Dynamic Membership for Regular Languages

Antoine Amarilli[1], Louis Jachiet[1], Charles Paperman[2]

January 13, 2022

[1]Télécom Paris

[2]Université de Lille

Problem: dynamic membership for regular languages

- Fix a regular language L
  - $\rightarrow$ E.g., $L = (ab)^*$
- Read an input word w with $n := |w|$
  - $\rightarrow$ E.g., $w = abbbab$

Problem: dynamic membership for regular languages

- Fix a regular language L
  - → E.g., $L = (ab)^*$
- Read an input word w with $n := |w|$
  - → E.g., $w = abbbab$
- Preprocess it in $O(n)$
  - → E.g., we have $w \notin L$

Problem: dynamic membership for regular languages

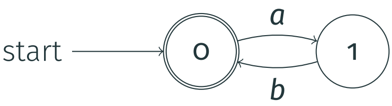
- Fix a regular language L
  - → E.g., $L = (ab)^*$
- Read an input word w with $n := |w|$
  - → E.g., $w = abbbab$
- Preprocess it in $O(n)$
  - → E.g., we have $w \notin L$
- Maintain the membership of w to L under substitution updates
  - → E.g., replace character at position 3 with a: we now have $w \in L$

Design choices

- Model: RAM model
  - Cell size in $\Theta(\log(n))$
  - Unit-cost arithmetics
- Updates: only substitutions (so n never changes)
  - Otherwise, already tricky to maintain the current state of the word
- Memory usage: always polynomial in n by definition of the model
  - Our upper bounds only need $O(n)$ space
  - The lower bounds apply without this assumption
- Preprocessing:
  - The upper bounds only need $O(n)$ preprocessing
  - The lower bounds apply without this assumption

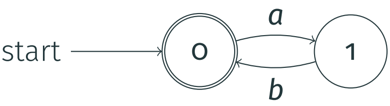
A general-purpose algorithm in $O(\log n)$

Fix the language $L = (ab)^*$:



A general-purpose algorithm in $O(\log n)$

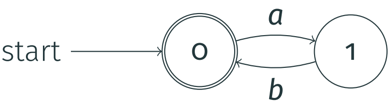
Fix the language $L = (ab)^*$:



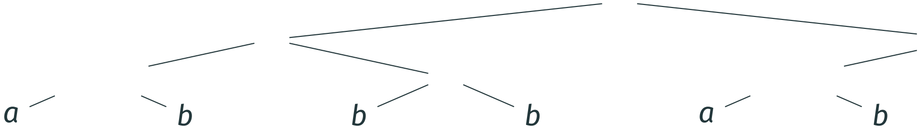
- Build a **balanced binary tree** on the input word $w = abaabb$

A general-purpose algorithm in $O(\log n)$

Fix the language $L = (ab)^*$:

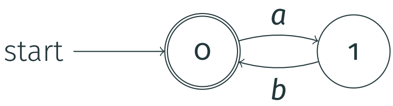


- Build a balanced binary tree on the input word $w = abaabb$

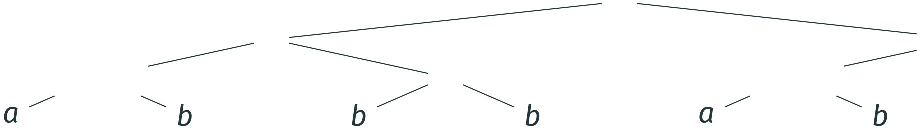


A general-purpose algorithm in $O(\log n)$

Fix the language $L = (ab)^*$:

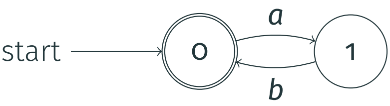


- Build a **balanced binary tree** on the input word $w = abaabb$
- Label each node n by the **transition monoid** element: all pairs $q \rightsquigarrow q'$ such that we can go from q to q' by reading the factor below n

