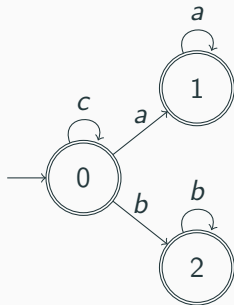
- Loopable states: 0 and 1
- 0 and 1 are connected, hence interchangeable

$\Rightarrow t = 1$

Example: $c^*a^* + c^*b^*$

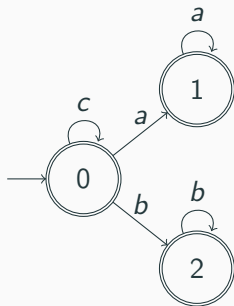


Example: $c^*a^* + c^*b^*$



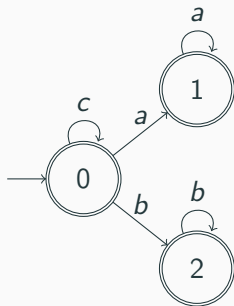
- Loopable states: 0, 1 and 2

Example: $c^*a^* + c^*b^*$



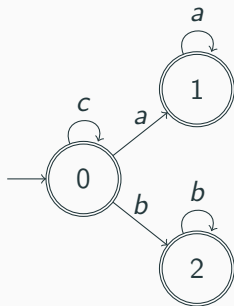
- Loopable states: 0, 1 and 2
- 0 and 1 are connected hence interchangeable

Example: $c^*a^* + c^*b^*$



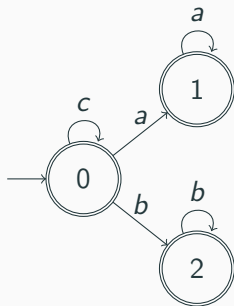
- Loopable states: 0, 1 and 2
- 0 and 1 are connected hence interchangeable
- 0 and 2 are connected hence interchangeable

Example: $c^*a^* + c^*b^*$



- Loopable states: 0, 1 and 2
- 0 and 1 are connected hence interchangeable
- 0 and 2 are connected hence interchangeable
- so 1 and 2 are also interchangeable

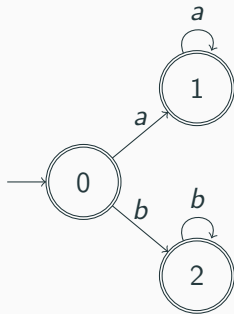
Example: $c^*a^* + c^*b^*$



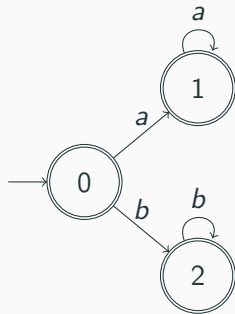
- Loopable states: 0, 1 and 2
- 0 and 1 are connected hence interchangeable
- 0 and 2 are connected hence interchangeable
- so 1 and 2 are also interchangeable

⇒ $t = 1$

Example: $a^* + b^*$

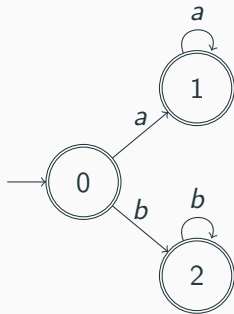


Example: $a^* + b^*$



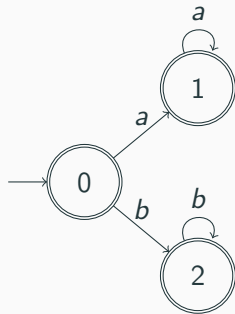
- Loopable states: 1 and 2

Example: $a^* + b^*$



- Loopable states: 1 and 2
- 1 and 2 are neither connected, nor compatible, so they are not interchangeable

