Topological Sorting under Regular Constraints

Antoine Amarilli[1], **Charles Paperman**[2]

December 7th, 2018

[1]Télécom ParisTech

[2]Université de Lille

Constrained Topological Sorting

- Fix an alphabet: e.g., $\Sigma = \{a, b\}$

Constrained Topological Sorting

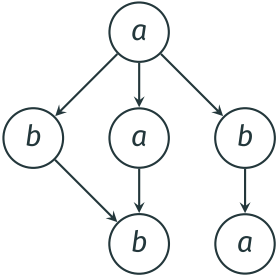
- Fix an **alphabet**: e.g., $\Sigma = \{a, b\}$
- Fix a **language**: e.g., $L = (ab)^*$

Constrained Topological Sorting

- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:

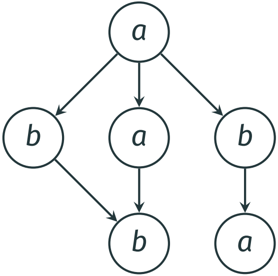
Constrained Topological Sorting

- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ



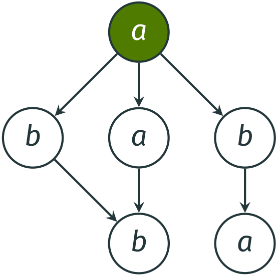
Constrained Topological Sorting

- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?



Constrained Topological Sorting

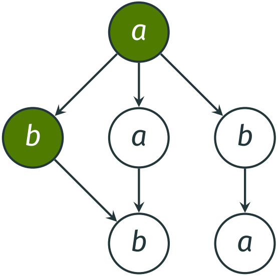
- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?



a

Constrained Topological Sorting

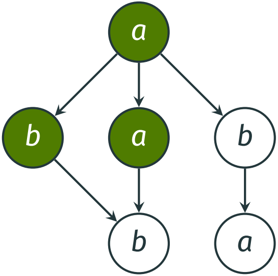
- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?



$a\ b$

Constrained Topological Sorting

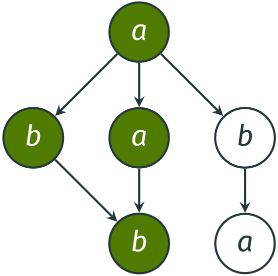
- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?



$a\ b\ a$

Constrained Topological Sorting

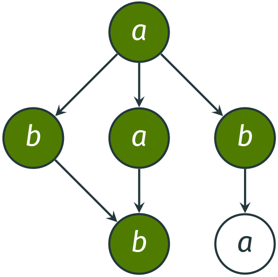
- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?



$a\ b\ a\ b$

Constrained Topological Sorting

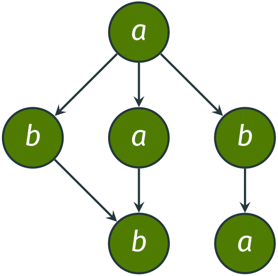
- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?



$a\ b\ a\ b\ b$

Constrained Topological Sorting

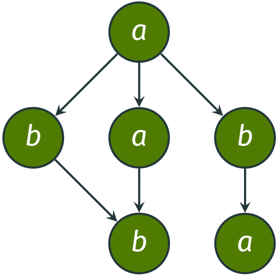
- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?



$a\ b\ a\ b\ b\ a$

Constrained Topological Sorting

- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?

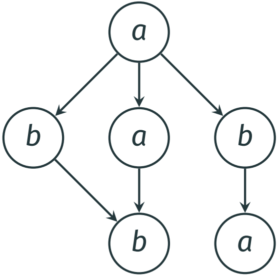


$a\ b\ a\ b\ b\ a$

... not in L!

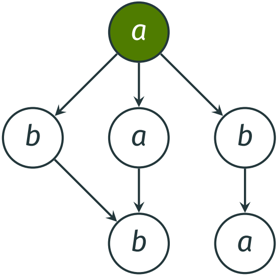
Constrained Topological Sorting

- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
    - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
    - **Output:** is there a topological sort that falls in L?



Constrained Topological Sorting

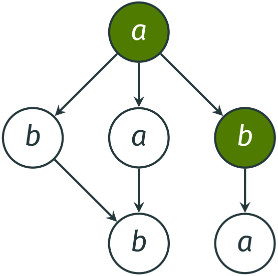
- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?



a

Constrained Topological Sorting

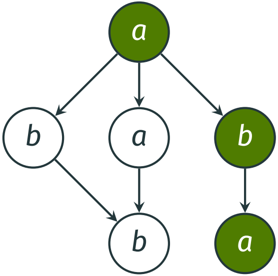
- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?



$a\ b$

Constrained Topological Sorting

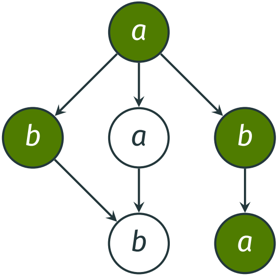
- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?



$a\ b\ a$

Constrained Topological Sorting

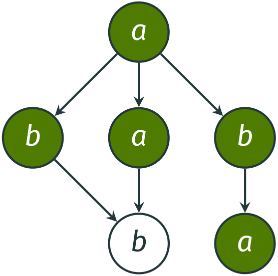
- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?



a b a b

Constrained Topological Sorting

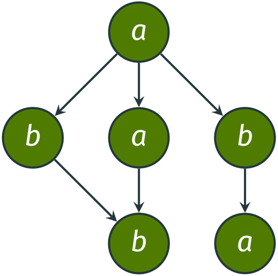
- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?



$a\ b\ a\ b\ a$

Constrained Topological Sorting

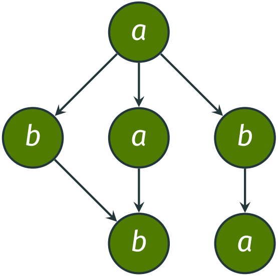
- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?



a b a b a b

Constrained Topological Sorting

- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?

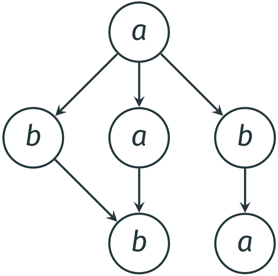


$a\ b\ a\ b\ a\ b$

... in L!

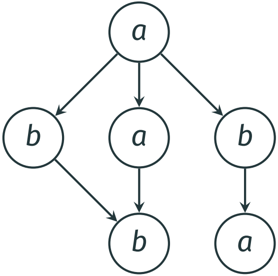
Constrained Topological Sorting

- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?



Constrained Topological Sorting

- Fix an alphabet: e.g., $\Sigma = \{a, b\}$
- Fix a language: e.g., $L = (ab)^*$
- We study constrained topological sorting:
  - **Input:** directed acyclic graph (DAG) with vertices labeled with Σ
  - **Output:** is there a topological sort that falls in L?
- **Question:** when is this problem tractable?

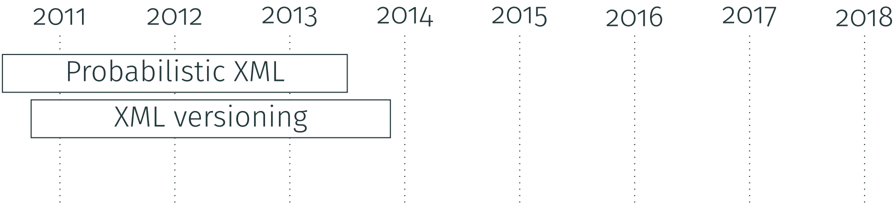


Motivation

- How we really ended up studying this problem:

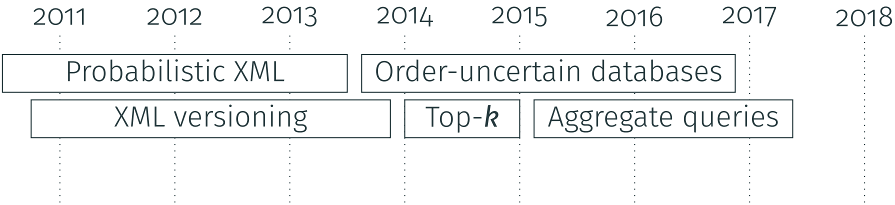
Motivation

- How we **really ended up** studying this problem:



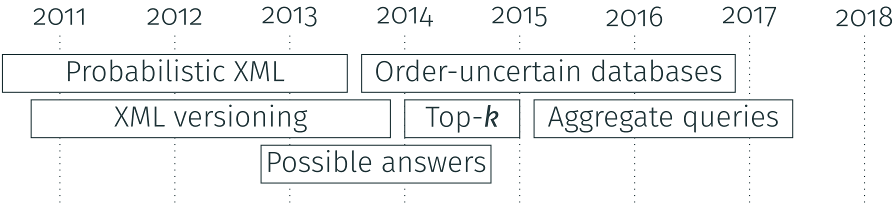
Motivation

- How we **really ended up** studying this problem:



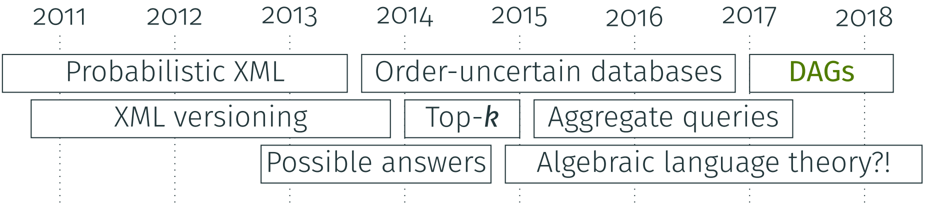
Motivation

- How we **really ended up** studying this problem:



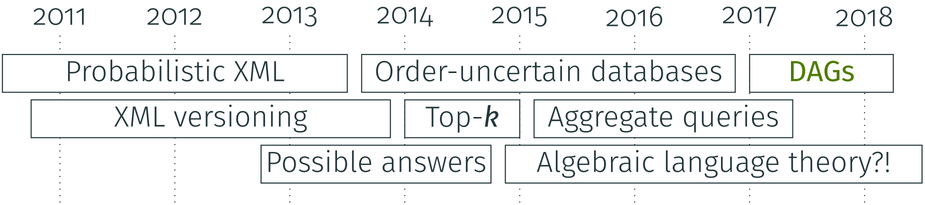
Motivation

- How we **really ended up** studying this problem:



Motivation

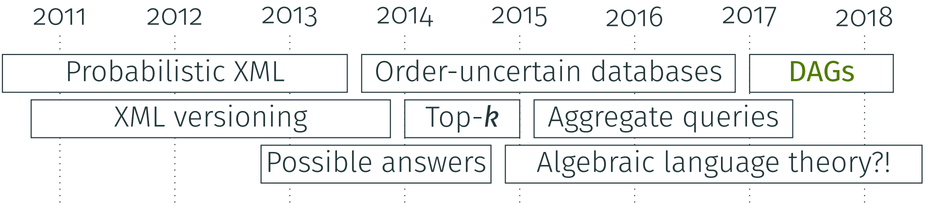
- How we really ended up studying this problem:



- Which a-posteriori motivation did we invent for the problem?

Motivation

- How we really ended up studying this problem:

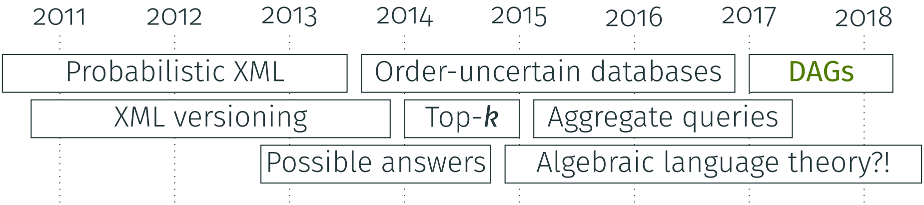


- Which a-posteriori motivation did we invent for the problem?

$\rightarrow$ Scheduling with constraints! $\quad$ $\rightarrow$ Verification for concurrent code!

Motivation

- How we really ended up studying this problem:

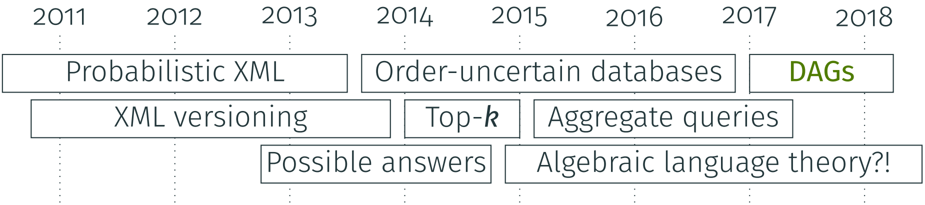


- Which a-posteriori motivation did we invent for the problem?

→ Scheduling with constraints! → Verification for concurrent code!
→ Computational biology! → Blockchain!

Motivation

- How we **really ended up** studying this problem:

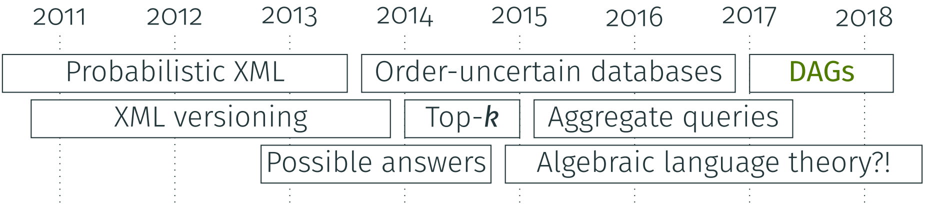


- Which **a-posteriori motivation** did we invent for the problem?

$\rightarrow$ **Scheduling** with constraints! $\rightarrow$ Verification for **concurrent code**!
$\rightarrow$ **Computational biology**! $\rightarrow$ **Blockchain**! (*joke*)

Motivation

- How we **really ended up** studying this problem:



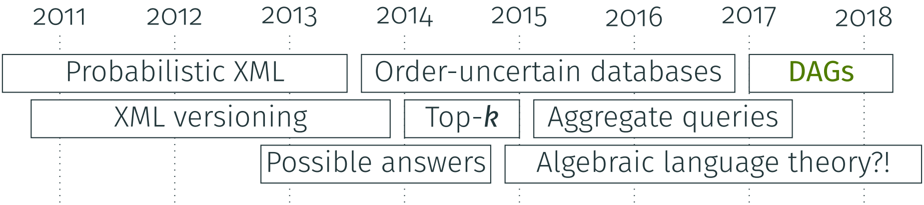
- Which **a-posteriori motivation** did we invent for the problem?

$\rightarrow$ **Scheduling** with constraints! $\rightarrow$ Verification for **concurrent code**!
$\rightarrow$ **Computational biology**! $\rightarrow$ **Blockchain**! (*joke*)

- But why do we **actually care**?

Motivation

- How we **really ended up** studying this problem:



- Which **a-posteriori motivation** did we invent for the problem?

→ **Scheduling** with constraints!
→ Verification for **concurrent code**!
→ **Computational biology**!
→ **Blockchain**! (*joke*)

- But why do we **actually care**?

→ **Natural** problem and apparently **nothing** was known about it!

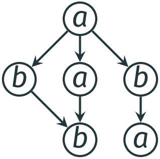
Formal problem statement

- Fix a regular language L on an finite alphabet Σ

Formal problem statement

- Fix a regular language L on an finite alphabet Σ

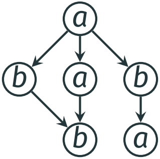
- Constrained topological sort problem CTS(L):
  - Input: a DAG G with vertices labeled by letters of Σ
  - Output: is there a topological sort of G such that the sequence of vertex labels is a word of L



Formal problem statement

- Fix a regular language L on an finite alphabet Σ

- Constrained topological sort problem $\mathsf{CTS}(L)$:
  - Input: a DAG G with vertices labeled by letters of Σ
  - Output: is there a topological sort of G such that the sequence of vertex labels is a word of L

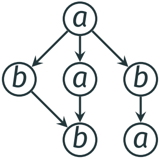


- Special case: the constrained shuffle problem $\mathsf{CSh}(L)$:
  - Input: a set of words $w_1, \ldots, w_n$ of Σ^*
  - Output: is there a shuffle of $w_1, \ldots, w_n$ which is in L

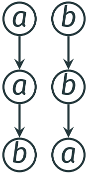
Formal problem statement

- Fix a **regular language** L on an **finite alphabet** Σ

- **Constrained topological sort** problem $\mathsf{CTS}(L)$:
  - **Input:** a **DAG** G with vertices labeled by letters of Σ
  - **Output:** is there a **topological sort** of G such that the sequence of vertex labels is a **word** of L



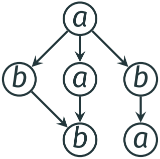
- Special case: the **constrained shuffle problem** $\mathsf{CSh}(L)$:
  - **Input:** a set of **words** $w_1, \ldots, w_n$ of Σ^*
  - **Output:** is there a **shuffle** of $w_1, \ldots, w_n$ which is in L
- This is like CTS but the input DAG is an **union of paths**



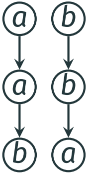
Formal problem statement

- Fix a **regular language** L on an **finite alphabet** Σ

- **Constrained topological sort** problem $\mathsf{CTS}(L)$:
  - **Input:** a **DAG** G with vertices labeled by letters of Σ
  - **Output:** is there a **topological sort** of G such that the sequence of vertex labels is a **word** of L



- Special case: the **constrained shuffle problem** $\mathsf{CSh}(L)$:
  - **Input:** a set of **words** $w_1, \ldots, w_n$ of Σ^*
  - **Output:** is there a **shuffle** of $w_1, \ldots, w_n$ which is in L
- This is like CTS but the input DAG is an **union of paths**



→ **Question:** What is the **complexity** of $\mathsf{CTS}(L)$ and $\mathsf{CSh}(L)$, depending on the fixed language L?