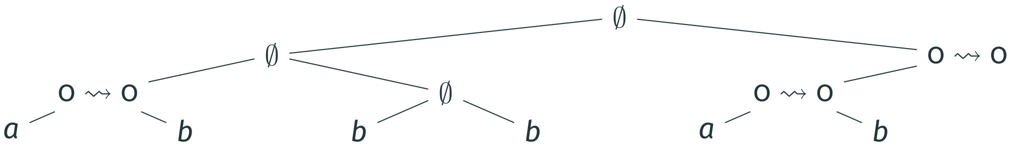


A general-purpose algorithm in $O(\log n)$

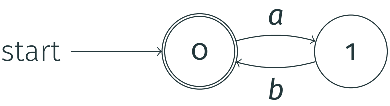
Fix the language $L = (ab)^*$:



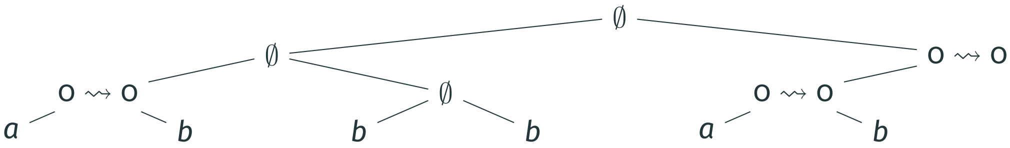
- Build a **balanced binary tree** on the input word $w = abaabb$
- Label each node n by the **transition monoid** element: all pairs $q \rightsquigarrow q'$ such that we can go from q to q' by reading the factor below n



A general-purpose algorithm in $O(\log n)$

Fix the language $L = (ab)^*$: 

- Build a balanced binary tree on the input word $w = abaabb$
- Label each node n by the transition monoid element: all pairs $q \rightsquigarrow q'$ such that we can go from q to q' by reading the factor below n



- The tree root describes if $w \in L$
- We can update the tree for each substitution in $O(\log n)$
- Can be improved to $O(\log n / \log \log n)$ with a log-ary tree

Can we do better than $O(\log n)$?

For our language $L = (ab)^*$ we can handle updates in $O(1)$:

Can we do better than $O(\log n)$?

For our language $L = (ab)^*$ we can handle updates in $O(1)$:

- Check that n is even
- Count violations: a's at even positions and b's at odd positions
- Maintain this counter in constant time
- We have $w \in L$ iff there are no violations

Can we do better than $O(\log n)$?

For our language $L = (ab)^*$ we can handle updates in $O(1)$:

- Check that n is even
- Count violations: a's at even positions and b's at odd positions
- Maintain this counter in constant time
- We have $w \in L$ iff there are no violations

Question: what is the complexity of dynamic membership, depending on the fixed regular language L?

Dynamic word problem for monoids

To answer the question, we study the dynamic word problem for monoids:

- Problem definition:
  - Fix a monoid M (set with associative law and neutral element)
  - Input: word w of elements of M
  - Maintain the product of the elements under substitution updates

Dynamic word problem for monoids

To answer the question, we study the **dynamic word problem for monoids**:

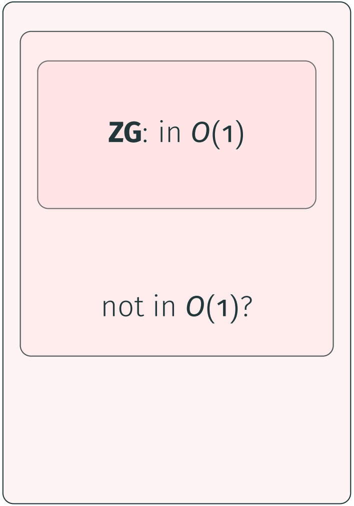
- Problem definition:
  - Fix a **monoid** M (set with associative law and neutral element)
  - **Input:** word w of elements of M
  - Maintain the **product** of the elements under substitution updates
- This is a **special case** of dynamic membership for regular languages
  - e.g., it assumes that there is a **neutral element**
- This problem was studied by [Skovbjerg Frandsen et al., 1997]:

  $\rightarrow$ in $O(1)$ for **commutative monoids**

  $\rightarrow$ in $O(\log \log n)$ for **group-free monoids**

  $\rightarrow$ in $\Theta(\log n / \log \log n)$ for a certain class of monoids

Our results on the dynamic word problem for monoids



- We identify the class **ZG** satisfying $x^{\omega+1}y = yx^{\omega+1}$:
  - for any monoid **in ZG**, the problem is **in $O(1)$**
  - for any monoid **not in ZG**, we can reduce from a problem that we **conjecture is not in $O(1)$**

Our results on the dynamic word problem for monoids

ZG: in $O(1)$

SG: in $O(\log \log n)$
not in $O(1)$?

All: in $\Theta(\log n / \log \log n)$

- We identify the class **ZG** satisfying $x^{\omega+1}y = yx^{\omega+1}$:
  - for any monoid in **ZG**, the problem is in $O(1)$
  - for any monoid not in **ZG**, we can reduce from a problem that we conjecture is not in $O(1)$
- We identify the class **SG** satisfying $x^{\omega+1}yx^{\omega} = x^{\omega}yx^{\omega+1}$
  - for any monoid in **SG**, the problem is in $O(\log \log n)$
  - for any monoid not in **SG**, it is in $\Omega(\log n / \log \log n)$ (lower bound of Skovbjerg Frandsen et al.)

Our results on the dynamic word problem for monoids

ZG: in $O(1)$

SG: in $O(\log\log n)$
not in $O(1)$?

All: in $\Theta(\log n / \log\log n)$

- We identify the class **ZG** satisfying $x^{\omega+1}y = yx^{\omega+1}$:
  - for any monoid in **ZG**, the problem is in $O(1)$
  - for any monoid not in **ZG**, we can reduce from a problem that we conjecture is not in $O(1)$
- We identify the class **SG** satisfying $x^{\omega+1}yx^{\omega} = x^{\omega}yx^{\omega+1}$
  - for any monoid in **SG**, the problem is in $O(\log\log n)$
  - for any monoid not in **SG**, it is in $\Omega(\log n / \log\log n)$ (lower bound of Skovbjerg Frandsen et al.)
- The problem is always in $O(\log n / \log\log n)$

Results on the dynamic membership problem for regular languages

QLZG: in $O(1)$

QSG: in $O(\log \log n)$
not in $O(1)$?

All: in $\Theta(\log n / \log \log n)$

Our results extend to regular language classes called **QLZG** and **QSG**

$\rightarrow$ We define them in the sequel

Roadmap of proof techniques

- First: show the results on monoids
    - The dynamic word problem is in $O(1)$ for monoids in **ZG**
    - The dynamic word problem is in $O(\log \log n)$ for monoids in **SG**
    - Lower bounds outside of **ZG** and outside of **SG**
- Second: extend the results to semigroups
- Third: extend the results to regular languages

Results on monoids

$O(1)$ upper bound for monoids

Theorem

The dynamic word problem for commutative monoids is in $O(1)$

Algorithm:

- Count the number n_m of occurrences of each element m of M in w
- Maintain the counts n_m under updates
- Evaluate the product as $\prod_{m \in M} m^{n_m}$ in $O(1)$

$O(1)$ upper bound for monoids

Theorem

The dynamic word problem for commutative monoids is in $O(1)$

Algorithm:

- Count the number n_m of occurrences of each element m of M in w
- Maintain the counts n_m under updates
- Evaluate the product as $\prod_{m \in M} m^{n_m}$ in $O(1)$

Lemma (Closure under monoid variety operations)

The submonoids, direct products, quotients of tractable monoids are also tractable

$O(1)$ upper bound for monoids (cont'd)

Theorem

The monoids S^1 where we add an identity to a nilpotent semigroup S are in $O(1)$

Idea of the proof: consider $e^* a e^* b e^*$

$O(1)$ upper bound for monoids (cont'd)

