# Query Evaluation over SLP-Compressed Data

Markus Schmid
HU Berlin

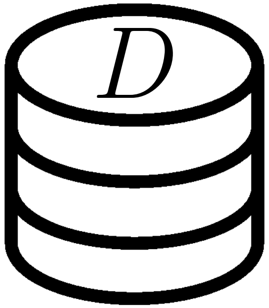
Forschungsvortrag
Magdeburg, 13.11.25

# Outline of My Recent Research Programme

DFG Project: "Query Evaluation over SLP-Compressed Data"

# Outline of My Recent Research Programme

## DFG Project: “Query Evaluation over SLP-Compressed Data”



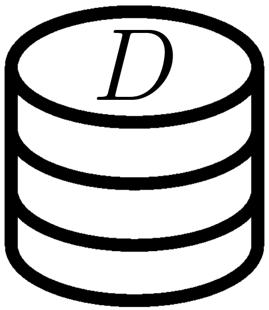

Data(base)

$q$

Query

# Outline of My Recent Research Programme

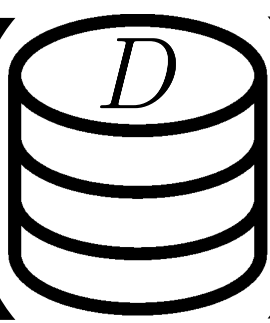
DFG Project: “Query Evaluation over SLP-Compressed Data”



Data(base)

$$q$$

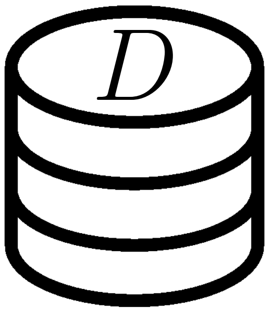
Query

$q\Big($  $\Big) = \{\, t_1, \quad t_2, \quad t_3, \quad t_4, \ldots \,\}$

Query result

# Outline of My Recent Research Programme

DFG Project: “Query Evaluation over SLP-Compressed Data”



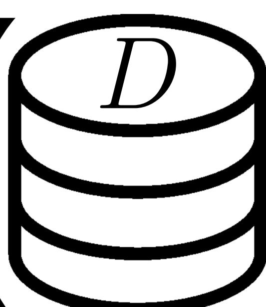

Data(base)

Relational data

$q$

Query

SQL-queries (FO-formulas)

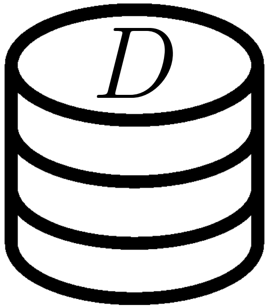
$q\Big($ 

$\Big) = \{\, t_1, \quad t_2, \quad t_3, \quad t_4, \ldots \quad \}$

Query result

# Outline of My Recent Research Programme

DFG Project: "Query Evaluation over SLP-Compressed Data"



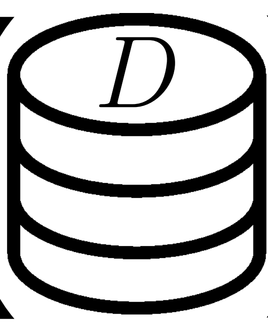

Data(base)

Relational data, strings, trees, graphs, etc.

$q$

Query

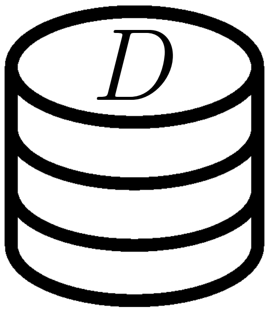
SQL-queries (FO-formulas), formulas in MSO, document spanners, etc.

$q\Big($  $\Big) = \{ t_1, \quad t_2, \quad t_3, \quad t_4, \ldots \}$


Query result

# Outline of My Recent Research Programme

DFG Project: “Query Evaluation over SLP-Compressed Data”




Data(base)

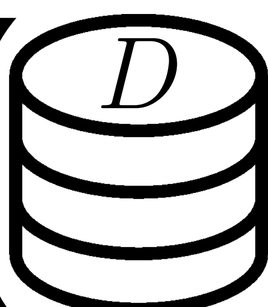
Relational data, strings, trees, graphs, etc.

