Enumerating Pattern Matches in Texts and Trees

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

October 3rd, 2019

[1]Télécom Paris

[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 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.
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 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.
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

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 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.
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:

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

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 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.
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:

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

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

Solution: Automata

- Convert the regular expression P to an automaton A

Solution: Automata

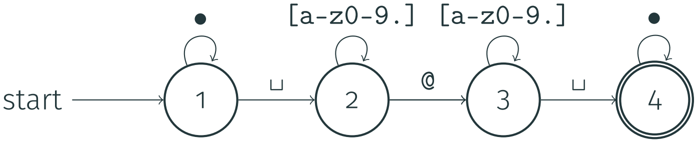
- Convert the regular expression P to an automaton A

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

Solution: Automata

- Convert the regular expression P to an automaton A

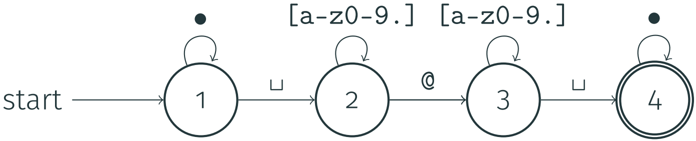
$$P := \text{␣ [a-z0-9.]* @ [a-z0-9.]* ␣}$$



Solution: Automata

- Convert the regular expression P to an automaton A

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

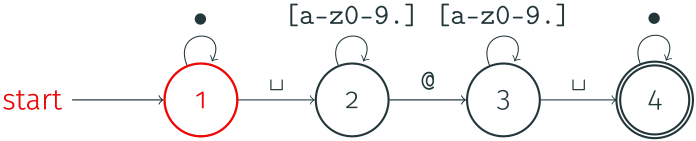


- Then, evaluate the automaton on the text T

Solution: Automata

- Convert the **regular expression** *P* to an **automaton** *A*

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



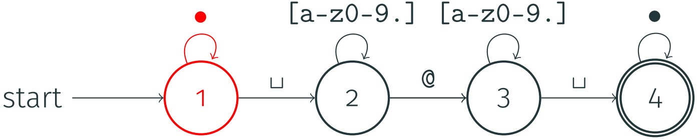
- Then, evaluate the automaton on the **text** *T*

E	m	a	i	l	␣	a	3	n	m	@	a	3	n	m	.	n	e	t	␣	A	f	f	i	l	i	a	t	i	o	n

Solution: Automata

- Convert the regular expression P to an automaton A

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



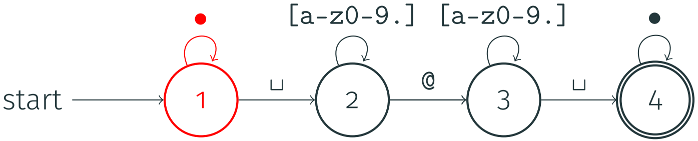
- Then, evaluate the automaton on the text T

E	m	a	i	l	␣	a	3	n	m	@	a	3	n	m	.	n	e	t	␣	A	f	f	i	l	i	a	t	i	o	n

Solution: Automata

- Convert the regular expression *P* to an automaton *A*

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



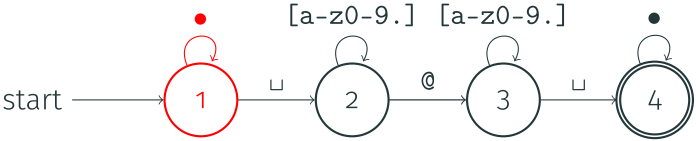
- Then, evaluate the automaton on the text *T*

`E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n`

Solution: Automata

- Convert the regular expression P to an automaton A

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



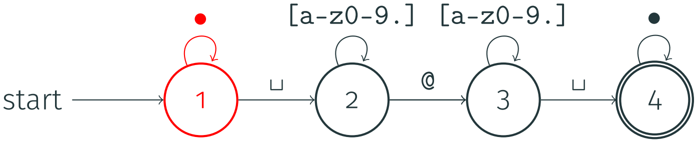
- Then, evaluate the automaton on the text T

`E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n`

Solution: Automata

- Convert the regular expression P to an automaton A

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



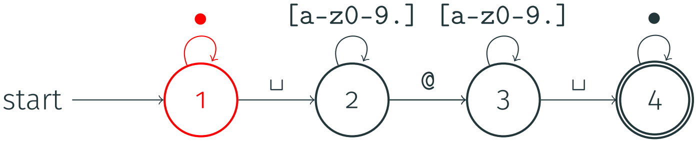
- Then, evaluate the automaton on the text T

E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n

Solution: Automata

- Convert the regular expression P to an automaton A

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



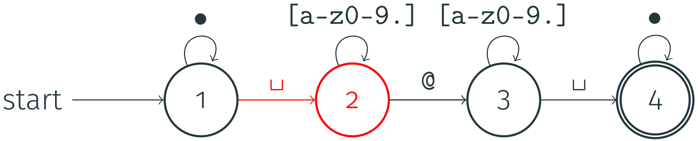
- Then, evaluate the automaton on the text T

`E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n`

Solution: Automata

- Convert the regular expression P to an automaton A

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



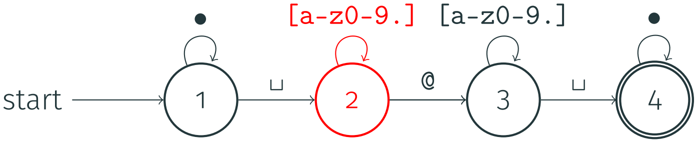
- Then, evaluate the automaton on the text T

E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n

Solution: Automata

- Convert the regular expression P to an automaton A

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



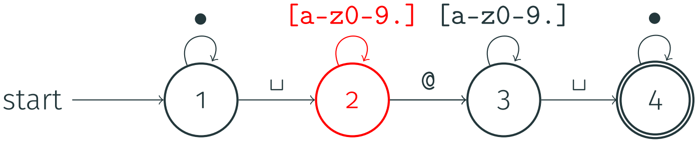
- Then, evaluate the automaton on the text T

E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n

Solution: Automata

- Convert the regular expression P to an automaton A

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



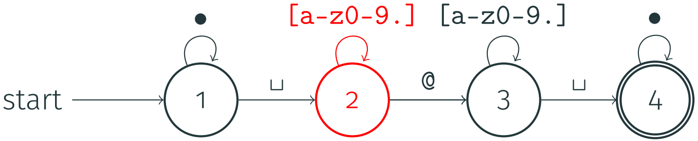
- Then, evaluate the automaton on the text T

`Email␣a3nm@a3nm.net␣Affiliation`

Solution: Automata

- Convert the regular expression *P* to an automaton *A*

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



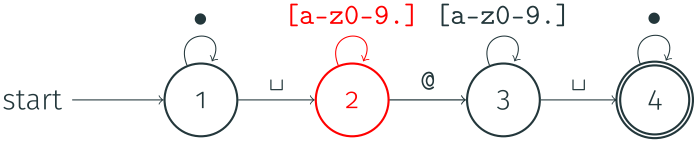
- Then, evaluate the automaton on the text *T*

`Email␣a3nm@a3nm.net␣Affiliation`

Solution: Automata

- Convert the **regular expression** *P* to an **automaton** *A*

$P :=$ `␣ [a-z0-9.]* @ [a-z0-9.]* ␣`



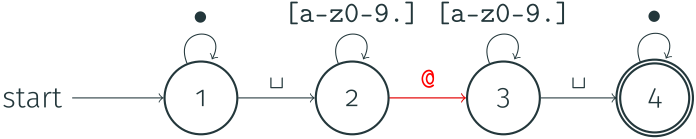
- Then, evaluate the automaton on the **text** *T*

`Email␣a3nm@a3nm.net␣Affiliation`

Solution: Automata

- Convert the regular expression P to an automaton A

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



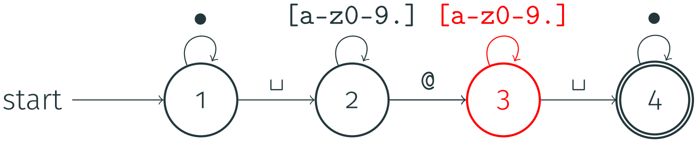
- Then, evaluate the automaton on the text T

`Email␣a3nm@a3nm.net␣Affiliation`

Solution: Automata

- Convert the regular expression P to an automaton A

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



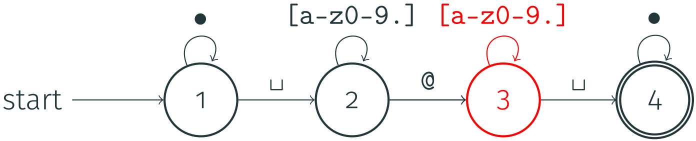
- Then, evaluate the automaton on the text T

E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n

Solution: Automata

- Convert the regular expression P to an automaton A

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



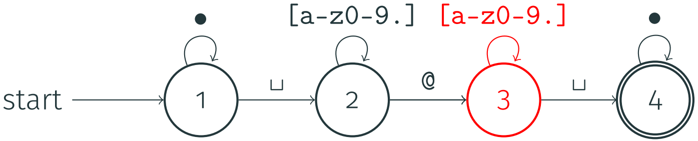
- Then, evaluate the automaton on the text T

