Enumerating Pattern Matches in Words and Trees

Antoine Amarilli[1], Pierre Bourhis[2], Stefan Mengel[3], Matthias Niewerth[4]

October 4th, 2018

[1]Télécom ParisTech

[2]CNRS CRIStAL

[3]CNRS CRIL

[4]Universität Bayreuth

Problem: Finding patterns in text

- We have a long text 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 ParisTech, 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.
More Résumé Location Other sites Blogging:  a3nm.net/blog Git:  a3nm.net/git ...
```

Problem: Finding patterns in text

- We have a long text 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 ParisTech, 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.
More Résumé Location Other sites Blogging:  a3nm.net/blog Git:  a3nm.net/git ...
```

- We want to find a pattern P in the text T:
  - $\rightarrow$ Example: find email addresses

Problem: Finding patterns in text

- We have a long text 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 ParisTech, 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.
More Résumé Location Other sites Blogging:  a3nm.net/blog Git:  a3nm.net/git ...
```

- We want to find a pattern P in the text T:
  - → Example: find email addresses
    - Write the pattern as a regular expression:

$$P := \texttt{␣}^{+}\ \texttt{[a-z0-9.]}^{*}\ \texttt{@}\ \texttt{[a-z0-9.]}^{*}\ \texttt{␣}^{+}$$

Problem: Finding patterns in text

- We have a long text 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 ParisTech, 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.
More Résumé Location Other sites Blogging:  a3nm.net/blog Git:  a3nm.net/git ...
```

- We want to find a pattern P in the text T:
  - → Example: find email addresses
    - Write the pattern as a regular expression:

$$P := \texttt{␣}^+ \ \texttt{[a-z0-9.]}^* \ \texttt{@} \ \texttt{[a-z0-9.]}^* \ \texttt{␣}^+$$

→ How to find the pattern P efficiently in the text T?

Solution: automata

- Convert the pattern from a regular expression to an automaton

Solution: automata

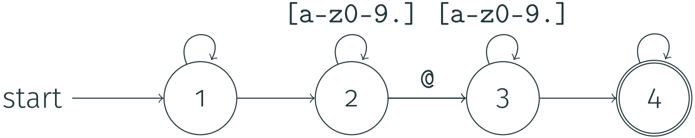
- Convert the pattern from a regular expression to an automaton

$$P := \texttt{␣}^{+}\ \texttt{[a-z0-9.]}^{*}\ \texttt{@}\ \texttt{[a-z0-9.]}^{*}\ \texttt{␣}^{+}$$

Solution: automata

- Convert the pattern from a regular expression to an automaton

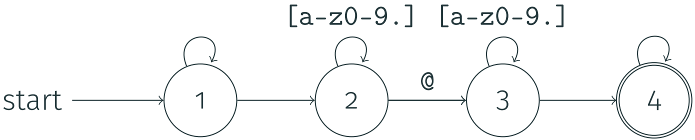
$$P := \textvisiblespace^{+}\ \texttt{[a-z0-9.]}^{*}\ \texttt{@}\ \texttt{[a-z0-9.]}^{*}\ \textvisiblespace^{+}$$



Solution: automata

- Convert the pattern from a regular expression to an automaton

$$P := \textvisiblespace^{+}\ \texttt{[a-z0-9.]}^{*}\ \texttt{@}\ \texttt{[a-z0-9.]}^{*}\ \textvisiblespace^{+}$$



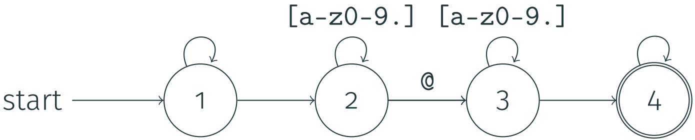
- Then, evaluate the automaton on the text T

```
...  Email and XMPP a3nm@a3nm.net Affiliation
Associate professor of computer ...
```

Solution: automata

- Convert the pattern from a regular expression to an automaton

$$P := \text{␣}^+ \ \texttt{[a-z0-9.]}^* \ \texttt{@} \ \texttt{[a-z0-9.]}^* \ \text{␣}^+$$



- Then, evaluate the automaton on the text T

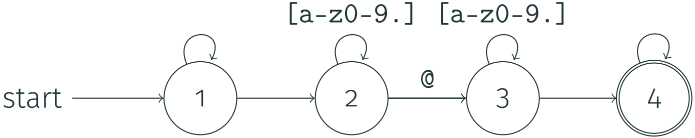
```
...  Email and XMPP a3nm@a3nm.net Affiliation
Associate professor of computer ...
```

- How efficient is this?

Solution: automata

- Convert the pattern from a **regular expression** to an **automaton**

$$P := \textvisiblespace^{+}\ \texttt{[a-z0-9.]}^{*}\ \texttt{@}\ \texttt{[a-z0-9.]}^{*}\ \textvisiblespace^{+}$$



- Then, evaluate the automaton on the **text** T

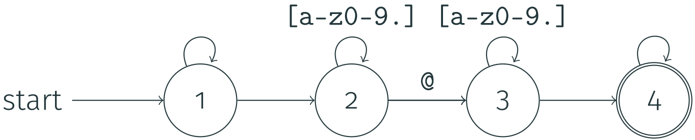
```
...  Email and XMPP a3nm@a3nm.net Affiliation
Associate professor of computer ...
```

- How **efficient** is this?
    - **Data complexity** in the text T: **linear**, i.e., $O(|T|)$

Solution: automata

- Convert the pattern from a regular expression to an automaton

$$P := \texttt{␣}^+ \ \texttt{[a-z0-9.]}^* \ \texttt{@} \ \texttt{[a-z0-9.]}^* \ \texttt{␣}^+$$



- Then, evaluate the automaton on the text T

```
...  Email and XMPP a3nm@a3nm.net Affiliation
Associate professor of computer ...
```

- How efficient is this?
  - Data complexity in the text T: linear, i.e., $O(|T|)$
  - Combined complexity in T and P: polynomial

Actual problem: Extracting all patterns

- **Problem:** This only tells us **if** the pattern is in the text!
  $\rightarrow$ ``YES''

Actual problem: Extracting all patterns

- **Problem:** This only tells us **if** the pattern is in the text!
  $\rightarrow$ ``YES''

- We want to **actually find** all pattern matches!

Actual problem: Extracting all patterns

- Problem: This only tells us if the pattern is in the text!
  $\rightarrow$ ‘‘YES’’
- We want to actually find all pattern matches!
- Write the pattern P as a regular expression with capture variables

$$P := \sqcup^{+}\ \alpha\ \texttt{[a-z0-9.]}^{*}\ \texttt{@}\ \texttt{[a-z0-9.]}^{*}\ \beta\ \sqcup^{+}$$

Actual problem: Extracting all patterns

- **Problem:** This only tells us **if** the pattern is in the text!
  $\rightarrow$ ``YES''

- We want to **actually find** all pattern matches!