Theorem

The monoids S^1 where we add an identity to a nilpotent semigroup S are in $O(1)$

Idea of the proof: consider $e^*ae^*be^*$

- Preprocessing: prepare a doubly-linked list L of the positions containing a's and b's
- Maintain the (unsorted) list when a's and b's are added/removed
- Evaluation:
  - If there are not exactly two positions in L, answer no
  - Otherwise, check that the smallest position of these two is an a and the largest is a b

$O(1)$ upper bound for monoids (cont'd)

Theorem

The monoids S^1 where we add an identity to a nilpotent semigroup S are in $O(1)$

Idea of the proof: consider $e^*ae^*be^*$

- Preprocessing: prepare a doubly-linked list L of the positions containing a's and b's
- Maintain the (unsorted) list when a's and b's are added/removed
- Evaluation:
  - If there are not exactly two positions in L, answer no
  - Otherwise, check that the smallest position of these two is an a and the largest is a b

This technique applies to monoids where we intuitively need to track a constant number of non-neutral elements

$O(1)$ upper bound for monoids (end)

Call **ZG** the variety of monoids satisfying $x^{\omega+1}y = yx^{\omega+1}$ for all x, y

$\rightarrow$ Elements of the form $x^{\omega+1}$ are those belonging to a subgroup of the monoid

$\rightarrow$ This includes in particular all idempotents ($xx = x$)

$\rightarrow$ The $x^{\omega+1}$ are central: they commute with all other elements

$O(1)$ upper bound for monoids (end)

Call **ZG** the variety of monoids satisfying $x^{\omega+1}y = yx^{\omega+1}$ for all x, y

→ Elements of the form $x^{\omega+1}$ are those belonging to a **subgroup** of the monoid

→ This includes in particular all **idempotents** ($xx = x$)

→ The $x^{\omega+1}$ are **central**: they commute with all other elements

Lemma

ZG *is exactly the monoids obtainable from commutative monoids and monoids of the form S^1 for a nilpotent semigroup* S *via the monoid variety operators*

Theorem

The dynamic word problem for monoids in **ZG** *is in $O(1)$*

$O(\log\log n)$ upper bound for monoids

Call **SG** the variety of monoids satisfying $x^{\omega+1}yx^{\omega} = x^{\omega}yx^{\omega+1}$ for all x, y

$\rightarrow$ Intuition: we can swap the elements of any given subgroup of the monoid

Examples:

- All **ZG** monoids (where elements $x^{\omega+1}$ commute with everything)
- All group-free monoids (where subgroups are trivial)
- Products of **ZG** monoids and group-free monoids

$O(\log \log n)$ upper bound for monoids

Call **SG** the variety of monoids satisfying $x^{\omega+1}yx^{\omega} = x^{\omega}yx^{\omega+1}$ for all x, y

$\rightarrow$ Intuition: we can swap the elements of any given subgroup of the monoid

Examples:

- All **ZG** monoids (where elements $x^{\omega+1}$ commute with everything)
- All group-free monoids (where subgroups are trivial)
- Products of **ZG** monoids and group-free monoids

Theorem

The dynamic word problem for monoids in **SG** *is in* $O(\log \log n)$

$O(\log\log n)$ upper bound for monoids (proof sketch)

Example: $\Sigma^*(ae^*a)\Sigma^*$ on $\Sigma = \{a, b, e\}$

$O(\log\log n)$ upper bound for monoids (proof sketch)

Example: $\Sigma^*(ae^*a)\Sigma^*$ on $\Sigma = \{a, b, e\}$

- Idea: maintain the count of factors ae^*a
- Problem: to do this, we need to “jump over” the e’s

$O(\log \log n)$ upper bound for monoids (proof sketch)

Example: $\Sigma^*(ae^*a)\Sigma^*$ on $\Sigma = \{a, b, e\}$

- Idea: maintain the count of factors ae^*a
- Problem: to do this, we need to “jump over” the e’s

$\rightarrow$ Van Emde Boas tree data structure:

- maintain a subset of $\{1, \ldots, n\}$ under insertions/deletions
- jump to the prev/next element in $O(\log \log n)$

$O(\log \log n)$ upper bound for monoids (proof sketch)

Example: $\Sigma^*(ae^*a)\Sigma^*$ on $\Sigma = \{a, b, e\}$

- **Idea:** maintain the count of factors ae^*a
- **Problem:** to do this, we need to "jump over" the e's

→ **Van Emde Boas tree** data structure:

  - maintain a subset of $\{1, \ldots, n\}$ under insertions/deletions
  - jump to the prev/next element in $O(\log \log n)$

Full proof: induction on $\mathcal{J}$-classes and Rees-Sushkevich theorem

Lower bounds

All lower bounds reduce from the prefix problem for some language L:

- Maintain a word under substitution updates
- Answer queries asking if a given prefix of the current word is in L

Lower bounds

All lower bounds reduce from the prefix problem for some language L:

- Maintain a word under substitution updates
- Answer queries asking if a given prefix of the current word is in L

Specifically:

- Prefix-$\mathbb{Z}_d$: for $\Sigma = \{0, \dots, d-1\}$, does the input prefix sum to 0 modulo d?
  $\rightarrow$ Known lower bound of $\Omega(\log n / \log \log n)$
- Prefix-U_1: for $\Sigma = \{0, 1\}$, does the queried prefix contain a 0?
  $\rightarrow$ We conjecture that this cannot be done in $O(1)$

Lower bounds

All lower bounds reduce from the prefix problem for some language L:

- Maintain a word under substitution updates
- Answer queries asking if a given prefix of the current word is in L

Specifically:

- Prefix-$\mathbb{Z}_d$: for $\Sigma = \{0, \ldots, d-1\}$, does the input prefix sum to 0 modulo d?
  - $\rightarrow$ Known lower bound of $\Omega(\log n / \log \log n)$
- Prefix-U_1: for $\Sigma = \{0, 1\}$, does the queried prefix contain a 0?
  - $\rightarrow$ We conjecture that this cannot be done in $O(1)$

Theorem (Lower bounds on a monoid M)

- *If M is not in* **SG***, then for some $d \in \mathbb{N}$ the Prefix-$\mathbb{Z}_d$ problem reduces to the dynamic word problem for M*
- *If M is in* **SG** $\setminus$ **ZG***, then Prefix-U_1 reduces to the dynamic word problem for M*

From monoids to semigroups

From monoids to semigroups

- Semigroup: like a monoid but possibly without a neutral element
- Dynamic word problem for semigroups: defined like for monoids

What is the difference?

- The language $\Sigma^*(ae^*a)\Sigma^*$ on $\Sigma = \{a, b, e\}$ has a neutral letter e that we intuitively need to "jump over"
- The language $\Sigma^*aa\Sigma^*$ on $\Sigma = \{a, b\}$ without e can be maintained in $O(1)$ by counting the factors aa

Submonoids in semigroups

- A submonoid of a semigroup S is a subset of S that has a neutral element
  - $\rightarrow$ If S has a submonoid M then the dynamic word problem for M reduces to S
  - $\rightarrow$ Lower bounds on M thus apply to S

Submonoids in semigroups

- A submonoid of a semigroup S is a subset of S that has a neutral element
  - $\rightarrow$ If S has a submonoid M then the dynamic word problem for M reduces to S
  - $\rightarrow$ Lower bounds on M thus apply to S
- Hence, we define:
  - **LSG**: all submonoids are **in SG**
  - **LZG**: all submonoids are **in ZG**

Extending SG to semigroups

We can show that, for semigroups:

Lemma

A semigroup satisfies the equation of **SG** *iff it is in* **LSG**

Hence, as the algorithm for **SG** works for semigroups as well as monoids:

Theorem

For any semigroup S:

- *If S is in* **SG***, then the dynamic word problem is in $O(\log \log n)$*
- *Otherwise, the dynamic word problem is in $\Theta(\log n / \log \log n)$*