`E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n`

Solution: Automata

- Convert the regular expression P to an automaton A

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



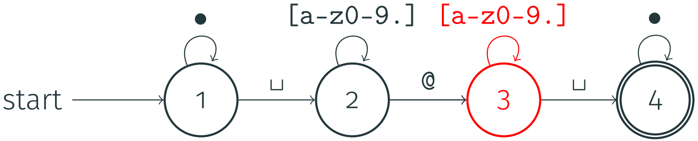
- Then, evaluate the automaton on the text T

`Email␣a3nm@a3nm.net␣Affiliation`

Solution: Automata

- Convert the regular expression P to an automaton A

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



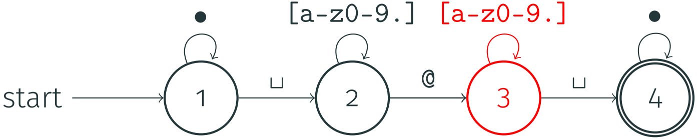
- Then, evaluate the automaton on the text T

E	m	a	i	l	␣	a	3	n	m	@	a	3	n	m	.	n	e	t	␣	A	f	f	i	l	i	a	t	i	o	n

Solution: Automata

- Convert the regular expression P to an automaton A

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



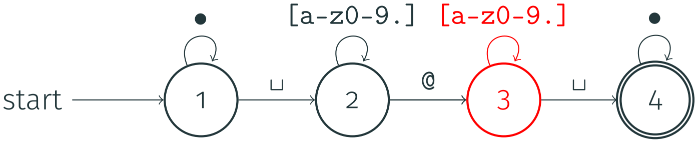
- Then, evaluate the automaton on the text T

E	m	a	i	l	␣	a	3	n	m	@	a	3	n	m	.	n	e	t	␣	A	f	f	i	l	i	a	t	i	o	n

Solution: Automata

- Convert the regular expression P to an automaton A

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



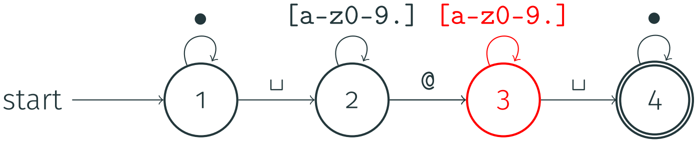
- Then, evaluate the automaton on the text T

E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n

Solution: Automata

- Convert the regular expression P to an automaton A

$P :=$ `␣ [a-z0-9.]* @ [a-z0-9.]* ␣`



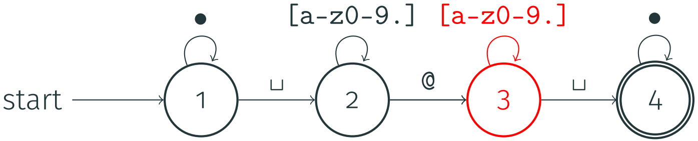
- Then, evaluate the automaton on the text T

`Email␣a3nm@a3nm.net␣Affiliation`

Solution: Automata

- Convert the **regular expression** P to an **automaton** A

$P :=$ ␣ [a-z0-9.]* @ [a-z0-9.]* ␣



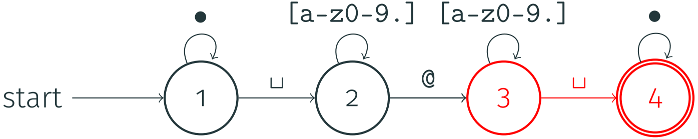
- Then, evaluate the automaton on the **text** T

E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n

Solution: Automata

- Convert the regular expression P to an automaton A

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



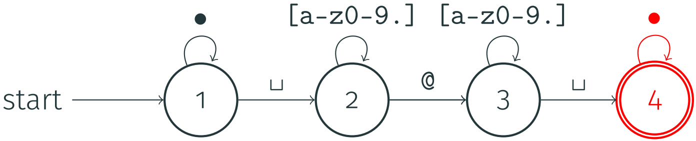
- Then, evaluate the automaton on the text T

`Email␣a3nm@a3nm.net␣Affiliation`

Solution: Automata

- Convert the regular expression *P* to an automaton *A*

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



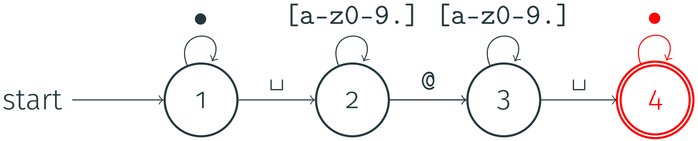
- Then, evaluate the automaton on the text *T*

E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n

Solution: Automata

- Convert the regular expression P to an automaton A

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



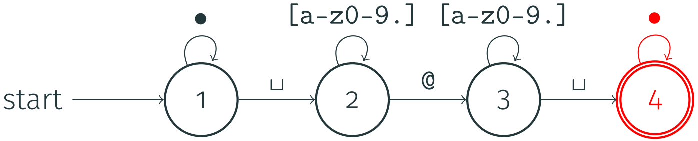
- Then, evaluate the automaton on the text T

`E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n`

Solution: Automata

- Convert the regular expression P to an automaton A

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



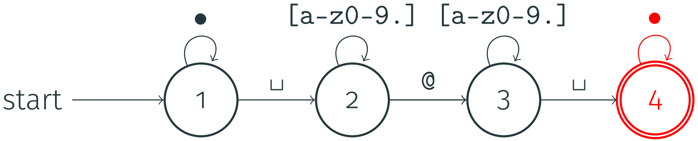
- Then, evaluate the automaton on the text T

`Email␣a3nm@a3nm.net␣Affiliation`

Solution: Automata

- Convert the regular expression P to an automaton A

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



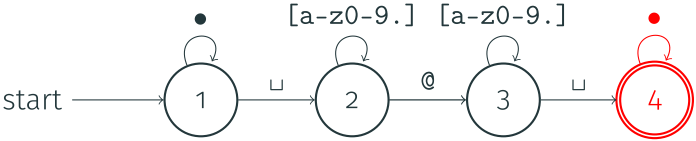
- Then, evaluate the automaton on the text T

E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n

Solution: Automata

- Convert the regular expression P to an automaton A

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



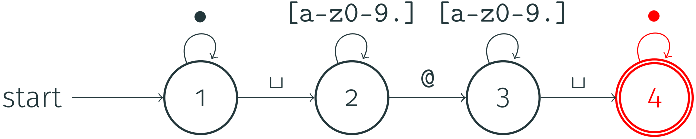
- Then, evaluate the automaton on the text T

E	m	a	i	l	␣	a	3	n	m	@	a	3	n	m	.	n	e	t	␣	A	f	f	i	l	i	a	t	i	o	n	

Solution: Automata

- Convert the regular expression P to an automaton A

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



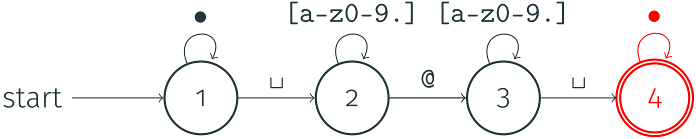
- Then, evaluate the automaton on the text T

`E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n`

Solution: Automata

- Convert the regular expression P to an automaton A

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



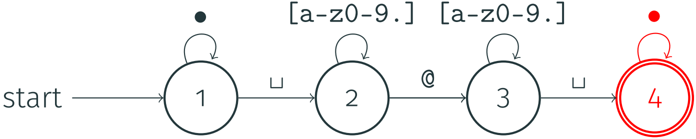
- Then, evaluate the automaton on the text T

`Email␣a3nm@a3nm.net␣Affiliation`

Solution: Automata

- Convert the regular expression P to an automaton A

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



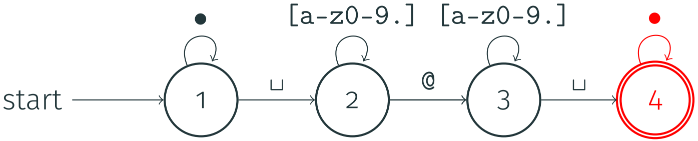
- Then, evaluate the automaton on the text T

`Email␣a3nm@a3nm.net␣Affiliation`

Solution: Automata

- Convert the regular expression P to an automaton A

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



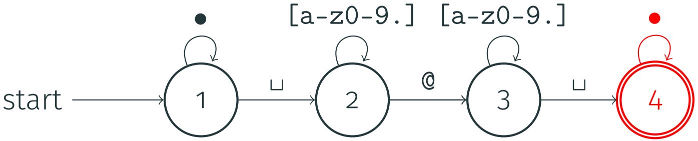
- Then, evaluate the automaton on the text T