- Write the pattern P as a regular expression with capture variables

$$P := \text{␣}^{+}\ \alpha\ \texttt{[a-z0-9.]}^{*}\ \texttt{@}\ \texttt{[a-z0-9.]}^{*}\ \beta\ \text{␣}^{+}$$

- **Semantics:** a **match** of P maps α and β to **positions** of T

Actual problem: Extracting all patterns

- **Problem:** This only tells us **if** the pattern is in the text!
  $\rightarrow$ "YES"

- We want to **actually find** all pattern matches!

- Write the pattern P as a regular expression with capture variables

$$P := \sqcup^+ \; \alpha \; \texttt{[a-z0-9.]}^* \; \texttt{@} \; \texttt{[a-z0-9.]}^* \; \beta \; \sqcup^+$$

- **Semantics:** a **match** of P maps α and β to **positions** of T

```
 ...  Email and XMPP a3nm@a3nm.net Affiliation
Associate professor of computer ...
```

Actual problem: Extracting all patterns

- Problem: This only tells us if the pattern is in the text!

  $\rightarrow$ "YES"

- We want to actually find all pattern matches!

- Write the pattern P as a regular expression with capture variables

$$P := \text{␣}^{+}\ \alpha\ \texttt{[a-z0-9.]}^{*}\ \texttt{@}\ \texttt{[a-z0-9.]}^{*}\ \beta\ \text{␣}^{+}$$

- Semantics: a match of P maps α and β to positions of T

```
 ...  Email and XMPP a3nm@a3nm.net Affiliation
Associate professor of computer ...
```

$\rightarrow$ One match: $\langle \alpha : 20, \beta : 32 \rangle$

Formal problem statement

- Problem description:

Formal problem statement

- Problem description:
  - Input:
    - A text 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
ParisTech, 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. More Résumé
Location Other sites Blogging: a3nm.net/blog Git: a3nm.net/git ...
```

Formal problem statement

- Problem description:
  - Input:
    - A text 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
ParisTech, 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. More Résumé
Location Other sites Blogging: a3nm.net/blog Git: a3nm.net/git ...
```

    - A pattern P given as a regular expression with capture variables

$$P := \text{␣}^{+}\ \alpha\ \texttt{[a-z0-9.]}^{*}\ \texttt{@}\ \texttt{[a-z0-9.]}^{*}\ \beta\ \text{␣}^{+}$$

Formal problem statement

- Problem description:
  - Input:
    - A text 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
ParisTech, 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.  More Résumé
Location Other sites Blogging:  a3nm.net/blog Git:  a3nm.net/git ...
```

    - A pattern P given as a regular expression with capture variables

$$P := \textvisiblespace^{+}\ {\color{red}\alpha}\ \texttt{[a-z0-9.]}^{*}\ \texttt{@}\ \texttt{[a-z0-9.]}^{*}\ {\color{red}\beta}\ \textvisiblespace^{+}$$

  - Output: the list of matches of P on T

$$\langle {\color{red}\alpha} : 187, {\color{red}\beta} : 199\rangle, \dots$$

Formal problem statement

- Problem description:
  - Input:
    - A text 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
ParisTech, 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.  More Résumé
Location Other sites Blogging:  a3nm.net/blog Git:  a3nm.net/git ...
```

    - A pattern P given as a regular expression with capture variables

$$P := \textvisiblespace^+ \ \alpha\ \texttt{[a-z0-9.]}^*\ \texttt{@}\ \texttt{[a-z0-9.]}^*\ \beta\ \textvisiblespace^+$$

  - Output: the list of matches of P on T

$$\langle \alpha : 187, \beta : 199 \rangle, \ldots$$

- We measure the complexity of the problem:
  - In data complexity, as a function of T
  - In combined complexity, as a function of P and T

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

l	o	l

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

$\alpha\beta$ `l    o    l`

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

α l β o l

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

α `l` `o` β `l`

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

$\boxed{\alpha \quad \texttt{l} \quad \texttt{o} \quad \texttt{l} \quad \beta}$

Measuring the complexity

- **Naive algorithm:** Consider **all ways** to assign capture variables and **test** for each of them if it satisfies the pattern

β `l` α `o` `l`

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

`l` $\alpha\beta$ `o` `l`

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

$$\boxed{\texttt{l}\ \alpha\ \ \texttt{o}\ \ \beta\ \texttt{l}}$$

Measuring the complexity

- **Naive algorithm:** Consider **all ways** to assign capture variables and **test** for each of them if it satisfies the pattern

`l` α `o` `l` β

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

$\boxed{\quad\beta\ \mathtt{l}\qquad \mathtt{o}\ \alpha\quad \mathtt{l}\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}$

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

l β o α l

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

`    l    o` $\alpha\beta$ `l`

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

l	o	α	l	β	

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

β `l` `o` `l` α

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

`l` β `o` `l` α

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

l o β l α

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

`l` `o` `l` $\alpha\beta$

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

`l o l`

$\rightarrow$ For k capture variables, data complexity...

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

`l o l`

  $\rightarrow$ For k capture variables, data complexity... $O(|T|^{k+1})$

Measuring the complexity

- **Naive algorithm:** Consider **all ways** to assign capture variables and **test** for each of them if it satisfies the pattern

l	o	l

  $\rightarrow$ For k capture variables, **data complexity**... $O(|T|^{k+1})$

- **Hope:** If T is big, we want data complexity to be in $O(|T|)$

Measuring the complexity

- Naive algorithm: Consider all ways to assign capture variables and test for each of them if it satisfies the pattern

l o l

  $\rightarrow$ For k capture variables, data complexity... $O(|T|^{k+1})$

- Hope: If T is big, we want data complexity to be in $O(|T|)$
- Challenge: This is impossible, there can be too many matches:

Measuring the complexity

- **Naive algorithm:** Consider **all ways** to assign capture variables and **test** for each of them if it satisfies the pattern

```
     l    o    l
```

$\rightarrow$ For k capture variables, **data complexity**... $O(|T|^{k+1})$

- **Hope:** If T is big, we want data complexity to be in $O(|T|)$

- **Challenge:** This is **impossible**, there can be **too many matches**:
  - Consider the **text** T:

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Measuring the complexity

- **Naive algorithm:** Consider **all ways** to assign capture variables and **test** for each of them if it satisfies the pattern

	l	o	l

  → For k capture variables, **data complexity**... $O(|T|^{k+1})$

- **Hope:** If T is big, we want data complexity to be in $O(|T|)$

- **Challenge:** This is **impossible**, there can be **too many matches**:
  - Consider the **text** T:

    `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`

  - Consider the **regexp with captures** $P := \alpha\ \mathtt{a}^*\ \beta$

Measuring the complexity

- **Naive algorithm:** Consider **all ways** to assign capture variables and **test** for each of them if it satisfies the pattern

