Regular Languages: Some New Problems and Algorithms

Antoine Amarilli

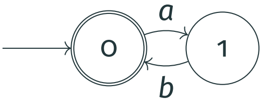
November 19, 2025

Joint work with: Corentin Barloy, Pierre Bourhis, İsmail İlkan Ceylan, Sven Dziadek, Octave Gaspard, Wolfgang Gatterbauer, Paweł Gawrychowski, Benoît Groz, Santiago Guzman Pro, Louis Jachiet, Sébastien Labbé, Neha Makhija, Kuldeep Meel, Stefan Mengel, Mikaël Monet, Martín Muñoz, Matthias Niewerth, Charles Paperman, Paul Raphaël, Tina Ringleb, Cristian Riveros, Sylvain Salvati, Luc Segoufin, Tim Van Bremen, Nicole Wein

Regular languages

Regular languages are a robust framework for constant-memory computation:

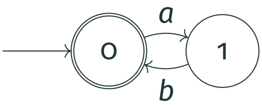
- Correspond to regular expressions, e.g., $(ab^*c|d)^*$
- Correspond to finite automata
  - Can be deterministic (DFA) or nondeterministic (NFA)



Regular languages

Regular languages are a robust framework for constant-memory computation:

- Correspond to regular expressions, e.g., $(ab^*c|d)^*$
- Correspond to finite automata
  - Can be deterministic (DFA) or nondeterministic (NFA)



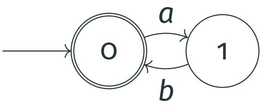
Key question: Membership problem

Given a word w and a regular language L, does $w \in L$?

Regular languages

Regular languages are a robust framework for constant-memory computation:

- Correspond to regular expressions, e.g., $(ab^*c|d)^*$
- Correspond to finite automata
  - Can be deterministic (DFA) or nondeterministic (NFA)



Key question: Membership problem

Given a word w and a regular language L, does $w \in L$?

The computational complexity can be studied in two settings:

- Data complexity: the language L is fixed and the input is w
- Combined complexity: the input is both w and some representation of L

Membership problem

What is the complexity of membership?

- In data complexity: can be decided in $O(|w|)$ on an input word w
  - $\rightarrow$ $O(1)$ possible for some languages [Aaronson et al., 2019]

Membership problem

What is the complexity of membership?

- In data complexity: can be decided in $O(|w|)$ on an input word w

  $\rightarrow$ $O(1)$ possible for some languages [Aaronson et al., 2019]

- In combined complexity:
  - Given a DFA A, can be decided in $O(|w| + |A|)$ by running the DFA

Membership problem

What is the complexity of membership?

- In data complexity: can be decided in $O(|w|)$ on an input word w

  $\rightarrow$ $O(1)$ possible for some languages [Aaronson et al., 2019]

- In combined complexity:
  - Given a DFA A, can be decided in $O(|w| + |A|)$ by running the DFA
  - Given an NFA A, can be decided in $O(|w| \cdot |A|)$ by state-set simulation

Membership problem

What is the complexity of membership?

- In data complexity: can be decided in $O(|w|)$ on an input word w
  $\rightarrow$ $O(1)$ possible for some languages [Aaronson et al., 2019]

- In combined complexity:
  - Given a DFA A, can be decided in $O(|w| + |A|)$ by running the DFA
  - Given an NFA A, can be decided in $O(|w| \cdot |A|)$ by state-set simulation
  - Conditional lower bound: no general $\Omega((|w| \cdot |A|)^{1-\epsilon})$ algorithm for any $\epsilon > 0$ assuming SETH [Backurs and Indyk, 2016]

Membership problem

What is the complexity of membership?

- In data complexity: can be decided in $O(|w|)$ on an input word w
  → $O(1)$ possible for some languages [Aaronson et al., 2019]
- In combined complexity:
  - Given a DFA A, can be decided in $O(|w| + |A|)$ by running the DFA
  - Given an NFA A, can be decided in $O(|w| \cdot |A|)$ by state-set simulation
  - Conditional lower bound: no general $\Omega((|w| \cdot |A|)^{1-\epsilon})$ algorithm for any $\epsilon > 0$ assuming SETH [Backurs and Indyk, 2016]

  → Better bounds possible for specific language families, e.g., subword/superword-closed languages [A., Manea, Ringleb, Schmid, 2025]

Membership problem

What is the complexity of membership?

- In **data complexity**: can be decided in $O(|w|)$ on an input word w
  - $\rightarrow$ $O(1)$ possible for some languages [Aaronson et al., 2019]
- In **combined complexity**:
  - Given a DFA A, can be decided in $O(|w| + |A|)$ by running the DFA
  - Given an NFA A, can be decided in $O(|w| \cdot |A|)$ by **state-set simulation**
  - **Conditional lower bound:** no general $\Omega((|w| \cdot |A|)^{1-\epsilon})$ algorithm for any $\epsilon > 0$ assuming SETH [Backurs and Indyk, 2016]
  - $\rightarrow$ Better bounds possible for **specific language families**, e.g., subword/superword-closed languages [A., Manea, Ringleb, Schmid, 2025]

This talk: study **extensions** of the membership problem

Roadmap

I will present four extensions of regular language membership…

- Membership for partial words and probabilistic words
- Incremental maintenance of membership
- Enumeration of word factors (beyond Boolean queries)
- Regular language problems on graphs

… and will sketch more directions at the end.

