Strings in Data Management: Enumeration and Incremental Maintenance

Antoine Amarilli

July 30, 2024

Télécom Paris

Introduction

Short presentation

- Associate prof at **Télécom Paris**
  - → Moving from Télécom Paris to **Inria Lille**

Short presentation

- Associate prof at **Télécom Paris**
  - → Moving from Télécom Paris to **Inria Lille**
- Core area: **database theory**
  - → Moving from **databases** to **theory**

Short presentation

- Associate prof at **Télécom Paris**
  - → Moving from Télécom Paris to **Inria Lille**
- Core area: **database theory**
  - → Moving from **databases** to **theory**
- Confession: **No prior stringology experience**!
  - → But attracted to the area via **database theory** problems

Talk overview

Main message: database theory problems sometimes lead to stringology problems

Talk overview

Main message: database theory problems sometimes lead to stringology problems

Talk contents:

- Regular document spanners, a formalism motivated by information extraction

Talk overview

Main message: database theory problems sometimes lead to stringology problems

Talk contents:

- Regular document spanners, a formalism motivated by information extraction
- Enumeration algorithms for regular document spanners on strings

Talk overview

Main message: database theory problems sometimes lead to stringology problems

Talk contents:

- Regular document spanners, a formalism motivated by information extraction
- Enumeration algorithms for regular document spanners on strings
- Incremental maintenance when the string is modified

Talk overview

Main message: database theory problems sometimes lead to stringology problems

Talk contents:

- Regular document spanners, a formalism motivated by information extraction
- Enumeration algorithms for regular document spanners on strings
- Incremental maintenance when the string is modified
- Directions for future research

Document spanners

Database motivation: Declarative information extraction

- **Standard setting in database research:** queries are posed over **relational tables**
- **In practice:** data is sometimes hidden in large textual documents

$\rightarrow$ **Information extraction** (IE): how to get from large textual data to structured data

Database motivation: Declarative information extraction

- **Standard setting in database research:** queries are posed over **relational tables**
- **In practice:** data is sometimes hidden in large textual documents

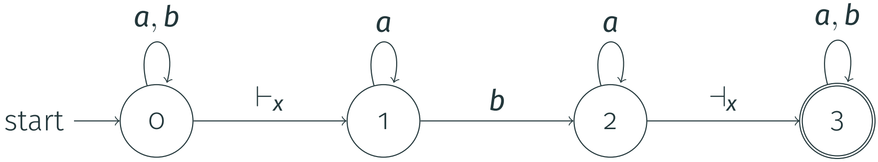
→ **Information extraction** (IE): how to get from large textual data to structured data

Guiding principle: **declarative** information extraction:

→ Specify **what** you want to extract, not **how** to extract it

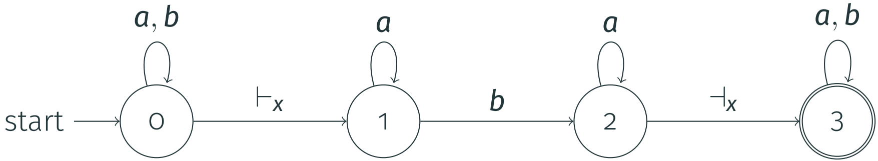
Basic idea for declarative IE: Regular document spanners

A finite automaton with special transitions that extract substrings of the input string



Basic idea for declarative IE: Regular document spanners

A finite automaton with special transitions that extract substrings of the input string



"Extract all couples of a nonempty factor with only ***a****'s and then a nonempty factor with only* ***b****'s"*

Formalizing document spanners

- **Document:** string over an alphabet

Formalizing document spanners

- **Document:** string over an alphabet

$T =$	J	o	h	n	␣	4	5	6	1	2	3	
	0	1	2	3	4	5	6	7	8	9	10	11

Formalizing document spanners

- **Document:** string over an alphabet

$$T = \texttt{J}\ \texttt{o}\ \texttt{h}\ \texttt{n}\ \texttt{␣}\ \texttt{4}\ \texttt{5}\ \texttt{6}\ \texttt{1}\ \texttt{2}\ \texttt{3}$$

0 1 2 3 4 5 6 7 8 9 10 11

- **Span:** interval of positions
  - $\rightarrow$ ex: $[0, 4\rangle$, $[5, 11\rangle$

Formalizing document spanners

- **Document:** string over an alphabet

$$T = \texttt{J}\ \texttt{o}\ \texttt{h}\ \texttt{n}\ \llcorner\!\lrcorner\ \texttt{4}\ \texttt{5}\ \texttt{6}\ \texttt{1}\ \texttt{2}\ \texttt{3}$$

0 1 2 3 4 5 6 7 8 9 10 11

- **Span:** interval of positions
  $\rightarrow$ ex: $[0, 4\rangle$, $[5, 11\rangle$

- **Mapping** over a set of variables X: function from X to spans
  $\rightarrow$ ex: for $X = \{x, y, z\}$, map x and y to $[0, 4\rangle$ and map z to $[11, 11\rangle$

Formalizing document spanners

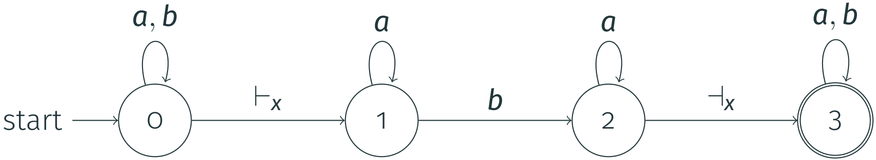
- **Document:** string over an alphabet

$$T = \texttt{J}_0\ \texttt{o}_1\ \texttt{h}_2\ \texttt{n}_3\ \sqcup_4\ \texttt{4}_5\ \texttt{5}_6\ \texttt{6}_7\ \texttt{1}_8\ \texttt{2}_9\ \texttt{3}_{10}\ {}_{11}$$

