Enumerating Pattern Matches in Words and Trees

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

October 8th, 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:
  - $\rightarrow$ 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

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:

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

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**:

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

→ **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

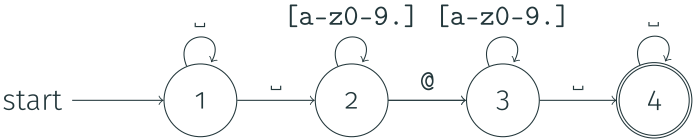
- Convert the pattern from a regular expression to an automaton

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

Solution: automata

- Convert the pattern from a regular expression to an automaton

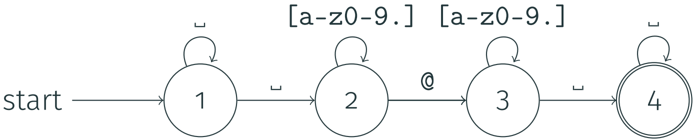
$$P := \text{␣}^{+}\ \texttt{[a-z0-9.]}^{*}\ \texttt{@}\ \texttt{[a-z0-9.]}^{*}\ \text{␣}^{+}$$



Solution: automata

- Convert the pattern from a regular expression to an automaton

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



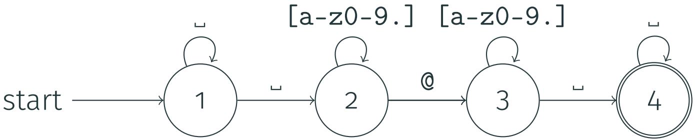
- Then, evaluate the automaton on the text T

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

Solution: automata

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

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



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

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

- How **efficient** is this?

Complexity of automaton evaluation

- Task: testing if an automaton A accepts a text T

Complexity of automaton evaluation

- Task: testing if an automaton A accepts a text T
- We measure complexity according to two metrics:

Complexity of automaton evaluation

- **Task:** testing if an automaton A accepts a text T
- We measure complexity according to **two metrics**:
  - **Data complexity**: in the text T

Complexity of automaton evaluation

- **Task:** testing if an automaton A accepts a text T
- We measure complexity according to **two metrics**:
  - **Data complexity**: in the text T
  - **Combined complexity**: in T and A

Complexity of automaton evaluation

- **Task:** testing if an automaton A accepts a text T
- We measure complexity according to **two metrics**:
  - **Data complexity**: in the text T
  - **Combined complexity**: in T and A
- If the automaton A is **deterministic**...

Complexity of automaton evaluation

- **Task:** testing if an automaton A accepts a text T
- We measure complexity according to **two metrics**:
  - **Data complexity**: in the text T
  - **Combined complexity**: in T and A
- If the automaton A is **deterministic**...
  - **Data complexity** is **linear** in T

Complexity of automaton evaluation

- **Task:** testing if an automaton A accepts a text T
- We measure complexity according to **two metrics**:
  - **Data complexity**: in the text T
  - **Combined complexity**: in T and A
- If the automaton A is **deterministic**...
  - **Data complexity** is **linear** in T
  - **Combined complexity** is **polynomial** in T and A

Complexity of automaton evaluation

- **Task:** testing if an automaton A accepts a text T
- We measure complexity according to **two metrics**:
  - **Data complexity**: in the text T
  - **Combined complexity**: in T and A
- If the automaton A is **deterministic**...
  - **Data complexity** is **linear** in T
  - **Combined complexity** is **polynomial** in T and A
- In general...

Complexity of automaton evaluation

- **Task:** testing if an automaton A accepts a text T
- We measure complexity according to **two metrics**:
  - **Data complexity**: in the text T
  - **Combined complexity**: in T and A
- If the automaton A is **deterministic**...
  - **Data complexity** is **linear** in T
  - **Combined complexity** is **polynomial** in T and A
- In general...
  - Compute a **deterministic automaton** from A...

Complexity of automaton evaluation

- **Task:** testing if an automaton A accepts a text T
- We measure complexity according to **two metrics**:
  - **Data complexity**: in the text T
  - **Combined complexity**: in T and A
- If the automaton A is **deterministic**...
  - **Data complexity** is **linear** in T
  - **Combined complexity** is **polynomial** in T and A
- In general...
  - Compute a **deterministic automaton** from A...
    - $\rightarrow$ **Linear** data complexity but **exponential** combined complexity

Complexity of automaton evaluation

- **Task:** testing if an automaton A accepts a text T

- We measure complexity according to **two metrics**:
  - **Data complexity**: in the text T
  - **Combined complexity**: in T and A

- If the automaton A is **deterministic**...
  - **Data complexity** is **linear** in T
  - **Combined complexity** is **polynomial** in T and A

- In general...
  - Compute a **deterministic automaton** from A...
    $\rightarrow$ **Linear** data complexity but **exponential** combined complexity
  - Better: Read T while remembering the current **set** of states (like determinizing A, but **on the fly**)

Complexity of automaton evaluation

- **Task:** testing if an automaton A accepts a text T
- We measure complexity according to **two metrics**:
  - **Data complexity**: in the text T
  - **Combined complexity**: in T and A
- If the automaton A is **deterministic**…
  - **Data complexity** is **linear** in T
  - **Combined complexity** is **polynomial** in T and A
- In general…
  - Compute a **deterministic automaton** from A…

    $\rightarrow$ **Linear** data complexity but **exponential** combined complexity
  - Better: Read T while remembering the current **set** of states (like determinizing A, but **on the fly**)

    $\rightarrow$ **Linear** data complexity and **polynomial** combined complexity

Actual problem: Extracting all patterns

- This only tests if the pattern exactly matches the whole text!
  $\rightarrow$ ‘‘YES’’

Actual problem: Extracting all patterns

- This only tests **if** the pattern **exactly matches** the whole text!
  $\rightarrow$ ‘‘YES’’
- We want to **actually find** all pattern matches!
  $\rightarrow$ Find all **pairs of positions** that are the endpoints of a match

Actual problem: Extracting all patterns

- This only tests **if** the pattern **exactly matches** the whole text!
  $\rightarrow$ ‘‘YES’’

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

  $\rightarrow$ Find all **pairs of positions** that are the endpoints of a match

- **Generalization:** patterns that can capture a **tuple** of positions
  $\rightarrow$ Find the email addresses **without leading/trailing spaces**
  $\rightarrow$ Find all **pairs** of a name followed by an email address

Patterns with capture variables

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

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

Patterns with capture variables

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

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

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

Patterns with capture variables

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

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

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

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

Patterns with capture variables

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

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

- 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

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

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

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

  - Output: the list of matches of P on T

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

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
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

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

  - **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

α	`l`	`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

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

l α 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

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

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

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`

  → 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`	

  → 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:

    `aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`

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:

    ```
    aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
    ```

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

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** $P := \bullet^* \ \alpha \ \mathtt{a}^* \ \beta \ \bullet^*$
  - 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** $P := \bullet^* \ \alpha \ \mathtt{a}^* \ \beta \ \bullet^*$
  - 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

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

→ 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

$$\bullet^* \sqcup^+ \alpha\, \texttt{[a-z0-9.]}^* \texttt{@}$$
$$\texttt{[a-z0-9.]}^* \beta \sqcup^+ \bullet^*$$

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

$\bullet^{*}\textvisiblespace^{+}\alpha\,\texttt{[a-z0-9.]}^{*}\texttt{@}$

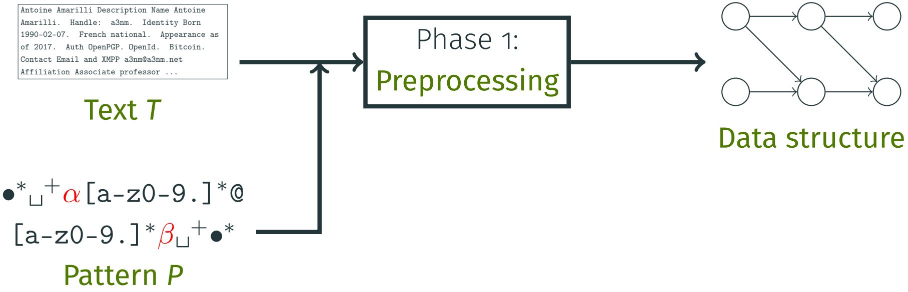
$\texttt{[a-z0-9.]}^{*}\beta\,\textvisiblespace^{+}\bullet^{*}$

Pattern P

Phase 1:

Preprocessing

Formalizing 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 ...
```

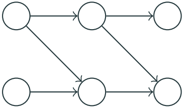
Text T

$\bullet^* \sqcup^+ \alpha$ `[a-z0-9.]*@`

`[a-z0-9.]*` $\beta \sqcup^+ \bullet^*$

Pattern P

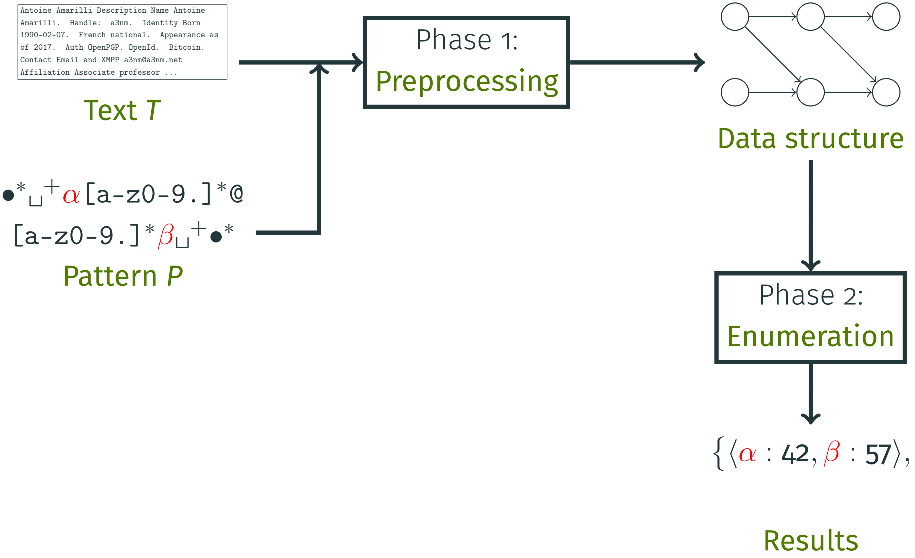
Phase 1:
Preprocessing



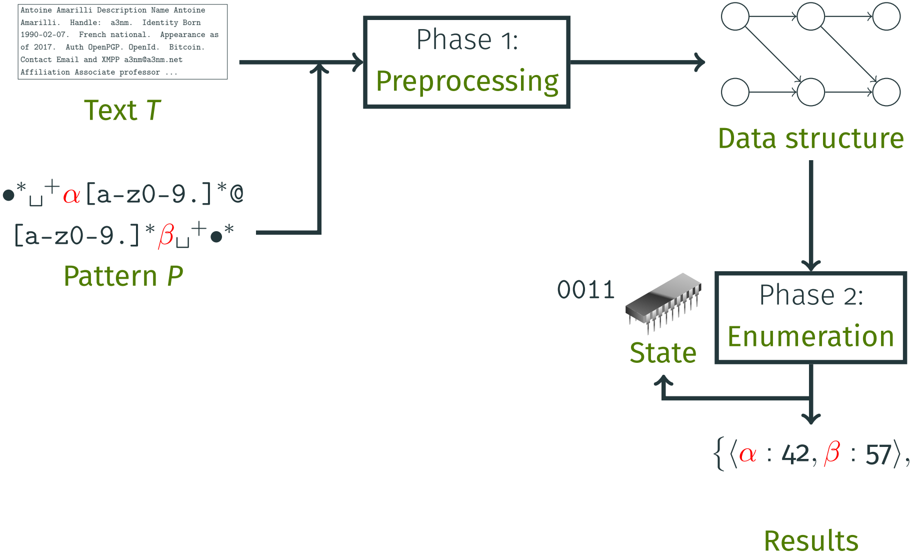
Data structure

Phase 2:
Enumeration

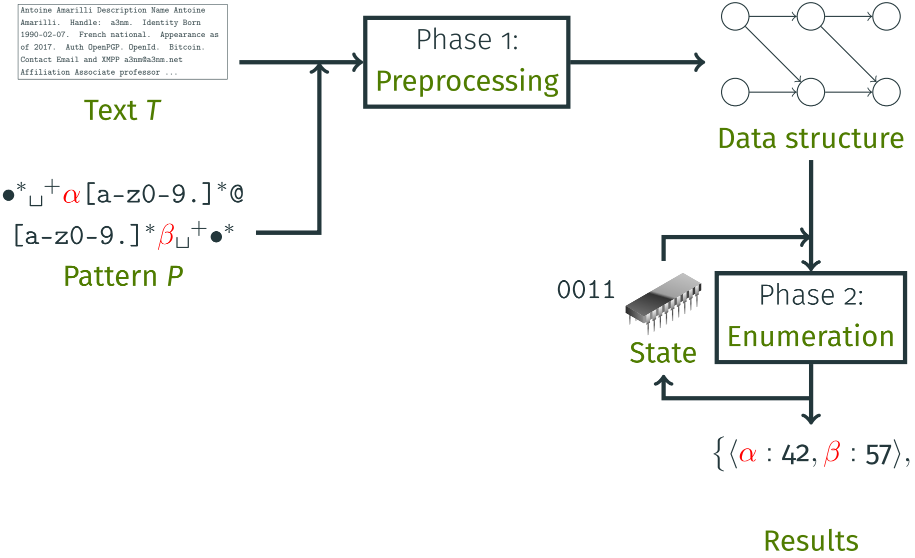
Formalizing enumeration algorithms



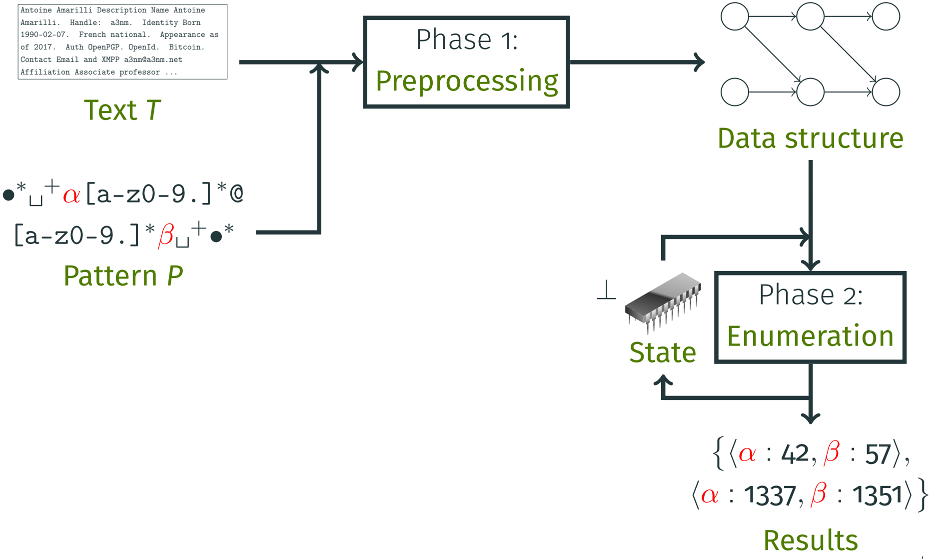
Formalizing enumeration algorithms



Formalizing enumeration algorithms



Formalizing enumeration algorithms



Formalizing enumeration algorithms

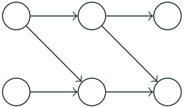
```
Antoine Amarilli Description Name Antoine
Amarilli.  Handle:  a3nm.  Identity Born
1990-02-07.  French national.  Appearance as
of 2017.  Auth OpenPGP. OpenId.  Bitcoin.
Contact Email and XMPP a3nm@a3nm.net
Affiliation Associate professor ...
```

Text T

$\bullet^* \sqcup^+ \alpha$ `[a-z0-9.]*@[a-z0-9.]*` $\beta \sqcup^+ \bullet^*$

Pattern P

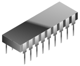
Phase 1: **Preprocessing**



Data structure

Phase 2: **Enumeration**

$\perp$



State

$\{\langle \alpha : 42, \beta : 57 \rangle, \langle \alpha : 1337, \beta : 1351 \rangle\}$

Results

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
  - $\rightarrow$ 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...

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

  → 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)*

$\rightarrow$ 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)*

Automaton formalism

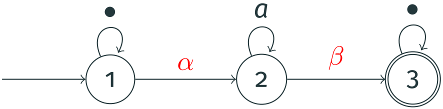
- We use automata that read letters and capture variables

Automaton formalism

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

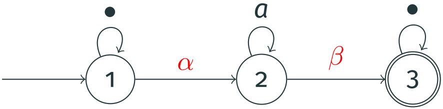
Automaton formalism

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



Automaton formalism

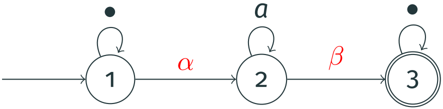
- We use automata that read letters and capture variables
  - → Example: $P := \bullet^* \ \alpha \ a^* \ \beta \ \bullet^*$



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

Automaton formalism

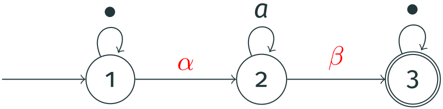
- We use automata that read letters and capture variables
  - → Example: $P := \bullet^* \;\; \alpha \;\; a^* \;\; \beta \;\; \bullet^*$



- 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

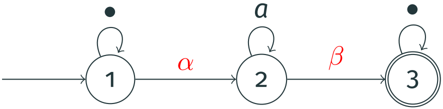
- We use automata that read letters and capture variables
  - → Example: $P := \bullet^* \ \alpha \ a^* \ \beta \ \bullet^*$



- 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
- 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
  - → 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 $\langle \alpha : \boldsymbol{i}, \beta : \boldsymbol{j} \rangle$ such that A has an accepting run reading α at position $\boldsymbol{i}$ and β at $\boldsymbol{j}$
- **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

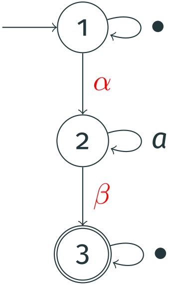
Compute a product DAG of the text T and of the automaton A

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

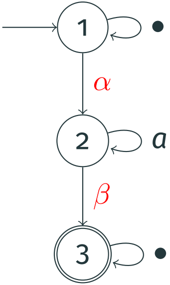


Proof idea: product DAG

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

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

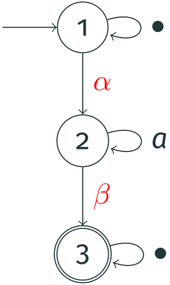
a	a	a	b	a

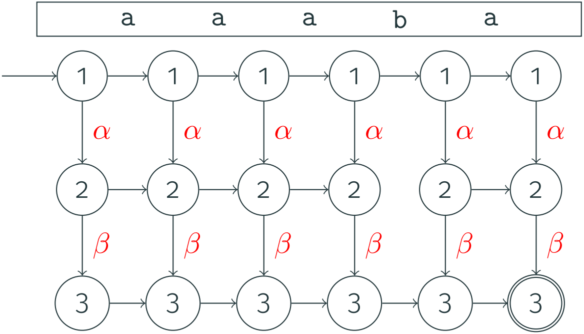


Proof idea: product DAG

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

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

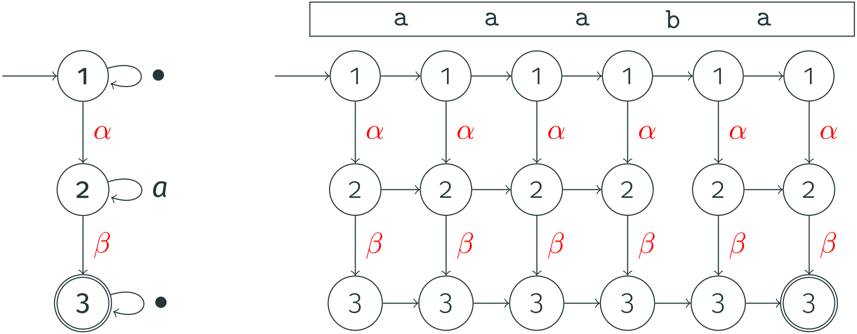




Proof idea: product DAG

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

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

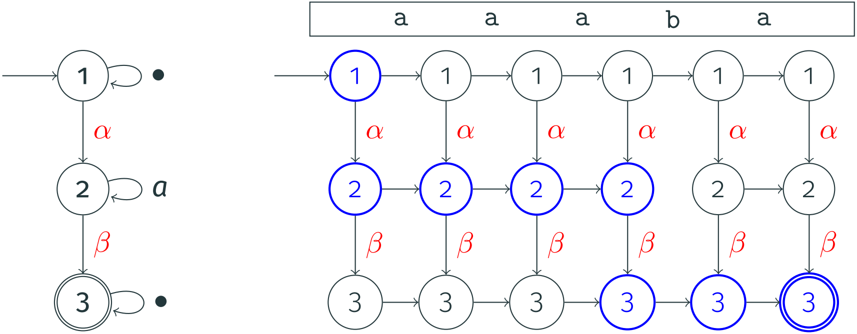


$\rightarrow$ Each path in the product DAG corresponds to a match

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 $\langle \alpha : 0, \beta : 3 \rangle$

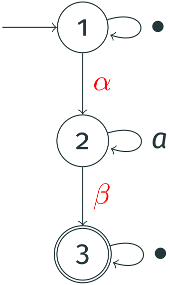


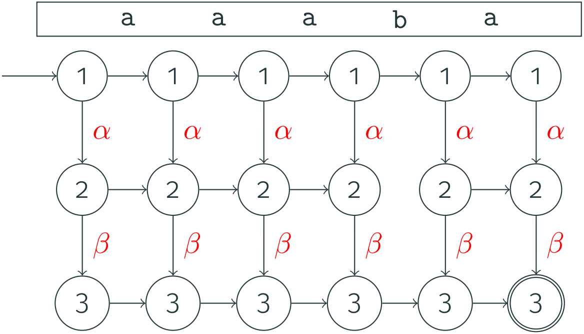
→ Each **path** in the **product DAG** corresponds to a **match**

Proof idea: product DAG

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

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

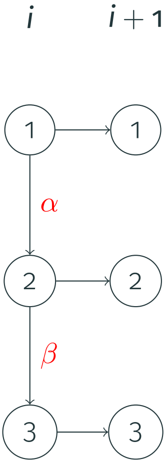




$\rightarrow$ Each **path** in the **product DAG** corresponds to a **match**

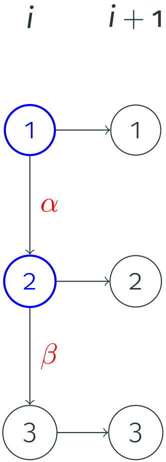