E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n

Solution: Automata

- Convert the regular expression P to an automaton A

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



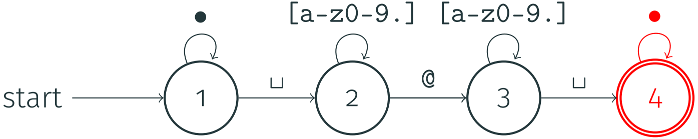
- Then, evaluate the automaton on the text T

E	m	a	i	l	␣	a	3	n	m	@	a	3	n	m	.	n	e	t	␣	A	f	f	i	l	i	a	t	i	o	n

Solution: Automata

- Convert the regular expression P to an automaton A

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



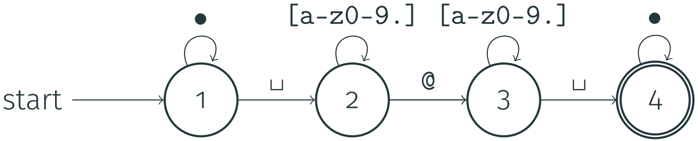
- Then, evaluate the automaton on the text T

`Email␣a3nm@a3nm.net␣Affiliation`

Solution: Automata

- Convert the regular expression P to an automaton A

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



- Then, evaluate the automaton on the text T

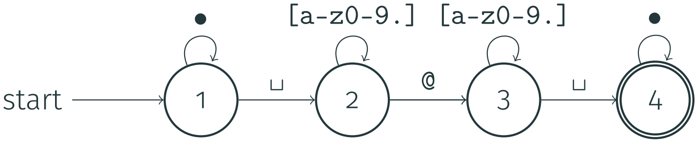
E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n

- The complexity is $O(|A| \times |T|)$, i.e., linear in T and polynomial in P

Solution: Automata

- Convert the regular expression P to an automaton A

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



- Then, evaluate the automaton on the text T

`E m a i l ␣ a 3 n m @ a 3 n m . n e t ␣ A f f i l i a t i o n`

- The complexity is $O(|A| \times |T|)$, i.e., linear in T and polynomial in P
  - $\rightarrow$ This is very efficient in T and reasonably efficient in P

Actual Problem: Extracting all Patterns

- This only tests if the pattern occurs in the text!
  - → ``YES''

Actual Problem: Extracting all Patterns

- This only tests if the pattern occurs in the text!
  → ‘‘YES’’

- Goal: find all substrings in the text *T* which match the pattern *P*

Actual Problem: Extracting all Patterns

- This only tests if the pattern occurs in the text!

  $\rightarrow$ "YES"

- Goal: find all substrings in the text T which match the pattern P

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
E	m	a	i	l	␣	a	3	n	m	@	a	3	n	m	.	n	e	t	␣	A	f	f	i	l	i	a	t	i	o	n

Actual Problem: Extracting all Patterns

- This only tests **if** the pattern **occurs in** the text!
  $\rightarrow$ "YES"

- Goal: find all **substrings** in the text T which match the pattern P

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
E	m	a	i	l	␣	a	3	n	m	@	a	3	n	m	.	n	e	t	␣	A	f	f	i	l	i	a	t	i	o	n

$\rightarrow$ **One match:** $[5, 20\rangle$

Actual Problem: Extracting all Patterns

- This only tests **if** the pattern **occurs in** the text!

  $\rightarrow$ "YES"

- Goal: find all **substrings** in the text T which match the pattern P

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
E	m	a	i	l	␣	a	3	n	m	@	a	3	n	m	.	n	e	t	␣	A	f	f	i	l	i	a	t	i	o	n

$\rightarrow$ **One match:** $[5, 20\rangle$

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 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 ...
```

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 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 pattern *P* given as a regular expression

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

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 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 pattern P given as a regular expression

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

  - Output: the list of substrings of T that match P:

$$[186, 200\rangle, \quad [483, 500\rangle, \quad \ldots$$

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 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 pattern P given as a regular expression

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

  - Output: the list of substrings of T that match P:

$$[186, 200\rangle, \quad [483, 500\rangle, \quad \ldots$$

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

Measuring the Complexity

- Naive algorithm: Run the automaton A on each substring of T

`l o l`

Measuring the Complexity

- Naive algorithm: Run the automaton A on each substring of T

[⟩ l o l

Measuring the Complexity

- Naive algorithm: Run the automaton *A* on each substring of *T*

[l ⟩ o   l

Measuring the Complexity

- Naive algorithm: Run the automaton A on each substring of T

[l o ⟩ l

Measuring the Complexity

- Naive algorithm: Run the automaton A on each substring of T

$[$ l o l $\rangle$

Measuring the Complexity

- Naive algorithm: Run the automaton A on each substring of T

`l` $[\rangle$ `o   l`

Measuring the Complexity

- Naive algorithm: Run the automaton A on each substring of T

l [o ⟩ l

Measuring the Complexity

- Naive algorithm: Run the automaton A on each substring of T

l [o l ⟩

Measuring the Complexity

- Naive algorithm: Run the automaton A on each substring of T

`l o ` $[\rangle$ ` l`

Measuring the Complexity

- Naive algorithm: Run the automaton A on each substring of T

l o [l ⟩

Measuring the Complexity

- Naive algorithm: Run the automaton A on each substring of T

l o l [⟩

Measuring the Complexity

- Naive algorithm: Run the automaton A on each substring of T

l	o	l

→ Complexity is $O(|T|^2 \times |A| \times |T|)$

Measuring the Complexity

- Naive algorithm: Run the automaton A on each substring of T

l	o	l

→ Complexity is $O(|T|^2 \times |A| \times |T|)$

→ Can be optimized to $O(|T|^2 \times |A|)$

Measuring the Complexity

- **Naive algorithm:** Run the automaton A on **each substring** of T

l	o	l	

  - → **Complexity** is $O(|T|^2 \times |A| \times |T|)$
  - → Can be **optimized** to $O(|T|^2 \times |A|)$

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

Measuring the Complexity

- **Naive algorithm:** Run the automaton A on **each substring** of T

l o l

  → **Complexity** is $O(|T|^2 \times |A| \times |T|)$
  → Can be **optimized** to $O(|T|^2 \times |A|)$

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

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Measuring the Complexity

- **Naive algorithm:** Run the automaton A on **each substring** of T

```
    l   o   l
```

  → **Complexity** is $O(|T|^2 \times |A| \times |T|)$

  → Can be **optimized** to $O(|T|^2 \times |A|)$

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

    ```
    aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
    ```

  - Consider the **pattern** $P :=$ `a`*

Measuring the Complexity

- **Naive algorithm:** Run the automaton A on **each substring** of T

```
    l   o   l
```

  $\rightarrow$ **Complexity** is $O(|T|^2 \times |A| \times |T|)$

  $\rightarrow$ Can be **optimized** to $O(|T|^2 \times |A|)$

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

    ```
    aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
    ```

  - Consider the **pattern** $P :=$ `a`*
  - The **number of matches** is $\Omega(|T|^2)$

Measuring the Complexity

- **Naive algorithm:** Run the automaton A on **each substring** of T

```
    l   o   l
```

$\rightarrow$ **Complexity** is $O(|T|^2 \times |A| \times |T|)$

$\rightarrow$ Can be **optimized** to $O(|T|^2 \times |A|)$

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

    `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`

  - Consider the **pattern** $P :=$ `a`*
  - The **number of matches** is $\Omega(|T|^2)$

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

Enumeration Algorithms

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

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

Enumeration Algorithms

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

Enumeration Algorithms

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

...

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

Enumeration Algorithms

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

␣ [a-z0-9.]*@
[a-z0-9.]* ␣

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