Partial and Probabilistic Words

Partial and probabilistic words

- Partial word: word with holes
  - → e.g., $a\textvisiblespace a\textvisiblespace$ on alphabet $\Sigma = \{a, b\}$
- Possible completions: filling the holes with letters
  - → here, 4 possible completions
- Partial membership to a fixed language L: given a partial word w, how many completions of w belong to L?

Partial and probabilistic words

- Partial word: word with holes
  → e.g., $a\_a\_$ on alphabet $\Sigma = \{a, b\}$
- Possible completions: filling the holes with letters
  → here, 4 possible completions
- Partial membership to a fixed language L: given a partial word w, how many completions of w belong to L?

Generalization: probabilistic words

- Each hole specifies a probability distribution over the alphabet
  → e.g., $w = a \begin{pmatrix} a : 1/2 \\ b : 1/2 \end{pmatrix} a \begin{pmatrix} a : 1/2 \\ b : 1/2 \end{pmatrix}$
- This defines a probability distribution on possible completions

Partial and probabilistic words

- Partial word: word with holes
  - $\rightarrow$ e.g., $a\_a\_$ on alphabet $\Sigma = \{a, b\}$
- Possible completions: filling the holes with letters
  - $\rightarrow$ here, 4 possible completions
- Partial membership to a fixed language L: given a partial word w, how many completions of w belong to L?

Generalization: probabilistic words

- Each hole specifies a probability distribution over the alphabet
  - $\rightarrow$ e.g., $w = a \begin{pmatrix} a : 1/2 \\ b : 1/2 \end{pmatrix} a \begin{pmatrix} a : 1/2 \\ b : 1/2 \end{pmatrix}$
- This defines a probability distribution on possible completions

Given a probabilistic word w, what is the total probability of the completions of w that belong to L?

Results on membership for probabilistic words

Data complexity (fixed regular language L):

Results on membership for probabilistic words

Data complexity (fixed regular language L): in linear time (up to arithmetic costs)

- Build a DFA for L (or just an unambiguous automaton or UFA)
- Do dynamic programming: read the probabilistic word w and remember the probability vector on the states

Results on membership for probabilistic words

Data complexity (fixed regular language L): in linear time (up to arithmetic costs)

- Build a DFA for L (or just an unambiguous automaton or UFA)
- Do dynamic programming: read the probabilistic word w and remember the probability vector on the states

In combined complexity:

- It is #P-hard in general but can be approximated (FPRAS) [Arenas et al., 2021]
- It is in PTIME when the input is a DFA or UFA
- Also PTIME for k-ambiguous automata? (following [Stearns and Hunt III, 1985])

Generalizations to context-free grammars: [A., Monet, Raphaël, Salvati, 2025]

Dynamic Membership

Dynamic membership for regular languages

- Fix a regular language L
  - $\rightarrow$ E.g., $L = (ab)^*$

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., $w = abbbab$

Dynamic membership for regular languages

- Fix a regular language L
  - → E.g., $L = (ab)^*$
- Read an input word w with $n := |w|$
  - → E.g., $w = abbbab$
- Maintain the membership of w to L under substitution updates
  - → Initially, we have $w \notin L$
  - → Replace character at position 3 with a: we now have $w \in L$
  - → The length n never changes

Dynamic membership for regular languages

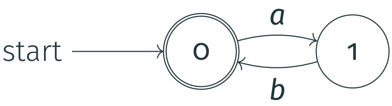
- Fix a regular language L
  - → E.g., $L = (ab)^*$
- Read an input word w with $n := |w|$
  - → E.g., $w = abbbab$
- Maintain the membership of w to L under substitution updates
  - → Initially, we have $w \notin L$
  - → Replace character at position 3 with a: we now have $w \in L$
  - → The length n never changes

Theorem (General result via balanced trees)

For any regular language L recognized by an NFA A, given a word w, we can maintain dynamic membership of w to L under substitution updates in $O(\mathit{Poly}(|A|) \times \log |w|)$ per update.