$\rightarrow$ **Challenge:** Enumerate paths but avoid **duplicate matches** and do not **waste time** to ensure constant delay

Proof idea: on-the-fly computation to avoid duplicates



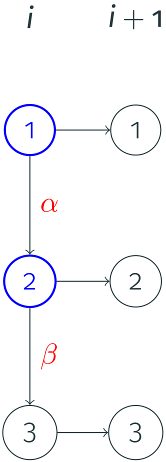
- We are at a position i and set of states in blue

Proof idea: on-the-fly computation to avoid duplicates



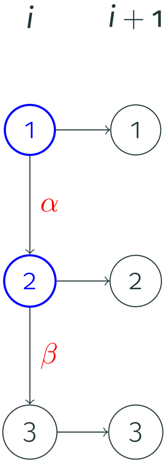
- We are at a position i and set of states in blue

Proof idea: on-the-fly computation to avoid duplicates



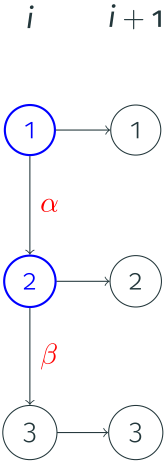
- We are at a position i and set of states in blue
- Partition tuples based on the set S of variables assigned at the current position

Proof idea: on-the-fly computation to avoid duplicates



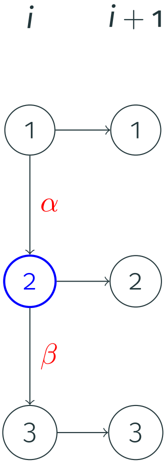
- We are at a position i and set of states in blue
- Partition tuples based on the set S of variables assigned at the current position
- For each S, consider the set of states where we can be at $i+1$ when reading S at i

Proof idea: on-the-fly computation to avoid duplicates



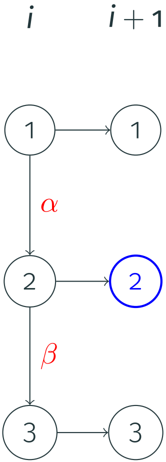
- We are at a position i and set of states in blue
- Partition tuples based on the set S of variables assigned at the current position
- For each S, consider the set of states where we can be at $i+1$ when reading S at i
    - Example: $S = \{\alpha\}$

Proof idea: on-the-fly computation to avoid duplicates



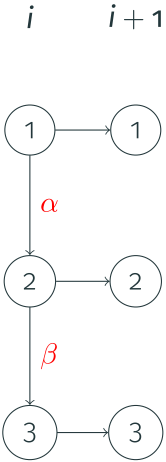
- We are at a position i and set of states in blue
- Partition tuples based on the set S of variables assigned at the current position
- For each S, consider the set of states where we can be at $i+1$ when reading S at i
  - Example: $S = \{\alpha\}$

Proof idea: on-the-fly computation to avoid duplicates



- We are at a **position** i and **set of states** in blue
- Partition tuples based on the **set S of variables** assigned at the current position
- For each S, consider the **set of states** where we can be at $i+1$ when reading S at i
  - Example: $S = \{\alpha\}$

Proof idea: on-the-fly computation to avoid duplicates



- We are at a **position** i and **set of states** in blue
- Partition tuples based on the **set S of variables** assigned at the current position
- For each S, consider the **set of states** where we can be at $i+1$ when reading S at i
  - Example: $S = \{\alpha\}$

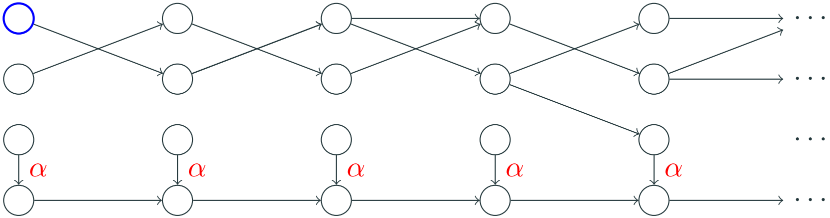
$\rightarrow$ We must have **preprocessed** the DAG to make sure that we can always finish the run

Proof idea: jump pointers to save time

- Issue: When we can't assign variables, we do not make progress

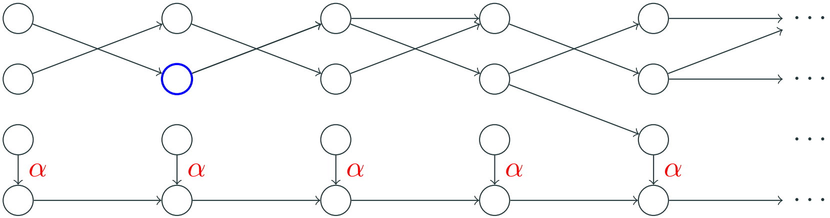
Proof idea: jump pointers to save time

- Issue: When we can't assign variables, we do not make progress



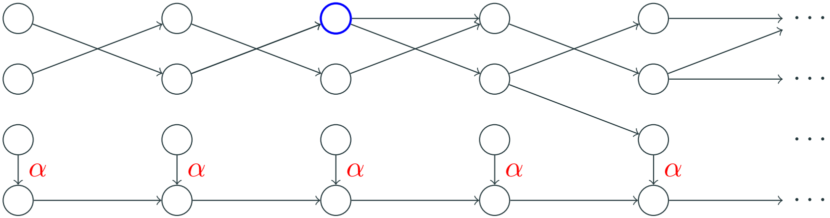
Proof idea: jump pointers to save time

- **Issue:** When we can't assign variables, we do not make **progress**



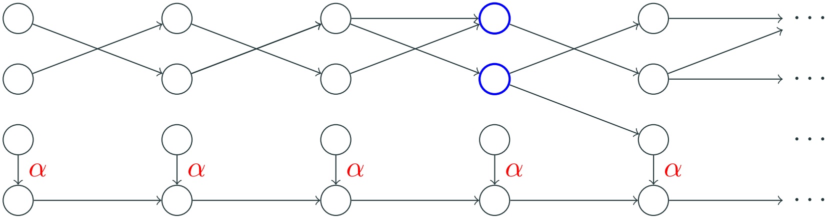
Proof idea: jump pointers to save time

- Issue: When we can't assign variables, we do not make progress



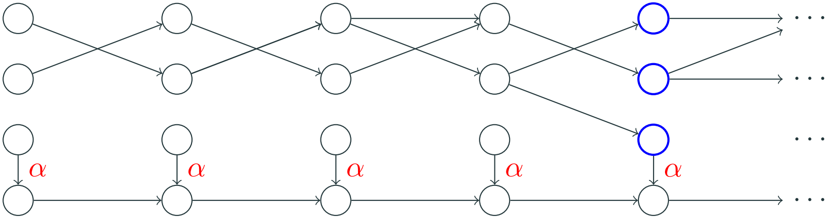
Proof idea: jump pointers to save time

- **Issue:** When we can't assign variables, we do not make **progress**



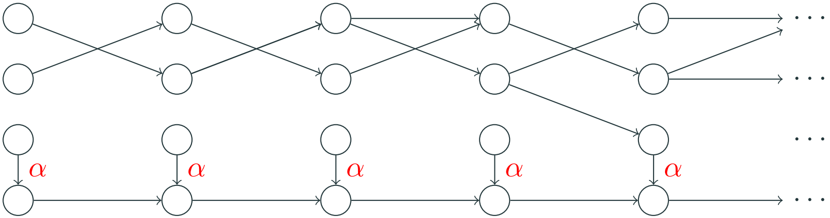
Proof idea: jump pointers to save time

- **Issue:** When we can’t assign variables, we do not make **progress**



Proof idea: jump pointers to save time

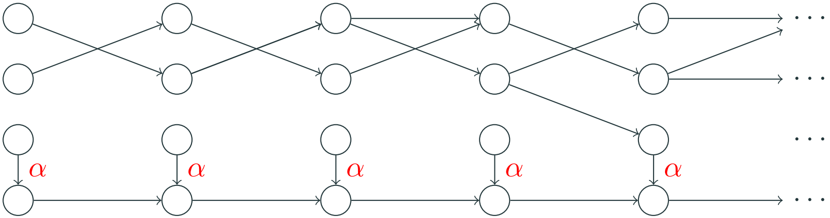
- Issue: When we can't assign variables, we do not make progress



- Idea: Directly jump to the reachable states
  at the next position where we can assign a variable

Proof idea: jump pointers to save time

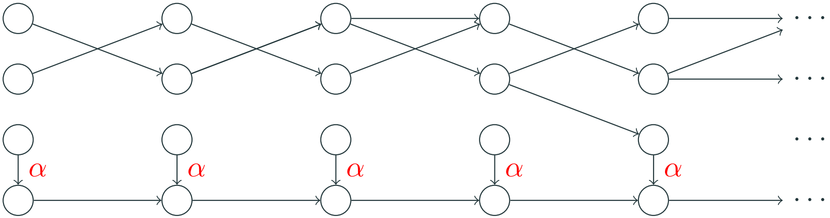
- Issue: When we can't assign variables, we do not make progress



- Idea: Directly jump to the reachable states
  at the next position where we can assign a variable
- Challenge: Preprocessing in linear time in T and polynomial in A:

Proof idea: jump pointers to save time

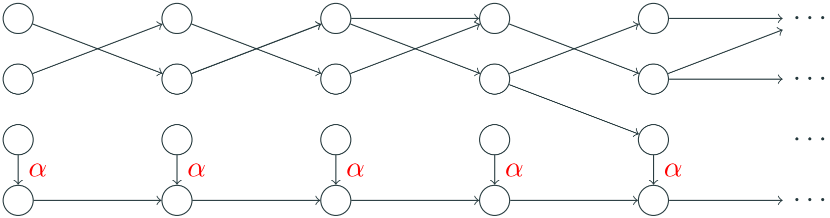
- **Issue:** When we can't assign variables, we do not make **progress**



- **Idea:** Directly **jump** to the reachable states
  at the next position where we can assign a variable
- **Challenge:** Preprocessing in **linear time** in T and **polynomial** in A:
  - $\rightarrow$ Compute for each state the **next position** where we can reach some state that can assign a variable