```
␣ [a-z0-9.]*@
[a-z0-9.]* ␣
```

Pattern P

Phase 1:
Preprocessing

Formalizing Enumeration Algorithms

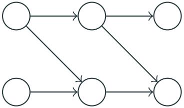
```
Antoine Amarilli Description Name Antoine
Amarilli.  Handle:  a3nm.  Identity Born
1990-02-07.  French national.  Appearance as
of 2017.  Auth OpenPGP. OpenId.  Bitcoin.
Contact Email and XMPP a3nm@a3nm.net
Affiliation Associate professor ...
```

Text T

␣ [a-z0-9.]*@[a-z0-9.]* ␣

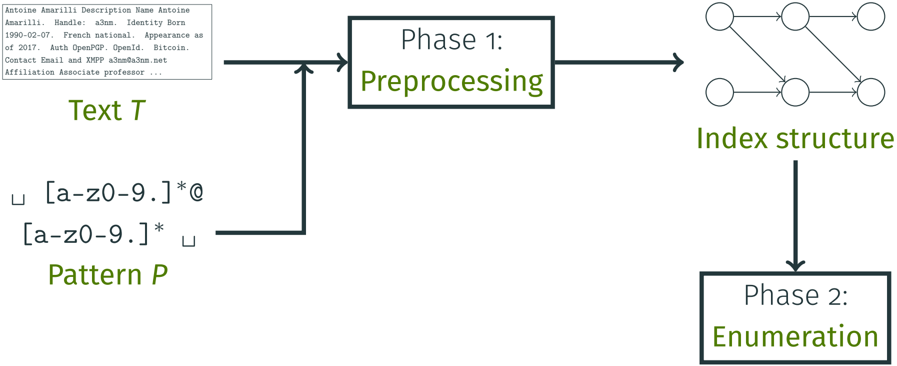
Pattern P

Phase 1: Preprocessing

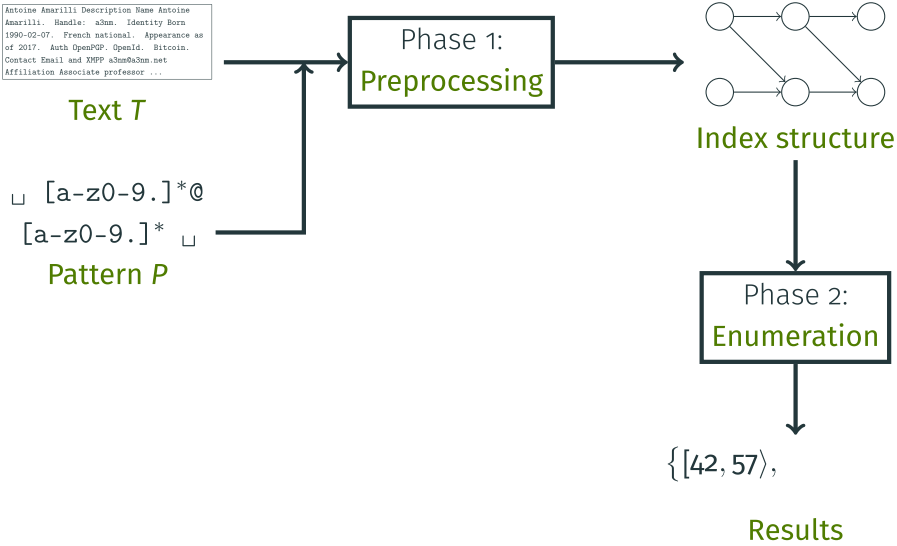


Index structure

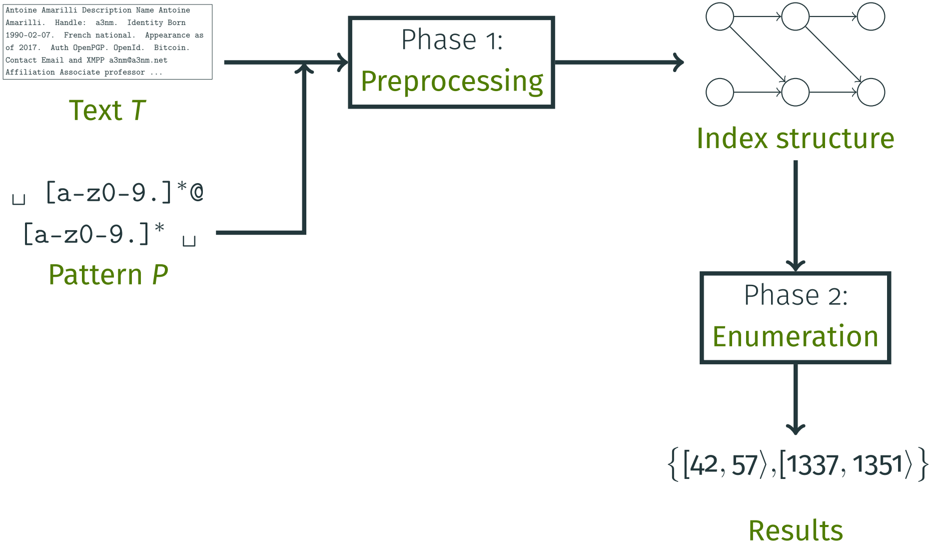
Formalizing Enumeration Algorithms



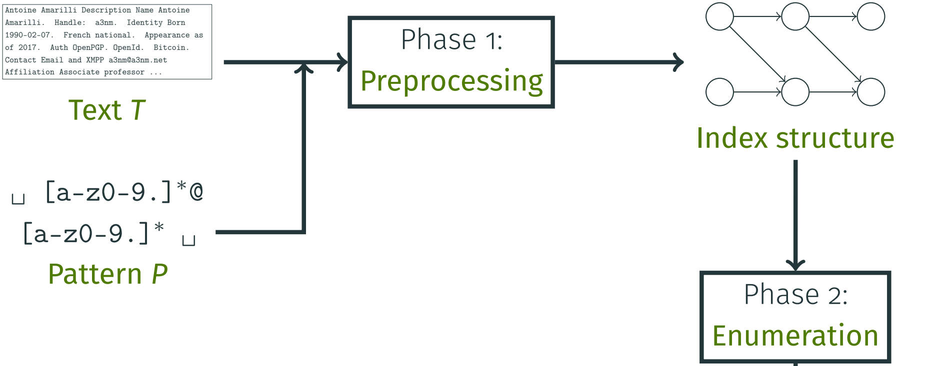
Formalizing Enumeration Algorithms

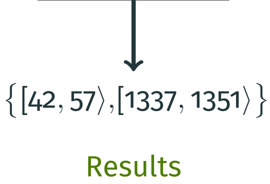


Formalizing Enumeration Algorithms



Formalizing Enumeration Algorithms





Two ways to measure performance:

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

... as a function of the text and pattern

Complexity of Enumeration Algorithms

- Recall the inputs to our problem:
  - 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 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.
More Résumé Location Other sites Blogging:  a3nm.net/blog Git:  a3nm.net/git ...
```

Complexity of Enumeration Algorithms

- Recall the **inputs** to our problem:
  - 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 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.
More Résumé Location Other sites Blogging:  a3nm.net/blog Git:  a3nm.net/git ...
```

  - A **pattern** P given as a regular expression

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

Complexity of Enumeration Algorithms

- Recall the **inputs** to our problem:
  - 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 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.
More Résumé Location Other sites Blogging:  a3nm.net/blog Git:  a3nm.net/git ...
```

  - A **pattern** P given as a regular expression

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

- What is the **delay** of the **naive algorithm**?

Complexity of Enumeration Algorithms

- Recall the **inputs** to our problem:
  - 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 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.
More Résumé Location Other sites Blogging:  a3nm.net/blog Git:  a3nm.net/git ...
```

  - A **pattern** P given as a regular expression

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

- What is the **delay** of the **naive algorithm**?

  → it is the **maximal time** to find the next **matching substring**

Complexity of Enumeration Algorithms

- Recall the **inputs** to our problem:
  - 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 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.
More Résumé Location Other sites Blogging:  a3nm.net/blog Git:  a3nm.net/git ...
```

  - A **pattern** P given as a regular expression

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

- What is the **delay** of the **naive algorithm**?

→ it is the **maximal time** to find the next **matching substring**
→ i.e. $O(|T|^2 \times |A|)$, e.g., if only the **beginning** and **end** match