- **Span:** interval of positions
  $\rightarrow$ ex: $[0, 4\rangle$, $[5, 11\rangle$
- **Mapping** over a set of variables X: function from X to spans
  $\rightarrow$ ex: for $X = \{x, y, z\}$, map x and y to $[0, 4\rangle$ and map z to $[11, 11\rangle$
- **Spanner**: function that maps each string to a set of mappings
  $\rightarrow$ ex: for $X = \{x, y, z\}$, each string is mapped to a **relational table** with columns x, y, z

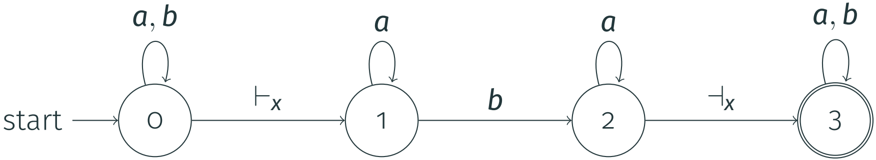
Document spanner example

Take the spanner from before:



Document spanner example

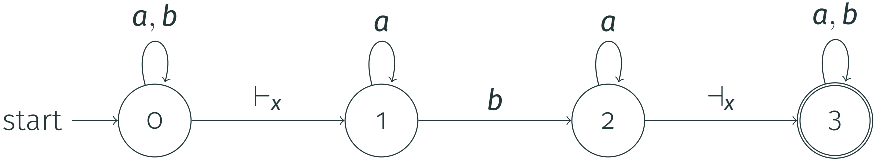
Take the spanner from before:



"Extract all substrings containing exactly one ***b****"*

Document spanner example

Take the spanner from before:



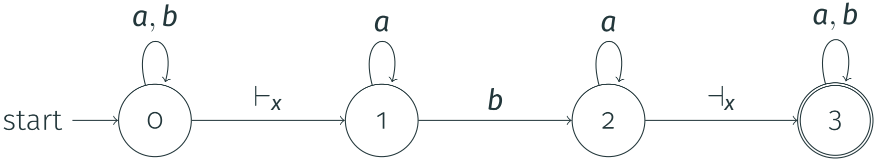
"Extract all substrings containing exactly one b*"*

$T =$ `b b a b`

0 1 2 3 4

Document spanner example

Take the spanner from before:



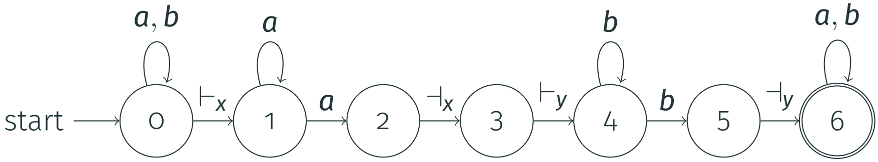
x
$[0, 1\rangle$
$[1, 2\rangle$
$[1, 3\rangle$
$[2, 4\rangle$
$[3, 4\rangle$

*"Extract all substrings containing exactly one **b**"*

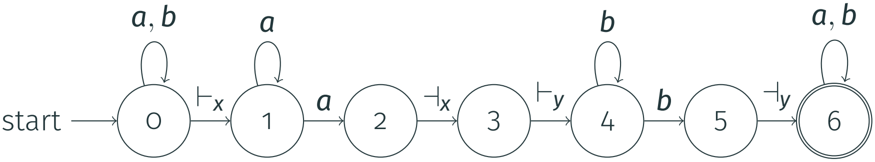
$T =$ `b b a b`

0 1 2 3 4

Document spanner example 2

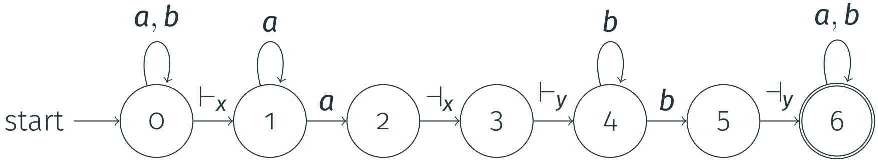


Document spanner example 2



*"Extract all nonempty contiguous substrings of **a**'s followed by nonempty contiguous substrings of **b**'s"*

Document spanner example 2

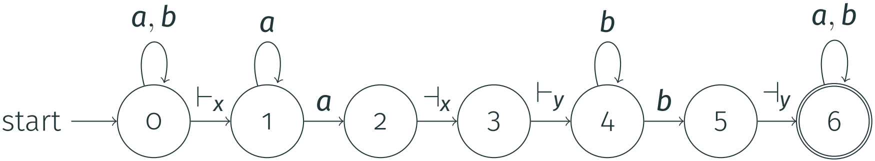


"Extract all nonempty contiguous substrings of a*'s followed by nonempty contiguous substrings of* b*'s"*

$T =$ `a b a b b`

0 1 2 3 4 5

Document spanner example 2



x	y
$[0,1\rangle$	$[1,2\rangle$
$[2,3\rangle$	$[3,4\rangle$
$[2,3\rangle$	$[3,5\rangle$

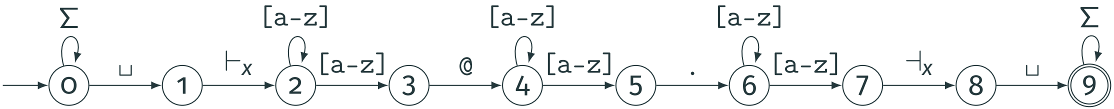
*"Extract all nonempty contiguous substrings of **a**'s followed by nonempty contiguous substrings of **b**'s"*

$T =$ `a b a b b`

0 1 2 3 4 5

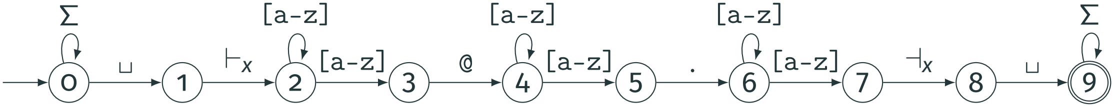
Formalizing regular document spanners

Regular spanners: those that can be expressed as variable-set automata (VAs)



Formalizing regular document spanners

Regular spanners: those that can be expressed as variable-set automata (VAs)