Case of ZG

We have **ZG** $\neq$ **LZG**, but we can still show:

Theorem

For any semigroup S:

- *If S is in* **LZG***, then the dynamic word problem is in $O(1)$*
- *Otherwise, it has a reduction from Prefix-U_1*

Case of ZG

We have **ZG** $\neq$ **LZG**, but we can still show:

Theorem

For any semigroup S:

- *If S is in **LZG**, then the dynamic word problem is in $O(1)$*
- *Otherwise, it has a reduction from Prefix-U_1*

Proof sketch: only need to show the upper bound:

- We show the $O(1)$ upper bound on the semidirect product $\mathbf{ZG} * \mathbf{D}$ of **ZG** with definite semigroups
- We show an independent locality result: $\mathbf{LZG} = \mathbf{ZG} * \mathbf{D}$
  - $\rightarrow$ Technical proof relying on finite categories and Straubing's delay theorem

From semigroups to languages

From semigroups to languages

We now move back to dynamic membership for regular languages

- Dynamic membership for a regular language L is like the dynamic word problem for its syntactic semigroup
  - $\rightarrow$ This is like the transition monoid but without the neutral element
- Difference: not all elements of the syntactic semigroup can be achieved as one letter

$\rightarrow$ We use instead the stable semigroup, which intuitively groups letters together into blocks of a constant size

From semigroups to languages (cont'd)

Call **QLZG** and **QSG** the languages whose stable semigroup is in **LZG** and **SG**

Theorem

Our results on semigroups in **SG** *and* **LZG** *extend to regular languages in* **QSG** *and* **QLZG**

From semigroups to languages (cont'd)

Call **QLZG** and **QSG** the languages whose stable semigroup is in **LZG** and **SG**

Theorem

Our results on semigroups in **SG** *and* **LZG** *extend to regular languages in* **QSG** *and* **QLZG**

For any regular language L*:*

- *If* L *is in* **QLZG** *then dynamic membership is in* $O(1)$
- *If* L *is in* **QSG** $\setminus$ **QLZG** *then dynamic membership is in* $O(\log\log n)$ *and has a reduction from prefix-*U_1
- *If* L *is not in* **QSG** *then dynamic membership is in* $\Theta(\log n / \log\log n)$

Conclusion and future work

Summary and open problems

We have shown a (conditional) **trichotomy** on the dynamic word problem for **monoids and semigroups**, and on dynamic membership for **regular languages**

Summary and open problems

We have shown a (conditional) **trichotomy** on the dynamic word problem for **monoids and semigroups**, and on dynamic membership for **regular languages**

- Can one show a **superconstant lower bound** on **prefix-U_1**?
  $\rightarrow$ **Help welcome!** but new techniques probably needed

Summary and open problems

We have shown a (conditional) **trichotomy** on the dynamic word problem for **monoids and semigroups**, and on dynamic membership for **regular languages**

- Can one show a **superconstant lower bound** on **prefix-U_1**?
  $\rightarrow$ **Help welcome!** but new techniques probably needed
- What about **intermediate cases** between $O(1)$ and $O(\log \log n)$
  - Yes with **randomization**: one language **in $\Theta(\log \log n)$** and one **in $O(\sqrt{\log \log n})$**
  - **Question:** can the intermediate classes be **characterized**?

Summary and open problems

We have shown a (conditional) trichotomy on the dynamic word problem for monoids and semigroups, and on dynamic membership for regular languages

- Can one show a superconstant lower bound on prefix-U_1?
  $\rightarrow$ Help welcome! but new techniques probably needed
- What about intermediate cases between $O(1)$ and $O(\log \log n)$
  - Yes with randomization: one language in $\Theta(\log \log n)$ and one in $O(\sqrt{\log \log n})$
  - Question: can the intermediate classes be characterized?
- Meta-dichotomy: what is the complexity of finding which case occurs?
  $\rightarrow$ Probably PSPACE-complete (depends on the representation)