`l o l`

$\rightarrow$ For k capture variables, **data complexity**... $O(|T|^{k+1})$

- **Hope:** If T is big, we want data complexity to be in $O(|T|)$

- **Challenge:** This is **impossible**, there can be **too many matches**:
  - Consider the **text** T:

`aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`

  - Consider the **regexp with captures** $P := \alpha\ \texttt{a}^*\ \beta$
  - The **number of matches** is $O(|T|^2)$

Measuring the complexity

- **Naive algorithm:** Consider **all ways** to assign capture variables and **test** for each of them if it satisfies the pattern

`l o l`

$\rightarrow$ For k capture variables, **data complexity**... $O(|T|^{k+1})$

- **Hope:** If T is big, we want data complexity to be in $O(|T|)$
- **Challenge:** This is **impossible**, there can be **too many matches**:
  - Consider the **text** T:

    `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`

  - Consider the **regexp with captures** $P := \alpha \ \mathtt{a}^* \ \beta$
  - The **number of matches** is $O(|T|^2)$

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

Enumeration algorithms

Idea: In real life, we do not want to compute all the answers
we just need to be able to enumerate answers quickly

Enumeration algorithms

Idea: In real life, we do not want to compute **all the answers**
we just need to be able to **enumerate** answers quickly

Idea: In real life, we do not want to compute **all the answers**
we just need to be able to **enumerate** answers quickly

Enumeration algorithms

Idea: In real life, we do not want to compute **all the answers**
we just need to be able to **enumerate** answers quickly

...

Idea: In real life, we do not want to compute **all the answers**
we just need to be able to **enumerate** answers quickly

Enumeration algorithms

Idea: In real life, we do not want to compute all the answers
we just need to be able to enumerate answers quickly

→ Formalization: enumeration algorithms

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 ...
```

Text T

$\sqcup^{+}$ α `[a-z0-9.]*` `@`
`[a-z0-9.]*` β $\sqcup^{+}$

Pattern 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 ...
```