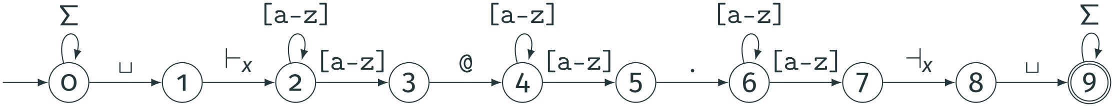


In practice, often more convenient to write in the subclass of regex-formulas:

$$\Sigma^* \vdash_x \texttt{[a-z]}^+ \; \texttt{@} \; \texttt{[a-z]}^+ \; \texttt{.} \; \texttt{[a-z]}^+ \dashv_x \Sigma^*$$

Formalizing regular document spanners

Regular spanners: those that can be expressed as **variable-set automata** (VAs)



In practice, often more convenient to write in the subclass of **regex-formulas**:

$$\Sigma^* \vdash_x \texttt{[a-z]}^+ \; \texttt{@} \; \texttt{[a-z]}^+ \; \texttt{.} \; \texttt{[a-z]}^+ \dashv_x \Sigma^*$$

Other **more general classes**:

- **Core spanners**: featuring **string equality selection**
- **Generalized core spanners**: featuring **difference**

What can you do with spanners?

- Find all occurrences of word “stringology”:

$$\Sigma^* \vdash_x \texttt{stringology} \dashv_x \Sigma^*$$

What can you do with spanners?

- Find all occurrences of word “stringology”:

$$\Sigma^* \vdash_x \texttt{stringology} \dashv_x \Sigma^*$$

- Find all occurrences of substrings satisfying a regular expression e

$$\Sigma^* \vdash_x e \dashv_x \Sigma^*$$

What can you do with spanners?

- Find **all occurrences** of word “stringology”:

$$\Sigma^* \vdash_x \texttt{stringology} \dashv_x \Sigma^*$$

- Find **all occurrences of substrings** satisfying a regular expression e

$$\Sigma^* \vdash_x e \dashv_x \Sigma^*$$

- Test if the **entire word** satisfies a regular expression e

Is there a result for $\vdash_x e \dashv_x$?

What can you do with spanners?

- Find all occurrences of word “stringology”:

$$\Sigma^* \vdash_x \texttt{stringology} \dashv_x \Sigma^*$$

- Find all occurrences of substrings satisfying a regular expression e

$$\Sigma^* \vdash_x e \dashv_x \Sigma^*$$

- Test if the entire word satisfies a regular expression e

Is there a result for $\vdash_x e \dashv_x$?

- Find all matches of pattern with variables xax, using string equality selection

$$\Sigma^* \vdash_x \Sigma^* \dashv_x a \vdash_{x'} \Sigma^* \dashv_{x'} \Sigma^* \text{ with string equality } x = x'$$

Which questions on spanners are investigated in database theory?

- Expressive power: can we express a given spanner in a given formalism? which formalisms are more expressive?

Which questions on spanners are investigated in database theory?

- **Expressive power:** can we express a given spanner in a given formalism? which formalisms are more expressive?
- **Closure under operators** and **state complexity:** which operations can we apply on spanners? how does the size change?

Which questions on spanners are investigated in database theory?

- Expressive power: can we express a given spanner in a given formalism? which formalisms are more expressive?
- Closure under operators and state complexity: which operations can we apply on spanners? how does the size change?
- Schema-based vs schemaless: what happens if the spanner does not always assign all variables in all results?

Which questions on spanners are investigated in database theory?

- **Expressive power:** can we express a given spanner in a given formalism? which formalisms are more expressive?
- **Closure under operators** and **state complexity:** which operations can we apply on spanners? how does the size change?
- **Schema-based vs schemaless**: what happens if the spanner does not always assign all variables in all results?
- **Extensions**:
  - **SLP-compressed** strings
  - **More expressive** spanner formalisms, e.g., based on **context-free languages**
  - Adding **weights** to spanners

Which questions on spanners are investigated in database theory?

- **Expressive power:** can we express a given spanner in a given formalism? which formalisms are more expressive?
- **Closure under operators** and **state complexity:** which operations can we apply on spanners? how does the size change?
- **Schema-based vs schemaless**: what happens if the spanner does not always assign all variables in all results?
- **Extensions**:
  - **SLP-compressed** strings
  - **More expressive** spanner formalisms, e.g., based on **context-free languages**
  - Adding **weights** to spanners

→ **Efficient evaluation**: can we efficiently compute the result of a spanner on a string?

Enumeration

Database motivation: Enumeration algorithms

Idea: When evaluating queries returning many results, we do not want to compute all results; we just need to be able to enumerate results quickly

Database motivation: Enumeration algorithms

Idea: When evaluating queries returning many results, we do not want to compute **all results**; we just need to be able to **enumerate** results quickly

Idea: When evaluating queries returning many results, we do not want to compute **all results**; we just need to be able to **enumerate** results quickly

Database motivation: Enumeration algorithms

Idea: When evaluating queries returning many results, we do not want to compute **all results**; we just need to be able to **enumerate** results quickly

...

Database motivation: Enumeration algorithms

Idea: When evaluating queries returning many results, we do not want to compute **all results**; we just need to be able to **enumerate** results quickly

Database motivation: Enumeration algorithms

Idea: When evaluating queries returning many results, we do not want to compute **all results**; we just need to be able to **enumerate** results quickly

→ Research area (in databases and outside): **enumeration algorithms**

A simpler problem: Enumerating results of regular spanners

- Problem description:

A simpler problem: Enumerating results of regular spanners

- Problem description:
  - Input:
    - A string T

```
Antoine Amarilli Description Name Antoine Amarilli. Handle: a3nm. Identity Born 1990-02-07. French national. Appearance as of 2017.
Auth OpenPGP. OpenId. Bitcoin. Contact Email and XMPP a3nm@a3nm.net Affiliation Associate professor of computer science (office C201-4)
in the DIG team of Télécom Paris, 46 rue Barrault, F-75634 Paris Cedex 13, France. Studies PhD in computer science awarded by Télécom
ParisTech on March 14, 2016. Former student of the École normale supérieure. test@example.com More Résumé Location Other sites Blogging:
a3nm.net/blog Git: a3nm.net/git ...
```