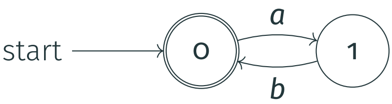
Proof sketch of the $O(\log n)$ algorithm

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



Proof sketch of the $O(\log n)$ algorithm

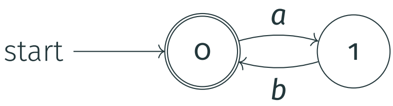
Fix the language $L = (ab)^*$:



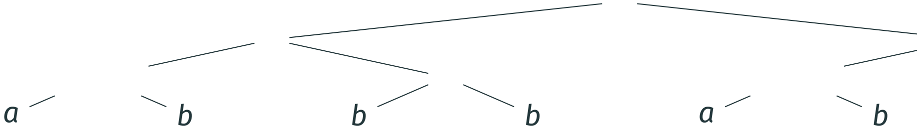
- Build a balanced binary tree on the input word $w = abbbab$

Proof sketch of the $O(\log n)$ algorithm

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

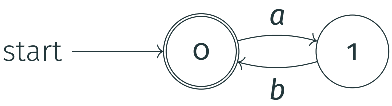


- Build a **balanced binary tree** on the input word $w = abbbab$

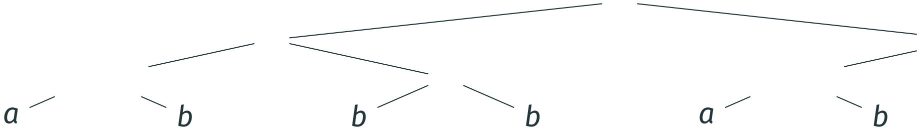


Proof sketch of the $O(\log n)$ algorithm

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

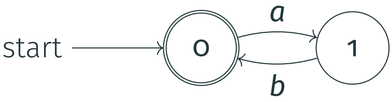


- Build a balanced binary tree on the input word $w = abbbab$
- Label each node n by the transition monoid element: all pairs $q \rightsquigarrow q'$ such that we can go from q to q' by reading the subword below n

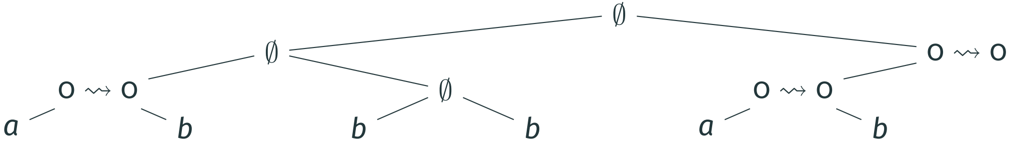


Proof sketch of the $O(\log n)$ algorithm

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

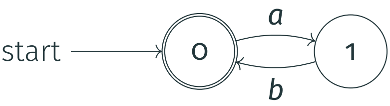


- Build a **balanced binary tree** on the input word $w = abbbab$
- Label each node n by the **transition monoid** element: all pairs $q \rightsquigarrow q'$ such that we can go from q to q' by reading the subword below n

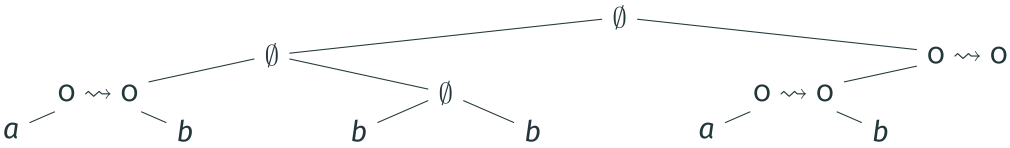


Proof sketch of the $O(\log n)$ algorithm

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



- Build a **balanced binary tree** on the input word $w = abbbab$
- 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 subword 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 **$O(\log n / \log \log n)$**

Improving on $O(\log n)$ for some languages

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

Improving on $O(\log n)$ for some languages

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

Improving on $O(\log n)$ for some languages

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 data complexity of dynamic membership, depending on the fixed regular language L?

Summary of our results [A., Jachiet, Paperman, 2021]

QLZG: in $O(1)$

QSG: in $O(\log \log n)$

not in $O(1)$?

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

- We identify a class **QLZG** of regular languages:
  - for any language in **QLZG**, dynamic membership is in $O(1)$
  - for any language not in **QLZG**, we can reduce from a problem that we conjecture is not in $O(1)$

Summary of our results [A., Jachiet, Paperman, 2021]

QLZG: in $O(1)$

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

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

- We identify a class **QLZG** of regular languages:
  - for any language in **QLZG**, dynamic membership is in $O(1)$
  - for any language not in **QLZG**, we can reduce from a problem that we conjecture is not in $O(1)$
- We identify a class **QSG** of regular languages:
  - for any language in **QSG**, the problem is in $O(\log \log n)$
  - for any language not in **QSG**, it is in $\Omega(\log n / \log \log n)$ (lower bound: [Skovbjerg Frandsen et al., 1997])

Summary of our results [A., Jachiet, Paperman, 2021]

QLZG: in $O(1)$

QSG: in $O(\log \log n)$

not in $O(1)$?