Dichotomy

*For every **regular language L**, exactly one of the following holds:*

- *L has [some nice property] and $\mathsf{CTS}(L)$ is **in NL***
- *L has [some nasty property] and $\mathsf{CTS}(L)$ is **NP-hard***

Dichotomy Conjecture

Conjecture

*For every **regular language** L, exactly one of the following holds:*

- *L has [some nice property] and* $\mathsf{CTS}(L)$ *is* ***in NL***
- *L has [some nasty property] and* $\mathsf{CTS}(L)$ *is* ***NP-hard***

¯_(ツ)_/¯

Dichotomy Conjecture

Conjecture

*For every **regular language L**, exactly one of the following holds:*

- *L has [some nice property] and $\mathsf{CTS}(L)$ is **in NL***
- *L has [some nasty property] and $\mathsf{CTS}(L)$ is **NP-hard***

¯_(ツ)_/¯

Here's what we actually know:

- CTS and CSh are NP-hard for some languages, including $(ab)^*$

Dichotomy Conjecture

Conjecture

*For every **regular language** L, exactly one of the following holds:*

- *L has [some nice property] and* CTS(L) *is **in NL***
- *L has [some nasty property] and* CTS(L) *is **NP-hard***

¯_(ツ)_/¯

Here's what we actually know:

- CTS and CSh are NP-hard for some languages, including $(ab)^*$
- They are in NL for some language families (monomials, groups)

Dichotomy Conjecture

Conjecture

*For every **regular language L**, exactly one of the following holds:*

- *L has [some nice property] and* CTS(L) *is **in NL***
- *L has [some nasty property] and* CTS(L) *is **NP-hard***

¯_(ツ)_/¯

Here's what we actually know:

- CTS and CSh are NP-hard for some languages, including $(ab)^*$
- They are in NL for some language families (monomials, groups)
- Some languages are tractable for seemingly unrelated reasons

Dichotomy Conjecture

Conjecture

*For every **regular language L**, exactly one of the following holds:*

- *L has [some nice property] and* CTS(L) *is **in NL***
- *L has [some nasty property] and* CTS(L) *is **NP-hard***

¯_(ツ)_/¯

Here's what we actually know:

- CTS and CSh are NP-hard for some languages, including $(ab)^*$
- They are in NL for some language families (monomials, groups)
- Some languages are tractable for seemingly unrelated reasons

$\rightarrow$ Very mysterious landscape! (to us)

Hardness Results

Existing Hardness Result

JOURNAL OF COMPUTER AND SYSTEM SCIENCES **28**, 345–358 (1984)

On the Complexity of Iterated Shuffle*

MANFRED K. WARMUTH† AND DAVID HAUSSLER‡

Department of Computer Science,
University of Colorado, Boulder, Colorado 80309

It is demonstrated that the following problems are *NP* complete:

(1) Given words w and $w_1, w_2, \ldots, w_n$, is w in the shuffle of $w_1, w_2, \ldots, w_n$?

Existing Hardness Result

JOURNAL OF COMPUTER AND SYSTEM SCIENCES **28**, 345–358 (1984)

On the Complexity of Iterated Shuffle*

MANFRED K. WARMUTH† AND DAVID HAUSSLER‡

Department of Computer Science,
University of Colorado, Boulder, Colorado 80309

It is demonstrated that the following problems are *NP* complete:

(1) Given words w and $w_1, w_2, \ldots, w_n$, is w in the shuffle of $w_1, w_2, \ldots, w_n$?

… but the target is a **word** which is **provided as input**!

Existing Hardness Result

JOURNAL OF COMPUTER AND SYSTEM SCIENCES **28**, 345–358 (1984)

On the Complexity of Iterated Shuffle*

MANFRED K. WARMUTH† AND DAVID HAUSSLER‡

Department of Computer Science,
University of Colorado, Boulder, Colorado 80309

It is demonstrated that the following problems are *NP* complete:

(1) Given words w and $w_1, w_2, \ldots, w_n$, is w in the shuffle of $w_1, w_2, \ldots, w_n$?

… but the target is a **word** which is **provided as input**!

$\rightarrow$ Does not directly apply for us, because we **fix** the target language

Hardness for CTS

- We can reduce their problem to CSh for the language $(aA + bB)^*$
- To determine if the shuffle of aab and bb contains $ababb$ …

Hardness for CTS

- We can reduce their problem to CSh for the language $(aA + bB)^*$
- To determine if the shuffle of aab and bb contains $ababb$ …
  solve the CSh-problem for aab and bb and $ABABB$

Hardness for CTS

- We can reduce their problem to CSh for the language $(aA + bB)^*$
- To determine if the shuffle of aab and bb contains $ababb$ ...
  solve the CSh-problem for aab and bb and $ABABB$

$\rightarrow$ CSh($(aA + bB)^*$) is NP-hard and the same holds for CTS

Hardness for CTS

- We can reduce their problem to CSh for the language $(aA + bB)^*$
- To determine if the shuffle of aab and bb contains $ababb$ ...
  solve the CSh-problem for aab and bb and $ABABB$

$\rightarrow$ CSh($(aA + bB)^*$) is NP-hard and the same holds for CTS

- Similar technique: CSh($(ab)^*$) is NP-hard

Hardness for CTS

- We can reduce their problem to CSh for the language $(aA + bB)^*$
- To determine if the shuffle of aab and bb contains $ababb$ ...
  solve the CSh-problem for aab and bb and $ABABB$

$\rightarrow$ CSh($(aA + bB)^*$) is NP-hard and the same holds for CTS

- Similar technique: CSh($(ab)^*$) is NP-hard

- Custom reduction technique to show NP-hardness

The reduction technique

- Say we want to solve CTS for $(ab)^*$ (NP-hard)

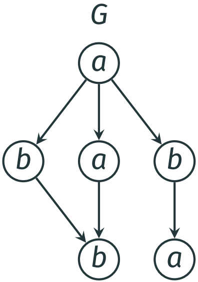
The reduction technique

- Say we want to solve CTS for $(ab)^*$ (NP-hard)
- Say we know how to solve CTS for $(abc)^*$

The reduction technique

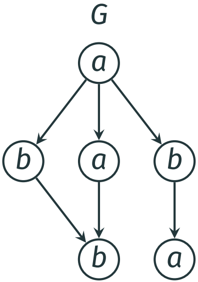
- Say we want to solve CTS for $(ab)^*$ (NP-hard)
- Say we know how to solve CTS for $(abc)^*$
- Take an instance G for $(ab)^*$, with $2n$ vertices

The reduction technique



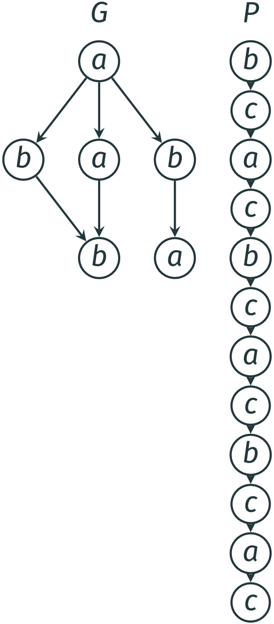
- Say we want to solve CTS for $(ab)^*$ (NP-hard)
- Say we know how to solve CTS for $(abc)^*$
- Take an instance G for $(ab)^*$, with $2n$ vertices

The reduction technique



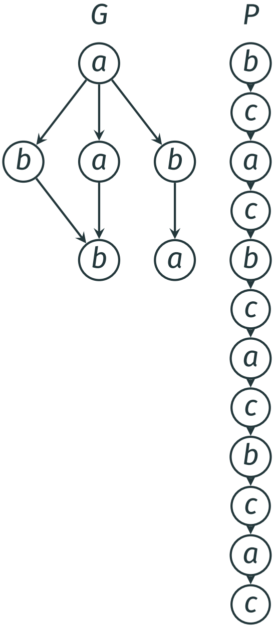
- Say we want to solve CTS for $(ab)^*$ (NP-hard)
- Say we know how to solve CTS for $(abc)^*$
- Take an instance G for $(ab)^*$, with $2n$ vertices
- Add the path P: $(bcac)^n$

The reduction technique



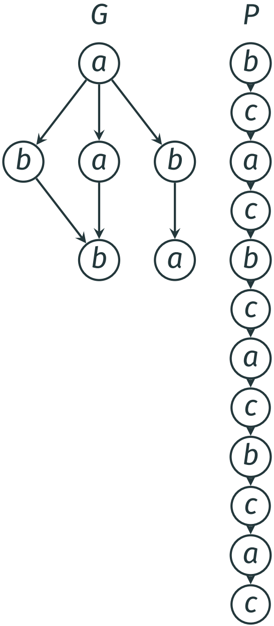
- Say we want to solve CTS for $(ab)^*$ (NP-hard)
- Say we know how to solve CTS for $(abc)^*$
- Take an instance G for $(ab)^*$, with $2n$ vertices
- Add the path P: $(bcac)^n$

The reduction technique



- Say we want to solve **CTS** for $(ab)^*$ (NP-hard)
- Say we know how to solve **CTS** for $(abc)^*$
- Take an instance G for $(ab)^*$, with $2n$ vertices
- Add the path P: $(bcac)^n$
- A topsort of $G \cup P$ achieving $(abc)^*$ gives a topsort of G achieving $(ab)^*$

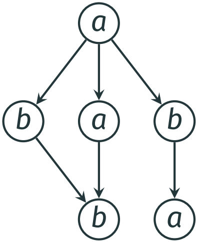
The reduction technique



- Say we want to solve **CTS** for $(ab)^*$ (NP-hard)
- Say we know how to solve **CTS** for $(abc)^*$
- Take an instance G for $(ab)^*$, with $2n$ vertices
- Add the path P: $(bcac)^n$
- A topsort of $G \cup P$ achieving $(abc)^*$ gives a topsort of G achieving $(ab)^*$
- Conversely, any topsort of G achieving $(ab)^*$ extends to a topsort of $G + P$ achieving $(abc)^*$

The reduction technique

G



P

- Say we want to solve **CTS** for $(ab)^*$ (NP-hard)
- Say we know how to solve **CTS** for $(abc)^*$
- Take an instance G for $(ab)^*$, with $2n$ vertices
- Add the path P: $(bcac)^n$
- A topsort of $G \cup P$ achieving $(abc)^*$ gives a topsort of G achieving $(ab)^*$
- Conversely, any topsort of G achieving $(ab)^*$ extends to a topsort of $G + P$ achieving $(abc)^*$
- Hence, **CTS**$((abc)^*)$ is NP-hard

Formalizing the reduction

Definition

A language L shuffle-reduces to a language L' if, given any n in unary, we can compute in PTIME a word P_i having the following property:

Formalizing the reduction

Definition

A language L shuffle-reduces to a language L' if, given any n in unary, we can compute in PTIME a word P_i having the following property: for any word w of length n, we have $w \in L$ iff the shuffle of w and P_i contains a word of L'.