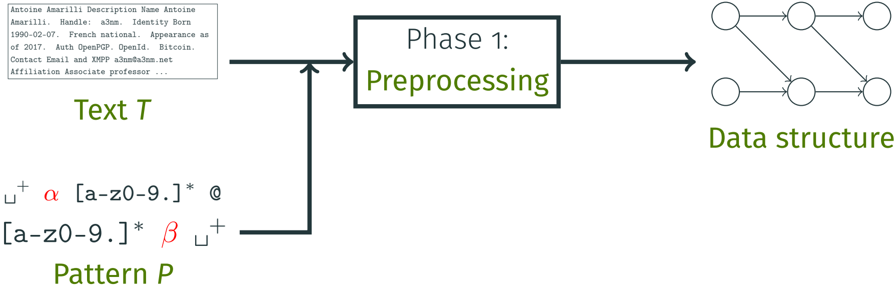
Text T

$\sqcup^{+}$ α [a-z0-9.]* @ [a-z0-9.]* β $\sqcup^{+}$

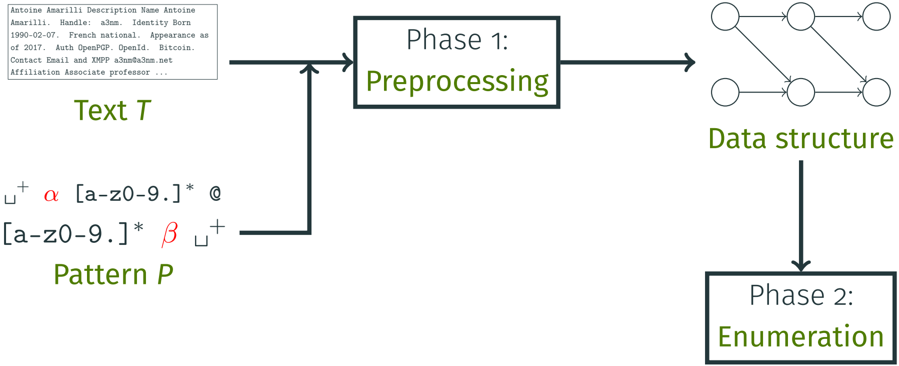
Pattern P

Phase 1:
Preprocessing

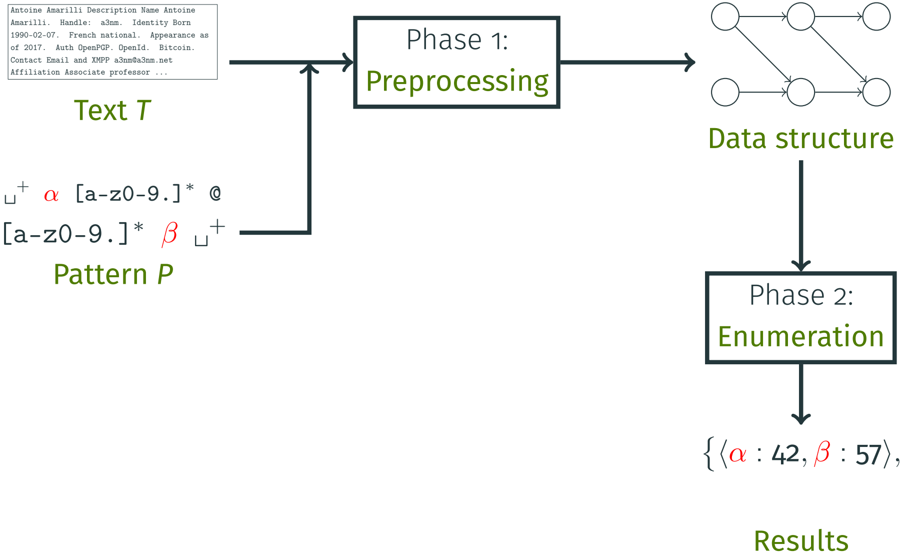
Formalizing enumeration algorithms



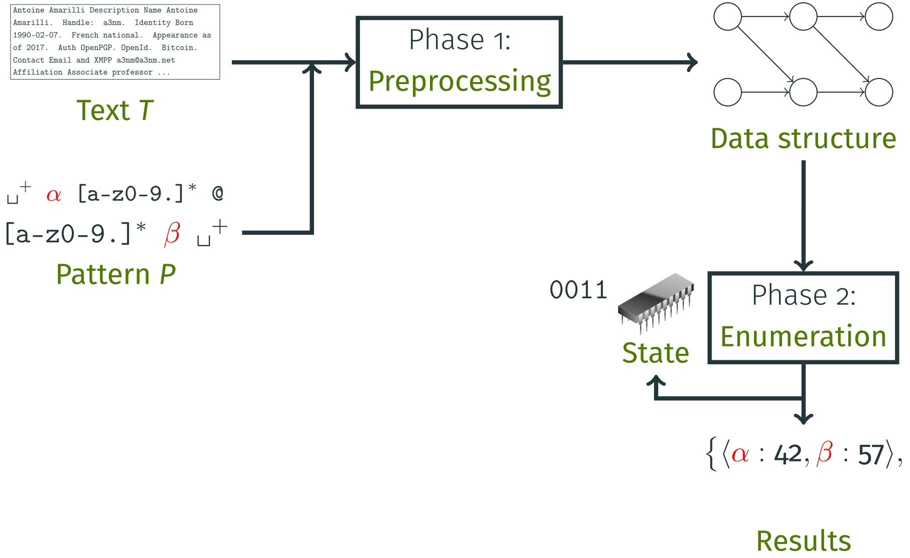
Formalizing enumeration algorithms



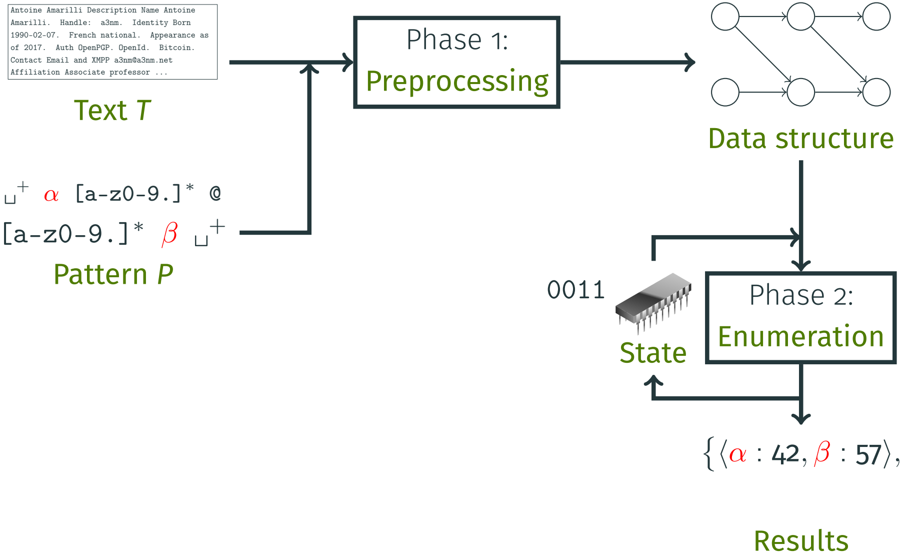
Formalizing enumeration algorithms



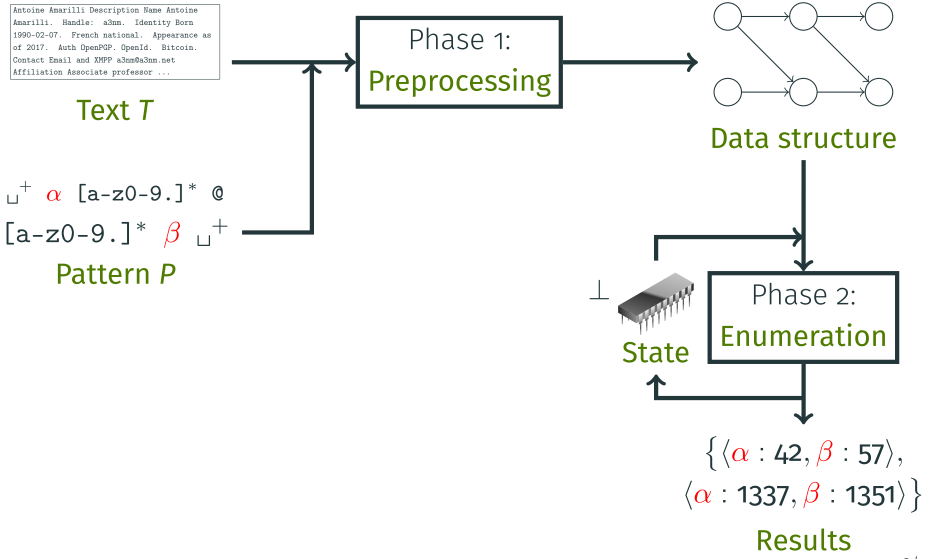
Formalizing enumeration algorithms



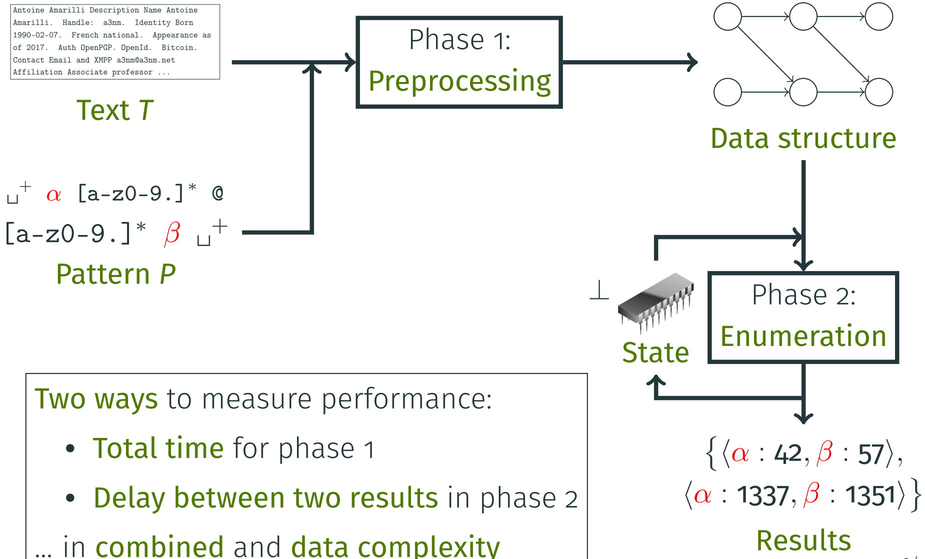
Formalizing enumeration algorithms



Formalizing enumeration algorithms



Formalizing enumeration algorithms



Two ways to measure performance:

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

... in combined and data complexity

Complexity of enumeration algorithms

- Recall the inputs to our problem:
  - The text T
  - The regexp with captures P

$\rightarrow$ Assumption: there is a constant number k of capture variables

Complexity of enumeration algorithms

- Recall the inputs to our problem:
  - The text T
  - The regexp with captures P

  $\rightarrow$ Assumption: there is a constant number k of capture variables

- What is the performance of the naive algorithm?
  - In terms of preprocessing...

Complexity of enumeration algorithms

- Recall the inputs to our problem:
  - The text T
  - The regexp with captures P
  - $\rightarrow$ Assumption: there is a constant number k of capture variables
- What is the performance of the naive algorithm?
  - In terms of preprocessing...
    - Combined complexity is...

Complexity of enumeration algorithms

- Recall the **inputs** to our problem:
  - The **text** T
  - The **regexp with captures** P

  $\rightarrow$ Assumption: there is a **constant number** k of **capture variables**

- What is the performance of the **naive algorithm**?
  - In terms of **preprocessing**...
    - **Combined complexity** is... **polynomial**: convert P to an automaton A

Complexity of enumeration algorithms

- Recall the inputs to our problem:
  - The text T
  - The regexp with captures P
  - → Assumption: there is a constant number k of capture variables

- What is the performance of the naive algorithm?
  - In terms of preprocessing...
    - Combined complexity is... polynomial: convert P to an automaton A
    - Data complexity is...