$q$

Query

SQL-queries (FO-formulas), formulas in MSO, document spanners, etc.

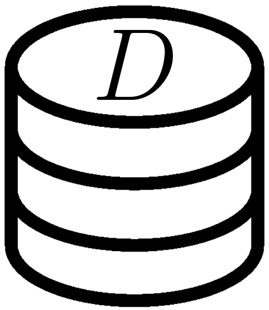
Preprocessing

$q$ (   ) $= \{ t_1, \quad t_2, \quad t_3, \quad t_4, \ldots \quad \}$

Query result

# Outline of My Recent Research Programme

DFG Project: “Query Evaluation over SLP-Compressed Data”



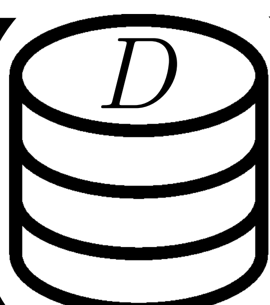

Data(base)

Relational data, strings,
trees, graphs, etc.

$q$

Query

SQL-queries (FO-formulas),
formulas in MSO,
document spanners, etc.

$q\Big($ 

$\Big)$

Query result

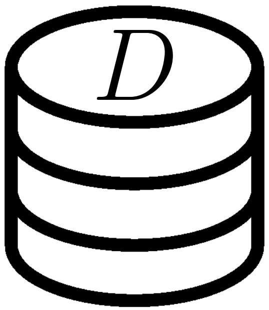
Preprocessing

$$= \{\, t_1, \underbrace{\qquad}_{\text{delay}} t_2, \underbrace{\qquad}_{\text{delay}} t_3, \underbrace{\qquad}_{\text{delay}} t_4, \ldots \,\}$$

enumerate

# Outline of My Recent Research Programme

DFG Project: "Query Evaluation over SLP-Compressed Data"



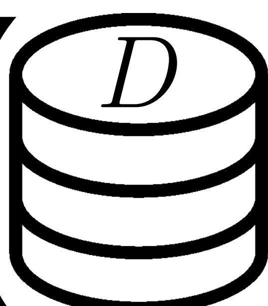

Data(base)

Relational data, strings, trees, graphs, etc.

$q$

Query

SQL-queries (FO-formulas), formulas in MSO, document spanners, etc.

$q$(  )


Query result

Preprocessing

$= \{ t_1, \quad t_2, \quad t_3, \quad t_4, \ldots \}$

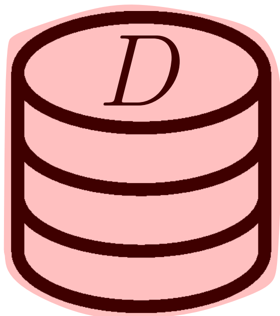
delay delay delay

enumerate

Preprocessing: linear in $|D|$ Delay: Independent from $|D|$

# Outline of My Recent Research Programme

DFG Project: “Query Evaluation over SLP-Compressed Data”



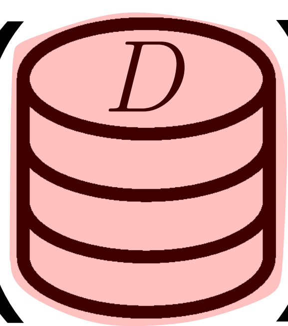

Data(base)

Relational data, strings,
trees, graphs, etc.

$q$

Query

SQL-queries (FO-formulas),
formulas in MSO,
document spanners, etc.

$q($

$)$

Query result

Preprocessing

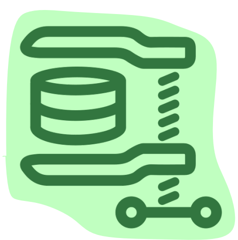
$$= \{ t_1, \underbrace{\quad}_{\text{delay}} t_2, \underbrace{\quad}_{\text{delay}} t_3, \underbrace{\quad}_{\text{delay}} t_4, \ldots \}$$

enumerate

Preprocessing: linear in $|D|$ Delay: Independent from $|D|$

# Outline of My Recent Research Programme

DFG Project: “Query Evaluation over SLP-Compressed Data”



Compressed data

Relational data, strings, trees, graphs, etc.

$q$

Query

SQL-queries (FO-formulas), formulas in MSO, document spanners, etc.