Complexity of Enumeration Algorithms

- Recall the **inputs** to our problem:
  - 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 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.
More Résumé Location Other sites Blogging:  a3nm.net/blog Git:  a3nm.net/git ...
```

  - A **pattern** P given as a regular expression

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

- What is the **delay** of the **naive algorithm**?

  → it is the **maximal time** to find the next **matching substring**
  → i.e. $O(|T|^2 \times |A|)$, e.g., if only the **beginning** and **end** 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 enumerate all matches of a pattern P on a text T with:

- *Preprocessing linear in T*
- *Delay constant (independent from T)*

Results for Enumerating Pattern Matches

- Existing work has shown the best possible bounds in T:

Theorem [Florenzano et al., 2018]

We can enumerate all matches of a pattern P on a text T with:

- *Preprocessing linear in T and exponential in P*
- *Delay constant (independent from T) and exponential in P*

→ Problem: They only measure the complexity as a function of T!

Results for Enumerating Pattern Matches

- Existing work has shown the best possible bounds **in T**:

Theorem [Florenzano et al., 2018]

We can enumerate all matches of a pattern P on a text T with:

- *Preprocessing linear in T and exponential in P*
- *Delay constant (independent from T) and exponential in P*

→ **Problem:** They only measure the complexity **as a function of T**!

- **Our contribution** is:

Results for Enumerating Pattern Matches

- Existing work has shown the best possible bounds **in T**:

Theorem [Florenzano et al., 2018]

We can enumerate all matches of a pattern P on a text T with:

- *Preprocessing linear in T and exponential in P*
- *Delay constant (independent from T) and exponential in P*

→ **Problem:** They only measure the complexity **as a function of T**!

- **Our contribution** is:

Theorem

We can enumerate all matches of a pattern P on a text T with:

- *Preprocessing in $O(|T| \times Poly(P))$*
- *Delay polynomial in P and independent from T*

Automaton Formalism

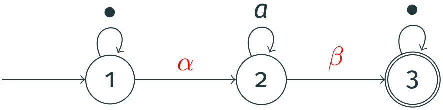
- We use automata that read letters and capture variables

Automaton Formalism

- We use automata that read letters and **capture variables**
  - → Example: $P := \bullet^* \ \alpha \ a^* \ \beta \ \bullet^*$

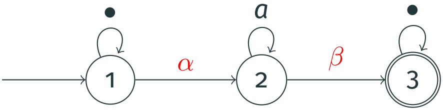
Automaton Formalism

- We use automata that read letters and capture variables
  - → Example: $P := \bullet^* \ \alpha \ a^* \ \beta \ \bullet^*$



Automaton Formalism

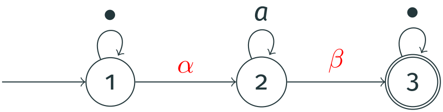
- We use automata that read letters and capture variables
  - → Example: $P := \bullet^* \ \alpha \ a^* \ \beta \ \bullet^*$



- Semantics of the automaton A:
  - Reads letters from the text
  - Guesses variables at positions in the text

Automaton Formalism

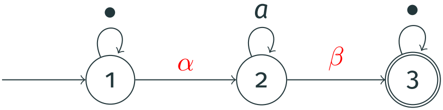
- We use automata that read letters and capture variables
  - → Example: $P := \bullet^* \ \alpha \ a^* \ \beta \ \bullet^*$



- Semantics of the automaton A:
  - **Reads** letters from the text
  - **Guesses** variables at positions in the text
  - → **Output:** tuples $\langle \alpha : i, \beta : j \rangle$ such that A has an accepting run reading α at position i and β at j

Automaton Formalism

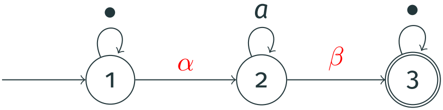
- We use automata that read letters and capture variables
  - $\rightarrow$ Example: $P := \bullet^* \;\; \alpha \;\; a^* \;\; \beta \;\; \bullet^*$



- Semantics of the automaton A:
  - Reads letters from the text
  - Guesses variables at positions in the text
  - $\rightarrow$ Output: tuples $\langle \alpha : i, \beta : j \rangle$ such that A has an accepting run reading α at position i and β at j
- Assumption: There is no run for which A reads the same capture variable twice at the same position

Automaton Formalism

- We use automata that read letters and capture variables
  - $\rightarrow$ Example: $P := \bullet^* \ \alpha \ a^* \ \beta \ \bullet^*$



- Semantics of the automaton A:
  - Reads letters from the text
  - Guesses variables at positions in the text
  - $\rightarrow$ Output: tuples $\langle \alpha : i, \beta : j \rangle$ such that A has an accepting run reading α at position i and β at j
- Assumption: There is no run for which A reads the same capture variable twice at the same position
- Challenge: Because of nondeterminism we can have many different runs of A producing the same tuple!

Proof Idea: Product DAG

Compute a product DAG of the text T and of the automaton A

Proof Idea: Product DAG

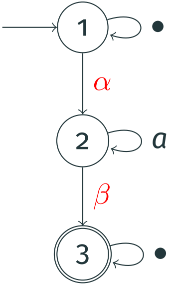
Compute a product DAG of the text T and of the automaton A

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

Proof Idea: Product DAG

Compute a product DAG of the text T and of the automaton A

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

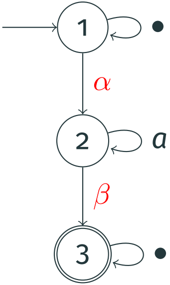