Formalizing the reduction

Definition

A language L shuffle-reduces to a language L' if, given any n in unary, we can compute in PTIME a word P_i having the following property: for any word w of length n, we have $w \in L$ iff the shuffle of w and P_i contains a word of L'.

Theorem

If L shuffle-reduces to L' then:

- $\mathsf{CSh}(L)$ *reduces in PTIME to* $\mathsf{CSh}(L')$
- $\mathsf{CTS}(L)$ *reduces in PTIME to* $\mathsf{CTS}(L')$

Other hard languages

- The reduction shows hardness for:
  - $(ab + b)^*$ (also simpler argument)
  - $(aa + bb)^*$ with $P_{2i} = (ab)^i$
  - u^* if u contains two different letters

Other hard languages

- The reduction shows hardness for:
  - $(ab + b)^*$ (also simpler argument)
  - $(aa + bb)^*$ with $P_{2i} = (ab)^i$
  - u^* if u contains two different letters

- Conjecture: if F is finite then $\mathrm{CTS}(F^*)$ is NP-hard unless it contains a power of each of its letters

Other hard languages

- The reduction shows hardness for:
  - $(ab + b)^*$ (also simpler argument)
  - $(aa + bb)^*$ with $P_{2i} = (ab)^i$
  - u^* if u contains two different letters

- Conjecture: if F is finite then $\mathsf{CTS}(F^*)$ is NP-hard unless it contains a power of each of its letters
  - Idea: reason about consumption rates of letters?

Other hard languages

- The reduction shows hardness for:
  - $(ab + b)^*$ (also simpler argument)
  - $(aa + bb)^*$ with $P_{2i} = (ab)^i$
  - u^* if u contains two different letters
- Conjecture: if F is finite then $\mathsf{CTS}(F^*)$ is NP-hard unless it contains a power of each of its letters
  - Idea: reason about consumption rates of letters?
  - Not even complete for F^* languages, as $(aa + bb)^*$ is NP-hard

Tractability Results

Tractability for Monomials

- **Monomial:** language of the form $A_1^* \, a_1 \, A_2^* \, a_2 \, \cdots \, A_n^* \, a_n \, A_{n+1}^*$ where $a_1, \ldots, a_n \in \Sigma$ and $A_1, \ldots, A_{n+1} \subseteq \Sigma$
- **Union of monomials:** union of finitely many such languages

Tractability for Monomials

- **Monomial:** language of the form $A_1^* \, a_1 \, A_2^* \, a_2 \, \cdots \, A_n^* \, a_n \, A_{n+1}^*$ where $a_1, \ldots, a_n \in \Sigma$ and $A_1, \ldots, A_{n+1} \subseteq \Sigma$
- **Union of monomials:** union of finitely many such languages
  - **Example:** pattern matching $\Sigma^* \, \mathit{word}1 \, \Sigma^* + \Sigma^* \, \mathit{word}2 \, \Sigma^*$

Tractability for Monomials

- **Monomial:** language of the form $A_1^* \, a_1 \, A_2^* \, a_2 \ \cdots \ A_n^* \, a_n \, A_{n+1}^*$ where $a_1, \ldots, a_n \in \Sigma$ and $A_1, \ldots, A_{n+1} \subseteq \Sigma$
- **Union of monomials:** union of finitely many such languages
  - **Example:** pattern matching $\Sigma^* \, \mathit{word}1 \, \Sigma^* + \Sigma^* \, \mathit{word}2 \, \Sigma^*$
  - **Logical interpretation:** languages definable in $\Sigma_2[<]$

Tractability for Monomials

- **Monomial:** language of the form $A_1^* \, a_1 \, A_2^* \, a_2 \, \cdots \, A_n^* \, a_n \, A_{n+1}^*$ where $a_1, \ldots, a_n \in \Sigma$ and $A_1, \ldots, A_{n+1} \subseteq \Sigma$
- **Union of monomials:** union of finitely many such languages
  - **Example:** pattern matching $\Sigma^* \, \mathit{word}1 \, \Sigma^* + \Sigma^* \, \mathit{word}2 \, \Sigma^*$
  - **Logical interpretation:** languages definable in $\Sigma_2[<]$

Theorem

For any union of monomials L*, the problem* $\mathsf{CTS}(L)$ *is* ***in NL***

Proof Idea for Monomials

- Tractable languages are clearly closed under union
  so it suffices to consider a monomial: $A_1^* \, a_1 \, A_2^* \, a_2 \, \cdots \, A_n^* \, a_n \, A_{n+1}^*$
  where $a_1, \ldots, a_n \in \Sigma$ and $A_1, \ldots, A_{n+1} \subseteq \Sigma$

Proof Idea for Monomials

- Tractable languages are clearly closed under union
  so it suffices to consider a monomial: $A_1^* \, a_1 \, A_2^* \, a_2 \, \cdots \, A_n^* \, a_n \, A_{n+1}^*$
  where $a_1, \ldots, a_n \in \Sigma$ and $A_1, \ldots, A_{n+1} \subseteq \Sigma$
- We can guess the positions of the individual a_i

Proof Idea for Monomials

- Tractable languages are clearly **closed under union** so it suffices to consider a monomial: $A_1^* \, a_1 \, A_2^* \, a_2 \, \cdots \, A_n^* \, a_n \, A_{n+1}^*$ where $a_1, \ldots, a_n \in \Sigma$ and $A_1, \ldots, A_{n+1} \subseteq \Sigma$
- We can **guess** the positions of the individual a_i
- Check that the other vertices **can fit** in the A_i^* (uses NL = co-NL)

Proof Idea for Monomials

- Tractable languages are clearly **closed under union** so it suffices to consider a monomial: $A_1^* \, a_1 \, A_2^* \, a_2 \, \cdots \, A_n^* \, a_n \, A_{n+1}^*$ where $a_1, \ldots, a_n \in \Sigma$ and $A_1, \ldots, A_{n+1} \subseteq \Sigma$
- We can **guess** the positions of the individual a_i
- Check that the other vertices **can fit** in the A_i^* (uses NL = co-NL)
  - Check that the **descendants** of a_n are all in A_{n+1}

Proof Idea for Monomials

- Tractable languages are clearly closed under union so it suffices to consider a monomial: $A_1^* \, a_1 \, A_2^* \, a_2 \, \cdots \, A_n^* \, a_n \, A_{n+1}^*$ where $a_1, \ldots, a_n \in \Sigma$ and $A_1, \ldots, A_{n+1} \subseteq \Sigma$
- We can guess the positions of the individual a_i
- Check that the other vertices can fit in the A_i^* (uses NL = co-NL)
  - Check that the descendants of a_n are all in A_{n+1}
  - Find the vertices that must be enumerated before a_n

Proof Idea for Monomials

- Tractable languages are clearly **closed under union**
  so it suffices to consider a monomial: $A_1^* \, a_1 \, A_2^* \, a_2 \, \cdots \, A_n^* \, a_n \, A_{n+1}^*$
  where $a_1, \ldots, a_n \in \Sigma$ and $A_1, \ldots, A_{n+1} \subseteq \Sigma$
- We can **guess** the positions of the individual a_i
- Check that the other vertices **can fit** in the A_i^* (uses NL = co-NL)
  - Check that the **descendants** of a_n are all in A_{n+1}
  - Find the vertices that **must** be enumerated before a_n
    - The **ancestors** of the a_i

Proof Idea for Monomials

- Tractable languages are clearly **closed under union** so it suffices to consider a monomial: $A_1^* \, a_1 \, A_2^* \, a_2 \, \cdots \, A_n^* \, a_n \, A_{n+1}^*$ where $a_1, \ldots, a_n \in \Sigma$ and $A_1, \ldots, A_{n+1} \subseteq \Sigma$
- We can **guess** the positions of the individual a_i
- Check that the other vertices **can fit** in the A_i^* (uses NL = co-NL)
  - Check that the **descendants** of a_n are all in A_{n+1}
  - Find the vertices that **must** be enumerated before a_n
    - The **ancestors** of the a_i
    - The **ancestors** of vertices with a letter **not in A_{n+1}**

Proof Idea for Monomials

- Tractable languages are clearly **closed under union** so it suffices to consider a monomial: $A_1^* \, a_1 \, A_2^* \, a_2 \, \cdots \, A_n^* \, a_n \, A_{n+1}^*$ where $a_1, \ldots, a_n \in \Sigma$ and $A_1, \ldots, A_{n+1} \subseteq \Sigma$
- We can **guess** the positions of the individual a_i
- Check that the other vertices **can fit** in the A_i^* (uses NL = co-NL)
  - Check that the **descendants** of a_n are all in A_{n+1}
  - Find the vertices that **must** be enumerated before a_n
    - The **ancestors** of the a_i
    - The **ancestors** of vertices with a letter **not in A_{n+1}**
  - Inductively solve the problem for these vertices and $A_1^* \, a_1 \, A_2^* \, a_2 \, \cdots \, A_n^*$

The Algebraic Approach

Can we just study algebraically the tractable languages?