Proof idea: jump pointers to save time

- Issue: When we can't assign variables, we do not make progress

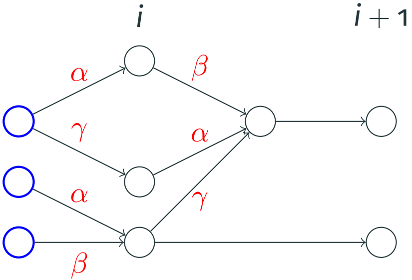


- Idea: Directly jump to the reachable states at the next position where we can assign a variable

- Challenge: Preprocessing in linear time in T and polynomial in A:
  - $\rightarrow$ Compute for each state the next position where we can reach some state that can assign a variable
  - $\rightarrow$ Compute at each position i the transitive closure to all positions j such that j is the next position of some state at i (there are $\leq |A|$)

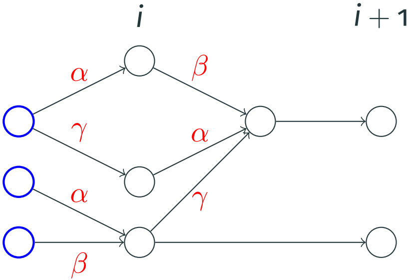
Proof idea: flashlight search

- **Issue:** Finding which **variable sets** we can assign at position i?



Proof idea: flashlight search

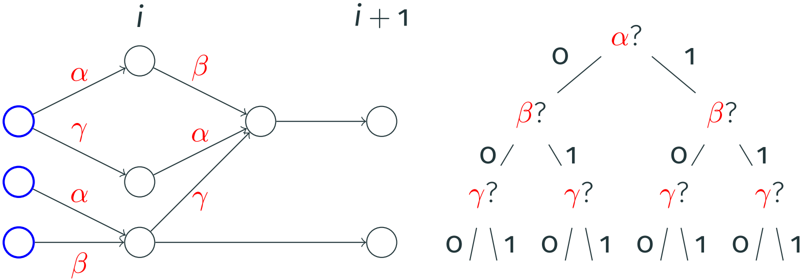
- Issue: Finding which variable sets we can assign at position i?



- Idea: Explore a decision tree on the variables (built on the fly)

Proof idea: flashlight search

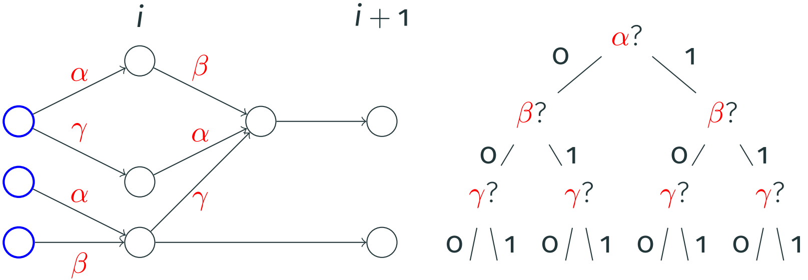
- **Issue:** Finding which **variable sets** we can assign at position i?



- **Idea:** Explore a **decision tree** on the variables (built on the fly)

Proof idea: flashlight search

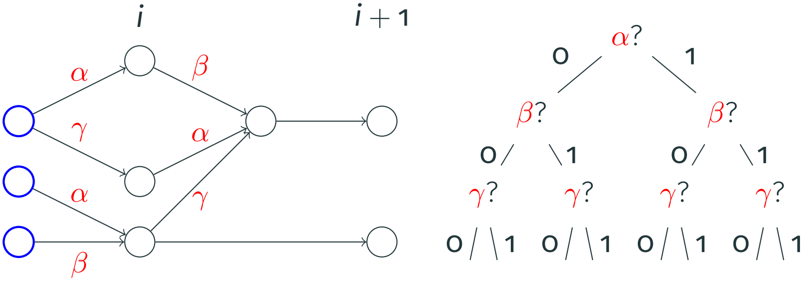
- **Issue:** Finding which **variable sets** we can assign at position i?



- **Idea:** Explore a **decision tree** on the variables (built on the fly)
- At each decision tree **node**, find the reachable **states** which have **all required variables** (1) and **no forbidden variables** (0)

Proof idea: flashlight search

- **Issue:** Finding which **variable sets** we can assign at position i?

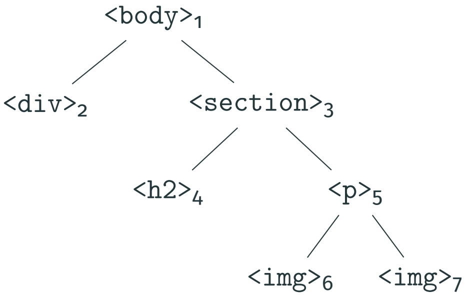


- **Idea:** Explore a **decision tree** on the variables (built on the fly)
- At each decision tree **node**, find the reachable **states** which have **all required variables** (1) and **no forbidden variables** (0)
  - $\rightarrow$ **Assumption**: we don't see the same variable **twice** on a path

Extension: From Text to Trees

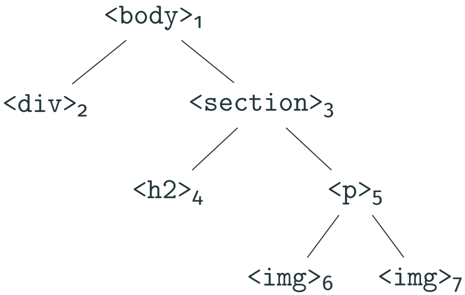
Pattern matching on trees

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



Pattern matching on trees

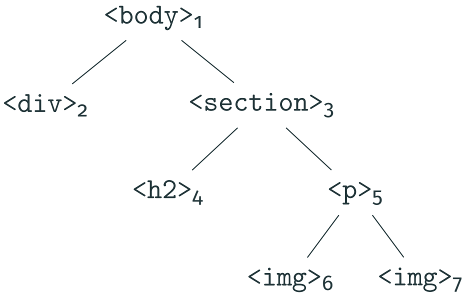
- The data T is no longer text but is now a tree:



- 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:

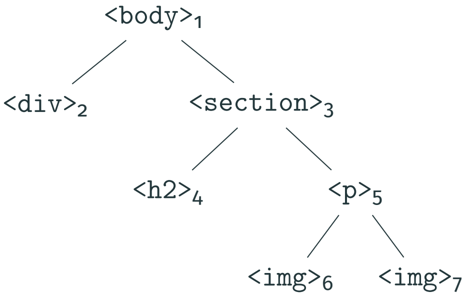


- The pattern P asks about the structure of the tree:
  Is there an h2 header and an image in the same section?

- Results:

Pattern matching on trees

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

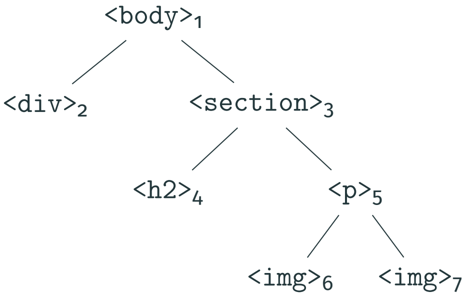


- The **pattern** *P* asks about the **structure** of the tree:
  *Is there α: an **h2** header and β: an **image** in the same section?*

- **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...

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)*

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)*

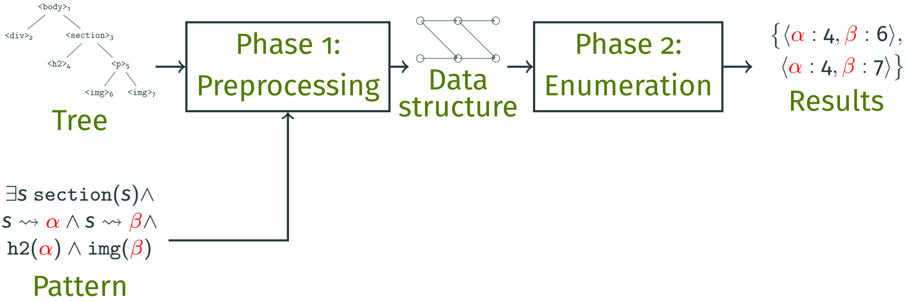
- Again, this is only in **data complexity**!
- We **conjecture** the following bounds for this task (ongoing work):

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)*

Proof idea for trees: structure

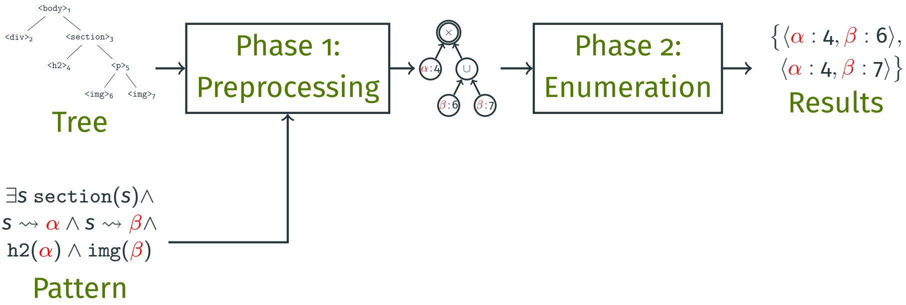
Similar structure to the previous proof, but with a circuit:



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

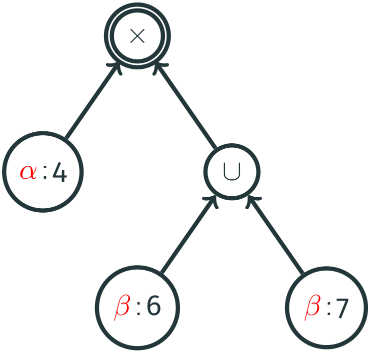
A set circuit represents a set of answers to a pattern $P(\alpha, \beta)$

- 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** $\alpha : 6 \rightarrow$ *"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., $\{\langle \alpha : 4, \beta : 6 \rangle, \langle \alpha : 4, \beta : 7 \rangle\}$

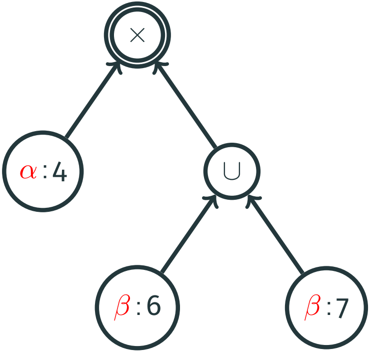


Three kinds of **set-valued gates**:

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:

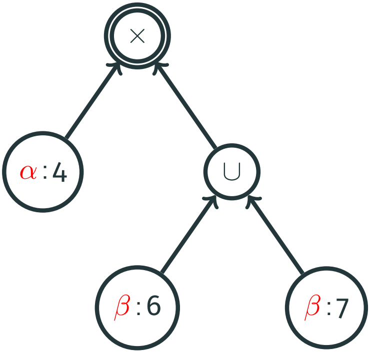
- Variable gate $\alpha : 4$:

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

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., $\{\langle \alpha\!:\!4, \beta\!:\!6 \rangle, \langle \alpha\!:\!4, \beta\!:\!7 \rangle\}$



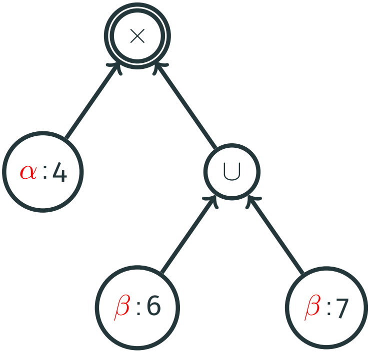
Three kinds of **set-valued gates**:

- **Variable gate** $\alpha\!:\!4$:
  $\rightarrow$ captures $\{\langle \alpha\!:\!4 \rangle\}$
- **Union gate** $\cup$:
  $\rightarrow$ union of sets of tuples

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., $\{\langle \alpha : 4, \beta : 6 \rangle, \langle \alpha : 4, \beta : 7 \rangle\}$



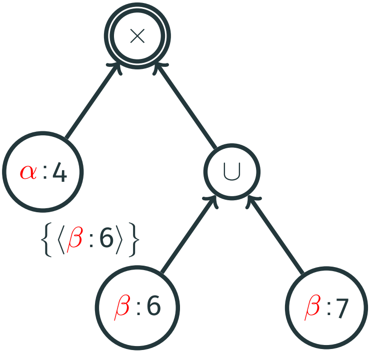
Three kinds of **set-valued gates**:

- **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: 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., $\{\langle \alpha : 4, \beta : 6 \rangle, \langle \alpha : 4, \beta : 7 \rangle\}$



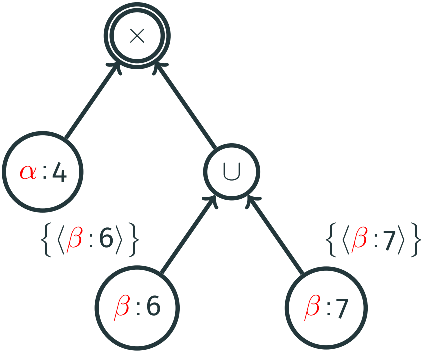
Three kinds of **set-valued gates**:

- **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: 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., $\{\langle \alpha:4, \beta:6 \rangle, \langle \alpha:4, \beta:7 \rangle\}$



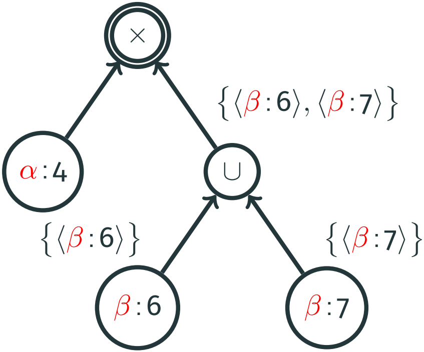
Three kinds of **set-valued gates**:

- **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: 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., $\{\langle \alpha : 4, \beta : 6 \rangle, \langle \alpha : 4, \beta : 7 \rangle\}$



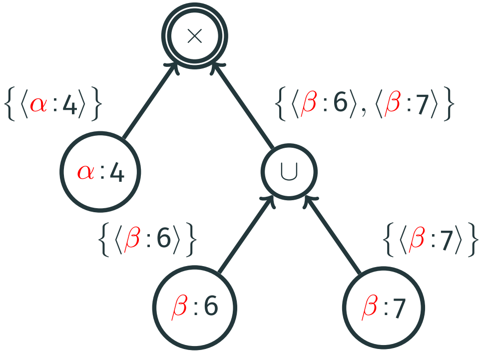
Three kinds of **set-valued gates**:

- **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: 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., $\{\langle \alpha : 4, \beta : 6 \rangle, \langle \alpha : 4, \beta : 7 \rangle\}$



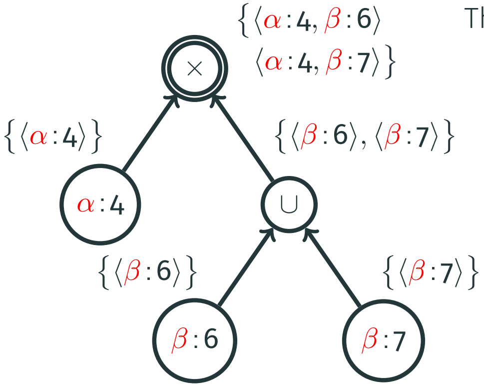
Three kinds of **set-valued gates**:

- **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 $P(\alpha, \beta)$

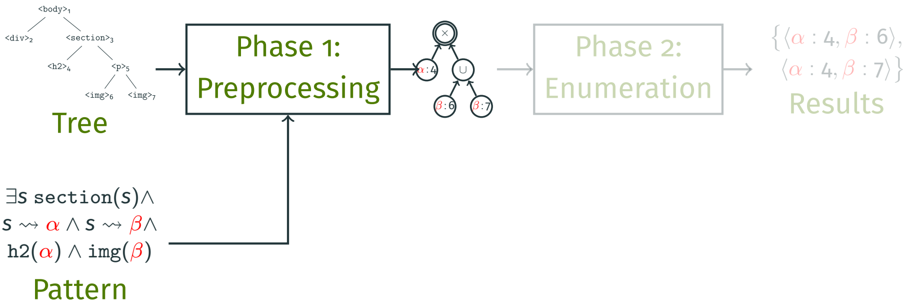
- **Singleton** $\alpha : 6 \rightarrow$ *"the variable α is mapped to node* 6*"*
- **Tuple** $\langle \alpha : 4, \beta : 6 \rangle$: tuple of singletons
- The circuit captures a **set** of tuples, e.g., $\{\langle \alpha : 4, \beta : 6 \rangle, \langle \alpha : 4, \beta : 7 \rangle\}$



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: set circuit construction



Theorem

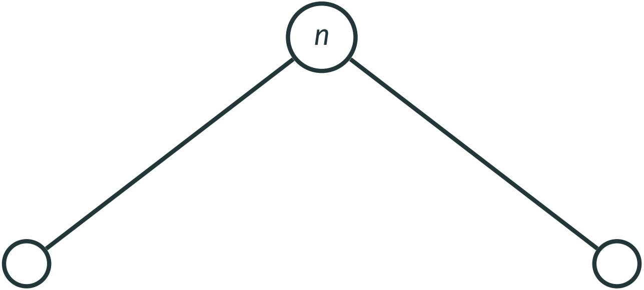
*For any **tree automaton** A with capture variables $\alpha_1, \ldots, \alpha_k$, given a **tree** T, we can build in $O(|T| \times |A|)$ a **set circuit** capturing exactly the set of tuples $\{\langle \alpha_1 : n_1, \ldots, \alpha_k : n_k \rangle$ in the output of A on T*

Proof idea for trees: set circuit construction (details)

- **Automaton:** *"Select all node pairs* (α, β)*"*
- **States:** $\{\emptyset, \alpha, \beta, \alpha\beta\}$

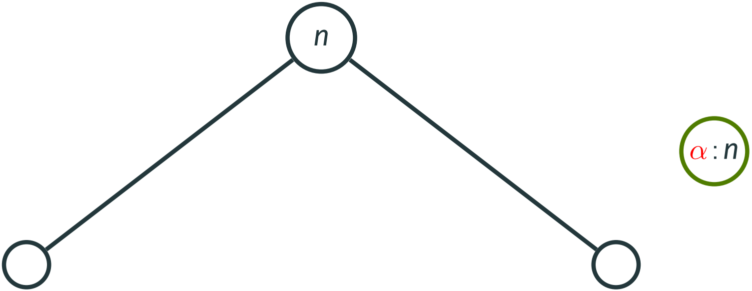
Proof idea for trees: set circuit construction (details)

- **Automaton:** *"Select all node pairs* (α, β)*"*
- **States:** $\{\emptyset, \alpha, \beta, \alpha\beta\}$



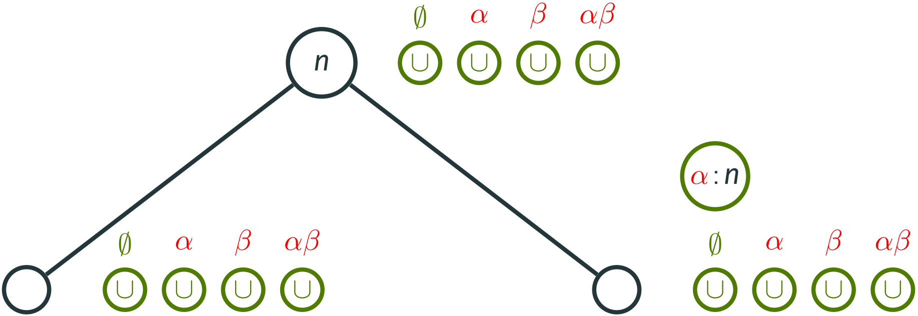
Proof idea for trees: set circuit construction (details)

- **Automaton:** *"Select all node pairs* (α, β)*"*
- **States:** $\{\emptyset, \alpha, \beta, \alpha\beta\}$



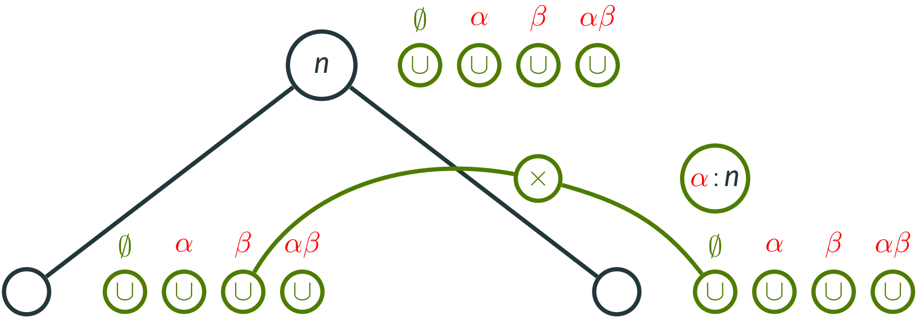
Proof idea for trees: set circuit construction (details)

- **Automaton:** *"Select all node pairs* (α, β)*"*
- **States:** $\{\emptyset, \alpha, \beta, \alpha\beta\}$



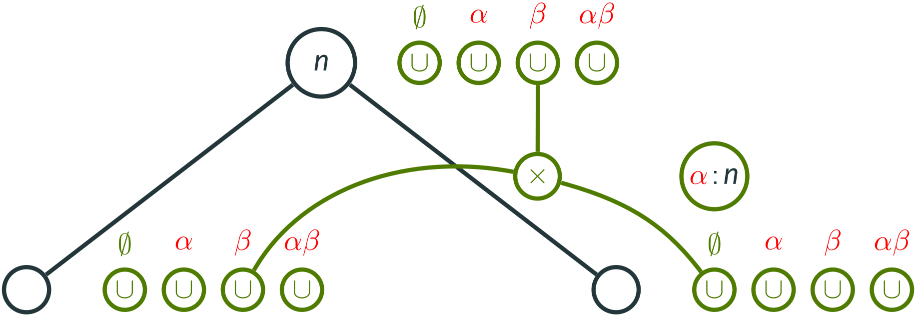
Proof idea for trees: set circuit construction (details)

- **Automaton:** *"Select all node pairs* (α, β)*"*
- **States:** $\{\emptyset, \alpha, \beta, \alpha\beta\}$



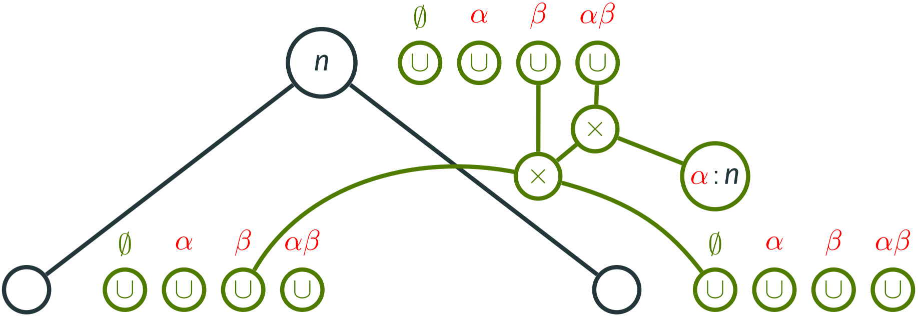
Proof idea for trees: set circuit construction (details)

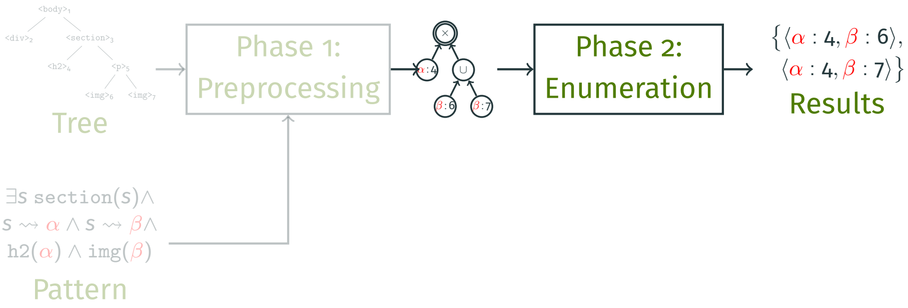
- **Automaton:** *"Select all node pairs* (α, β)*"*
- **States:** $\{\emptyset, \alpha, \beta, \alpha\beta\}$



Proof idea for trees: set circuit construction (details)

- **Automaton:** *"Select all node pairs* (α, β)*"*
- **States:** $\{\emptyset, \alpha, \beta, \alpha\beta\}$





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 $\langle \alpha : 4, \beta : 7 \rangle$

Proof idea for trees: general enumeration approach

- → Enumerate the set $T(g)$ captured by each gate g
- → Do it by top-down induction on the circuit

Proof idea for trees: general enumeration approach

- $\rightarrow$ Enumerate the set $T(g)$ captured by each gate g
- $\rightarrow$ Do it by **top-down induction** on the circuit

Base case: variable $\alpha : n$:

Proof idea for trees: general enumeration approach

- $\rightarrow$ Enumerate the set $T(g)$ captured by each gate g
- $\rightarrow$ Do it by **top-down induction** on the circuit

Base case: variable $\alpha : n$: enumerate $\langle \alpha : n \rangle$ and stop