Complexity of enumeration algorithms

- Recall the inputs to our problem:
  - The text T
  - The regexp with captures P

  $\rightarrow$ Assumption: there is a constant number k of capture variables

- What is the performance of the naive algorithm?
  - In terms of preprocessing...
    - Combined complexity is... polynomial: convert P to an automaton A
    - Data complexity is... constant: nothing to do on T

Complexity of enumeration algorithms

- Recall the inputs to our problem:
  - The text T
  - The regexp with captures P

  $\rightarrow$ Assumption: there is a constant number k of capture variables

- What is the performance of the naive algorithm?
  - In terms of preprocessing...
    - Combined complexity is... polynomial: convert P to an automaton A
    - Data complexity is... constant: nothing to do on T
  - In terms of delay...

Complexity of enumeration algorithms

- Recall the inputs to our problem:
  - The text T
  - The regexp with captures P

  $\rightarrow$ Assumption: there is a constant number k of capture variables

- What is the performance of the naive algorithm?
  - In terms of preprocessing...
    - Combined complexity is... polynomial: convert P to an automaton A
    - Data complexity is... constant: nothing to do on T
  - In terms of delay...
    - Combined complexity is...

Complexity of enumeration algorithms

- Recall the inputs to our problem:
  - The text T
  - The regexp with captures P
  - $\rightarrow$ Assumption: there is a constant number k of capture variables

- What is the performance of the naive algorithm?
  - In terms of preprocessing...
    - Combined complexity is... polynomial: convert P to an automaton A
    - Data complexity is... constant: nothing to do on T
  - In terms of delay...
    - Combined complexity is... polynomial: check if A accepts T

Complexity of enumeration algorithms

- Recall the inputs to our problem:
  - The text T
  - The regexp with captures P

  $\rightarrow$ Assumption: there is a constant number k of capture variables

- What is the performance of the naive algorithm?
  - In terms of preprocessing...
    - Combined complexity is... polynomial: convert P to an automaton A
    - Data complexity is... constant: nothing to do on T
  - In terms of delay...
    - Combined complexity is... polynomial: check if A accepts T
    - Data complexity is...

Complexity of enumeration algorithms

- Recall the inputs to our problem:
  - The text T
  - The regexp with captures P

  $\rightarrow$ Assumption: there is a constant number k of capture variables

- What is the performance of the naive algorithm?
  - In terms of preprocessing...
    - Combined complexity is... polynomial: convert P to an automaton A
    - Data complexity is... constant: nothing to do on T
  - In terms of delay...
    - Combined complexity is... polynomial: check if A accepts T
    - Data complexity is... polynomial in T: time to find the next match

Complexity of enumeration algorithms

- Recall the inputs to our problem:
  - The text T
  - The regexp with captures P
  - → Assumption: there is a constant number k of capture variables

- What is the performance of the naive algorithm?
  - In terms of preprocessing...
    - Combined complexity is... polynomial: convert P to an automaton A
    - Data complexity is... constant: nothing to do on T
  - In terms of delay...
    - Combined complexity is... polynomial: check if A accepts T
    - Data complexity is... polynomial in T: time to find the next match

→ Can we do better?

Results for enumerating pattern matches

- Existing work has shown the best possible bounds:

Results for enumerating pattern matches

- Existing work has shown the best possible bounds:

Theorem [Florenzano et al., 2018]

We can find all matches of a regexp with captures P on text T with:

- *Preprocessing linear in T*
- *Delay constant in T*

Results for enumerating pattern matches

- Existing work has shown the best possible bounds:

Theorem [Florenzano et al., 2018]

We can find all matches of a regexp with captures P on text T with:

- *Preprocessing linear in T (data)*
- *Delay constant in T (data)*

→ Problem: They only measure data complexity!
The combined complexity is exponential with their approach!

Results for enumerating pattern matches

- Existing work has shown the best possible bounds:

Theorem [Florenzano et al., 2018]

We can find all matches of a regexp with captures P on text T with:

- *Preprocessing **linear** in T (data)*
- *Delay **constant** in T (data)*

→ **Problem:** They only measure **data complexity**!
The combined complexity is **exponential** with their approach!

- **Our contribution** is:

Theorem

We can find all matches of a regexp with captures P on text T with:

- *Preprocessing **linear** in T (data) and **polynomial** in T and P (combined)*
- *Delay **constant** in T (data) and **polynomial** in T and P (combined)*

Key proof idea

Compute a product DAG of the text T and of the pattern P

Key proof idea

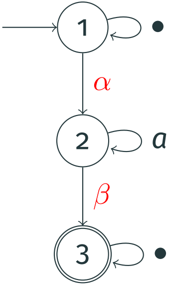
Compute a **product DAG** of the text T and of the pattern P

Example: Text $T := \boxed{\texttt{aaaba}}$ and $P := \bullet^* \alpha a^* \beta \bullet^*$,

Key proof idea

Compute a **product DAG** of the text T and of the pattern P

Example: Text $T := \boxed{\texttt{aaaba}}$ and $P := \bullet^* \alpha a^* \beta \bullet^*$,

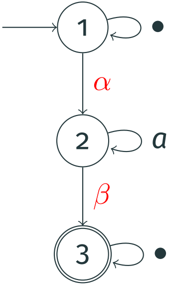


Key proof idea

Compute a product DAG of the text T and of the pattern P

Example: Text $T :=$ $\boxed{\texttt{aaaba}}$ and $P := \bullet^* \alpha a^* \beta \bullet^*$,

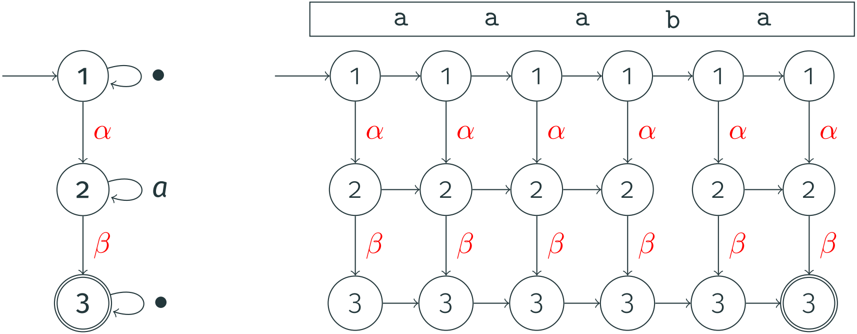
a	a	a	b	a



Key proof idea

Compute a product DAG of the text T and of the pattern P

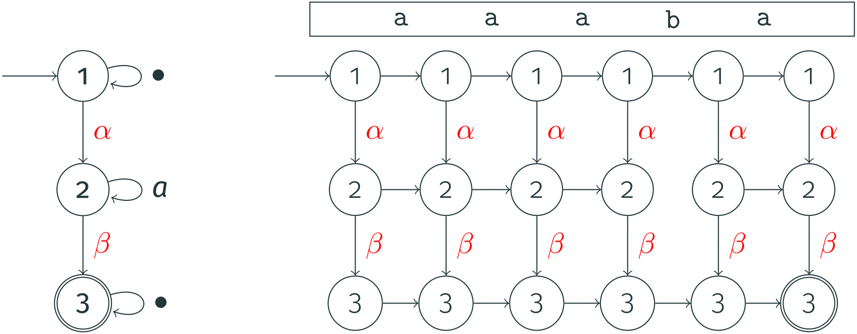
Example: Text $T :=$ `aaaba` and $P := \bullet^* \alpha a^* \beta \bullet^*$,