A simpler problem: Enumerating results of regular spanners

- Problem description:
  - Input:
    - A string T

```
Antoine Amarilli Description Name Antoine Amarilli.  Handle:  a3nm.  Identity Born 1990-02-07.  French national.  Appearance as of 2017.
Auth OpenPGP. OpenId.  Bitcoin.  Contact Email and XMPP a3nm@a3nm.net Affiliation Associate professor of computer science (office C201-4)
in the DIG team of Télécom Paris, 46 rue Barrault, F-75634 Paris Cedex 13, France.  Studies PhD in computer science awarded by Télécom
ParisTech on March 14, 2016.  Former student of the École normale supérieure.  test@example.com More Résumé Location Other sites Blogging:
a3nm.net/blog Git:  a3nm.net/git ...
```

    - A regular spanner P

$$\Sigma^* \vdash_x \texttt{[a-z]}^+ \texttt{ @ } \texttt{[a-z]}^+ \texttt{ . } \texttt{[a-z]}^+ \dashv_x \Sigma^*$$

A simpler problem: Enumerating results of regular spanners

- Problem description:
  - Input:
    - A string T

```
Antoine Amarilli Description Name Antoine Amarilli.  Handle:  a3nm.  Identity Born 1990-02-07.  French national.  Appearance as of 2017.
Auth OpenPGP. OpenId.  Bitcoin.  Contact Email and XMPP a3nm@a3nm.net Affiliation Associate professor of computer science (office C201-4)
in the DIG team of Télécom Paris, 46 rue Barrault, F-75634 Paris Cedex 13, France.  Studies PhD in computer science awarded by Télécom
ParisTech on March 14, 2016.  Former student of the École normale supérieure.  test@example.com More Résumé Location Other sites Blogging:
a3nm.net/blog Git:  a3nm.net/git ...
```

    - A regular spanner P

      $\Sigma^* \vdash_x \texttt{[a-z]}^+ \texttt{ @ } \texttt{[a-z]}^+ \texttt{ . } \texttt{[a-z]}^+ \dashv_x \Sigma^*$

  - Output: the list of results (mappings) of P on T

$$\{x : [186, 200\rangle\}, \quad \{x : [483, 500\rangle\}, \quad \ldots$$

A simpler problem: Enumerating results of regular spanners

- Problem description:
  - Input:
    - A string T

```
Antoine Amarilli Description Name Antoine Amarilli. Handle: a3nm. Identity Born 1990-02-07. French national. Appearance as of 2017.
Auth OpenPGP. OpenId. Bitcoin. Contact Email and XMPP a3nm@a3nm.net Affiliation Associate professor of computer science (office C201-4)
in the DIG team of Télécom Paris, 46 rue Barrault, F-75634 Paris Cedex 13, France. Studies PhD in computer science awarded by Télécom
ParisTech on March 14, 2016. Former student of the École normale supérieure. test@example.com More Résumé Location Other sites Blogging:
a3nm.net/blog Git: a3nm.net/git ...
```

    - A regular spanner P

      $\Sigma^* \vdash_x \texttt{[a-z]}^+ \texttt{ @ } \texttt{[a-z]}^+ \texttt{ . } \texttt{[a-z]}^+ \dashv_x \Sigma^*$
  - Output: the list of results (mappings) of P on T

$$\{x : [186, 200\rangle\}, \quad \{x : [483, 500\rangle\}, \quad \ldots$$

- Goal: be very efficient in T and reasonably efficient in P

A simpler problem: Enumerating results of regular spanners

- **Problem description:**
  - **Input:**
    - A **string** T

      ```
      Antoine Amarilli Description Name Antoine Amarilli. Handle: a3nm. Identity Born 1990-02-07. French national. Appearance as of 2017.
      Auth OpenPGP. OpenId. Bitcoin. Contact Email and XMPP a3nm@a3nm.net Affiliation Associate professor of computer science (office C201-4)
      in the DIG team of Télécom Paris, 46 rue Barrault, F-75634 Paris Cedex 13, France. Studies PhD in computer science awarded by Télécom
      ParisTech on March 14, 2016. Former student of the École normale supérieure. test@example.com More Résumé Location Other sites Blogging:
      a3nm.net/blog Git: a3nm.net/git ...
      ```

    - A **regular spanner** P

      $\Sigma^* \vdash_x \texttt{[a-z]}^+ \texttt{@} \texttt{[a-z]}^+ \texttt{.} \texttt{[a-z]}^+ \dashv_x \Sigma^*$
  - **Output:** the list of **results** (mappings) of P on T

$$\{x : [186, 200\rangle\}, \quad \{x : [483, 500\rangle\}, \quad \ldots$$

- **Goal:** be **very efficient** in T and **reasonably efficient** in P

We mostly focus on a **simpler case**: $P = \Sigma^* \vdash_x e \dashv_x \Sigma^*$ where e is a regular expression

$\rightarrow$ Find all substrings of T satisfying e

Naive algorithms

What is the complexity of finding all substrings of a word T satisfying a regular expression e?

Naive algorithms

What is the complexity of finding all substrings of a word T satisfying a regular expression e?

- **Naive algorithm:** Run an automaton A for e on **each substring** of T
  $\rightarrow$ **Complexity** is $O(|T|^2 \times |T|)$

Naive algorithms

What is the complexity of finding all substrings of a word T satisfying a regular expression e?

- **Naive algorithm:** Run an automaton A for e on **each substring** of T
  - $\rightarrow$ **Complexity** is $O(|T|^2 \times |T|)$
  - $\rightarrow$ Can be **optimized** to $O(|T|^2)$

Naive algorithms

What is the complexity of finding all substrings of a word T satisfying a regular expression e?