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

- We identify a class **QLZG** of regular languages:
  - for any language in **QLZG**, dynamic membership is in $O(1)$
  - for any language not in **QLZG**, we can reduce from a problem that we conjecture is not in $O(1)$
- We identify a class **QSG** of regular languages:
  - for any language in **QSG**, the problem is in $O(\log \log n)$
  - for any language not in **QSG**, it is in $\Omega(\log n / \log \log n)$ (lower bound: [Skovbjerg Frandsen et al., 1997])
- The problem is always in $O(\log n / \log \log n)$

Summary of our results [A., Jachiet, Paperman, 2021]

QLZG: in $O(1)$

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

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

- We identify a class **QLZG** of regular languages:
  - for any language **in QLZG**, dynamic membership is **in $O(1)$**
  - for any language **not in QLZG**, we can reduce from a problem that we **conjecture is not in $O(1)$**
- We identify a class **QSG** of regular languages:
  - for any language **in QSG**, the problem is **in $O(\log \log n)$**
  - for any language **not in QSG**, it is **in $\Omega(\log n / \log \log n)$** (lower bound: [Skovbjerg Frandsen et al., 1997])
- The problem is always in $O(\log n / \log \log n)$

Generalizations to **trees**: [A., Barloy, Jachiet, Paperman, 2025] and Labbé's PhD

Enumeration of Factors

Enumeration of factors

- Membership of a word w to a language L is a **Boolean** query: yes/no answer
- What if we want to find the **matches** of L in w?

Enumeration of factors

- Membership of a word w to a language L is a **Boolean** query: yes/no answer
- What if we want to find the **matches** of L in w?

Factor enumeration problem:

- **Fix:** regular language L
- **Input:** word $w = a_1 \cdots a_n$
- **Output:** enumerate all pairs $[i, j)$ such that $a_i \cdots a_{j-1} \in L$

→ This is like `re.findall` but returning all pairs (including overlapping ones)

Enumeration of factors

- Membership of a word w to a language L is a **Boolean** query: yes/no answer
- What if we want to find the **matches** of L in w?

Factor enumeration problem:

- **Fix:** regular language L
- **Input:** word $w = a_1 \cdots a_n$
- **Output:** enumerate all pairs $[i, j)$ such that $a_i \cdots a_{j-1} \in L$

→ This is like `re.findall` but returning all pairs (including overlapping ones)

There can be $\Theta(n^2)$ results, so we want an **output-sensitive algorithm**

- **Preprocessing:** worst-case running time before the first result
- **Delay** worst-case time between results

Complexity of factor enumeration

Tractable in data complexity or in combined complexity with a DFA:

Theorem (follows from [Florenzano et al., 2018])

Given a word w and a DFA A, we can enumerate the factors in w that match A with preprocessing $O(|w| \times |A|)$ and delay $O(1)$.

Complexity of factor enumeration

Tractable in data complexity or in combined complexity with a DFA:

Theorem (follows from [Florenzano et al., 2018])

Given a word w and a DFA A, we can enumerate the factors in w that match A with preprocessing $O(|w| \times |A|)$ and delay $O(1)$.

We can show (with more effort) tractability in combined complexity for NFAs:

Theorem ([A., Bourhis, Mengel, Niewerth, 2019a])

For a NFA A, we can enumerate the factors of w that match A with preprocessing $O(|w| \times Poly(|A|))$ and delay $O(Poly(|A|))$.

Complexity of factor enumeration

Tractable in data complexity or in combined complexity with a DFA:

Theorem (follows from [Florenzano et al., 2018])

Given a word w and a DFA A, we can enumerate the factors in w that match A with preprocessing $O(|w| \times |A|)$ and delay $O(1)$.

We can show (with more effort) tractability in combined complexity for NFAs:

Theorem ([A., Bourhis, Mengel, Niewerth, 2019a])

For a NFA A, we can enumerate the factors of w that match A with preprocessing $O(|w| \times Poly(|A|))$ and delay $O(Poly(|A|))$.

Beyond factor listing: generalizes to automata with capture variables

Generalizations to tree automata [A., Bourhis, Mengel, Niewerth, 2019b] and context-free grammars: [A., Jachiet, Muñoz, Riveros, 2022]

Regular Languages on Graphs

Regular Path Queries (RPQs)

- Fix an finite **alphabet** Σ

Regular Path Queries (RPQs)

- Fix an finite alphabet Σ

$\Sigma = \{a, b\}$

Regular Path Queries (RPQs)

- Fix an finite alphabet Σ
- Regular path query RPQ_L
  - $\rightarrow$ Given by a regular language L on Σ

$$\Sigma = \{a, b\}$$

Regular Path Queries (RPQs)

- Fix an finite alphabet Σ
- Regular path query RPQ_L
  - → Given by a regular language L on Σ

$$\Sigma = \{a, b\}$$

$$L = a^* \, b \, a^*$$

Regular Path Queries (RPQs)