The Algebraic Approach Fails

Can we just study algebraically the tractable languages? Not really...

The Algebraic Approach Fails

Can we just study algebraically the tractable languages? Not really...

- Not closed under intersection
- Not closed under complement
- Not closed under inverse morphism
- Not closed under concatenation (not in paper, only known for CTS)
- For CSh: not closed under quotient

¯_(ツ)_/¯

Side Remark: CTS and CSh are Different

Consider the language $L = b\Sigma^* + aa\Sigma^* + (ab)^*$

Side Remark: CTS **and** CSh **are Different**

Consider the language $L = b\Sigma^* + aa\Sigma^* + (ab)^*$

- CTS(L) is NP-hard because $(ab)^{-1}L = (ab)^*$

Side Remark: CTS **and** CSh **are Different**

Consider the language $L = b\Sigma^* + aa\Sigma^* + (ab)^*$

- CTS(L) is NP-hard because $(ab)^{-1}L = (ab)^*$
- CSh(L) is in NL: trivial if there is more than one word

Side Remark: CTS **and** CSh **are Different**

Consider the language $L = b\Sigma^* + aa\Sigma^* + (ab)^*$

- CTS(L) is NP-hard because $(ab)^{-1}L = (ab)^*$
- CSh(L) is in NL: trivial if there is more than one word

Hence, some languages are tractable for CSh and hard for CTS

Tractability Based on Width

- CSh(L) is in NL for any regular language L if we assume that there are at most k input words $w_1, \ldots, w_k$ for a constant $k \in \mathbb{N}$

Tractability Based on Width

- CSh(L) is in NL for any regular language L if we assume that there are at most k input words $w_1, \ldots, w_k$ for a constant $k \in \mathbb{N}$
  - $\rightarrow$ Need k counters to remember the current position in each word, plus automaton state

Tractability Based on Width

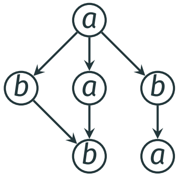
- CSh(L) is in NL for any regular language L if we assume that there are at most k input words $w_1, \ldots, w_k$ for a constant $k \in \mathbb{N}$
  - $\rightarrow$ Need k counters to remember the current position in each word, plus automaton state

- CTS(L) is in in NL for any regular language L if the input DAG G has width $\leq k$ for constant $k \in \mathbb{N}$

Tractability Based on Width

- CSh(L) is **in NL** for **any regular language L** if we assume that there are **at most k input words** $w_1, \ldots, w_k$ for a constant $k \in \mathbb{N}$
  - $\rightarrow$ Need **k counters** to remember the current position in each word, plus **automaton state**

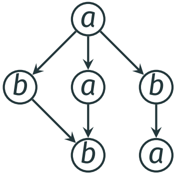
- CTS(L) is in **in NL** for **any regular language L** if the input DAG G has **width $\leq k$** for constant $k \in \mathbb{N}$



Tractability Based on Width

- CSh(L) is in NL for any regular language L if we assume that there are at most k input words $w_1, \ldots, w_k$ for a constant $k \in \mathbb{N}$
  - $\rightarrow$ Need k counters to remember the current position in each word, plus automaton state

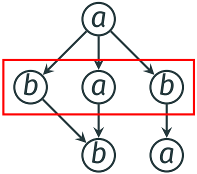
- CTS(L) is in in NL for any regular language L if the input DAG G has width $\leq k$ for constant $k \in \mathbb{N}$
  - Width: size of the largest antichain (subset of pairwise incomparable vertices)



Tractability Based on Width

- CSh(L) is in NL for any regular language L if we assume that there are at most k input words $w_1, \ldots, w_k$ for a constant $k \in \mathbb{N}$
  - $\rightarrow$ Need k counters to remember the current position in each word, plus automaton state

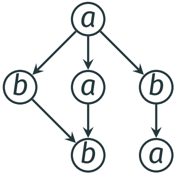
- CTS(L) is in in NL for any regular language L if the input DAG G has width $\leq k$ for constant $k \in \mathbb{N}$
  - Width: size of the largest antichain (subset of pairwise incomparable vertices)



Tractability Based on Width

- CSh(L) is in NL for any regular language L if we assume that there are at most k input words $w_1, \ldots, w_k$ for a constant $k \in \mathbb{N}$
  - $\rightarrow$ Need k counters to remember the current position in each word, plus automaton state

- CTS(L) is in in NL for any regular language L if the input DAG G has width $\leq k$ for constant $k \in \mathbb{N}$
  - Width: size of the largest antichain (subset of pairwise incomparable vertices)
  - $\rightarrow$ Partition G in k chains (Dilworth's theorem), and conclude by NL algorithm



Tractability Based on Width

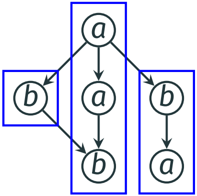
- CSh(L) is in NL for any regular language L if we assume that there are at most k input words $w_1, \ldots, w_k$ for a constant $k \in \mathbb{N}$

  $\rightarrow$ Need k counters to remember the current position in each word, plus automaton state

- CTS(L) is in in NL for any regular language L if the input DAG G has width $\leq k$ for constant $k \in \mathbb{N}$

  - Width: size of the largest antichain (subset of pairwise incomparable vertices)

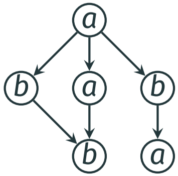
  $\rightarrow$ Partition G in k chains (Dilworth's theorem), and conclude by NL algorithm



Tractability Based on Width

- CSh(L) is in NL for any regular language L if we assume that there are at most k input words $w_1, \ldots, w_k$ for a constant $k \in \mathbb{N}$
  - $\rightarrow$ Need k counters to remember the current position in each word, plus automaton state

- CTS(L) is in in NL for any regular language L if the input DAG G has width $\leq k$ for constant $k \in \mathbb{N}$
  - Width: size of the largest antichain (subset of pairwise incomparable vertices)
  - $\rightarrow$ Partition G in k chains (Dilworth's theorem), and conclude by NL algorithm



$\rightarrow$ These results are making an additional assumption, but...

Tractability Based on Width (2)

- Fix $\Sigma = \{a, b\}$, take any regular language L and constant $k \in \mathbb{N}$, we know that CTS is in NL for $L + \Sigma^*(a^k + b^k)\Sigma^*$

Tractability Based on Width (2)

- Fix $\Sigma = \{a, b\}$, take any regular language L and constant $k \in \mathbb{N}$, we know that CTS is in NL for $L + \Sigma^*(a^k + b^k)\Sigma^*$
  - If the input DAG has width $< 2k$, use the result for bounded width

Tractability Based on Width (2)

- Fix $\Sigma = \{a, b\}$, take any regular language L and constant $k \in \mathbb{N}$, we know that CTS is in NL for $L + \Sigma^*(a^k + b^k)\Sigma^*$
  - If the input DAG has width $< 2k$, use the result for bounded width
  - Otherwise we can achieve a^k or b^k with a large antichain

Tractability Based on Width (2)

- Fix $\Sigma = \{a, b\}$, take any regular language L and constant $k \in \mathbb{N}$, we know that CTS is in NL for $L + \Sigma^*(a^k + b^k)\Sigma^*$
  - If the input DAG has width $< 2k$, use the result for bounded width
  - Otherwise we can achieve a^k or b^k with a large antichain

- A similar technique shows that $(ab)^* + \Sigma^* aa \Sigma^*$ is tractable

$\rightarrow$ Does it suffice to bound the width of all letters but one?

Tractability Based on Width (2)

- Fix $\Sigma = \{a, b\}$, take any regular language L and constant $k \in \mathbb{N}$, we know that CTS is in NL for $L + \Sigma^*(a^k + b^k)\Sigma^*$
  - If the input DAG has width $< 2k$, use the result for bounded width
  - Otherwise we can achieve a^k or b^k with a large antichain

- A similar technique shows that $(ab)^* + \Sigma^* aa \Sigma^*$ is tractable

$\rightarrow$ Does it suffice to bound the width of all letters but one?