Key proof idea

Compute a **product DAG** of the text T and of the pattern P

Example: Text $T :=$ `aaaba` and $P := \bullet^* \alpha a^* \beta \bullet^*$,

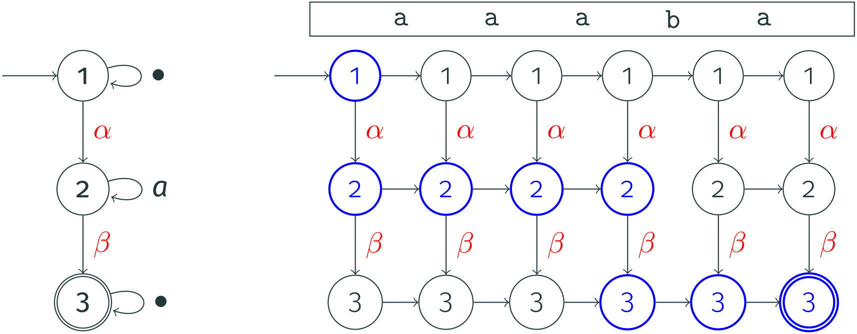


$\rightarrow$ Each **path** in the **product DAG** corresponds to a **match**

Key proof idea

Compute a **product DAG** of the text T and of the pattern P

Example: Text $T :=$ `aaaba` and $P := \bullet^* \alpha a^* \beta \bullet^*$, match $\langle \alpha : 0, \beta : 3 \rangle$

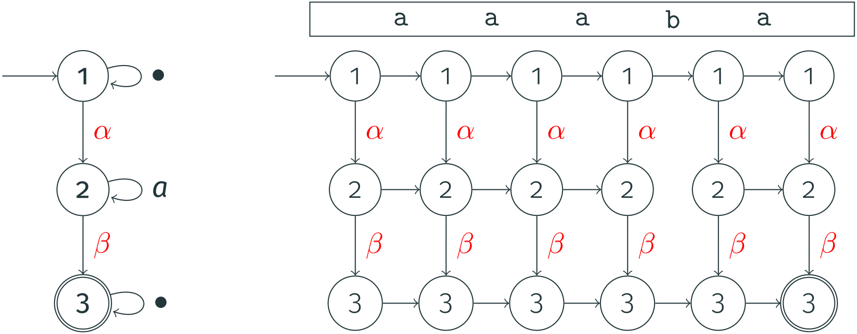


$\rightarrow$ Each **path** in the **product DAG** corresponds to a **match**

Key proof idea

Compute a **product DAG** of the text T and of the pattern P

Example: Text $T :=$ `aaaba` and $P := \bullet^* \alpha a^* \beta \bullet^*$,

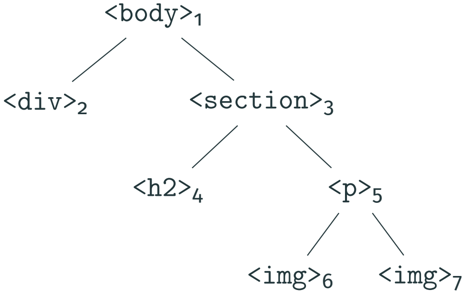


$\rightarrow$ Each **path** in the **product DAG** corresponds to a **match**

$\rightarrow$ **Challenge:** Enumerate paths but avoid **duplicate matches** and do not **waste time** to ensure constant delay

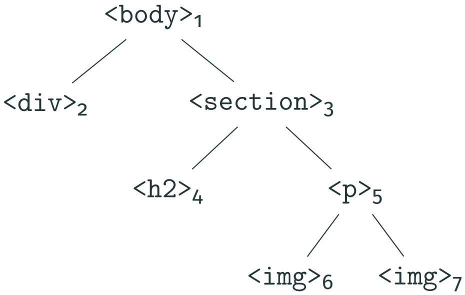
Extension: From Text to Trees

- The data T is no longer text but is now a tree:



Pattern matching on trees

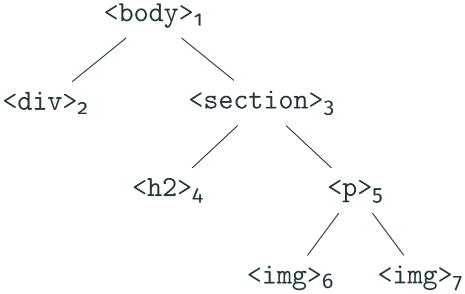
- The data T is no longer text but is now a tree:



- The pattern P asks about the structure of the tree:
  Is there an h2 header and an image in the same section?

Pattern matching on trees

- The data T is no longer text but is now a tree:

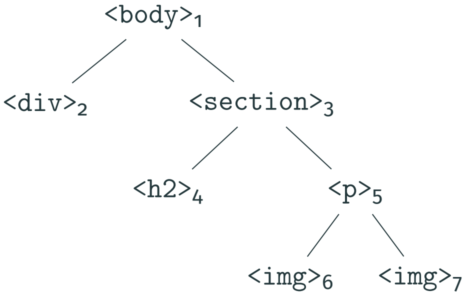


- The pattern P asks about the structure of the tree:
  Is there an h2 header and an image in the same section?

- Results:

Pattern matching on trees

- The **data** *T* is no longer **text** but is now a **tree**:

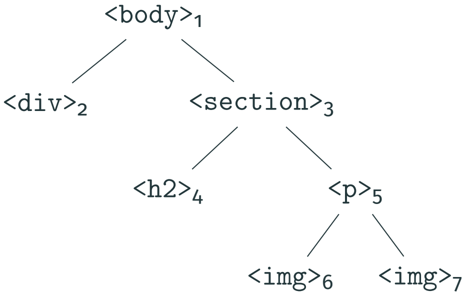


- The **pattern** *P* asks about the **structure** of the tree:
  *Is there α: an **h2** header and β: an **image** in the same section?*

- **Results:**

Pattern matching on trees

- The **data** T is no longer **text** but is now a **tree**:



- The **pattern** P asks about the **structure** of the tree:
  *Is there α: an **h2** header and β: an **image** in the same section?*

- **Results:** $\langle \alpha : 4, \beta : 6 \rangle, \langle \alpha : 4, \beta : 7 \rangle$

Definitions and Results on Trees

- Tree patterns can be written as a tree automaton with captures

Definitions and Results on Trees

- Tree patterns can be written as a **tree automaton** with captures
- Like for **text**, we can enumerate the matches of **tree automata**...