- Fix an finite alphabet Σ
- Regular path query RPQ_L
  - $\rightarrow$ Given by a regular language L on Σ
- Graph database $D = (V, E)$
  - $\rightarrow$ Vertices V and edges $E \subseteq V \times \Sigma \times V$

$$\Sigma = \{a, b\}$$

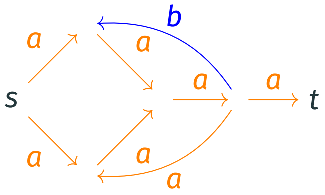
$$L = a^* \, b \, a^*$$

Regular Path Queries (RPQs)

- Fix an finite alphabet Σ
- Regular path query RPQ_L
  - → Given by a regular language L on Σ
- Graph database $D = (V, E)$
  - → Vertices V and edges $E \subseteq V \times \Sigma \times V$

$\Sigma = \{a, b\}$

$L = a^* \, b \, a^*$

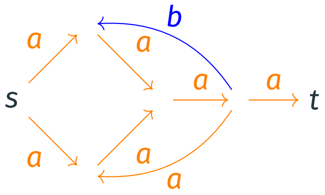


Regular Path Queries (RPQs)

- Fix an finite alphabet Σ
- Regular path query RPQ_L
  → Given by a regular language L on Σ
- Graph database $D = (V, E)$
  → Vertices V and edges $E \subseteq V \times \Sigma \times V$
- We have $D \models \mathrm{RPQ}_L(s, t)$ for endpoints s, t if:
  - We have a walk w in D from the source s to the target t
  - The concatenation of the edge labels of w is in L

$\Sigma = \{a, b\}$

$L = a^* \, b \, a^*$

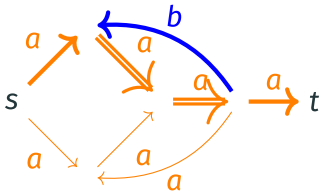


Regular Path Queries (RPQs)

- Fix an finite alphabet Σ
- Regular path query RPQ_L
  - $\rightarrow$ Given by a regular language L on Σ
- Graph database $D = (V, E)$
  - $\rightarrow$ Vertices V and edges $E \subseteq V \times \Sigma \times V$
- We have $D \models \mathrm{RPQ}_L(s, t)$ for endpoints s, t if:
  - We have a walk w in D from the source s to the target t
  - The concatenation of the edge labels of w is in L

$\Sigma = \{a, b\}$

$L = a^* \, b \, a^*$

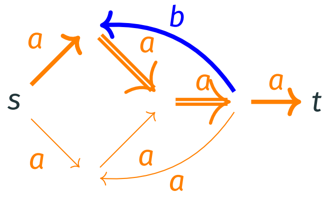


Regular Path Queries (RPQs)

- Fix an finite alphabet Σ
- Regular path query RPQ_L
  - $\rightarrow$ Given by a regular language L on Σ
- Graph database $D = (V, E)$
  - $\rightarrow$ Vertices V and edges $E \subseteq V \times \Sigma \times V$
- We have $D \models \mathrm{RPQ}_L(s, t)$ for endpoints s, t if:
  - We have a walk w in D from the source s to the target t
  - The concatenation of the edge labels of w is in L
  - Note: w is not necessarily a simple path

$\Sigma = \{a, b\}$

$L = a^* \, b \, a^*$



Evaluating RPQs on graph databases

Consider the generalization of the membership problem (in data complexity)

- Fix: a regular language L giving the regular path query RPQ_L
- Input a graph database D, source s, target t
- Output: decide whether $D \models \mathrm{RPQ}_L(s, t)$

Evaluating RPQs on graph databases

Consider the generalization of the membership problem (in data complexity)

- Fix: a regular language L giving the regular path query RPQ_L
- Input a graph database D, source s, target t
- Output: decide whether $D \models \mathrm{RPQ}_L(s, t)$

This can be solved in PTIME:

- Proof 1: write RPQ_L in Datalog and evaluate on D
- Proof 2: see D as an NFA and build the product automaton with an NFA for L

Evaluating RPQs on graph databases

Consider the generalization of the membership problem (in data complexity)

- Fix: a regular language L giving the regular path query RPQ_L
- Input a graph database D, source s, target t
- Output: decide whether $D \models \mathrm{RPQ}_L(s, t)$

This can be solved in PTIME:

- Proof 1: write RPQ_L in Datalog and evaluate on D
- Proof 2: see D as an NFA and build the product automaton with an NFA for L

Many possible generalizations of membership on graph databases:

→ I will focus on one: the smallest witness problem

→ *What is the smallest sub-database of D that satisfies RPQ_L?*

Smallest Witness for RPQs

- Decision problem SW_L for a fixed regular language L:
  - Input: graph database D, vertices s and t, integer $k \in \mathbb{N}$

Smallest Witness for RPQs

- Decision problem SW_L for a fixed regular language L:
  - Input: graph database D, vertices s and t, integer $k \in \mathbb{N}$
  - Output: is there a subdatabase $D' \subseteq D$ with $\leq k$ edges with $D' \models \mathsf{RPQ}_L(s, t)$

Smallest Witness for RPQs

- Decision problem SW_L for a fixed regular language L:
  - Input: graph database D, vertices s and t, integer $k \in \mathbb{N}$
  - Output: is there a subdatabase $D' \subseteq D$ with $\leq k$ edges with $D' \models \mathsf{RPQ}_L(s, t)$

→ What is the complexity of SW_L depending on the language L?