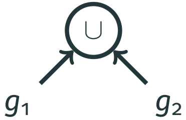
Proof idea for trees: general enumeration approach

→ Enumerate the set $T(g)$ captured by each gate g

→ Do it by **top-down induction** on the circuit

Base case: variable $\alpha : n$: enumerate $\langle \alpha : n \rangle$ and stop

∪-gate



Concatenation: enumerate $T(g_1)$
and then enumerate $T(g_2)$

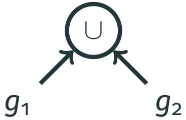
Proof idea for trees: general enumeration approach

→ Enumerate the set $T(g)$ captured by each gate g

→ Do it by **top-down induction** on the circuit

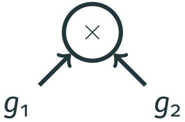
Base case: variable $\alpha : n$: enumerate $\langle \alpha : n \rangle$ and stop

$\cup$-gate



Concatenation: enumerate $T(g_1)$ and then enumerate $T(g_2)$

$\times$-gate

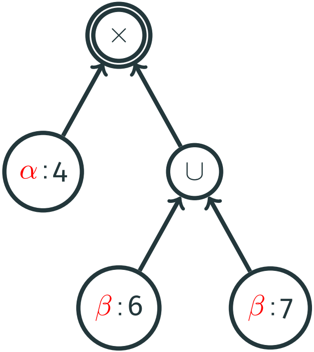


Lexicographic product:

for every t_1 in $T(g_1)$:
  for every t_2 in $T(g_2)$:
    output $t_1 + t_2$

Proof idea for trees: circuit conditions

Enumeration relies on some **conditions** on the input circuit (d-DNNF):



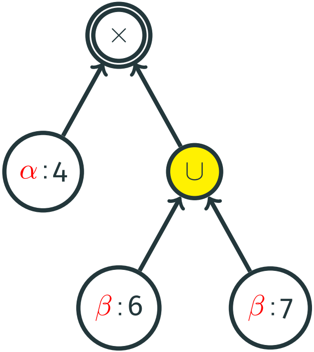
Proof idea for trees: circuit conditions

Enumeration relies on some **conditions** on the input circuit (d-DNNF):

- $\cup$ are all **deterministic**:

For any two inputs g_1 and g_2 of a $\cup$-gate, the captured sets $T(g_1)$ and $T(g_2)$ are **disjoint** (they have no tuple in common)

$\rightarrow$ Avoids **duplicate tuples**



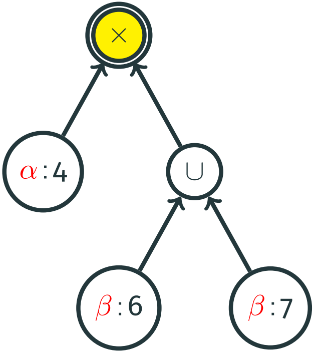
Proof idea for trees: circuit conditions

Enumeration relies on some **conditions** on the input circuit (d-DNNF):

- $\cup$ are all **deterministic**:

For any two inputs g_1 and g_2 of a $\cup$-gate, the captured sets $T(g_1)$ and $T(g_2)$ are **disjoint** (they have no tuple in common)

$\rightarrow$ Avoids **duplicate tuples**



- $\times$ are all **decomposable**:

For any two inputs g_1 and g_2 of a $\times$-gate, no variable has a path to both g_1 and g_2

$\rightarrow$ Avoids **duplicate singletons**

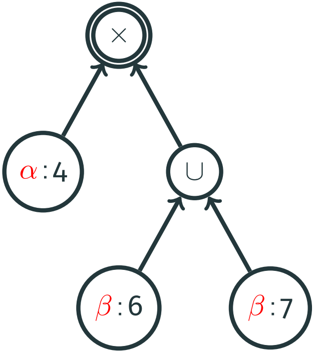
Proof idea for trees: circuit conditions

Enumeration relies on some **conditions** on the input circuit (d-DNNF):

- $\cup$ are all **deterministic**:

For any two inputs g_1 and g_2 of a $\cup$-gate, the captured sets $T(g_1)$ and $T(g_2)$ are **disjoint** (they have no tuple in common)

$\rightarrow$ Avoids **duplicate tuples**



- $\times$ are all **decomposable**:

For any two inputs g_1 and g_2 of a $\times$-gate, no variable has a path to both g_1 and g_2

$\rightarrow$ Avoids **duplicate singletons**

- Also an additional **upwards-determinism** condition

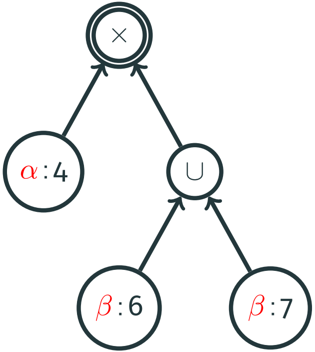
Proof idea for trees: circuit conditions

Enumeration relies on some **conditions** on the input circuit (d-DNNF):

- $\cup$ are all **deterministic**:

For any two inputs g_1 and g_2 of a $\cup$-gate, the captured sets $T(g_1)$ and $T(g_2)$ are **disjoint** (they have no tuple in common)

$\rightarrow$ Avoids **duplicate tuples**



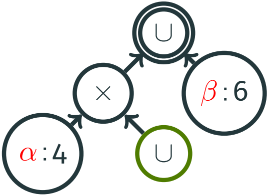
- $\times$ are all **decomposable**:

For any two inputs g_1 and g_2 of a $\times$-gate, no variable has a path to both g_1 and g_2

$\rightarrow$ Avoids **duplicate singletons**

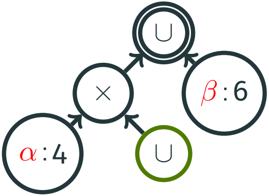
- Also an additional **upwards-determinism** condition
- Our circuit satisfies these thanks to **automaton determinism**

Proof idea for trees: enumeration subtleties



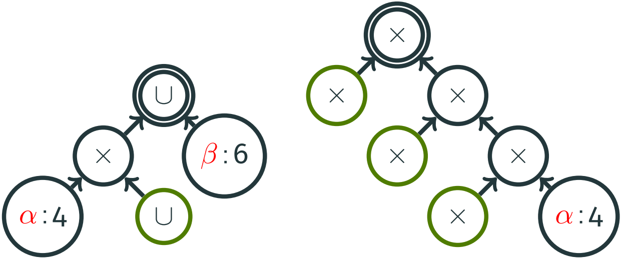
- We must not waste time in gates capturing $\emptyset$

Proof idea for trees: enumeration subtleties



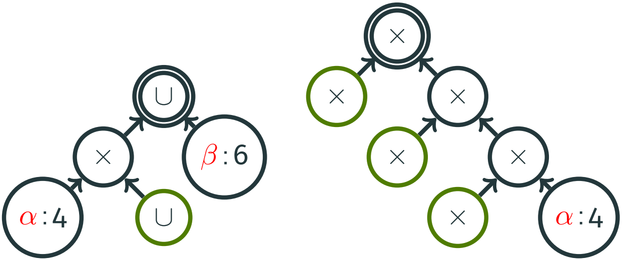
- We must not waste time in gates capturing $\emptyset$
  - $\rightarrow$ Label them during the preprocessing

Proof idea for trees: enumeration subtleties



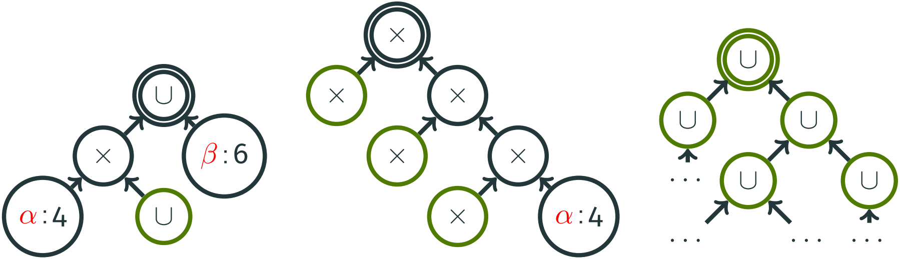
- We must not waste time in gates capturing $\emptyset$
  - $\rightarrow$ **Label** them during the preprocessing
- We must not waste time because of gates capturing $\{\langle\rangle\}$

Proof idea for trees: enumeration subtleties



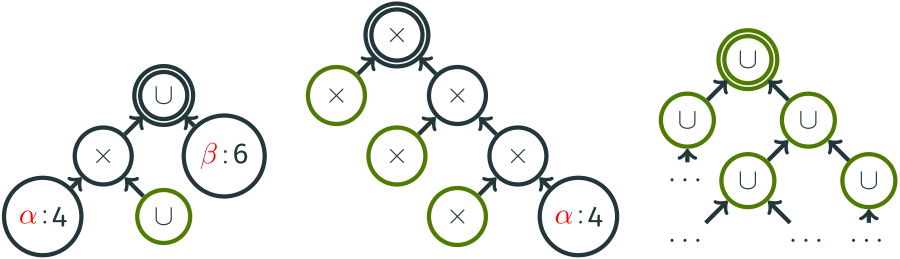
- We must not waste time in gates capturing $\emptyset$
  - $\rightarrow$ **Label** them during the preprocessing
- We must not waste time because of gates capturing $\{\langle\rangle\}$
  - $\rightarrow$ **Homogenization** to set them aside

Proof idea for trees: enumeration subtleties