- **Naive algorithm:** Run an automaton A for e on **each substring** of T
  - → **Complexity** is $O(|T|^2 \times |T|)$
  - → Can be **optimized** to $O(|T|^2)$
- **Problem:** We may need to output $\Omega(|T|^2)$ matching substrings:

Naive algorithms

What is the complexity of finding all substrings of a word T satisfying a regular expression e?

- **Naive algorithm:** Run an automaton A for e on **each substring** of T
  - → **Complexity** is $O(|T|^2 \times |T|)$
  - → Can be **optimized** to $O(|T|^2)$
- **Problem:** We may need to output $\Omega(|T|^2)$ matching substrings:
  - Consider the **string** T:

    aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Naive algorithms

What is the complexity of finding all substrings of a word T satisfying a regular expression e?

- **Naive algorithm:** Run an automaton A for e on **each substring** of T
  - → **Complexity** is $O(|T|^2 \times |T|)$
  - → Can be **optimized** to $O(|T|^2)$
- **Problem:** We may need to output $\Omega(|T|^2)$ matching substrings:
  - Consider the **string** T:

    `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`

  - Consider the **regular expression** $e := \texttt{a}^*$

Naive algorithms

What is the complexity of finding all substrings of a word T satisfying a regular expression e?

- **Naive algorithm:** Run an automaton A for e on **each substring** of T
  - $\rightarrow$ **Complexity** is $O(|T|^2 \times |T|)$
  - $\rightarrow$ Can be **optimized** to $O(|T|^2)$

- **Problem:** We may need to output $\Omega(|T|^2)$ matching substrings:
  - Consider the **string** T:

    `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`

  - Consider the **regular expression** $e := \mathtt{a}^*$
  - The **number of matches** is $\Omega(|T|^2)$

Naive algorithms

What is the complexity of finding all substrings of a word T satisfying a regular expression e?

- **Naive algorithm:** Run an automaton A for e on **each substring** of T
  - $\rightarrow$ **Complexity** is $O(|T|^2 \times |T|)$
  - $\rightarrow$ Can be **optimized** to $O(|T|^2)$
- **Problem:** We may need to output $\Omega(|T|^2)$ matching substrings:
  - Consider the **string** T:

    `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`

  - Consider the **regular expression** $e := \texttt{a}^*$
  - The **number of matches** is $\Omega(|T|^2)$

$\rightarrow$ We need a **different way** to measure complexity

Formalizing enumeration algorithms

```
Antoine Amarilli Description Name Antoine
Amarilli. Handle: a3nm. Identity Born
1990-02-07. French national. Appearance as
of 2017. Auth OpenPGP. OpenId. Bitcoin.
Contact Email and XMPP a3nm@a3nm.net
Affiliation Associate professor ...
```

String T

$$\Sigma^* \vdash_x \texttt{[a-z]}^+$$
$$\texttt{@}\ \texttt{[a-z]}^+\ \texttt{.}$$
$$\texttt{[a-z]}^+\ \dashv_x \Sigma^*$$

Regular spanner P

Formalizing enumeration algorithms

```
Antoine Amarilli Description Name Antoine
Amarilli.  Handle:  a3nm.  Identity Born
1990-02-07.  French national.  Appearance as
of 2017.  Auth OpenPGP. OpenId.  Bitcoin.
Contact Email and XMPP a3nm@a3nm.net
Affiliation Associate professor ...
```

String T

$\Sigma^* \vdash_x$ `[a-z]`$^+$ `@` `[a-z]`$^+$ `.` `[a-z]`$^+$ $\dashv_x \Sigma^*$

Regular spanner P

Phase 1: Preprocessing

Formalizing enumeration algorithms

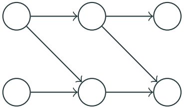
```
Antoine Amarilli Description Name Antoine
Amarilli. Handle: a3nm. Identity Born
1990-02-07. French national. Appearance as
of 2017. Auth OpenPGP. OpenId. Bitcoin.
Contact Email and XMPP a3nm@a3nm.net
Affiliation Associate professor ...
```

String T

$\Sigma^* \vdash_x$ `[a-z]`$^+$ `@` `[a-z]`$^+$ `.` `[a-z]`$^+$ $\dashv_x \Sigma^*$

Regular spanner P

Phase 1: Preprocessing



Index structure

Formalizing enumeration algorithms

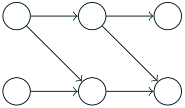
```
Antoine Amarilli Description Name Antoine
Amarilli. Handle: a3nm. Identity Born
1990-02-07. French national. Appearance as
of 2017. Auth OpenPGP. OpenId. Bitcoin.
Contact Email and XMPP a3nm@a3nm.net
Affiliation Associate professor ...
```

String T

$$\Sigma^* \vdash_x \texttt{[a-z]}^+ \ \texttt{@} \ \texttt{[a-z]}^+ \ \texttt{.} \ \texttt{[a-z]}^+ \dashv_x \Sigma^*$$

Regular spanner P

Phase 1: Preprocessing



Index structure

Phase 2: Enumeration

Formalizing enumeration algorithms

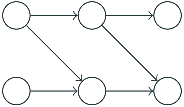
```
Antoine Amarilli Description Name Antoine
Amarilli.  Handle:  a3nm.  Identity Born
1990-02-07.  French national.  Appearance as
of 2017.  Auth OpenPGP. OpenId.  Bitcoin.
Contact Email and XMPP a3nm@a3nm.net
Affiliation Associate professor ...
```

String T

$\Sigma^* \vdash_x$ `[a-z]`$^+$ `@` `[a-z]`$^+$ `.` `[a-z]`$^+$ $\dashv_x \Sigma^*$

Regular spanner P

Phase 1: **Preprocessing**



Index structure

Phase 2: **Enumeration**

$\{x : [42, 57\rangle\},$

Results

Formalizing enumeration algorithms

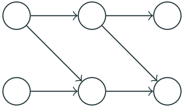
```
Antoine Amarilli Description Name Antoine
Amarilli. Handle: a3nm. Identity Born
1990-02-07. French national. Appearance as
of 2017. Auth OpenPGP. OpenId. Bitcoin.
Contact Email and XMPP a3nm@a3nm.net
Affiliation Associate professor ...
```