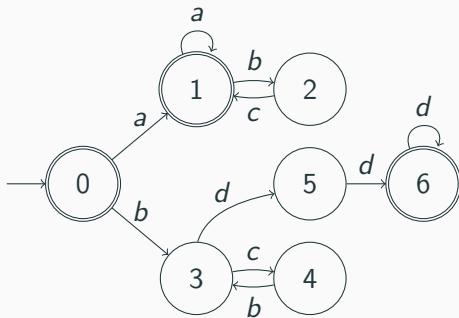
Example: $a^* + b^*$



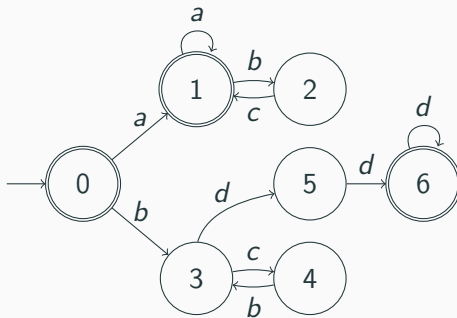
- Loopable states: 1 and 2
- 1 and 2 are neither connected, nor compatible, so they are not interchangeable

$\Rightarrow t = 2$

Example: $a(a + bc)^* + b(cb)^* ddd^*$

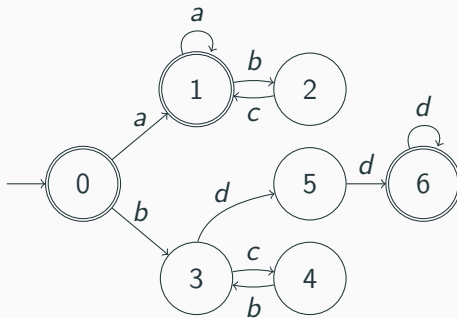


Example: $a(a + bc)^* + b(cb)^* ddd^*$



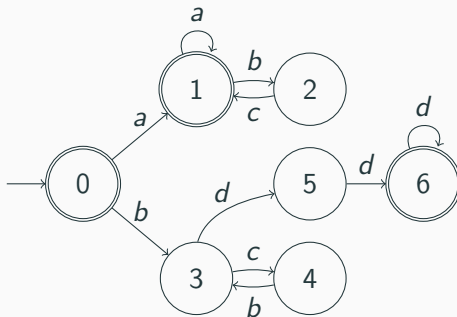
- Loopable states: 1, 2, 3, 4 and 6

Example: $a(a + bc)^* + b(cb)^* ddd^*$



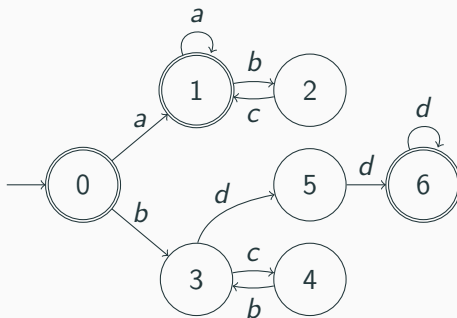
- Loopable states: 1, 2, 3, 4 and 6
- 1 and 2 are connected hence interchangeable

Example: $a(a + bc)^* + b(cb)^* ddd^*$



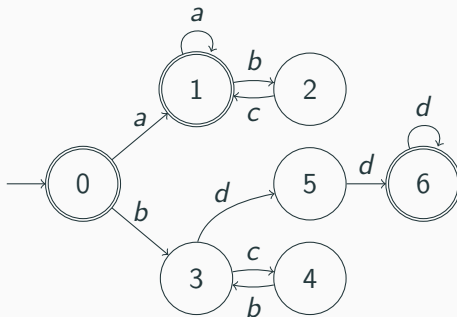
- Loopable states: 1, 2, 3, 4 and 6
- 1 and 2 are connected hence interchangeable
- 4, 3 and 6 are connected hence interchangeable

Example: $a(a + bc)^* + b(cb)^* ddd^*$



- Loopable states: 1, 2, 3, 4 and 6
- 1 and 2 are connected hence interchangeable
- 4, 3 and 6 are connected hence interchangeable
- 1 and 4 are compatible (with the word bc), hence interchangeable

Example: $a(a + bc)^* + b(cb)^* ddd^*$



- Loopable states: 1, 2, 3, 4 and 6
- 1 and 2 are connected hence interchangeable
- 4, 3 and 6 are connected hence interchangeable
- 1 and 4 are compatible (with the word bc), hence interchangeable

⇒ $t = 1$

The partition

Let $\mathcal{C}_1, \dots, \mathcal{C}_t$ be the interchangeability classes of loopable states of A .