Smallest Witness for RPQs

- Decision problem SW_L for a fixed regular language L:
  - Input: graph database D, vertices s and t, integer $k \in \mathbb{N}$
  - Output: is there a subdatabase $D' \subseteq D$ with $\leq k$ edges with $D' \models \mathrm{RPQ}_L(s,t)$

→ What is the complexity of SW_L depending on the language L?

First results:

- SW_L is always in NP

  → Guess the subdatabase D' with $|D'| \leq k$ and verify $D' \models \mathrm{RPQ}_L(s,t)$

Smallest Witness for RPQs

- Decision problem SW_L for a fixed regular language L:
  - Input: graph database D, vertices s and t, integer $k \in \mathbb{N}$
  - Output: is there a subdatabase $D' \subseteq D$ with $\leq k$ edges with $D' \models \mathsf{RPQ}_L(s, t)$

$\rightarrow$ What is the complexity of SW_L depending on the language L?

First results:

- SW_L is always in NP
  - $\rightarrow$ Guess the subdatabase D' with $|D'| \leq k$ and verify $D' \models \mathsf{RPQ}_L(s, t)$
- If L is a finite language, then SW_L is in PTIME
  - $\rightarrow$ Polynomial number of possible matches so we can bruteforce

Smallest Witness for RPQs

- **Decision problem** SW_L for a fixed regular language L:
  - **Input**: graph database D, vertices s and t, integer $k \in \mathbb{N}$
  - **Output**: is there a subdatabase $D' \subseteq D$ with $\leq k$ edges with $D' \models \mathrm{RPQ}_L(s,t)$

→ **What is the complexity of SW_L depending on the language L?**

First results:

- SW_L is always **in NP**
  → **Guess** the subdatabase D' with $|D'| \leq k$ and **verify** $D' \models \mathrm{RPQ}_L(s,t)$
- If L is a **finite language**, then SW_L is **in PTIME**
  → Polynomial number of possible matches so we can **bruteforce**
- For $L = a^*$, we have that SW_L is...

Smallest Witness for RPQs

- Decision problem SW_L for a fixed regular language L:
  - Input: graph database D, vertices s and t, integer $k \in \mathbb{N}$
  - Output: is there a subdatabase $D' \subseteq D$ with $\leq k$ edges with $D' \models \text{RPQ}_L(s, t)$

→ What is the complexity of SW_L depending on the language L?

First results:

- SW_L is always in NP
  → Guess the subdatabase D' with $|D'| \leq k$ and verify $D' \models \text{RPQ}_L(s, t)$
- If L is a finite language, then SW_L is in PTIME
  → Polynomial number of possible matches so we can bruteforce
- For $L = a^*$, we have that SW_L is... in PTIME
  → Compute the shortest path from s to t and check that it has $\leq k$ edges

Tractability for modularity constraints

$L = (a^q)^* a^r$

Tractability for modularity constraints

$L = (a^q)^* a^r$ (st-walk of length $r \bmod q$ with min. #distinct edges)

Tractability for modularity constraints

$L = (a^q)^* a^r$ (st-walk of length $r \bmod q$ with min. #distinct edges)

Theorem ([A., Groz, Wein, 2025])

For any fixed $q > 0$ *and* $0 \leq r < q$*, letting* $L = (a^q)^* a^r$*, the problem* SW_L *is in PTIME*

Tractability for modularity constraints

$L = (a^q)^* a^r$ (st-walk of length $r \bmod q$ with min. #distinct edges)

Theorem ([A., Groz, Wein, 2025])

For any fixed $q > 0$ and $0 \leq r < q$, letting $L = (a^q)^* a^r$, the problem SW_L is in PTIME

Proof sketch:

- An optimal walk w will not have too many detours

Tractability for modularity constraints

$L = (a^q)^* a^r$ (st-walk of length $r \bmod q$ with min. #distinct edges)

Theorem ([A., Groz, Wein, 2025])

For any fixed $q > 0$ and $0 \leq r < q$, letting $L = (a^q)^* a^r$, the problem SW_L is in PTIME

Proof sketch:

- An optimal walk w will not have too many detours
  - Detour: taking a new edge (u, v) then going back to a vertex already reachable from u

Tractability for modularity constraints

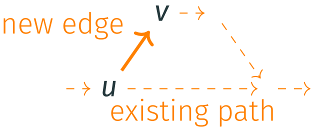
$L = (a^q)^* a^r$ (st-walk of length r mod q with min. #distinct edges)

Theorem ([A., Groz, Wein, 2025])