Proof Idea: Product DAG

Compute a product DAG of the text T and of the automaton A

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

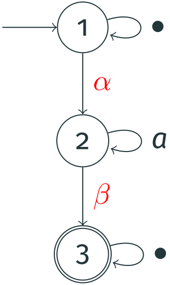
a	a	a	b	a

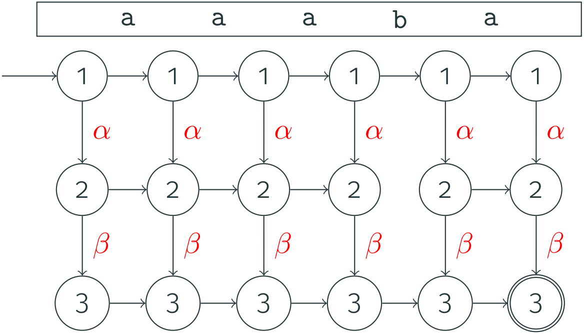


Proof Idea: Product DAG

Compute a product DAG of the text T and of the automaton A

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

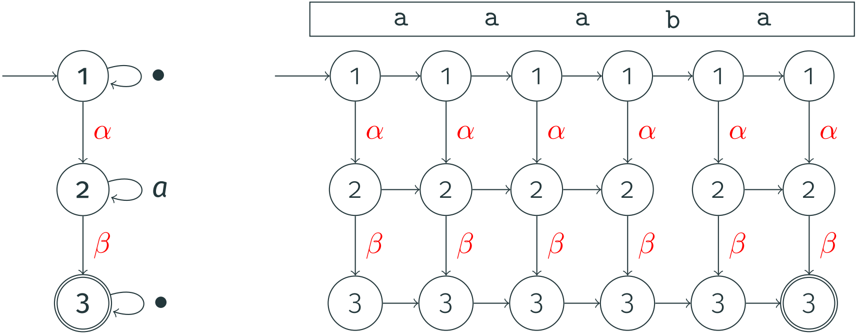




Proof Idea: Product DAG

Compute a **product DAG** of the text T and of the automaton A

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

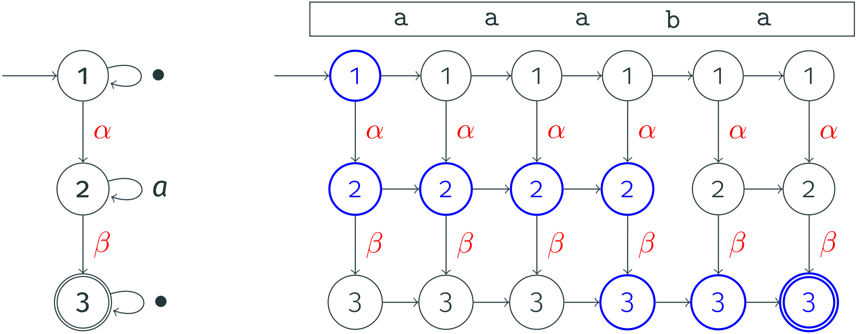


$\rightarrow$ Each **path** in the **product DAG** corresponds to a **match**

Proof Idea: Product DAG

Compute a **product DAG** of the text T and of the automaton A

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

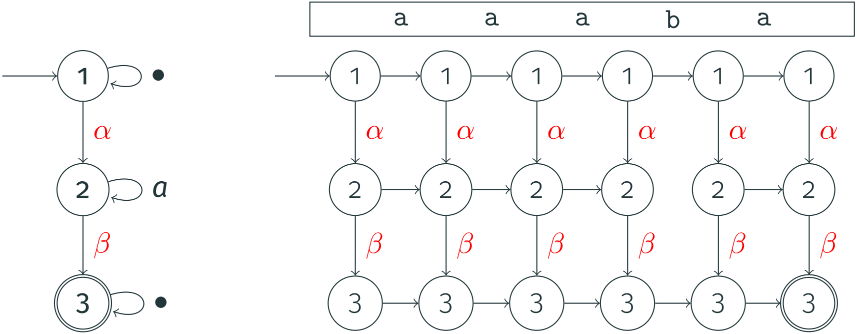


→ Each **path** in the **product DAG** corresponds to a **match**

Proof Idea: Product DAG

Compute a **product DAG** of the text T and of the automaton A

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



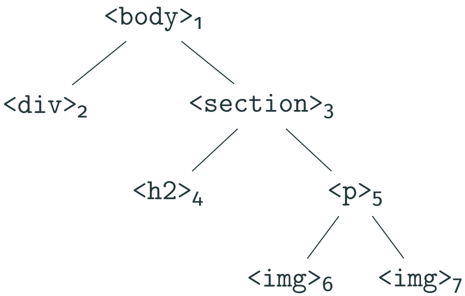
→ Each **path** in the **product DAG** corresponds to a **match**

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

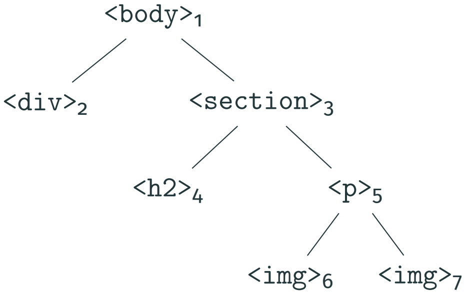
Extension: From Text to Trees

Pattern Matching on Trees

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



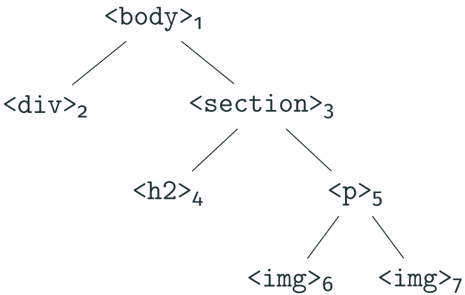
- The **data** T is no longer **text** but is now a **tree**:



- 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**:

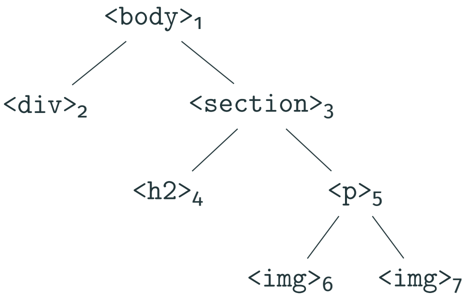


- The **pattern** P asks about the **structure** of the tree:
  *Is there an **h2** header and an **image** in the same section?*

- **Results:**

Pattern Matching on Trees

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

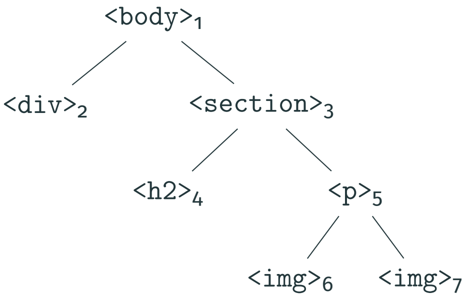


- The **pattern** *P* asks about the **structure** of the tree:
  *Is there α: an **h2** header and β: an **image** in the same section?*

- **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 P can be written as a kind of tree automaton…

Definitions and Results on Trees

- Tree patterns P can be written as a kind of tree automaton...
- Existing work has studied this problem and shown:

Definitions and Results on Trees

- Tree patterns P can be written as a kind of tree automaton...

- Existing work has studied this problem and shown:

Theorem [Bagan, 2006]

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

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

Definitions and Results on Trees

- Tree patterns P can be written as a kind of tree automaton...
- Existing work has studied this problem and shown:

Theorem [Bagan, 2006]

We can find all matches on a tree T *of a tree pattern* P *(with constantly many capture variables) with:*

- *Preprocessing linear in* T *and exponential in* P
- *Delay constant in* T *and exponential in* P

- Again, this only measures the complexity in T! We show:

Definitions and Results on Trees

- Tree patterns P can be written as a kind of tree automaton...
- Existing work has studied this problem and shown:

Theorem [Bagan, 2006]

We can find all matches on a tree T *of a tree pattern* P *(with constantly many capture variables) with:*

- *Preprocessing linear in* T *and exponential in* P
- *Delay constant in* T *and exponential in* P

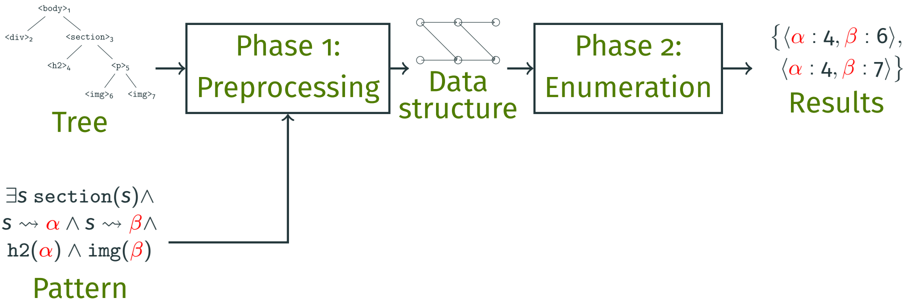
- Again, this only measures the complexity in T! We show:

Theorem [Amarilli et al., 2019]

- *Preprocessing in* $O(|T| \times Poly(P))$
- *Delay polynomial in* P *and independent from* T

Proof Idea for Trees: Structure

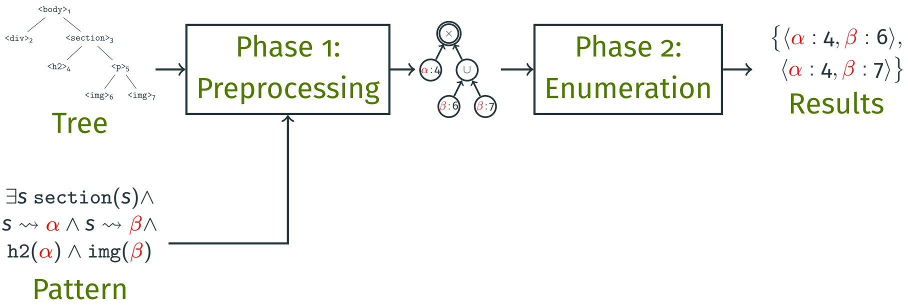
Similar structure to the previous proof, but with a circuit:



Proof Idea for Trees: Structure

Similar structure to the previous proof, but with a circuit:

- **Preprocessing:** Compute a circuit representation of the answers
- **Enumeration:** Apply a generic algorithm on the circuit



Proof Idea for Trees: Set Circuits

A set circuit represents a set of answers to a pattern $P(\alpha, \beta)$

Proof Idea for Trees: Set Circuits

A set circuit represents a set of answers to a pattern $P(\alpha, \beta)$

- Singleton $\alpha{:}6 \rightarrow$ *"the variable α is mapped to node 6"*

Proof Idea for Trees: Set Circuits

A set circuit represents a set of answers to a pattern $P(\alpha, \beta)$

- Singleton $\alpha : 6 \rightarrow$ *"the variable α is mapped to node 6"*
- Tuple $\langle \alpha : 4, \beta : 6 \rangle$: tuple of singletons

Proof Idea for Trees: Set Circuits

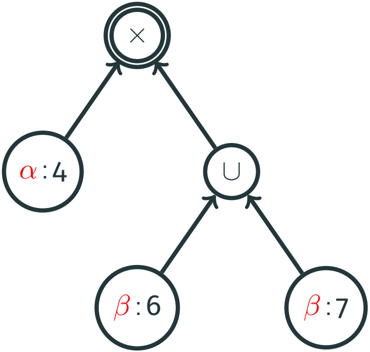
A **set circuit** represents a **set of answers** to a pattern $P(\alpha, \beta)$

- **Singleton** $\alpha : 6 \rightarrow$ *"the variable α is mapped to node 6"*
- **Tuple** $\langle \alpha : 4, \beta : 6 \rangle$: tuple of singletons
- The circuit captures a **set** of tuples, e.g., $\{ \langle \alpha : 4, \beta : 6 \rangle, \langle \alpha : 4, \beta : 7 \rangle \}$