Definition

For $1 \leq i \leq t$, define

$$L_i = \{w \in L(A) \mid \text{the run of } w \text{ goes through a state of } \mathcal{C}_i\}.$$

Also define

$$NL = \{w \in L(A) \mid \text{the run of } w \text{ does not use loopable states}\}.$$

The partition

Let $\mathcal{C}_1, \dots, \mathcal{C}_t$ be the interchangeability classes of loopable states of A .

Definition

For $1 \leq i \leq t$, define

$$L_i = \{w \in L(A) \mid \text{the run of } w \text{ goes through a state of } \mathcal{C}_i\}.$$

Also define

$$NL = \{w \in L(A) \mid \text{the run of } w \text{ does not use loopable states}\}.$$

Proposition

We have $L = NL \sqcup L_1 \sqcup \dots \sqcup L_t$

Proof: (BLACKBOARD)

Proof of the lower bound

Proposition

We have $L = \text{NL} \sqcup L_1 \sqcup \dots \sqcup L_t$

Proof of the lower bound

Proposition

We have $L = \text{NL} \sqcup L_1 \sqcup \dots \sqcup L_t$

Proposition

L cannot be partitioned into less than t languages that each are orderable for the Levenshtein distance.

Proof: we only do the case $t = 2$ and $\text{NL} = \emptyset$ (so $L = L_1 \sqcup L_2$).

We prove (BLACKBOARD): for any distance $d \in \mathbb{N}$, there is a threshold $I \in \mathbb{N}$ such that for any two words $u \in L_1$ and $v \in L_2$ with $i \neq j$ and $|u| \geq I$ and $|v| \geq I$, we have $\delta_{\text{Lev}}(u, v) > d$.

Indeed this is enough, using the same argument as for $a^* + b^*$

Proof (sketch) of the upper bound

Upper bound: existence of an ordering

We have shown:

Theorem

Given a DFA A , we can partition $L(A)$ into

$$L = L_1 \sqcup \dots \sqcup L_t$$

such that L cannot be partitioned into less than t orderable languages for the Levenshtein distance.

Upper bound: existence of an ordering

We have shown:

Theorem

Given a DFA A , we can partition $L(A)$ into

$$L = L_1 \sqcup \dots \sqcup L_t$$

such that L cannot be partitioned into less than t orderable languages for the Levenshtein distance.

We now show that each L_i is orderable for the push-pop distance

We want

We want to show:

Upper bound: existence

Let A be a DFA that has only one class of interchangeable loopable states.
Then $L(A)$ is orderable for the push-pop distance.

We want

We want to show:

Upper bound: existence

Let A be a DFA that has only one class of interchangeable loopable states.
Then $L(A)$ is orderable for the push-pop distance.

Let δ_{pp} denote the push-pop distance on Σ^*

Definition

Two words w, w' in a language L are d -connected in L if there exists a sequence w_0, \dots, w_n of words of L with $w_0 = w$, $w_n = w'$, and $\delta_{\text{pp}}(w_i, w_{i+1}) \leq d$ for all $0 \leq i < n$.

We say that L is d -connected if every pair of words of L is d -connected in L .

Definition

Two words w, w' in a language L are d -connected in L if there exists a sequence w_0, \dots, w_n of words of L with $w_0 = w$, $w_n = w'$, and $\delta_{\text{pp}}(w_i, w_{i+1}) \leq d$ for all $0 \leq i < n$.