Summary and open problems

We have shown a (conditional) **trichotomy** on the dynamic word problem for **monoids and semigroups**, and on dynamic membership for **regular languages**

- Can one show a **superconstant lower bound** on **prefix-U_1**?
  $\rightarrow$ **Help welcome!** but new techniques probably needed
- What about **intermediate cases** between $O(1)$ and $O(\log\log n)$
  - Yes with **randomization**: one language in $\Theta(\log\log n)$ and one in $O(\sqrt{\log\log n})$
  - **Question:** can the intermediate classes be **characterized**?
- **Meta-dichotomy**: what is the complexity of finding which case occurs?
  $\rightarrow$ Probably **PSPACE-complete** (depends on the representation)
- What about a dichotomy for the **prefix problem** or **infix problem**?
  $\rightarrow$ We have an **inelegant characterization**

Summary and open problems

We have shown a (conditional) trichotomy on the dynamic word problem for monoids and semigroups, and on dynamic membership for regular languages

- Can one show a superconstant lower bound on prefix-U_1?
  $\rightarrow$ Help welcome! but new techniques probably needed
- What about intermediate cases between $O(1)$ and $O(\log \log n)$
  - Yes with randomization: one language in $\Theta(\log \log n)$ and one in $O(\sqrt{\log \log n})$
  - Question: can the intermediate classes be characterized?
- Meta-dichotomy: what is the complexity of finding which case occurs?
  $\rightarrow$ Probably PSPACE-complete (depends on the representation)
- What about a dichotomy for the prefix problem or infix problem?
  $\rightarrow$ We have an inelegant characterization
- Is there an intuitive way to understand **QSG** and **QLZG**?

Big-picture directions

- Extending from **words** to **trees**
  - $\rightarrow$ Probably **challenging**: the algebraic tools for trees are not as powerful

Big-picture directions

- Extending from **words** to **trees**
  - $\rightarrow$ Probably **challenging**: the algebraic tools for trees are not as powerful
- Extending from **regular languages** to **context-free languages**
  - $\rightarrow$ Also **missing algebraic tools**; probably related to trees

Big-picture directions

- Extending from **words** to **trees**
  - $\rightarrow$ Probably **challenging**: the algebraic tools for trees are not as powerful
- Extending from **regular languages** to **context-free languages**
  - $\rightarrow$ Also **missing algebraic tools**; probably related to trees
- Supporting **more expressive updates**: insertion, deletion, cut and paste (?)
  - $\rightarrow$ May be able to support insert/delete in a "**linked list**" model
  - $\rightarrow$ Other interesting setting: insert/delete at the **extremities** (streaming)

Big-picture directions

- Extending from **words** to **trees**
  - → Probably **challenging**: the algebraic tools for trees are not as powerful
- Extending from **regular languages** to **context-free languages**
  - → Also **missing algebraic tools**; probably related to trees
- Supporting **more expressive updates**: insertion, deletion, cut and paste (?)
  - → May be able to support insert/delete in a "**linked list**" model
  - → Other interesting setting: insert/delete at the **extremities** (streaming)
- Going beyond **Boolean queries**
  - → Natural questions: **counting** matches, or **enumerating** matches
  - → Idea: achieve **efficient enumeration** under updates

Big-picture directions

- Extending from **words** to **trees**
  - → Probably **challenging**: the algebraic tools for trees are not as powerful
- Extending from **regular languages** to **context-free languages**
  - → Also **missing algebraic tools**; probably related to trees
- Supporting **more expressive updates**: insertion, deletion, cut and paste (?)
  - → May be able to support insert/delete in a "**linked list**" model
  - → Other interesting setting: insert/delete at the **extremities** (streaming)
- Going beyond **Boolean queries**
  - → Natural questions: **counting** matches, or **enumerating** matches
  - → Idea: achieve **efficient enumeration** under updates

Thanks for your attention!

References i

Amarilli, A., Jachiet, L., and Paperman, C. (2021).
Dynamic Membership for Regular Languages.
In *ICALP*.