$\rightarrow$ Unknown for $L + \Sigma^* a^k \Sigma^*$ with arbitrary L and $k > 2$! ¯_(ツ)_/¯

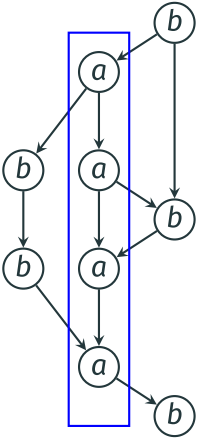
An Annoying Open Problem

- Fix the alphabet $\Sigma = \{a, b\}$

An Annoying Open Problem

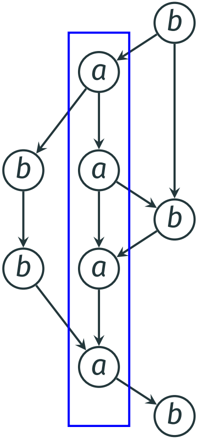
- Fix the alphabet $\Sigma = \{a, b\}$
- Assume that the input DAG has a-width 1, i.e., there is a total order on the a-labeled elements

An Annoying Open Problem



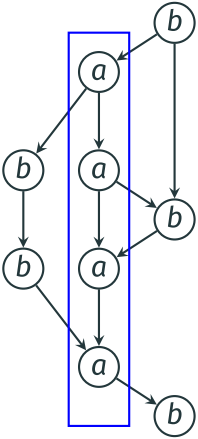
- Fix the alphabet $\Sigma = \{a, b\}$
- Assume that the input DAG has a-width 1, i.e., there is a total order on the a-labeled elements

An Annoying Open Problem



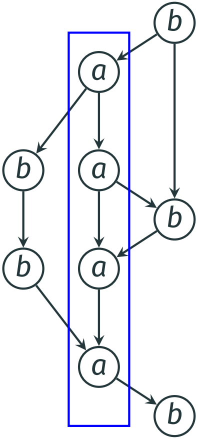
- Fix the alphabet $\Sigma = \{a, b\}$
- Assume that the input DAG has **a-width 1**, i.e., there is a **total order** on the a-labeled elements
- Easy **greedy PTIME algorithm** for $\mathrm{CTS}((ab)^*)$:

An Annoying Open Problem



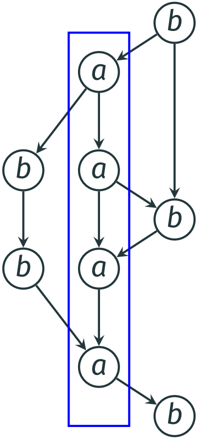
- Fix the alphabet $\Sigma = \{a, b\}$
- Assume that the input DAG has a-width 1, i.e., there is a total order on the a-labeled elements
- Easy greedy PTIME algorithm for $\mathrm{CTS}((ab)^*)$:
  - If we want an a, take the next one (no choice)

An Annoying Open Problem



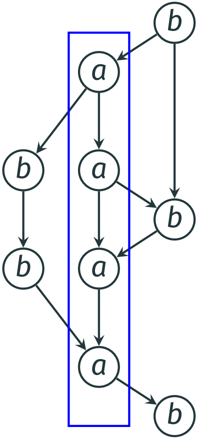
- Fix the alphabet $\Sigma = \{a, b\}$
- Assume that the input DAG has a-width 1, i.e., there is a total order on the a-labeled elements
- Easy greedy PTIME algorithm for $\mathrm{CTS}((ab)^*)$:
  - If we want an a, take the next one (no choice)
  - If we want a b, take an available b-vertex whose first a-descendant is as high as possible (idea: consume the most blocking b's)

An Annoying Open Problem



- Fix the alphabet $\Sigma = \{a, b\}$
- Assume that the input DAG has a-width 1, i.e., there is a total order on the a-labeled elements
- Easy greedy PTIME algorithm for $\mathsf{CTS}((ab)^*)$:
  - If we want an a, take the next one (no choice)
  - If we want a b, take an available b-vertex whose first a-descendant is as high as possible (idea: consume the most blocking b's)
- Should generalizes to $\mathsf{CTS}(L)$ for any L...

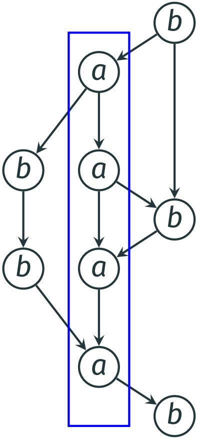
An Annoying Open Problem



- Fix the alphabet $\Sigma = \{a, b\}$
- Assume that the input DAG has *a*-width 1, i.e., there is a total order on the *a*-labeled elements
- Easy greedy PTIME algorithm for CTS($(ab)^*$):
  - If we want an *a*, take the next one (no choice)
  - If we want a *b*, take an available *b*-vertex whose first *a*-descendant is as high as possible (idea: consume the most blocking *b*'s)
- Should generalizes to CTS(L) for any L... right?!

¯_(ツ)_/¯

An Annoying Open Problem



- Fix the alphabet $\Sigma = \{a, b\}$
- Assume that the input DAG has a-width 1, i.e., there is a total order on the a-labeled elements
- Easy greedy PTIME algorithm for CTS($(ab)^*$):
  - If we want an a, take the next one (no choice)
  - If we want a b, take an available b-vertex whose first a-descendant is as high as possible (idea: consume the most blocking b's)
- Should generalizes to CTS(L) for any L... right?!

¯_(ツ)_/¯

Open problem

Fix $\Sigma = \{a, b\}$ and an arbitrary regular language L. Given a DAG without two incomparable a's, can you solve CTS(L)?

Tractability Based on the Structure of Groups

- Group language: the underlying monoid is a finite group
  - $\rightarrow$ Automata where each letter acts bijectively

Tractability Based on the Structure of Groups

- **Group language:** the underlying monoid is a finite group
  - $\rightarrow$ Automata where each letter acts bijectively

- **District group monomial:** language $G_1\ a_1\ \cdots\ G_n\ a_n\ G_{n+1}$ where $a_1, \ldots, a_n \in \Sigma$ and $G_1, \ldots, G_n$ are group languages on subsets of the alphabet Σ

Tractability Based on the Structure of Groups

- **Group language:** the underlying monoid is a **finite group**
  - $\rightarrow$ Automata where each letter **acts bijectively**
- **District group monomial:** language $G_1\, a_1 \,\cdots\, G_n\, a_n\, G_{n+1}$ where $a_1, \ldots, a_n \in \Sigma$ and $G_1, \ldots, G_n$ are **group languages** on **subsets** of the alphabet Σ

Theorem

For any union L of district group monomials, $\mathsf{CSh}(L)$ is in NL

Tractability Based on the Structure of Groups

- Group language: the underlying monoid is a finite group
  - → Automata where each letter acts bijectively

- District group monomial: language $G_1 \, a_1 \, \cdots \, G_n \, a_n \, G_{n+1}$ where $a_1, \ldots, a_n \in \Sigma$ and $G_1, \ldots, G_n$ are group languages on subsets of the alphabet Σ

Theorem

For any union L of district group monomials, CSh(L) is in NL

→ Only for CSh; complexity for CTS is unknown! ¯_(ツ)_/¯

Proof Structure for Groups

- By far the most technical proof of the paper

Proof Structure for Groups

- By far the most technical proof of the paper
- From district group monomials to group languages:
  - Guess the vertices where the a_i are mapped
  - Guess (in succession) how each input word is split

Proof Structure for Groups

- By far the most technical proof of the paper
- From district group monomials to group languages:
  - Guess the vertices where the a_i are mapped
  - Guess (in succession) how each input word is split
- For groups: distinguish the rare and frequent letters of Σ

Proof Structure for Groups

- By far the most technical proof of the paper
- From district group monomials to group languages:
  - Guess the vertices where the a_i are mapped
  - Guess (in succession) how each input word is split
- For groups: distinguish the rare and frequent letters of Σ
  - Rare letters are in constantly many strings: NL algorithm on them
  - Frequent letters are in enough strings to generate anything

Proof Structure for Groups

- By far the most technical proof of the paper
- From district group monomials to group languages:
  - Guess the vertices where the a_i are mapped
  - Guess (in succession) how each input word is split
- For groups: distinguish the rare and frequent letters of Σ
  - Rare letters are in constantly many strings: NL algorithm on them
  - Frequent letters are in enough strings to generate anything

$\rightarrow$ Key (CSh): find an antichain with all frequent letters many times

Proof Structure for Groups

- By far the most technical proof of the paper
- From district group monomials to group languages:
  - Guess the vertices where the a_i are mapped
  - Guess (in succession) how each input word is split
- For groups: distinguish the rare and frequent letters of Σ
  - Rare letters are in constantly many strings: NL algorithm on them
  - Frequent letters are in enough strings to generate anything

  $\rightarrow$ Key (CSh): find an antichain with all frequent letters many times
- Two main challenges:
  - Even on frequent letters, we can only achieve all group elements up to commutative information

    $\rightarrow$ E.g., in a group $G \times (\mathbb{Z}/2\mathbb{Z})$ with generators of the form $(g_i, 1)$, a large odd number of generators will never achieve $(g, 0)$

Proof Structure for Groups

- By far the most technical proof of the paper
- From district group monomials to group languages:
  - Guess the vertices where the a_i are mapped
  - Guess (in succession) how each input word is split
- For groups: distinguish the rare and frequent letters of Σ
  - Rare letters are in constantly many strings: NL algorithm on them
  - Frequent letters are in enough strings to generate anything
  - $\rightarrow$ Key (CSh): find an antichain with all frequent letters many times
- Two main challenges:
  - Even on frequent letters, we can only achieve all group elements up to commutative information
    - $\rightarrow$ E.g., in a group $G \times (\mathbb{Z}/2\mathbb{Z})$ with generators of the form $(g_i, 1)$, a large odd number of generators will never achieve $(g, 0)$
    - $\rightarrow$ Antichain lemma: Constantly many elements suffice to achieve anything in the spanned subgroup up to "commutative information"