We say that L is d -connected if every pair of words of L is d -connected in L .

In other words, the graph $G_{L,d}$ whose nodes are words of L and where two words are connected by an edge if they are at push-pop distance $\leq d$ is connex.

Definition

Two words w, w' in a language L are d -connected in L if there exists a sequence w_0, \dots, w_n of words of L with $w_0 = w$, $w_n = w'$, and $\delta_{\text{pp}}(w_i, w_{i+1}) \leq d$ for all $0 \leq i < n$.

We say that L is d -connected if every pair of words of L is d -connected in L .

In other words, the graph $G_{L,d}$ whose nodes are words of L and where two words are connected by an edge if they are at push-pop distance $\leq d$ is connex.

- **Note:** if L is d -orderable, then L is d -connected.

Definition

Two words w, w' in a language L are d -connected in L if there exists a sequence w_0, \dots, w_n of words of L with $w_0 = w$, $w_n = w'$, and $\delta_{\text{pp}}(w_i, w_{i+1}) \leq d$ for all $0 \leq i < n$.

We say that L is d -connected if every pair of words of L is d -connected in L .

In other words, the graph $G_{L,d}$ whose nodes are words of L and where two words are connected by an edge if they are at push-pop distance $\leq d$ is connex.

- **Note:** if L is d -orderable, then L is d -connected.
→ the converse is not true! E.g., $a^* + b^*$ is 1-connected (but not orderable)

Definition

Two words w, w' in a language L are d -connected in L if there exists a sequence w_0, \dots, w_n of words of L with $w_0 = w$, $w_n = w'$, and $\delta_{\text{pp}}(w_i, w_{i+1}) \leq d$ for all $0 \leq i < n$.

We say that L is d -connected if every pair of words of L is d -connected in L .

In other words, the graph $G_{L,d}$ whose nodes are words of L and where two words are connected by an edge if they are at push-pop distance $\leq d$ is connex.

- **Note:** if L is d -orderable, then L is d -connected.
- the converse is not true! E.g., $a^* + b^*$ is 1-connected (but not orderable)
- We show a kind of converse for finite languages in the next slide

d -connectivity implies $3d$ -orderability for finite languages

Proposition

If L is finite and d -connected then it is $3d$ -orderable.

d -connectivity implies $3d$ -orderability for finite languages

Proposition

If L is finite and d -connected then it is $3d$ -orderable.

Proof: take a spanning tree T of $G_{L,d}$. For $n \in T$, let $h(n)$ be its depth. Apply the following algorithm to the root of T :

```
void visit(node n){
    if(h(n) is even){
        enumerate(n);
        for (child ch of n)
            visit(ch);
    }
    if(h(n) is odd){
        for (child ch of n)
            visit(ch);
        enumerate(n);
    }
}
```

Example (BLACKBOARD)

d -connectivity implies $3d$ -orderability for finite languages

Proposition

If L is finite and d -connected then it is $3d$ -orderable.

Proof: take a spanning tree T of $G_{L,d}$. For $n \in T$, let $h(n)$ be its depth. Apply the following algorithm to the root of T :

```
void visit(node n){
    if(h(n) is even){
        enumerate(n);
        for (child ch of n)
            visit(ch);
    }
    if(h(n) is odd){
        for (child ch of n)
            visit(ch);
        enumerate(n);
    }
}
```

Example (BLACKBOARD)

Two consecutive nodes enumerated by this algorithm are at distance ≤ 3 in T , hence in $G_{L,d}$, hence the corresponding words are at distance $\leq 3d$ for δ_{pp} .

Using this for infinite languages

Definition

For L a language and $i, \ell \in \mathbb{N}$, define the i -th ℓ -stratum of L as

$$S_i = \{w \in L \mid (i-1)\ell \leq |w| < i\ell\}$$

Using this for infinite languages

Definition

For L a language and $i, \ell \in \mathbb{N}$, define the i -th ℓ -stratum of L as

$$S_i = \{w \in L \mid (i-1)\ell \leq |w| < i\ell\}$$

We can show (technical!):

Proposition

Let $L = L(A)$ with A having only one interchangeable class of loopable states. Let, letting $\ell = 8|A|^2$ and $d = 16|A|^2$, each S_i is d -connected.