Proof Idea for Trees: Set Circuits

A set circuit represents a set of answers to a pattern $P(\alpha, \beta)$

- Singleton $\alpha : 6$ → *"the variable α is mapped to node 6"*
- Tuple $\langle \alpha : 4, \beta : 6 \rangle$: tuple of singletons
- The circuit captures a set of tuples, e.g., $\{\langle \alpha : 4, \beta : 6 \rangle, \langle \alpha : 4, \beta : 7 \rangle\}$

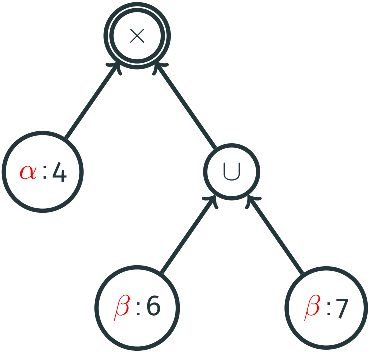


Three kinds of set-valued gates:

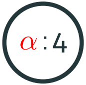
Proof Idea for Trees: Set Circuits

A **set circuit** represents a **set of answers** to a pattern $P(\alpha, \beta)$

- **Singleton** $\alpha : 6 \rightarrow$ *"the variable α is mapped to node* 6*"*
- **Tuple** $\langle \alpha : 4, \beta : 6 \rangle$: tuple of singletons
- The circuit captures a **set** of tuples, e.g., $\{\langle \alpha : 4, \beta : 6 \rangle, \langle \alpha : 4, \beta : 7 \rangle\}$



Three kinds of **set-valued gates**:

- **Variable gate** 

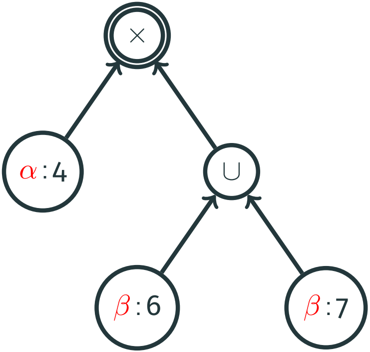
:

  $\rightarrow$ captures $\{\langle \alpha : 4 \rangle\}$

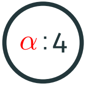
Proof Idea for Trees: Set Circuits

A **set circuit** represents a **set of answers** to a pattern $P(\alpha, \beta)$

- **Singleton** $\alpha : 6 \rightarrow$ *"the variable α is mapped to node 6"*
- **Tuple** $\langle \alpha : 4, \beta : 6 \rangle$: tuple of singletons
- The circuit captures a **set** of tuples, e.g., $\{\langle \alpha : 4, \beta : 6 \rangle, \langle \alpha : 4, \beta : 7 \rangle\}$



Three kinds of **set-valued gates**:

- **Variable gate**  :

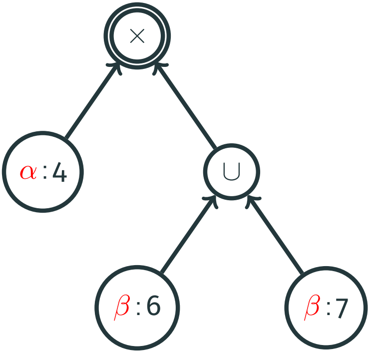
  $\rightarrow$ captures $\{\langle \alpha : 4 \rangle\}$
- **Union gate**  :

  $\rightarrow$ union of sets of tuples

Proof Idea for Trees: Set Circuits

A **set circuit** represents a **set of answers** to a pattern $P(\alpha, \beta)$

- **Singleton** $\alpha : 6$ → *"the variable α is mapped to node* 6*"*
- **Tuple** $\langle \alpha : 4, \beta : 6 \rangle$: tuple of singletons
- The circuit captures a **set** of tuples, e.g., $\{\langle \alpha : 4, \beta : 6 \rangle, \langle \alpha : 4, \beta : 7 \rangle\}$



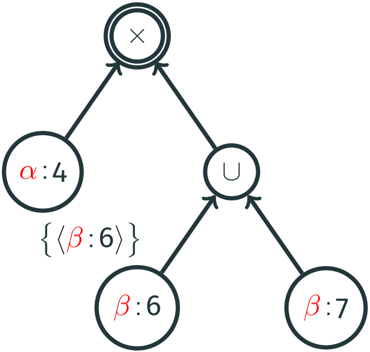
Three kinds of **set-valued gates**:

- **Variable gate** ($\alpha : 4$) :
  → captures $\{\langle \alpha : 4 \rangle\}$
- **Union gate** ($\cup$) :
  → union of sets of tuples
- **Product gate** ($\times$) :
  → relational product

Proof Idea for Trees: Set Circuits

A **set circuit** represents a **set of answers** to a pattern $P(\alpha, \beta)$

- **Singleton** $\alpha{:}6$ → *"the variable α is mapped to node* 6*"*
- **Tuple** $\langle \alpha{:}4, \beta{:}6 \rangle$: tuple of singletons
- The circuit captures a **set** of tuples, e.g., $\{\langle \alpha{:}4, \beta{:}6 \rangle, \langle \alpha{:}4, \beta{:}7 \rangle\}$



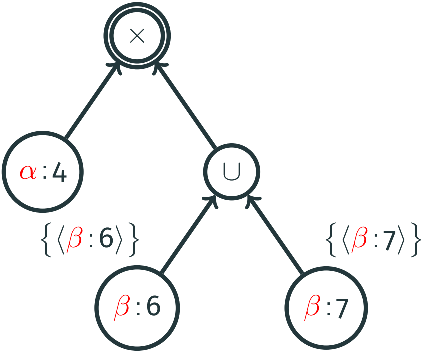
Three kinds of **set-valued gates**:

- **Variable gate** ($\alpha{:}4$):
  → captures $\{\langle \alpha{:}4 \rangle\}$
- **Union gate** ($\cup$):
  → union of sets of tuples
- **Product gate** ($\times$):
  → relational product

Proof Idea for Trees: Set Circuits

A **set circuit** represents a **set of answers** to a pattern $P(\alpha, \beta)$

- **Singleton** $\alpha : 6$ → *"the variable α is mapped to node 6"*
- **Tuple** $\langle \alpha : 4, \beta : 6 \rangle$: tuple of singletons
- The circuit captures a **set** of tuples, e.g., $\{\langle \alpha : 4, \beta : 6 \rangle, \langle \alpha : 4, \beta : 7 \rangle\}$



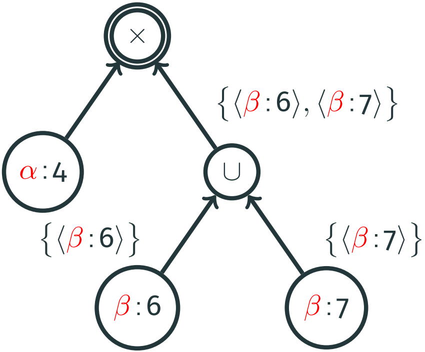
Three kinds of **set-valued gates**:

- **Variable gate** $\alpha : 4$:
  → captures $\{\langle \alpha : 4 \rangle\}$
- **Union gate** $\cup$:
  → union of sets of tuples
- **Product gate** $\times$:
  → relational product

Proof Idea for Trees: Set Circuits

A **set circuit** represents a **set of answers** to a pattern $P(\alpha, \beta)$

- **Singleton** $\alpha : 6$ → *"the variable α is mapped to node 6"*
- **Tuple** $\langle \alpha : 4, \beta : 6 \rangle$: tuple of singletons
- The circuit captures a **set** of tuples, e.g., $\{\langle \alpha : 4, \beta : 6 \rangle, \langle \alpha : 4, \beta : 7 \rangle\}$



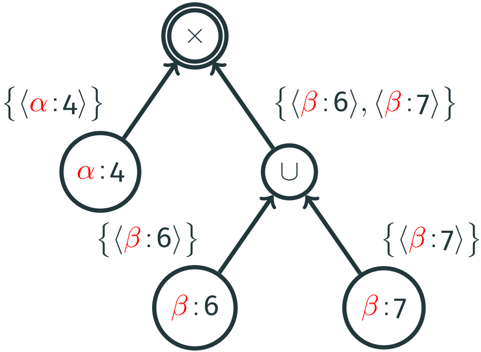
Three kinds of **set-valued gates**:

- **Variable gate** ($\alpha : 4$):
  → captures $\{\langle \alpha : 4 \rangle\}$
- **Union gate** ($\cup$):
  → union of sets of tuples
- **Product gate** ($\times$):
  → relational product

Proof Idea for Trees: Set Circuits

A **set circuit** represents a **set of answers** to a pattern $P(\alpha, \beta)$

- **Singleton** $\alpha : 6$ → *"the variable α is mapped to node* 6*"*
- **Tuple** $\langle \alpha : 4, \beta : 6 \rangle$: tuple of singletons
- The circuit captures a **set** of tuples, e.g., $\{\langle \alpha : 4, \beta : 6 \rangle, \langle \alpha : 4, \beta : 7 \rangle\}$



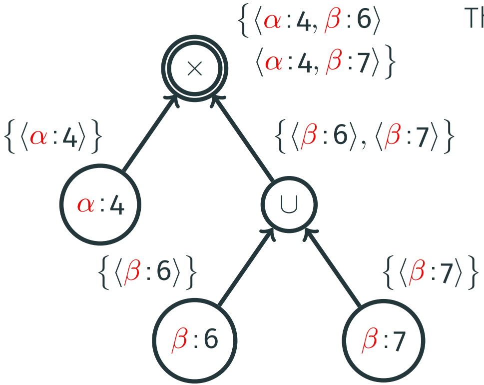
Three kinds of **set-valued gates**:

- **Variable gate** ($\alpha : 4$):