String T

$\Sigma^* \vdash_x$ `[a-z]`$^+$
`@` `[a-z]`$^+$ `.`
`[a-z]`$^+$ $\dashv_x \Sigma^*$

Regular spanner P

Phase 1:
Preprocessing



Index structure

Phase 2:
Enumeration

$\{x : [42, 57\rangle\}$,
$\{x : [1337, 1351\rangle\}$

Results

Formalizing enumeration algorithms

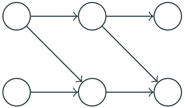
```
Antoine Amarilli Description Name Antoine
Amarilli.  Handle:  a3nm.  Identity Born
1990-02-07.  French national.  Appearance as
of 2017.  Auth OpenPGP. OpenId.  Bitcoin.
Contact Email and XMPP a3nm@a3nm.net
Affiliation Associate professor ...
```

String T

$\Sigma^* \vdash_x$ `[a-z]`$^+$ `@` `[a-z]`$^+$ `.` `[a-z]`$^+$ $\dashv_x \Sigma^*$

Regular spanner P

Phase 1: Preprocessing



Index structure

Phase 2: Enumeration

$\{x : [42, 57\rangle\}$, $\{x : [1337, 1351\rangle\}$

Results

Two ways to measure performance:

- Total time for phase 1
- Delay between two results in phase 2

... as a function of the string T and regular spanner P

Results for enumerating regexp matches and regular spanner mappings

For the problem of enumerating regexp matches, we can show:

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

Given a string T and a deterministic automaton A, we can enumerate the subword occurrences in T that match A with preprocessing $O(|T| \times |A|)$ and constant delay.

Results for enumerating regexp matches and regular spanner mappings

For the problem of enumerating regexp matches, we can show:

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

Given a string T and a deterministic automaton A, we can enumerate the subword occurrences in T that match A with preprocessing $O(|T| \times |A|)$ and constant delay.

We can show (with more effort):

Theorem (joint work with Pierre Bourhis, Stefan Mengel, Matthias Niewerth)

For a nondeterministic automaton A, we can enumerate the subword occurrences in T that match A with preprocessing $O(|T| \times Poly(|A|))$ and delay $O(Poly(|A|))$.

Results for enumerating regexp matches and regular spanner mappings

For the problem of enumerating **regexp matches**, we can show:

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

*Given a string T and a **deterministic automaton** A, we can enumerate the subword occurrences in T that match A with preprocessing $O(|T| \times |A|)$ and constant delay.*

We can show (with more effort):

Theorem (joint work with Pierre Bourhis, Stefan Mengel, Matthias Niewerth)

*For a **nondeterministic automaton** A, we can enumerate the subword occurrences in T that match A with preprocessing $O(|T| \times Poly(|A|))$ and delay $O(Poly(|A|))$.*

- Remark: if the regexp is just **a word** or a **set of words**, the complexity is $O(|T| \times |A|)$: not helpful compared to Knuth-Morris-Pratt / Aho-Corasick.
- We can achieve the **same complexity** for regular spanners (not just regexps)

Incremental maintenance

Database motivation: Incremental validation

- Validate if an XML document satisfies a schema
- Maintain this information as the document is updated
- Relates to incremental view maintenance in relational databases

Database motivation: Incremental validation

- Validate if an XML document satisfies a schema
- Maintain this information as the document is updated
- Relates to incremental view maintenance in relational databases

Idea: if the document is large, we want to avoid re-validating the document from scratch

Database motivation: Incremental validation

- Validate if an **XML document** satisfies a **schema**
- **Maintain** this information as the document is updated
- Relates to **incremental view maintenance** in relational databases

Idea: if the document is large, we want to **avoid re-validating the document from scratch**

$\rightarrow$ **Forget trees**: what about incremental validation on **strings**?

Simpler problem: Dynamic membership for regular languages

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

Simpler problem: Dynamic membership for regular languages

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

Simpler problem: Dynamic membership for regular languages

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

Simpler problem: Dynamic membership for regular languages

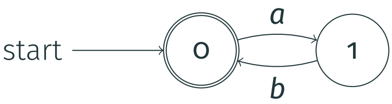
- Fix a **regular language** L
  - → E.g., $L = (ab)^*$
- Read an **input string** T with $n := |T|$
  - → E.g., $T = abbbab$
- **Maintain** the membership of T to L under **substitution updates**
  - → Initially, we have $T \notin L$
  - → Replace character at position 3 with a: we now have $T \in L$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

Theorem

For any regular language L recognized by an automaton A, given a string T, we can maintain dynamic membership of T to L under substitution updates in $O(Poly(|A|) \times \log |T|)$ per update.

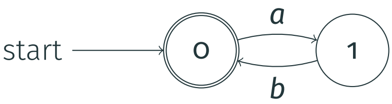
Proof of the $O(\log n)$ algorithm

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



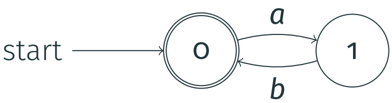
Proof of the $O(\log n)$ algorithm

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

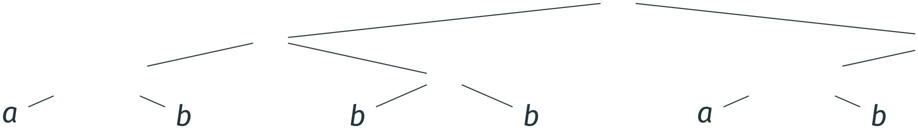


- Build a **balanced binary tree** on the input string $T = abbbab$

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

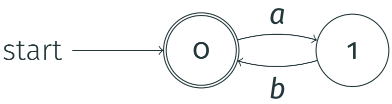
Fix the language $L = (ab)^*$: 