Proof Structure for Groups

- By far the most technical proof of the paper
- From district group monomials to group languages:
  - Guess the vertices where the a_i are mapped
  - Guess (in succession) how each input word is split
- For groups: distinguish the rare and frequent letters of Σ
  - Rare letters are in constantly many strings: NL algorithm on them
  - Frequent letters are in enough strings to generate anything

  $\rightarrow$ Key (CSh): find an antichain with all frequent letters many times
- Two main challenges:
  - Even on frequent letters, we can only achieve all group elements up to commutative information
    - $\rightarrow$ E.g., in a group $G \times (\mathbb{Z}/2\mathbb{Z})$ with generators of the form $(g_i, 1)$, a large odd number of generators will never achieve $(g, 0)$
    - $\rightarrow$ Antichain lemma: Constantly many elements suffice to achieve anything in the spanned subgroup up to "commutative information"
  - When doing the NL algorithm on rare letters, constant bound on the number of frequent letter insertions

Tractability Based on All Sorts of Strange Reasons

- $(aa + b)^*$ is in NL for CSh:

Tractability Based on All Sorts of Strange Reasons

- $(aa+b)^*$ is in NL for CSh:
  - Ad-hoc greedy algorithm: consume string with most odd a blocks

Tractability Based on All Sorts of Strange Reasons

- $(aa + b)^*$ is in NL for CSh:
  - Ad-hoc greedy algorithm: consume string with most odd a blocks
  - Complexity open for CTS! ¯_(ツ)_/¯

Tractability Based on All Sorts of Strange Reasons

- $(aa + b)^*$ is in NL for CSh:
  - Ad-hoc greedy algorithm: consume string with most odd a blocks
  - Complexity open for CTS! ¯_(ツ)_/¯
  - Complexity open for $(a^k + b)^*$ for $k > 2$! ¯_(ツ)_/¯

Tractability Based on All Sorts of Strange Reasons

- $(aa + b)^*$ is in NL for CSh:
  - Ad-hoc greedy algorithm: consume string with most odd a blocks
  - Complexity open for CTS! ¯_(ツ)_/¯
  - Complexity open for $(a^k + b)^*$ for $k > 2$! ¯_(ツ)_/¯
  - What about similar languages like $(aa + bb + ab)^*$? ¯_(ツ)_/¯

Tractability Based on All Sorts of Strange Reasons

- $(aa + b)^*$ is in NL for CSh:
  - Ad-hoc greedy algorithm: consume string with most odd a blocks
  - Complexity open for CTS! ¯_(ツ)_/¯
  - Complexity open for $(a^k + b)^*$ for $k > 2$! ¯_(ツ)_/¯
  - What about similar languages like $(aa + bb + ab)^*$? ¯_(ツ)_/¯

- $(caa)^*d(cbb)^*d\Sigma^* + \Sigma^*cc\Sigma^*$ is in NL for CSh but NP-hard for CTS
  - Tractability argument: another ad hoc greedy algorithm
  - Hardness argument: from k-clique encoded to a bipartite graph

A Kind of Dichotomy

Prelude to the Kind of Dichotomy

- We were aiming for a dichotomy, but... ¯_(ツ)_/¯

Prelude to the Kind of Dichotomy

- We were aiming for a dichotomy, but... ¯_(ツ)_/¯
- Let's try to make the problem simpler

Prelude to the Kind of Dichotomy

- We were aiming for a dichotomy, but... ¯_(ツ)_/¯
- Let's try to make the problem simpler
- Idea: If we don't fix a target language but a language "family" then we can hope for a coarser dichotomy
- We can restrict to "families" closed under algebraic operations and go back to the algebraic approach

A Kind of Dichotomy

- Fix a **semiautomaton** $S = (Q, \Sigma, \delta)$ with Q the set of **states**, with Σ a **finite alphabet**, and with δ the **transitions**.

A Kind of Dichotomy

- Fix a semiautomaton $S = (Q, \Sigma, \delta)$ with Q the set of states, with Σ a finite alphabet, and with δ the transitions.

A Kind of Dichotomy

- Fix a **semiautomaton** $S = (Q, \Sigma, \delta)$ with Q the set of **states**, with Σ a **finite alphabet**, and with δ the **transitions**.
- Idea: we will give in the input a **specification**, i.e., a set $\{(i_1, F_1), \ldots, (i_k, F_k)\}$ with $(i_j, F_j) \in Q \times 2^Q$

A Kind of Dichotomy

- Fix a semiautomaton $S = (Q, \Sigma, \delta)$ with Q the set of states, with Σ a finite alphabet, and with δ the transitions.
- Idea: we will give in the input a specification, i.e., a set $\{(i_1, F_1), \ldots, (i_k, F_k)\}$ with $(i_j, F_j) \in Q \times 2^Q$
  - We specify the initial and final states (= closure by quotient)

A Kind of Dichotomy

- Fix a semiautomaton $S = (Q, \Sigma, \delta)$ with Q the set of states, with Σ a finite alphabet, and with δ the transitions.
- Idea: we will give in the input a specification, i.e., a set $\{(i_1, F_1), \ldots, (i_k, F_k)\}$ with $(i_j, F_j) \in Q \times 2^Q$
  - We specify the initial and final states (= closure by quotient)
  - We can toggle the final states (= closure by complement)

A Kind of Dichotomy

- Fix a semiautomaton $S = (Q, \Sigma, \delta)$ with Q the set of states, with Σ a finite alphabet, and with δ the transitions.
- Idea: we will give in the input a specification, i.e., a set $\{(i_1, F_1), \ldots, (i_k, F_k)\}$ with $(i_j, F_j) \in Q \times 2^Q$
  - We specify the initial and final states (= closure by quotient)
  - We can toggle the final states (= closure by complement)
  - We will do a conjunction over the (i_j, F_j) (= closure by intersection)

A Kind of Dichotomy

- Fix a semiautomaton $S = (Q, \Sigma, \delta)$ with Q the set of states, with Σ a finite alphabet, and with δ the transitions.
- Idea: we will give in the input a specification, i.e., a set $\{(i_1, F_1), \ldots, (i_k, F_k)\}$ with $(i_j, F_j) \in Q \times 2^Q$
  - We specify the initial and final states (= closure by quotient)
  - We can toggle the final states (= closure by complement)
  - We will do a conjunction over the (i_j, F_j) (= closure by intersection)
- Semiautomaton Constrained topological sort problem $\mathsf{CTS}(S)$:
  - Input:
    - a DAG G with vertices labeled by letters of Σ,
    - a specification of S, i.e., $\{(i_1, F_1), \ldots, (i_k, F_k)\}$ with $(i_j, F_j) \in Q \times 2^Q$
  - Output: is there a topological sort of G such that the sequence of vertex labels is accepted by the automaton $(Q, \Sigma, \delta, i_j, F_j)$ for all $1 \leq j \leq k$

A Kind of Dichotomy (2)

Theorem

For every *semiautomaton S, exactly one of the following holds:*

- *The transition semigroup of* S *belongs to* ... *and* $\mathsf{CTS}(S)$ *is in NL*
- *The transition semigroup of* S *is not in* ... *and* $\mathsf{CTS}(S)$ *is NP-hard*

A Kind of Dichotomy (2)

Theorem

For every counterfree semiautomaton S, exactly one of the following holds:

- *The transition semigroup of S belongs to DA and $\mathsf{CTS}(S)$ is in NL*
- *The transition semigroup of S is not in DA and $\mathsf{CTS}(S)$ is NP-hard*

- *DA* is a classic variety of semigroups

A Kind of Dichotomy (2)

Theorem

For every counterfree semiautomaton S, exactly one of the following holds:

- *The transition semigroup of* $\mathcal{S}$ *belongs to DA and* $\mathsf{CTS}(S)$ *is in NL*
- *The transition semigroup of* $\mathcal{S}$ *is not in DA and* $\mathsf{CTS}(S)$ *is NP-hard*

- *DA* is a classic variety of semigroups

A Kind of Dichotomy (2)

Theorem

For every counterfree semiautomaton S, exactly one of the following holds:

- *The transition semigroup of S belongs to DA and $\mathsf{CTS}(S)$ is in NL*
- *The transition semigroup of S is not in DA and $\mathsf{CTS}(S)$ is NP-hard*

- *DA* is a classic variety of semigroups
- Counterfree is equivalent to being first-order definable and "not containing any groups"

A Kind of Dichotomy (2)

Theorem

For every ~~*counterfree*~~ *semiautomaton S, exactly one of the following holds:*

- *The transition semigroup of* S *belongs to DO and* CSh(*S*) *is in NL*
- *The transition semigroup of* S *is not in DS and* CSh(*S*) *is NP-hard*

- *DA* is a classic variety of semigroups
- Counterfree is equivalent to being first-order definable and “not containing any groups”
- *DO*, *DS* are much less well understood varieties of semigroups

Conclusion

Summary and Future Work

Language	**CSh** (shuffle)	**CTS** (top. sort)
$(ab)^*$, u^* with different letters	NP-hard	NP-hard

Summary and Future Work

Language	**CSh** (shuffle)	**CTS** (top. sort)
$(ab)^*$, u^* with different letters	NP-hard	NP-hard
Monomials $A_1^* a_1 \cdots A_n^* a_n A_{n+1}^*$	in NL	in NL
Groups, district group monomials	in NL	¯_(ツ)_/¯