Theorem [Bagan, 2006]

We can find all matches on a tree T of a tree automaton A (with constantly many capture variables) with:

- *Preprocessing **linear** in T*
- *Delay **constant** in T*

Definitions and Results on Trees

- Tree patterns can be written as a **tree automaton** with captures
- Like for **text**, we can enumerate the matches of **tree automata**...

Theorem [Bagan, 2006]

We can find all matches on a tree T of a tree automaton A (with constantly many capture variables) with:

- *Preprocessing **linear** in T (data)*
- *Delay **constant** in T (data)*

- Again, this is only in **data complexity**!

Definitions and Results on Trees

- Tree patterns can be written as a **tree automaton** with captures
- Like for **text**, we can enumerate the matches of **tree automata**...

Theorem [Bagan, 2006]

We can find all matches on a tree T of a tree automaton A (with constantly many capture variables) with:

- *Preprocessing **linear** in T (data)*
- *Delay **constant** in T (data)*

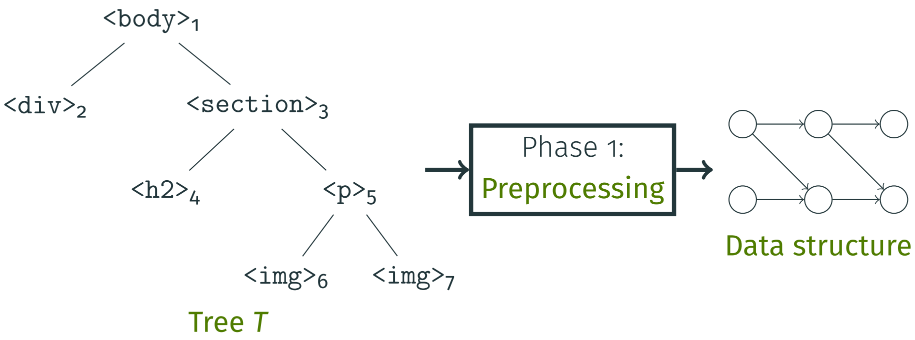
- Again, this is only in **data complexity**!
- We **conjecture** the following bounds for this task (ongoing work):

Conjecture

- *Preprocessing **linear** in T (data) and **polynomial** in A and T (combined)*
- *Delay **constant** in T (data) and **polynomial** in A and T (combined)*

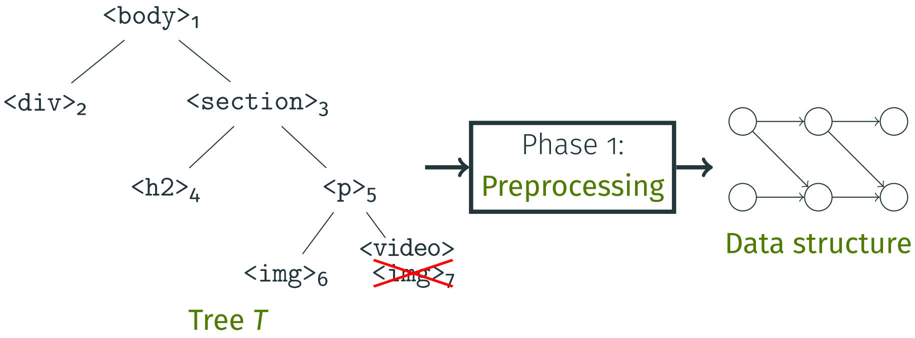
Extension: Handling Updates

Updates



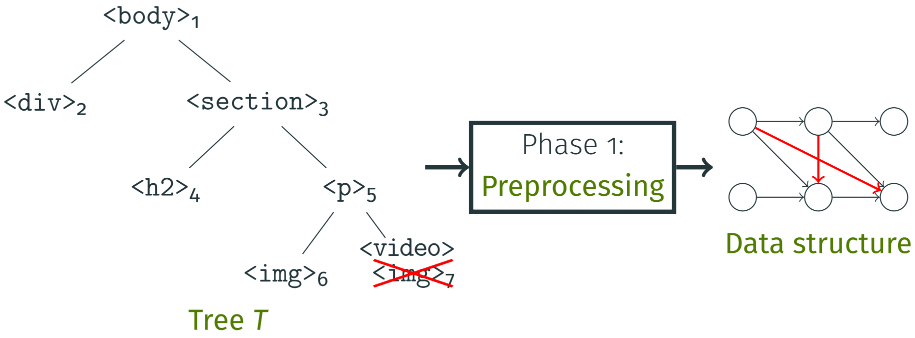
- The input data can be **modified** after the preprocessing

Updates



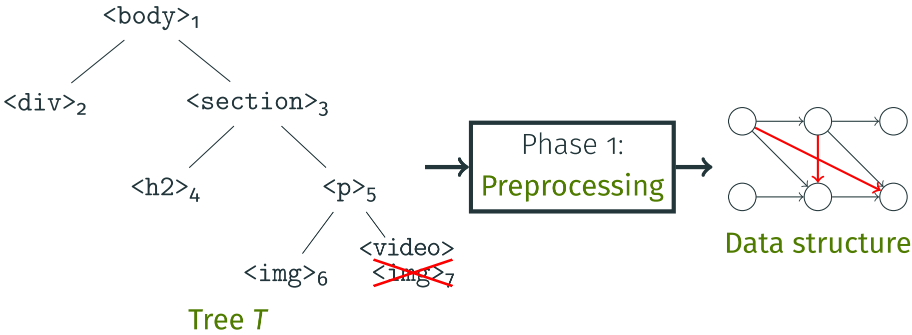
- The input data can be **modified** after the preprocessing

Updates



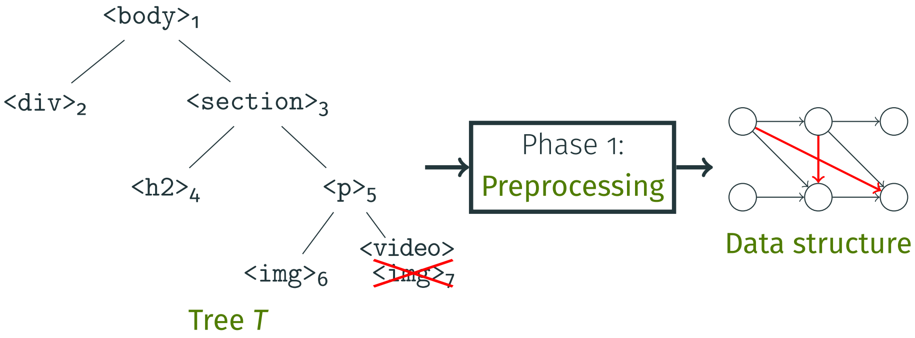
- The input data can be **modified** after the preprocessing

Updates



- The input data can be **modified** after the preprocessing
- If this happen, we must rerun the **preprocessing** from scratch

Updates



- The input data can be **modified** after the preprocessing
- If this happen, we must rerun the **preprocessing** from scratch

→ Can we **do better**?