- We must not waste time in gates capturing $\emptyset$
  - $\rightarrow$ **Label** them during the preprocessing
- We must not waste time because of gates capturing $\{\langle\rangle\}$
  - $\rightarrow$ **Homogenization** to set them aside
- We must not waste time in **hierarchies of $\cup$-gates**

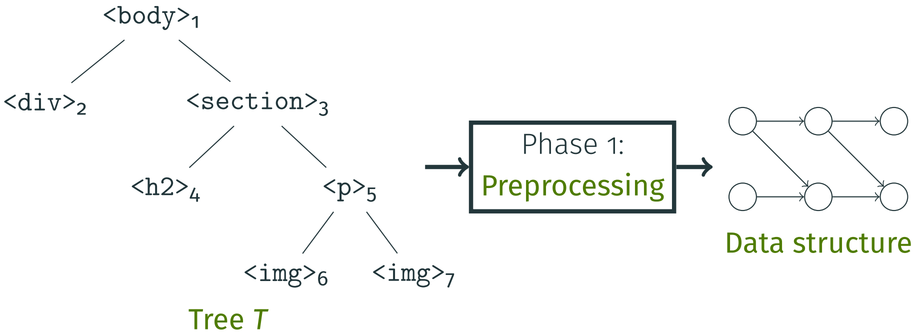
Proof idea for trees: enumeration subtleties



- We must not waste time in gates capturing $\emptyset$
  - $\rightarrow$ **Label** them during the preprocessing
- We must not waste time because of gates capturing $\{\langle\rangle\}$
  - $\rightarrow$ **Homogenization** to set them aside
- We must not waste time in **hierarchies of $\cup$-gates**
  - $\rightarrow$ Precompute a **reachability index** (uses **upwards-determinism**)

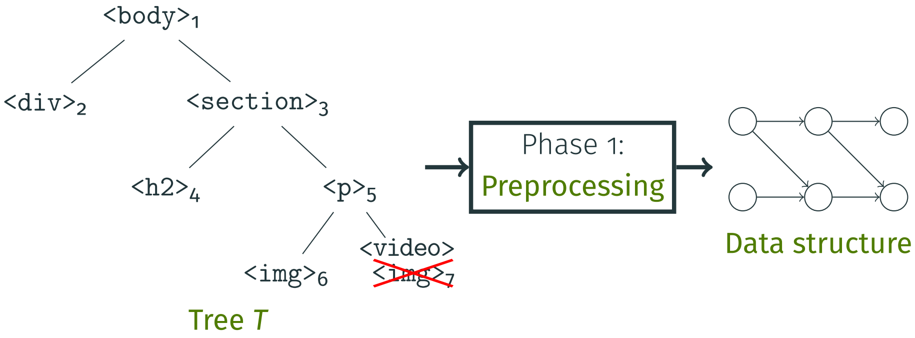
Extension: Handling Updates

Updates



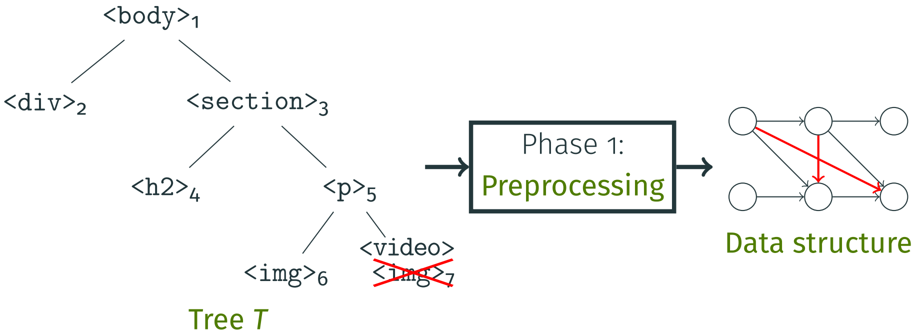
- The input data can be **modified** after the preprocessing

Updates



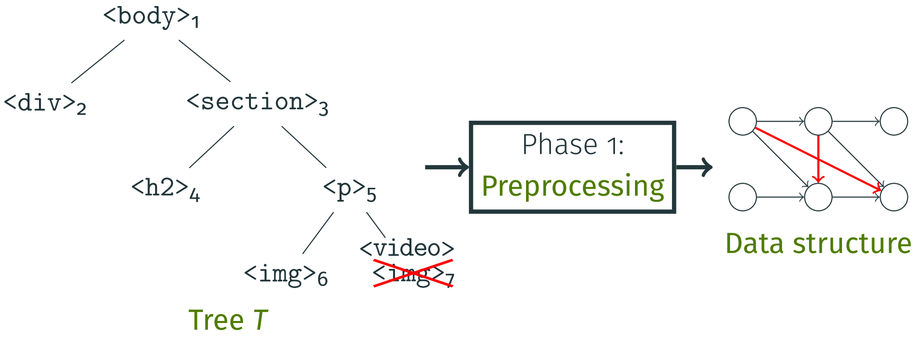
- The input data can be **modified** after the preprocessing

Updates



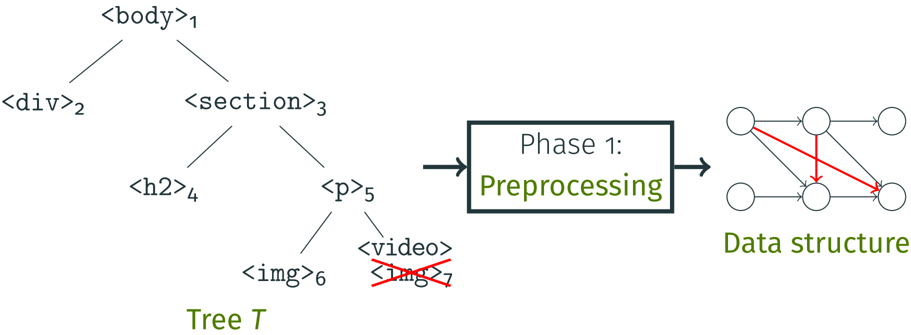
- The input data can be **modified** after the preprocessing

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):

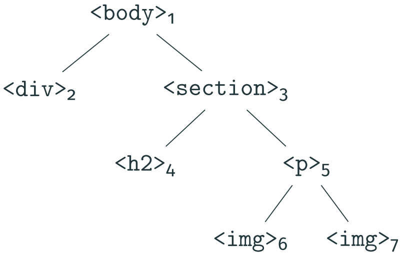
Work	Data	Preproc.	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$
[Losemann and Martens, 2014]	trees	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$
[Losemann and Martens, 2014]	text	$O(T)$	$O(\log T)$	$O(\log T)$

Known results on dynamic trees

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

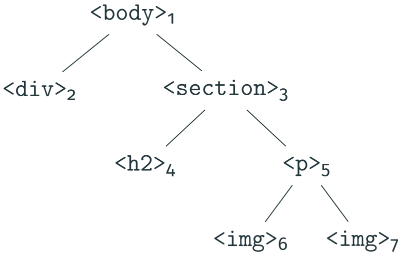
Work	Data	Preproc.	Delay	Updates
[Bagan, 2006], [Kazana and Segoufin, 2013]	trees	$O(T)$	$O(1)$	$O(T)$
[Losemann and Martens, 2014]	trees	$O(T)$	$O(\log^2 T)$	$O(\log^2 T)$
[Losemann and Martens, 2014]	text	$O(T)$	$O(\log T)$	$O(\log T)$
[Niewerth and Segoufin, 2018]	text	$O(T)$	$O(1)$	$O(\log T)$

Relabelings



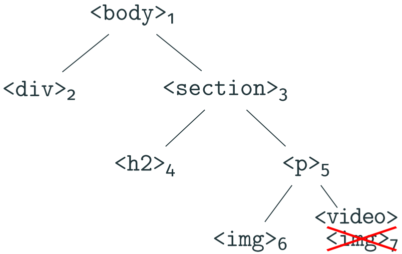
- Special kind of updates: relabelings that change the label of a node

Relabelings



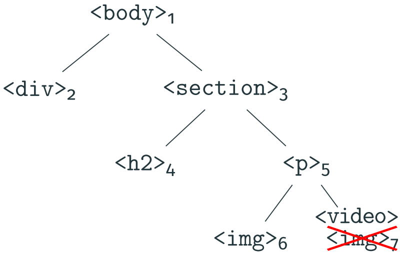
- Special kind of updates: relabelings that change the label of a node
- Example: relabel node 7 to `<video>`

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

- Current proof uses hybrid circuits but we want to simplify it

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

- Current proof uses hybrid circuits but we want to simplify it
- Remaining open questions:
  - $\rightarrow$ Does this hold for more general updates (insert/delete, etc.)?
  - $\rightarrow$ Can we also achieve tractable combined complexity?

Extension: Connection to Circuits

Connections with Boolean circuits

- Mapping DAGs and set circuits can be seen as variants of Boolean circuits

Connections with Boolean circuits

- Mapping DAGs and set circuits can be seen as variants of Boolean circuits
- The answers to enumerate are their satisfying assignments

Connections with Boolean circuits

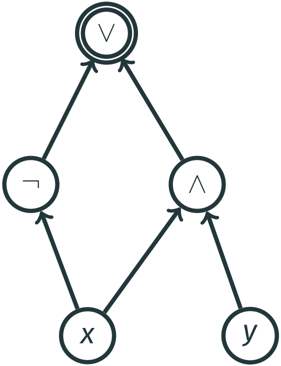
- Mapping DAGs and set circuits can be seen as variants of Boolean circuits
- The answers to enumerate are their satisfying assignments
- These circuits fall in restricted circuit classes
  that allow for efficient enumeration

Connections with Boolean circuits

- Mapping DAGs and set circuits can be seen as variants of Boolean circuits
- The answers to enumerate are their satisfying assignments
- These circuits fall in restricted circuit classes that allow for efficient enumeration

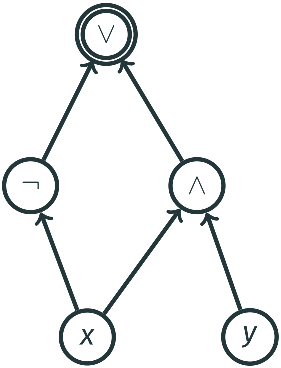
$\rightarrow$ Task: Given a Boolean circuit, how to efficiently enumerate its satisfying valuations?

Boolean circuits



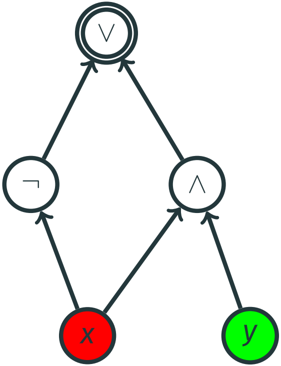
- Directed acyclic graph of **gates**
- **Output** gate: 
- **Variable** gates: x
- **Internal** gates: $\lor$ $\land$ $\neg$

Boolean circuits



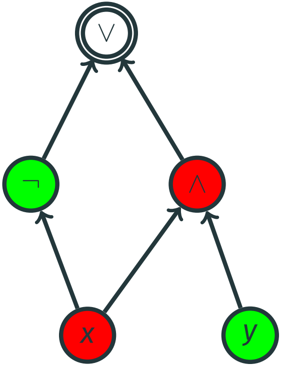
- Directed acyclic graph of **gates**
- **Output** gate:
- **Variable** gates: x
- **Internal** gates: $\vee$ $\wedge$ $\neg$
- **Valuation**: function from variables to $\{0, 1\}$
  Example: $\nu = \{x \mapsto 0,\ y \mapsto 1\}$...

Boolean circuits



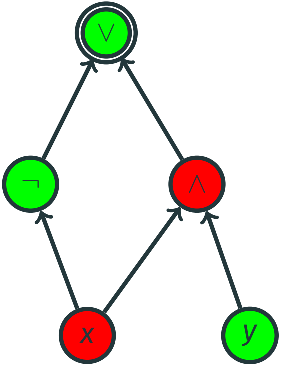
- Directed acyclic graph of **gates**
- **Output** gate: 
- **Variable** gates: x
- **Internal** gates: $\lor$ $\land$ $\neg$