Summary and Future Work

Language	**CSh** (shuffle)	**CTS** (top. sort)
$(ab)^*$, u^* with different letters	NP-hard	NP-hard
Monomials $A_1^* a_1 \cdots A_n^* a_n A_{n+1}^*$	in NL	in NL
Groups, district group monomials	in NL	¯_(ツ)_/¯
$b\Sigma^* + aa\Sigma^* + (ab)^*$	in NL	NP-hard

Summary and Future Work

Language	**CSh** (shuffle)	**CTS** (top. sort)
$(ab)^*$, u^* with different letters	NP-hard	NP-hard
Monomials $A_1^* a_1 \cdots A_n^* a_n A_{n+1}^*$	in NL	in NL
Groups, district group monomials	in NL	¯_(ツ)_/¯
$b\Sigma^* + aa\Sigma^* + (ab)^*$	in NL	NP-hard
$L + \Sigma^*(a^k + b^k)\Sigma^*$	in NL	in NL
$(ab)^* + \Sigma^* a^2 \Sigma^*$	in NL	in NL
$L + \Sigma^* a^k \Sigma^*$	¯_(ツ)_/¯	¯_(ツ)_/¯

Summary and Future Work

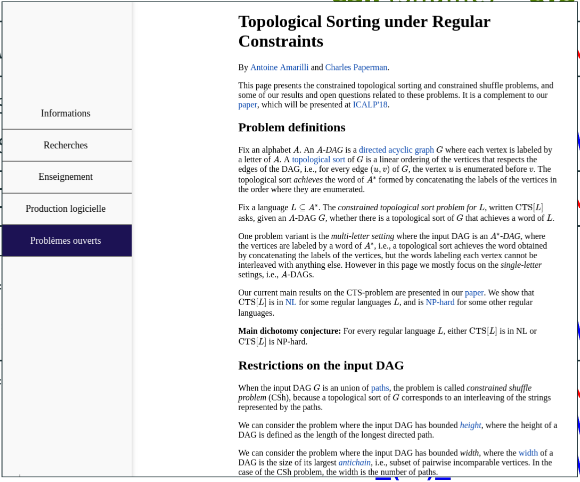
Language	**CSh** (shuffle)	**CTS** (top. sort)
$(ab)^*$, u^* with different letters	NP-hard	NP-hard
Monomials $A_1^* a_1 \cdots A_n^* a_n A_{n+1}^*$	in NL	in NL
Groups, district group monomials	in NL	¯_(ツ)_/¯
$b\Sigma^* + aa\Sigma^* + (ab)^*$	in NL	NP-hard
$L + \Sigma^*(a^k + b^k)\Sigma^*$	in NL	in NL
$(ab)^* + \Sigma^* a^2 \Sigma^*$	in NL	in NL
$L + \Sigma^* a^k \Sigma^*$	¯_(ツ)_/¯	¯_(ツ)_/¯
$(aa + bb)^*$, $(ab + a)^*$	NP-hard	NP-hard
$(aa + b)^*$	in NL	¯_(ツ)_/¯
$(a^k + b)^*$	¯_(ツ)_/¯	¯_(ツ)_/¯

Summary and Future Work

Language	**CSh** (shuffle)	**CTS** (top. sort)
$(ab)^*$, u^* with different letters	NP-hard	NP-hard
Monomials $A_1^* a_1 \cdots A_n^* a_n A_{n+1}^*$	in NL	in NL
Groups, district group monomials	in NL	¯_(ツ)_/¯
$b\Sigma^* + aa\Sigma^* + (ab)^*$	in NL	NP-hard
$L + \Sigma^*(a^k + b^k)\Sigma^*$	in NL	in NL
$(ab)^* + \Sigma^* a^2 \Sigma^*$	in NL	in NL
$L + \Sigma^* a^k \Sigma^*$	¯_(ツ)_/¯	¯_(ツ)_/¯
$(aa + bb)^*$, $(ab + a)^*$	NP-hard	NP-hard
$(aa + b)^*$	in NL	¯_(ツ)_/¯
$(a^k + b)^*$	¯_(ツ)_/¯	¯_(ツ)_/¯
Essentially all other languages...	¯_(ツ)_/¯	¯_(ツ)_/¯

Summary and Future Work

Language	CSh (shuffle)	CTS (top. sort)
$(ab)^*$, u^* w		NP-hard
Monomials Groups, di		in NL ¯_(ツ)_/¯
$b\Sigma^* + aa\Sigma$		NP-hard
$L + \Sigma^*(a^k$ $(ab)^* + \Sigma^*$ $L + \Sigma^* a^k \Sigma$		in NL in NL ¯_(ツ)_/¯
$(aa+bb)^*$ $(aa+b)^*$ $(a^k+b)^*$		NP-hard ¯_(ツ)_/¯ ¯_(ツ)_/¯
Essentially all other languages...	¯_(ツ)_/¯	¯_(ツ)_/¯



Informations

Recherches

Enseignement

Production logicielle

Problèmes ouverts

Topological Sorting under Regular Constraints

By Antoine Amarilli and Charles Paperman.

This page presents the constrained topological sorting and constrained shuffle problems, and some of our results and open questions related to these problems. It is a complement to our paper, which will be presented at ICALP'18.

Problem definitions

Fix an alphabet A. An *A-DAG* is a directed acyclic graph G where each vertex is labeled by a letter of A. A topological sort of G is a linear ordering of the vertices that respects the edges of the DAG, i.e., for every edge (u, v) of G, the vertex u is enumerated before v. The topological sort *achieves* the word of A^* formed by concatenating the labels of the vertices in the order where they are enumerated.

Fix a language $L \subseteq A^*$. The *constrained topological sort problem for* L, written $\mathrm{CTS}[L]$ asks, given an A-DAG G, whether there is a topological sort of G that achieves a word of L.

One problem variant is the *multi-letter setting* where the input DAG is an *A^*-DAG*, where the vertices are labeled by a word of A^*, i.e., a topological sort achieves the word obtained by concatenating the labels of the vertices, but the words labeling each vertex cannot be interleaved with anything else. However in this page we mostly focus on the *single-letter* setings, i.e., A-DAGs.

Our current main results on the CTS-problem are presented in our paper. We show that $\mathrm{CTS}[L]$ is in NL for some regular languages L, and is NP-hard for some other regular languages.

Main dichotomy conjecture: For every regular language L, either $\mathrm{CTS}[L]$ is in NL or $\mathrm{CTS}[L]$ is NP-hard.

Restrictions on the input DAG

When the input DAG G is an union of paths, the problem is called *constrained shuffle problem* (CSh), because a topological sort of G corresponds to an interleaving of the strings represented by the paths.

We can consider the problem where the input DAG has bounded *height*, where the height of a DAG is defined as the length of the longest directed path.

We can consider the problem where the input DAG has bounded *width*, where the width of a DAG is the size of its largest *antichain*, i.e., subset of pairwise incomparable vertices. In the case of the CSh problem, the width is the number of paths.

Summary and Future Work

Language		CTS (top. sort)
$(ab)^*$, u^* with		NP-hard
Monomials $A_1^* a$ Groups, distric		in NL ¯_(ツ)_/¯
$b\Sigma^* + aa\Sigma^* +$		NP-hard
$L + \Sigma^*(a^k + b^k$ $(ab)^* + \Sigma^* a^2 \Sigma$ $L + \Sigma^* a^k \Sigma^*$		in NL in NL ¯_(ツ)_/¯
$(aa + bb)^*$, $(a$ $(aa + b)^*$ $(a^k + b)^*$		NP-hard ¯_(ツ)_/¯ ¯_(ツ)_/¯
Essentially all other languages...	¯_(ツ)_/¯	¯_(ツ)_/¯

Summary and Future Work

Language	**CSh** (shuffle)	**CTS** (top. sort)
$(ab)^*$, u^* with different letters	NP-hard	NP-hard
Monomials $A_1^* a_1 \cdots A_n^* a_n A_{n+1}^*$	in NL	in NL
Groups, district group monomials	in NL	¯_(ツ)_/¯
$b\Sigma^* + aa\Sigma^* + (ab)^*$	in NL	NP-hard
$L + \Sigma^*(a^k + b^k)\Sigma^*$	in NL	in NL
$(ab)^* + \Sigma^* a^2 \Sigma^*$	in NL	in NL
$L + \Sigma^* a^k \Sigma^*$	¯_(ツ)_/¯	¯_(ツ)_/¯
$(aa + bb)^*$, $(ab + a)^*$	NP-hard	NP-hard
$(aa + b)^*$	in NL	¯_(ツ)_/¯
$(a^k + b)^*$	¯_(ツ)_/¯	¯_(ツ)_/¯
Essentially all other languages...	¯_(ツ)_/¯	¯_(ツ)_/¯

Thanks for your attention!

References

Amarilli, A. and Paperman, C. (2018).
Topological Sorting under Regular Constraints.
In *ICALP*.

Warmuth, M. K. and Haussler, D. (1984).
On the complexity of iterated shuffle.
JCSS, 28(3).

Image Credits

Super-Dupont (slide 24) : *Oui nide iou*, Superdupont, Lob & Gotlib, drawn by Neal Adams, Alexis, Al Coutelis, Daniel Goossens, Solé, Gotlib. Fair use.