  → captures $\{\langle \alpha : 4 \rangle\}$
- **Union gate** (∪):
  → union of sets of tuples
- **Product gate** (×):
  → relational product

Proof Idea for Trees: Set Circuits

A **set circuit** represents a **set of answers** to a pattern $P(\alpha, \beta)$

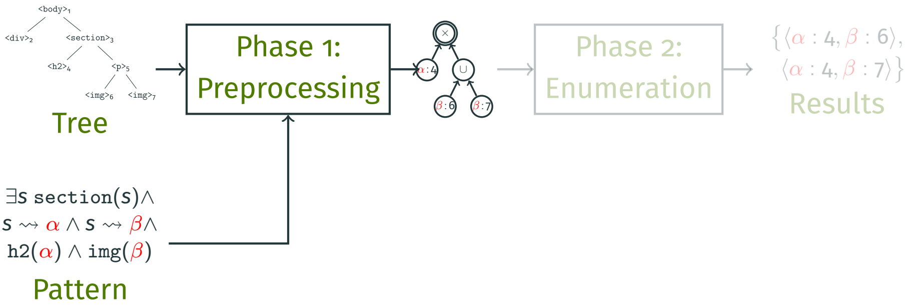
- **Singleton** $\alpha : 6 \rightarrow$ *"the variable α is mapped to node* 6*"*
- **Tuple** $\langle \alpha : 4, \beta : 6 \rangle$: tuple of singletons
- The circuit captures a **set** of tuples, e.g., $\{\langle \alpha : 4, \beta : 6 \rangle, \langle \alpha : 4, \beta : 7 \rangle\}$



Three kinds of **set-valued gates**:

- **Variable gate** $(\alpha : 4)$:
  $\rightarrow$ captures $\{\langle \alpha : 4 \rangle\}$
- **Union gate** $(\cup)$:
  $\rightarrow$ union of sets of tuples
- **Product gate** $(\times)$:
  $\rightarrow$ relational product

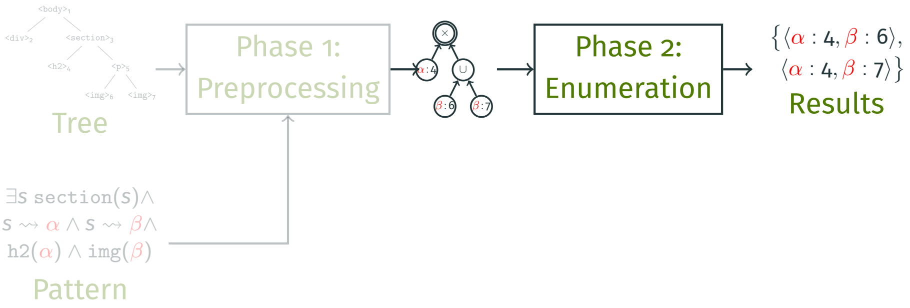
Proof Idea for Trees: Results



Theorem

For any ***tree automaton*** A *with capture variables* $\alpha_1, \ldots, \alpha_k$*, given a* ***tree*** T*, we can build in* $O(|T| \times |A|)$ *a* ***set circuit*** *capturing exactly the set of tuples* $\{\langle \alpha_1 : n_1, \ldots, \alpha_k : n_k \rangle$ *in the output of* A *on* T

Proof Idea for Trees: Results



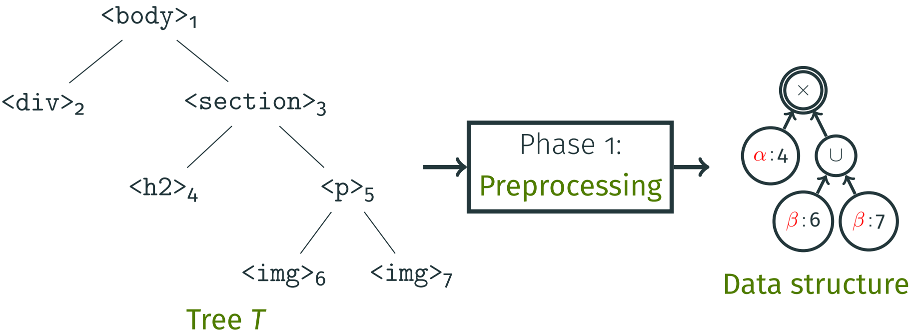
Theorem

Given a set circuit satisfying some conditions, we can enumerate all tuples that it captures with linear preprocessing and constant delay

E.g., for $\{\langle \alpha : 4, \beta : 6\rangle, \langle \alpha : 4, \beta : 7\rangle\}$: enumerate $\langle \alpha : 4, \beta : 6\rangle$ then $\langle \alpha : 4, \beta : 7\rangle$

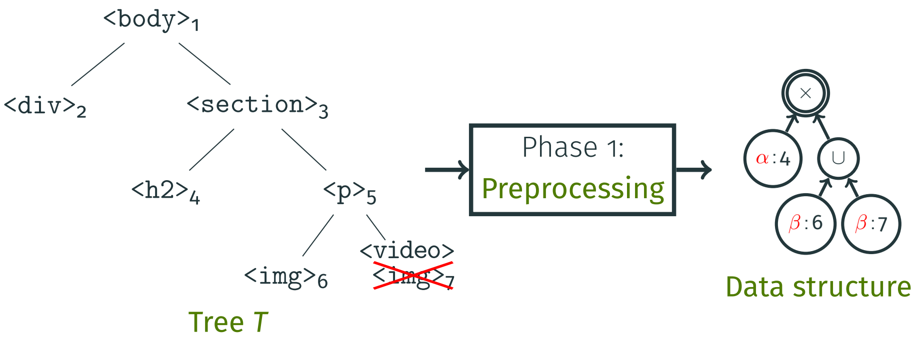
Extension: Supporting Updates

Updates



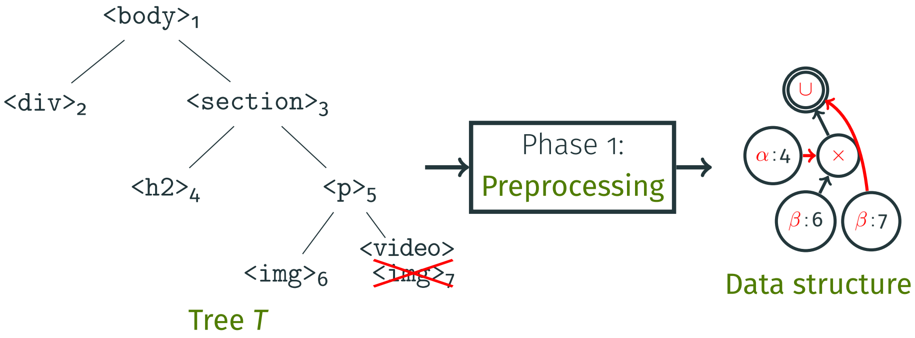
- The input data can be **modified** after the preprocessing

Updates



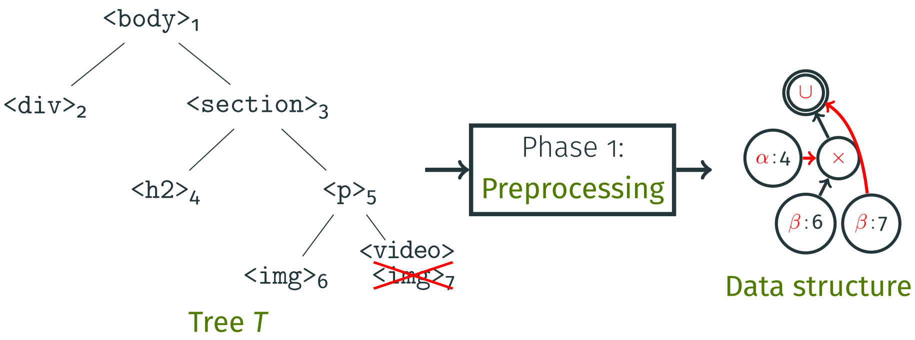
- The input data can be **modified** after the preprocessing

Updates



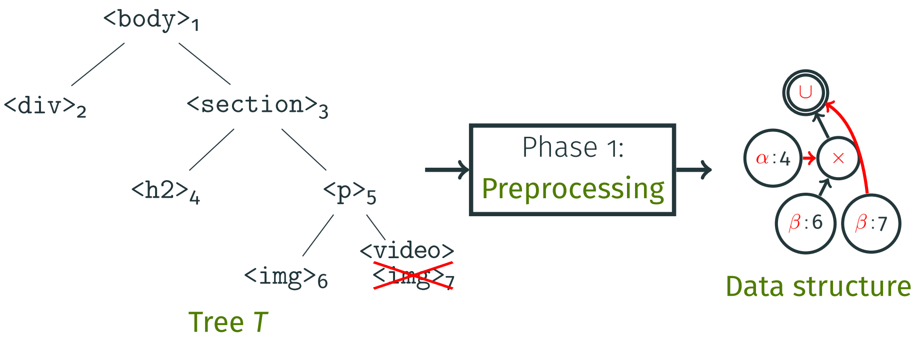
- The input data can be **modified** after the preprocessing

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**?

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	$O(T)$	$O(1)$	$O(T)$

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	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$

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	$O(T)$	$O(\log T)$	$O(\log T)$

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	$O(T)$	$O(1)$	$O(\log T)$

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	$O(T)$	$O(1)$	$O(\log T)$
[Amarilli et al., 2019]	trees	$O(T)$	$O(1)$	$O(\log T)$

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 reasonable complexity in P
  and with linear preprocessing and constant delay in 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 reasonable complexity in P and with linear preprocessing and constant delay in T

Extensions 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 reasonable complexity in P and with linear preprocessing and constant delay in T

Extensions 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 **reasonable complexity** in P and with **linear** preprocessing and **constant** delay in T

Extensions and future work:

- Extending the results from text to **trees**
- Supporting **updates** on the input data
- Understanding the connections with **circuit classes**

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 **reasonable complexity** in P and with **linear** preprocessing and **constant** delay in T

Extensions and future work:

- Extending the results from text to **trees**
- Supporting **updates** on the input data
- Understanding the connections with **circuit classes**
- Enumerating results in a relevant **order**?

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 **reasonable complexity** in P and with **linear** preprocessing and **constant** delay in T

Extensions and future work:

- Extending the results from text to **trees**
- Supporting **updates** on the input data
- Understanding the connections with **circuit classes**
- Enumerating results in a relevant **order**?
- Testing how well our methods perform in **practice**
  - $\rightarrow$ **Implementation:** `https://github.com/PoDMR/enum-spanner-rs`

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 **reasonable complexity** in P and with **linear** preprocessing and **constant** delay in T

Extensions and future work:

- Extending the results from text to **trees**
- Supporting **updates** on the input data
- Understanding the connections with **circuit classes**
- Enumerating results in a relevant **order**?
- Testing how well our methods perform in **practice**
  - $\rightarrow$ **Implementation:** `https://github.com/PoDMR/enum-spanner-rs`

Thanks for your attention!

Amarilli, A., Bourhis, P., Mengel, S., and Niewerth, M. (2019).
Enumeration on Trees with Tractable Combined Complexity and Efficient Updates.
In *PODS*.

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.