- Build a balanced binary tree on the input string $T = abbbab$

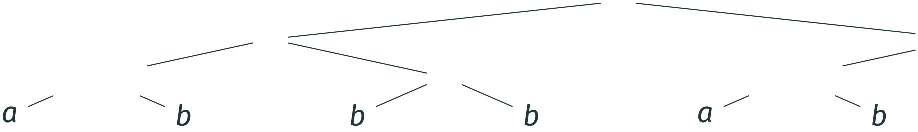


Proof of the $O(\log n)$ algorithm

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

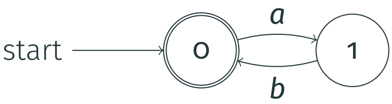


- Build a balanced binary tree on the input string $T = abbbab$
- Label each node n by the transition monoid element: all pairs $q \rightsquigarrow q'$ such that we can go from q to q' by reading the substring below n

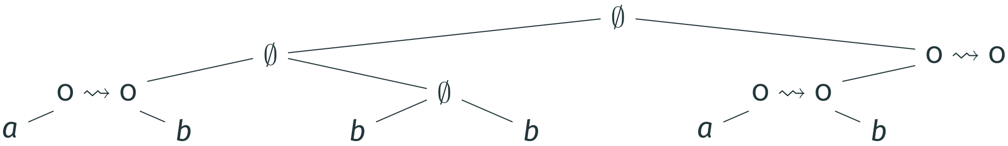


Proof of the $O(\log n)$ algorithm

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

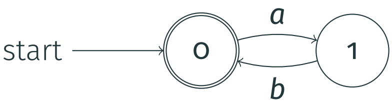


- Build a **balanced binary tree** on the input string $T = abbbab$
- Label each node n by the **transition monoid** element: all pairs $q \rightsquigarrow q'$ such that we can go from q to q' by reading the substring below n

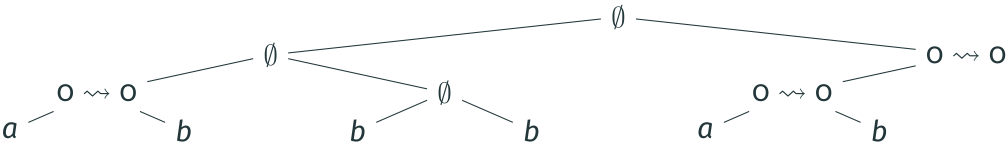


Proof of the $O(\log n)$ algorithm

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



- Build a balanced binary tree on the input string $T = abbbab$
- 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 substring below n



- The tree root describes if $T \in L$
- 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

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 $T \in L$ 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 $T \in L$ iff there are no violations

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

Summary of our results (joint work with Louis Jachiet and Charles Paperman)

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 (joint work with Louis Jachiet and Charles Paperman)

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 by Skovbjerg Frandsen et al.)

Summary of our results (joint work with Louis Jachiet and Charles Paperman)

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 by Skovbjerg Frandsen et al.)
- The problem is always in $O(\log n / \log \log n)$

Combining incremental maintenance and enumeration

- We have looked at **dynamic membership** for regular languages…
  - $\rightarrow$ … and showed how to maintain it in $O(\log |T|)$ **time per update**
- We have looked at **enumeration** for regular spanners…
  - $\rightarrow$ … and showed how to perform it with **linear preprocessing and constant delay**

Can we get the best of both worlds?

Combining incremental maintenance and enumeration

- We have looked at **dynamic membership** for regular languages...
  - → ... and showed how to maintain it in $O(\log |T|)$ **time per update**
- We have looked at **enumeration** for regular spanners...
  - → ... and showed how to perform it with **linear preprocessing and constant delay**

Can we get the best of both worlds?

Theorem (follows from [Niewerth and Segoufin, 2018])

*Given a string T and a **fixed** document spanner P, we can enumerate the results of P on T with **preprocessing** $O(|T|)$ and **delay independent from** $|T|$, and we can maintain the enumeration structure in $O(\log |T|)$ **time per substitution update**.*

Combining incremental maintenance and enumeration

- We have looked at **dynamic membership** for regular languages...
  - → ... and showed how to maintain it in $O(\log |T|)$ **time per update**
- We have looked at **enumeration** for regular spanners...
  - → ... and showed how to perform it with **linear preprocessing and constant delay**

Can we get the best of both worlds?

Theorem (follows from [Niewerth and Segoufin, 2018])

Given a string T and a fixed document spanner P, we can enumerate the results of P on T with preprocessing $O(|T|)$ and delay independent from $|T|$, and we can maintain the enumeration structure in $O(\log |T|)$ time per substitution update.

Theorem (joint work with Pierre Bourhis, Stefan Mengel, Matthias Niewerth)

We can do the same with preprocessing $O(|T| \times \mathit{Poly}(|A|))$, delay $O(\mathit{Poly}(|A|))$, and updates $O(\mathit{Poly}(|A|) \times \log |T|)$, where A is a nondeterministic automaton for P.

Combining incremental maintenance and enumeration

- We have looked at **dynamic membership** for regular languages...
  - → ... and showed how to maintain it in **$O(\log |T|)$ time per update**
- We have looked at **enumeration** for regular spanners...
  - → ... and showed how to perform it with **linear preprocessing and constant delay**

Can we get the best of both worlds?

Theorem (follows from [Niewerth and Segoufin, 2018])

Given a string T and a fixed document spanner P, we can enumerate the results of P on T with preprocessing $O(|T|)$ and delay independent from $|T|$, and we can maintain the enumeration structure in $O(\log |T|)$ time per substitution update.

Theorem (joint work with Pierre Bourhis, Stefan Mengel, Matthias Niewerth)

We can do the same with preprocessing $O(|T| \times \mathit{Poly}(|A|))$, delay $O(\mathit{Poly}(|A|))$, and updates $O(\mathit{Poly}(|A|) \times \log |T|)$, where A is a nondeterministic automaton for P.