*For any fixed $q > 0$ and $0 \leq r < q$, letting $L = (a^q)^* a^r$, the problem SW_L is **in PTIME***

Proof sketch:

- An optimal walk w will not have too many **detours**
  - **Detour**: taking a new edge (u, v) then going back to a vertex already reachable from u



Tractability for modularity constraints

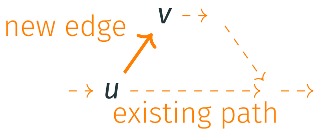
$$L = (a^q)^* a^r \qquad (st\text{-walk of length } r \bmod q \text{ with min. \#distinct edges})$$

Theorem ([A., Groz, Wein, 2025])

For any fixed $q > 0$ and $0 \leq r < q$, letting $L = (a^q)^* a^r$, the problem SW_L is in PTIME

Proof sketch:

- An optimal walk w will not have too many **detours**
  - **Detour**: taking a new edge (u, v) then going back to a vertex already reachable from u
  - Only useful to **change the remainder** of the length



Tractability for modularity constraints

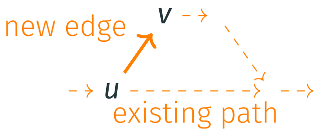
$L = (a^q)^* a^r$ (st-walk of length $r \bmod q$ with min. #distinct edges)

Theorem ([A., Groz, Wein, 2025])

For any fixed $q > 0$ and $0 \leq r < q$, letting $L = (a^q)^* a^r$, the problem SW_L is in PTIME

Proof sketch:

- An optimal walk w will not have too many **detours**
  - **Detour**: taking a new edge (u, v) then going back to a vertex already reachable from u
  - Only useful to **change the remainder** of the length
  - After $O(\log q)$ detours, **all possible remainders** achieved



Tractability for modularity constraints

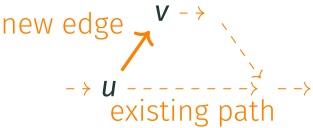
$L = (a^q)^* a^r$ (st-walk of length $r \bmod q$ with min. #distinct edges)

Theorem ([A., Groz, Wein, 2025])

For any fixed $q > 0$ *and* $0 \leq r < q$, *letting* $L = (a^q)^* a^r$, *the problem* SW_L *is in PTIME*

Proof sketch:

- An optimal walk w will not have too many **detours**
  - **Detour**: taking a new edge (u, v) then going back to a vertex already reachable from u
  - Only useful to **change the remainder** of the length
  - After $O(\log q)$ detours, **all possible remainders** achieved



- The **cutwidth** of a graph spanned by walk w is bounded by $O(\text{\#detours of } w)$

Tractability for modularity constraints

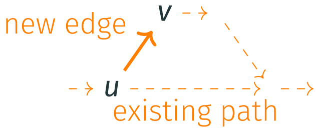
$$L = (a^q)^* a^r \qquad (st\text{-walk of length } r \bmod q \text{ with min. \#distinct edges})$$

Theorem ([A., Groz, Wein, 2025])

For any fixed $q > 0$ and $0 \leq r < q$, letting $L = (a^q)^* a^r$, the problem SW_L is in PTIME

Proof sketch:

- An optimal walk w will not have too many **detours**
  - **Detour**: taking a new edge (u, v) then going back to a vertex already reachable from u
  - Only useful to **change the remainder** of the length
  - After $O(\log q)$ detours, **all possible remainders** achieved



- The **cutwidth** of a graph spanned by walk w is bounded by $O(\#\text{detours of } w)$
- Bounded-cutwidth subgraphs can be found by **dynamic programming**

Other Directions and Future Work

Many other directions (talk to me to know more!)

Many generalizations of the membership problem of words to regular languages:

- Partial and probabilistic words
  - What about RPQs on probabilistic graphs? [A., van Bremen, Gaspard, Meel, 2025]

Many other directions (talk to me to know more!)

Many generalizations of the **membership problem** of words to regular languages:

- Partial and probabilistic words
  - What about **RPQs on probabilistic graphs**? [A., van Bremen, Gaspard, Meel, 2025]
- Dynamic membership
  - What about **other updates** (insert/delete, cut and paste...)
  - What about **trees**? **graphs**?

Many other directions (talk to me to know more!)

Many generalizations of the membership problem of words to regular languages:

- Partial and probabilistic words
  - What about RPQs on probabilistic graphs? [A., van Bremen, Gaspard, Meel, 2025]
- Dynamic membership
  - What about other updates (insert/delete, cut and paste...)
  - What about trees? graphs?
- Enumeration
  - What about incremental maintenance of enumeration structures?
    → [A., Bourhis, Mengel, Niewerth, 2019b], [A., Dziadek, Segoufin, 2025]

Many other directions (talk to me to know more!)