Amarilli, A. and Paperman, C. (2021).
Locality and Centrality: The Variety ZG.
Under review.

Fredman, M. and Saks, M. (1989).
The cell probe complexity of dynamic data structures.
In *STOC*.

Patrascu, M. (2008).
Lower bound techniques for data structures.
PhD thesis, Massachusetts Institute of Technology.

Skovbjerg Frandsen, G., Miltersen, P. B., and Skyum, S. (1997).
Dynamic word problems.
JACM, 44(2).

Other research themes

- 00
- 01
- 10

⋮

- Enumeration algorithms, links to circuit classes
  - Enumeration for regular spanners and grammars
  - In-order enumeration
  - Connections to knowledge compilation

Other research themes

- 00
- 01
- 10

$\vdots$

- **Enumeration algorithms**, links to **circuit classes**
  - Enumeration for **regular spanners** and **grammars**
  - **In-order** enumeration
  - Connections to **knowledge compilation**

$0\ \overset{1}{\cancel{0}}\ 0\ 1$

- **Efficient maintenance** of query results on **dynamic data**
  - Supporting **membership queries**, counts, enumeration structures...
  - For **regular languages**, regular tree languages, context-free languages...
  - On **words**, trees, graphs...
  - Under **substitution updates** or other updates

Other research themes

- 00
- 01
- 10

⋮

- **Enumeration algorithms**, links to **circuit classes**
  - Enumeration for **regular spanners** and **grammars**
  - **In-order** enumeration
  - Connections to **knowledge compilation**

1

0 ~~0~~ 0 1

- **Efficient maintenance** of query results on **dynamic data**
  - Supporting **membership queries**, counts, enumeration structures...
  - For **regular languages**, regular tree languages, context-free languages...
  - On **words**, trees, graphs...
  - Under **substitution updates** or other updates

0? 50%

1? 50%

- **Query evaluation** on **probabilistic data**
  - Dichotomies for **homomorphism-closed queries**
  - **Uniform** model counting
  - **Treewidth-based** and grid-minor-based methods

Other research themes

- 00
- 01
- 10

⋮

- Enumeration algorithms, links to circuit classes
    - Enumeration for regular spanners and grammars
    - In-order enumeration
    - Connections to knowledge compilation

1

0 ~~0~~ 0 1

- Efficient maintenance of query results on dynamic data
    - Supporting membership queries, counts, enumeration structures...
    - For regular languages, regular tree languages, context-free languages...
    - On words, trees, graphs...
    - Under substitution updates or other updates

0? 50%

1? 50%

- Query evaluation on probabilistic data
    - Dichotomies for homomorphism-closed queries
    - Uniform model counting
    - Treewidth-based and grid-minor-based methods
- Database theory, provenance, logics...

Advertisement: TCS4F and “No free view? No review!”

TheoretiCS

A new open-access journal for theoretical computer science (managing editor with Nathanaël Fijalkow)

Advertisement: TCS4F and “No free view? No review!”

TheoretiCS

A new **open-access journal** for **theoretical computer science** (managing editor with Nathanaël Fijalkow)

A pledge to **reduce the carbon footprint** of your research travels

`www.tcs4f.org`

(with Thomas Schwentick, Thomas Colcombet, Hugo Férée)

TCS4F
THEORETICAL COMPUTER SCIENTISTS FOR FUTURE

Advertisement: TCS4F and “No free view? No review!”

TheoretiCS

A new open-access journal for theoretical computer science
(managing editor with Nathanaël Fijalkow)

A pledge to reduce the carbon footprint of your research travels

`www.tcs4f.org`

(with Thomas Schwentick, Thomas Colcombet, Hugo Férée)

TCS4F
THEORETICAL
COMPUTER
SCIENTISTS
FOR FUTURE

NO FREE VIEW?
NO REVIEW!

A pledge not to review for conferences and journals that do not publish their research as open access

`www.nofreeviewnoreview.org`

(with Antonin Delpeuch)