Known results on dynamic trees

All these results are on data complexity in T (for a fixed pattern):

Work	Data	Preproc.	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$

Known results on dynamic trees

All these results are on data complexity in T (for a fixed pattern):

Work	Data	Preproc.	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$
[Losemann and Martens, 2014]	trees	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$

Known results on dynamic trees

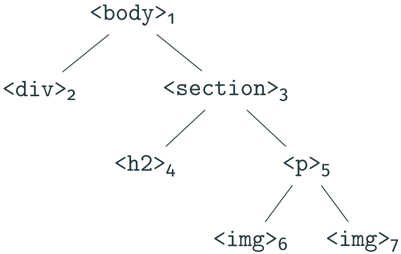
All these results are on data complexity in T (for a fixed pattern):

Work	Data	Preproc.	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$
[Losemann and Martens, 2014]	trees	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$
[Losemann and Martens, 2014]	text	$O(T)$	$O(\log T)$	$O(\log T)$

Known results on dynamic trees

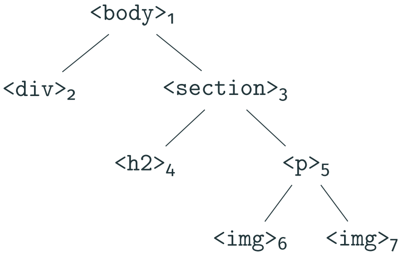
All these results are on data complexity in T (for a fixed pattern):

Work	Data	Preproc.	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$
[Losemann and Martens, 2014]	trees	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$
[Losemann and Martens, 2014]	text	$O(T)$	$O(\log T)$	$O(\log T)$
[Niewerth and Segoufin, 2018]	text	$O(T)$	$O(1)$	$O(\log T)$



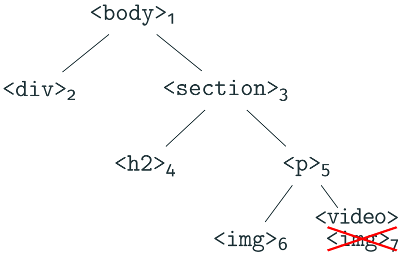
- Special kind of updates: relabelings that change the label of a node

Relabelings



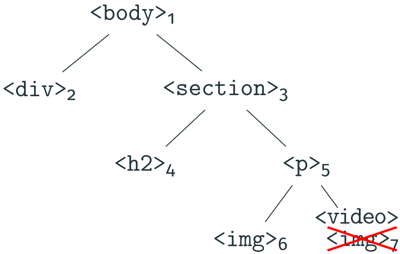
- Special kind of updates: relabelings that change the label of a node
- Example: relabel node 7 to `<video>`

Relabelings



- Special kind of updates: **relabelings** that change the label of a node
- **Example:** relabel node 7 to `<video>`

Relabelings



- Special kind of updates: **relabelings** that change the label of a node
- **Example:** relabel node `7` to `<video>`
- The tree's **structure** never changes

New results on dynamic trees

If we allow only relabeling updates, we can show:

Work	**Data**	**Preproc.**	**Delay**	**Updates**
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$
[Losemann and Martens, 2014]	trees	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$

New results on dynamic trees

If we allow only relabeling updates, we can show:

Work	Data	Preproc.	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$
[Losemann and Martens, 2014]	trees	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$
[Amarilli et al., 2018]	trees	$O(T)$	$O(1)$	$O(\log T)$

New results on dynamic trees

If we allow only relabeling updates, we can show:

Work	**Data**	**Preproc.**	**Delay**	**Updates**
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$
[Losemann and Martens, 2014]	trees	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$
[Amarilli et al., 2018]	trees	$O(T)$	$O(1)$	$O(\log T)$

New results on dynamic trees

If we allow only relabeling updates, we can show:

Work	Data	Preproc.	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$
[Losemann and Martens, 2014]	trees	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$
[Amarilli et al., 2018]	trees	$O(T)$	$O(1)$	$O(\log T)$

Remaining open questions:

→ Does this hold for more general updates (insert/delete, etc.)?

→ Can we also achieve tractable combined complexity?

Summary and Future Work

Summary and future work

Summary:

- Problem: given a text T and a pattern P, enumerate efficiently all matches of P on T

Summary and future work

Summary:

- Problem: given a text T and a pattern P, enumerate efficiently all matches of P on T
- Result: we can do this with tractable combined complexity and linear preprocessing and constant delay in data complexity

Summary and future work

Summary:

- Problem: given a text T and a pattern P, enumerate efficiently all matches of P on T
- Result: we can do this with tractable combined complexity and linear preprocessing and constant delay in data complexity

Ongoing and future work:

- Extending the results from text to trees

Summary and future work

Summary:

- **Problem:** given a text T and a pattern P, enumerate efficiently all matches of P on T
- **Result:** we can do this with **tractable combined complexity** and **linear** preprocessing and **constant** delay in data complexity

Ongoing and future work:

- Extending the results from text to **trees**
- Supporting **updates** on the input data

Summary and future work

Summary:

- Problem: given a text T and a pattern P, enumerate efficiently all matches of P on T
- Result: we can do this with tractable combined complexity and linear preprocessing and constant delay in data complexity

Ongoing and future work:

- Extending the results from text to trees
- Supporting updates on the input data
- Testing how well our methods perform in practice

Summary and future work

Summary:

- **Problem:** given a text T and a pattern P, enumerate efficiently all matches of P on T
- **Result:** we can do this with **tractable combined complexity** and **linear** preprocessing and **constant** delay in data complexity

Ongoing and future work:

- Extending the results from text to **trees**
- Supporting **updates** on the input data
- Testing how well our methods perform in **practice**

Thanks for your attention!

References i

Amarilli, A., Bourhis, P., and Mengel, S. (2018).
Enumeration on trees under relabelings.
In *ICDT*.

Bagan, G. (2006).
MSO queries on tree decomposable structures are computable with linear delay.
In *CSL*.

Florenzano, F., Riveros, C., Ugarte, M., Vansummeren, S., and Vrgoc, D. (2018).
Constant delay algorithms for regular document spanners.
In *PODS*.

References ii

Kazana, W. and Segoufin, L. (2013).
Enumeration of monadic second-order queries on trees.
TOCL, 14(4).

Losemann, K. and Martens, W. (2014).
MSO queries on trees: Enumerating answers under updates.
In *CSL-LICS*.

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