Many generalizations of the **membership problem** of words to regular languages:

- Partial and probabilistic words
  - What about **RPQs on probabilistic graphs**? [A., van Bremen, Gaspard, Meel, 2025]
- Dynamic membership
  - What about **other updates** (insert/delete, cut and paste...)
  - What about **trees**? **graphs**?
- Enumeration
  - What about **incremental maintenance** of enumeration structures?
    - → [A., Bourhis, Mengel, Niewerth, 2019b], [A., Dziadek, Segoufin, 2025]
- RPQs on graphs
  - **Resilience:** can we find the **minimum number of facts to kill all L-walks**?
    - → [A., Gatterbauer, Makhija, Monet, 2025]

Many other directions (talk to me to know more!)

Many generalizations of the **membership problem** of words to regular languages:

- Partial and probabilistic words
  - What about **RPQs on probabilistic graphs**? [A., van Bremen, Gaspard, Meel, 2025]
- Dynamic membership
  - What about **other updates** (insert/delete, cut and paste...)
  - What about **trees**? **graphs**?
- Enumeration
  - What about **incremental maintenance** of enumeration structures?
    - → [A., Bourhis, Mengel, Niewerth, 2019b], [A., Dziadek, Segoufin, 2025]
- RPQs on graphs
  - **Resilience:** can we find the **minimum number of facts to kill all L-walks**?
    - → [A., Gatterbauer, Makhija, Monet, 2025]
- Other problems: Tuple testing, conditional membership, **topological sorts** [A., Paperman, 2018], **perfect matchings**, simple paths, enumerating big results...

Many other directions (talk to me to know more!)

Many generalizations of the **membership problem** of words to regular languages:

- Partial and probabilistic words
  - What about **RPQs on probabilistic graphs**? [A., van Bremen, Gaspard, Meel, 2025]
- Dynamic membership
  - What about **other updates** (insert/delete, cut and paste...)
  - What about **trees**? **graphs**?
- Enumeration
  - What about **incremental maintenance** of enumeration structures?

    → [A., Bourhis, Mengel, Niewerth, 2019b], [A., Dziadek, Segoufin, 2025]
- RPQs on graphs
  - **Resilience:** can we find the **minimum number of facts to kill all L-walks**?

    → [A., Gatterbauer, Makhija, Monet, 2025]
- Other problems: Tuple testing, conditional membership, **topological sorts** [A., Paperman, 2018], **perfect matchings**, simple paths, enumerating big results...

Thanks for your attention!

References

Aaronson, S., Grier, D., and Schaeffer, L. (2019). A quantum query complexity trichotomy for regular languages. In *FOCS*.

Amarilli, A., Barloy, C., Jachiet, L., and Paperman, C. (2025a). Dynamic membership for regular tree languages. In *MFCS*.

Amarilli, A., Bourhis, P., Mengel, S., and Niewerth, M. (2019a). Constant-Delay Enumeration for Nondeterministic Document Spanners. In *ICDT*.

Amarilli, A., Bourhis, P., Mengel, S., and Niewerth, M. (2019b). Enumeration on trees with tractable combined complexity and efficient updates. In *PODS*.

References ii

Amarilli, A., Dziadek, S., and Segoufin, L. (2025b). Constant-time dynamic enumeration of word infixes in a regular language.

Amarilli, A., Gatterbauer, W., Makhija, N., and Monet, M. (2025c). Resilience for regular path queries: Towards a complexity classification. In *PODS*.

Amarilli, A., Groz, B., and Wein, N. (2025d). Edge-minimum walk of modular length in polynomial time. In *ITCS*.

Amarilli, A., Jachiet, L., Muñoz, M., and Riveros, C. (2022). Efficient enumeration algorithms for annotated grammars. In *PODS*.

Amarilli, A., Jachiet, L., and Paperman, C. (2021). Dynamic membership for regular languages. In *ICALP*.

References iii

Amarilli, A., Manea, F., Ringleb, T., and Schmid, M. L. (2025e). Linear time subsequence and supersequence regex matching. In *MFCS*.

Amarilli, A., Monet, M., Raphaël, P., and Salvati, S. (2025f). On the complexity of language membership for probabilistic words. Under review.

Amarilli, A. and Paperman, C. (2018). Topological sorting under regular constraints. In *ICALP*.

Amarilli, A., van Bremen, T., Gaspard, O., and Meel, K. S. (2025g). Approximating queries on probabilistic graphs. *LMCS*. To appear.

Arenas, M., Croquevielle, L. A., Jayaram, R., and Riveros, C. (2021). #NFA admits an FPRAS: Efficient enumeration, counting, and uniform generation for logspace classes. *Journal of the ACM*, 68(6).

References iv

Backurs, A. and Indyk, P. (2016). Which regular expression patterns are hard to match? In *FOCS*.

Florenzano, F., Riveros, C., Ugarte, M., Vansummeren, S., and Vrgoc, D. (2018). Constant Delay Algorithms for Regular Document Spanners. In *PODS*.

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

Stearns, R. E. and Hunt III, H. B. (1985). On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM Journal on Computing*, 14(3).