Also generalizes to **trees** for a suitable notion of tree automata with captures

Conclusion

Summary, ongoing research, and open problems

- The message: some database theory questions are better answered by stringology
  - **Regular spanners** as a formalism for expressive pattern matching tasks on strings
  - **Enumeration problems** on strings to produce large sets of results efficiently
  - **Incremental maintenance problems** to maintain results under changes to the string
  - Other domains, e.g., **Regular path queries** on graph databases

Summary, ongoing research, and open problems

- The message: some database theory questions are better answered by stringology
  - **Regular spanners** as a formalism for expressive pattern matching tasks on strings
  - **Enumeration problems** on strings to produce large sets of results efficiently
  - **Incremental maintenance problems** to maintain results under changes to the string
  - Other domains, e.g., **Regular path queries** on graph databases

Directions for further research (talk to me to know more!):

- **Better update complexity than $O(\log n)$** for some **enumeration** tasks

  → Ongoing work with Sven Dziadek and Luc Segoufin

Summary, ongoing research, and open problems

- The message: some database theory questions are better answered by stringology
  - **Regular spanners** as a formalism for expressive pattern matching tasks on strings
  - **Enumeration problems** on strings to produce large sets of results efficiently
  - **Incremental maintenance problems** to maintain results under changes to the string
  - Other domains, e.g., **Regular path queries** on graph databases

Directions for further research (talk to me to know more!):

- **Better update complexity than $O(\log n)$** for some **enumeration** tasks
  → Ongoing work with Sven Dziadek and Luc Segoufin
- **Better update complexity than $O(\log n)$** for **tree languages**
  → Ongoing with Corentin Barloy, Pawel Gawrychowski, Louis Jachiet, Charles Paperman

Summary, ongoing research, and open problems

- The message: some database theory questions are better answered by stringology
  - **Regular spanners** as a formalism for expressive pattern matching tasks on strings
  - **Enumeration problems** on strings to produce large sets of results efficiently
  - **Incremental maintenance problems** to maintain results under changes to the string
  - Other domains, e.g., **Regular path queries** on graph databases

Directions for further research (talk to me to know more!):

- **Better update complexity than $O(\log n)$** for some **enumeration** tasks
  $\rightarrow$ Ongoing work with Sven Dziadek and Luc Segoufin
- **Better update complexity than $O(\log n)$** for **tree languages**
  $\rightarrow$ Ongoing with Corentin Barloy, Pawel Gawrychowski, Louis Jachiet, Charles Paperman
- **More general updates**: insertions/deletions, tree modifications, search/replace...

Summary, ongoing research, and open problems

- The message: some database theory questions are better answered by stringology
  - **Regular spanners** as a formalism for expressive pattern matching tasks on strings
  - **Enumeration problems** on strings to produce large sets of results efficiently
  - **Incremental maintenance problems** to maintain results under changes to the string
  - Other domains, e.g., **Regular path queries** on graph databases

Directions for further research (talk to me to know more!):

- **Better update complexity than $O(\log n)$** for some **enumeration** tasks
  → Ongoing work with Sven Dziadek and Luc Segoufin
- **Better update complexity than $O(\log n)$** for **tree languages**
  → Ongoing with Corentin Barloy, Pawel Gawrychowski, Louis Jachiet, Charles Paperman
- **More general updates**: insertions/deletions, tree modifications, search/replace...
- **Ranked enumeration**, direct access, membership testing queries, etc.

Summary, ongoing research, and open problems

- The message: some database theory questions are better answered by stringology
  - **Regular spanners** as a formalism for expressive pattern matching tasks on strings
  - **Enumeration problems** on strings to produce large sets of results efficiently
  - **Incremental maintenance problems** to maintain results under changes to the string
  - Other domains, e.g., **Regular path queries** on graph databases

Directions for further research (talk to me to know more!):

- **Better update complexity than $O(\log n)$** for some **enumeration** tasks
  $\rightarrow$ Ongoing work with Sven Dziadek and Luc Segoufin
- **Better update complexity than $O(\log n)$** for **tree languages**
  $\rightarrow$ Ongoing with Corentin Barloy, Pawel Gawrychowski, Louis Jachiet, Charles Paperman
- **More general updates**: insertions/deletions, tree modifications, search/replace...
- **Ranked enumeration**, direct access, membership testing queries, etc.
- Enumeration for document spanners with **string equality selection**

Summary, ongoing research, and open problems

- The message: some database theory questions are better answered by stringology
  - **Regular spanners** as a formalism for expressive pattern matching tasks on strings
  - **Enumeration problems** on strings to produce large sets of results efficiently
  - **Incremental maintenance problems** to maintain results under changes to the string
  - Other domains, e.g., **Regular path queries** on graph databases

Directions for further research (talk to me to know more!):

- **Better update complexity than $O(\log n)$** for some **enumeration** tasks
  → Ongoing work with Sven Dziadek and Luc Segoufin
- **Better update complexity than $O(\log n)$** for **tree languages**
  → Ongoing with Corentin Barloy, Pawel Gawrychowski, Louis Jachiet, Charles Paperman
- **More general updates**: insertions/deletions, tree modifications, search/replace...
- **Ranked enumeration**, direct access, membership testing queries, etc.
- Enumeration for document spanners with **string equality selection**
- Enumerating large answers via **diffs**

Summary, ongoing research, and open problems

- The message: some database theory questions are better answered by stringology
  - **Regular spanners** as a formalism for expressive pattern matching tasks on strings
  - **Enumeration problems** on strings to produce large sets of results efficiently
  - **Incremental maintenance problems** to maintain results under changes to the string
  - Other domains, e.g., **Regular path queries** on graph databases

Directions for further research (talk to me to know more!):

- **Better update complexity than $O(\log n)$** for some **enumeration** tasks
  → Ongoing work with Sven Dziadek and Luc Segoufin
- **Better update complexity than $O(\log n)$** for **tree languages**
  → Ongoing with Corentin Barloy, Pawel Gawrychowski, Louis Jachiet, Charles Paperman
- **More general updates**: insertions/deletions, tree modifications, search/replace...
- **Ranked enumeration**, direct access, membership testing queries, etc.
- Enumeration for document spanners with **string equality selection**
- Enumerating large answers via **diffs**

Thanks for your attention!

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*.

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

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

Niewerth, M. and Segoufin, L. (2018).
Enumeration of MSO queries on strings with constant delay and logarithmic updates.
In *PODS*.