Using this for infinite languages

Definition

For L a language and $i, \ell \in \mathbb{N}$, define the i -th ℓ -stratum of L as

$$S_i = \{w \in L \mid (i-1)\ell \leq |w| < i\ell\}$$

We can show (technical!):

Proposition

Let $L = L(A)$ with A having only one interchangeable class of loopable states. Let, letting $\ell = 8|A|^2$ and $d = 16|A|^2$, each S_i is d -connected.

We conclude by concatenating orderings for S_1, S_2, \dots obtained with the enumeration technique of the previous slide, with well-chosen starting and ending points (BLACKBOARD).

Conclusion

Main results (Levenshtein and push-pop)

Let L be regular. Then:

- There exists $t \in \mathbb{N}$ and regular languages L_1, \dots, L_t such that

$$L = L_1 \sqcup \dots \sqcup L_t$$

and each L_i is orderable for the **push-pop** distance

Main results (Levenshtein and push-pop)

Let L be regular. Then:

- There exists $t \in \mathbb{N}$ and regular languages L_1, \dots, L_t such that

$$L = L_1 \sqcup \dots \sqcup L_t$$

and each L_i is orderable for the **push-pop** distance

- This t is optimal, even for the Levenshtein distance: L cannot be partitioned into less than t orderable languages for the **Levenshtein** distance.

Main results (Levenshtein and push-pop)

Let L be regular. Then:

- There exists $t \in \mathbb{N}$ and regular languages L_1, \dots, L_t such that

$$L = L_1 \sqcup \dots \sqcup L_t$$

and each L_i is orderable for the **push-pop** distance

- This t is optimal, even for the Levenshtein distance: L cannot be partitioned into less than t orderable languages for the **Levenshtein** distance.
 - This shows that L is orderable for Levenshtein **iff** it is for push-pop!

Main results (Levenshtein and push-pop)

Let L be regular. Then:

- There exists $t \in \mathbb{N}$ and regular languages L_1, \dots, L_t such that

$$L = L_1 \sqcup \dots \sqcup L_t$$

and each L_i is orderable for the **push-pop** distance

- This t is optimal, even for the Levenshtein distance: L cannot be partitioned into less than t orderable languages for the **Levenshtein** distance.
 - This shows that L is orderable for Levenshtein **iff** it is for push-pop!
- When L is orderable for push-pop then, in a suitable pointer machine model, we have an algorithm that outputs **push-pop edit scripts** to enumerate L , with **bounded delay** (i.e., independent from the current word length)

Other results:

- It is *NP-hard*, given a DFA A such that $L(A)$ is orderable (for Levenshtein or push-pop), to *determine the minimal d* such that $L(A)$ is d -orderable.
- A regular language is partitionable into finitely many orderable languages for the push-pop-right distance if and only if it is *slender*.
 - Further, the optimal number of languages can also be computed from the automaton
 - We can also enumerate in bounded delay

Open questions and future work:

- Make the delay **polynomial** in $|A|$? (currently it is exp)
- What about enumeration in **radix order**? in lexicographic order?
- What about the **push-left pop-right distance**? the **padded Hamming** distance?
- What about regular **tree languages**?
- Other uses of the **enumeration model**?
- Implementation and real-life use-cases?

Open questions and future work:

- Make the delay **polynomial** in $|A|$? (currently it is exp)
- What about enumeration in **radix order**? in lexicographic order?
- What about the **push-left pop-right distance**? the **padded Hamming** distance?
- What about regular **tree languages**?
- Other uses of the **enumeration model**?
- Implementation and real-life use-cases?

Thanks for your attention!

Acknowledgements

Thanks to Mikaël Monet for preparing the original version of these slides.