- **Valuation**: function from variables to $\{0, 1\}$
  Example: $\nu = \{x \mapsto 0,\ y \mapsto 1\}$...

Boolean circuits



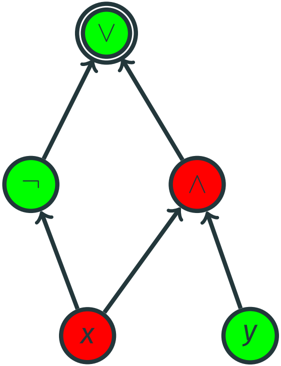
- Directed acyclic graph of **gates**
- **Output** gate:
- **Variable** gates: x
- **Internal** gates: $\lor$ $\land$ $\neg$
- **Valuation**: function from variables to $\{0, 1\}$
  Example: $\nu = \{x \mapsto 0,\ y \mapsto 1\}$...

Boolean circuits



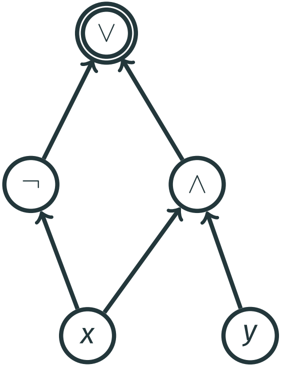
- Directed acyclic graph of **gates**
- **Output** gate:
- **Variable** gates: x
- **Internal** gates: $\vee$ $\wedge$ $\neg$
- **Valuation**: function from variables to $\{0, 1\}$
  Example: $\nu = \{x \mapsto 0,\ y \mapsto 1\}$... mapped to 1

Boolean circuits



- Directed acyclic graph of **gates**
- **Output** gate:
- **Variable** gates: x
- **Internal** gates: $\lor$ $\land$ $\neg$
- **Valuation**: function from variables to $\{0, 1\}$
  Example: $\nu = \{x \mapsto 0,\ y \mapsto 1\}$... mapped to 1
- **Assignment**: set of variables mapped to 1
  Example: $S_\nu = \{y\}$; more concise than ν

Boolean circuits



- Directed acyclic graph of **gates**
- **Output** gate: 
- **Variable** gates: x
- **Internal** gates: $\vee$ $\wedge$ $\neg$
- **Valuation**: function from variables to $\{0, 1\}$
  Example: $\nu = \{x \mapsto 0,\ y \mapsto 1\}$... mapped to 1
- **Assignment**: set of variables mapped to 1
  Example: $S_\nu = \{y\}$; more concise than ν

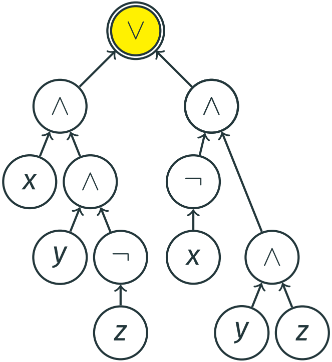
Our task: Enumerate all **satisfying assignments** of an input circuit

Circuit restrictions

d-DNNF:

- $\lor$ are all **deterministic**:

The inputs are **mutually exclusive** (= no valuation ν makes two inputs simultaneously evaluate to 1)



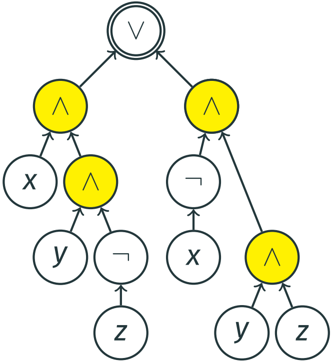
d-DNNF:

- $\vee$ are all **deterministic**:

The inputs are **mutually exclusive** (= no valuation ν makes two inputs simultaneously evaluate to 1)

- $\wedge$ are all **decomposable**:

The inputs are **independent** (= no variable x has a path to two different inputs)



Circuit restrictions

d-DNNF:

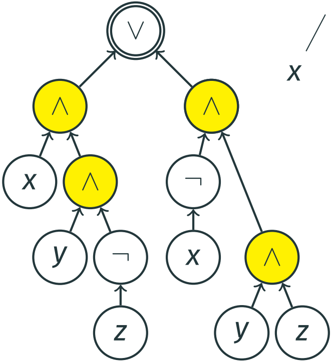
- $\vee$ are all **deterministic**:

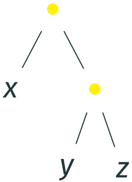
The inputs are **mutually exclusive** (= no valuation ν makes two inputs simultaneously evaluate to 1)

- $\wedge$ are all **decomposable**:

The inputs are **independent** (= no variable x has a path to two different inputs)

v-tree: $\wedge$-gates follow a **tree** on the variables





Main results

Theorem

Given a d-DNNF circuit C with a v-tree T, we can enumerate its satisfying assignments with preprocessing linear in $|C| + |T|$ and delay linear in each assignment

Main results

Theorem

Given a d-DNNF circuit C with a v-tree T, we can enumerate its satisfying assignments with preprocessing linear in $|C| + |T|$ and delay linear in each assignment

Also: restrict to assignments of **constant size** $k \in \mathbb{N}$ (at most k variables are set to 1):

Theorem

Given a d-DNNF circuit C with a v-tree T, we can enumerate its satisfying assignments of size $\leq k$ with preprocessing linear in $|C| + |T|$ and constant delay

Main results

Theorem

Given a d-DNNF circuit C with a v-tree T, we can enumerate its satisfying assignments with preprocessing linear in $|C| + |T|$ and delay linear in each assignment

Also: restrict to assignments of **constant size** $k \in \mathbb{N}$ (at most k variables are set to 1):

Theorem

Given a d-DNNF circuit C with a v-tree T, we can enumerate its satisfying assignments of size $\leq k$ with preprocessing linear in $|C| + |T|$ and constant delay

Subtleties: Must **complete** to a set circuit; memory usage problems

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
- Understanding the connections with circuits

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
- Understanding the connections with **circuits**
- 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 **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
- Understanding the connections with **circuits**
- Enumerating results in a relevant **order**?
- 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
- Understanding the connections with **circuits**
- Enumerating results in a relevant **order**?
- Testing how well our methods perform in **practice**

Thanks for your attention!

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.