$q\Big($ 

$\Big) = \{ t_1, \; t_2, \; t_3, \; t_4, \; \ldots \}$

Query result

Preprocessing

delay delay delay

enumerate

Preprocessing: linear in $|D|$  Delay: Independent from $|D|$

# Outline of My Recent Research Programme

DFG Project: “Query Evaluation over SLP-Compressed Data”

Compressed data

Relational data, strings, trees, graphs, etc.

$q$

Query

SQL-queries (FO-formulas), formulas in MSO, document spanners, etc.

Preprocessing

$q($  $)$ $= \{ t_1, \underbrace{\quad\quad}_{\text{delay}} t_2, \underbrace{\quad\quad}_{\text{delay}} t_3, \underbrace{\quad\quad}_{\text{delay}} t_4, \ldots \}$

Query result

enumerate

Preprocessing: linear in $|D|$ Delay: Independent from $|D|$

# The Advantages of Querying Compressed Data

Manage data completely in compressed form!

# The Advantages of Querying Compressed Data

Manage data completely in compressed form!

Obtain faster algorithms!

# The Advantages of Querying Compressed Data

Manage data completely in compressed form!

Obtain faster algorithms!

Algorithm A

- solves problem $P$
- needs **uncompressed** inputs
- running time: $O(n)$

Algorithm B

- solves problem $P$
- allows **compressed** inputs
- running time: $O(m^3)$

# The Advantages of Querying Compressed Data

Manage data completely in compressed form!

Obtain faster algorithms!

Algorithm A

- solves problem $P$
- needs **uncompressed** inputs
- running time: $O(n)$

Algorithm B

- solves problem $P$
- allows **compressed** inputs
- running time: $O(m^3)$

Algorithm $B$ might outperform algorithm $A$ on strongly compressible instances!

We will survey three results about query enumeration over SLP-compressed data.

| | Query class | Data domain |
|---|---|---|
| (1) | Document spanners | Strings |
| (2) | MSO-queries | Node-labelled unranked forests |
| (3) | FO-queries | Relational structures of bounded degree |

## Querying Compressed Strings, Trees, and Relational Data

We will survey three results about query enumeration over SLP-compressed data.

| | Query class | Data domain |
|---|---|---|
| (1) | Document spanners | Strings |
| (2) | MSO-queries | Node-labelled unranked forests |
| (3) | FO-queries | Relational structures of bounded degree |

Note: We measure algorithmic performance in data complexity. This means that the size of the query is constant.

# The Compression Scheme

We need a compression scheme that ...

- ... is lossless.
- ... is versatile, i.e., we can use it for different data domains.
- ... potentially achieves very strong compression.
- ... is suitable for algorithmics on compressed inputs.
- ... fast compression algorithms.

# The Compression Scheme

We need a compression scheme that ...

- ... is lossless.
- ... is versatile, i.e., we can use it for different data domains.
- ... potentially achieves very strong compression.
- ... is suitable for algorithmics on compressed inputs.
- ... fast compression algorithms.

$\longrightarrow$ Straight-Line Programs (SLP)

# Straight-Line Programs (SLPs)

Basic Idea: Represent a data object $D$ by a context-free grammar $G$ that derives only $D$, i.e., $L(G) = \{D\}$.

# Straight-Line Programs (SLPs)

Basic Idea: Represent a data object $D$ by a context-free grammar $G$ that derives only $D$, i.e., $L(G) = \{D\}$.

## Data Domain: Strings

# Straight-Line Programs (SLPs)

Basic Idea: Represent a data object $D$ by a context-free grammar $G$ that derives only $D$, i.e., $L(G) = \{D\}$.

## Data Domain: Strings

Data: a a b b a b a a b a a b b a b

# Straight-Line Programs (SLPs)

Basic Idea: Represent a data object $D$ by a context-free grammar $G$ that derives only $D$, i.e., $L(G) = \{D\}$.

## Data Domain: Strings

Data: $\mathtt{a\,a\,b\,b\,a\,b\,a\,a\,b\,a\,a\,b\,b\,a\,b}$

Compression: $S \to C\,B\,C, \quad B \to \mathtt{a}\,A$

$C \to B\,\mathtt{b}\,A, \quad A \to \mathtt{a\,b}$

# Straight-Line Programs (SLPs)

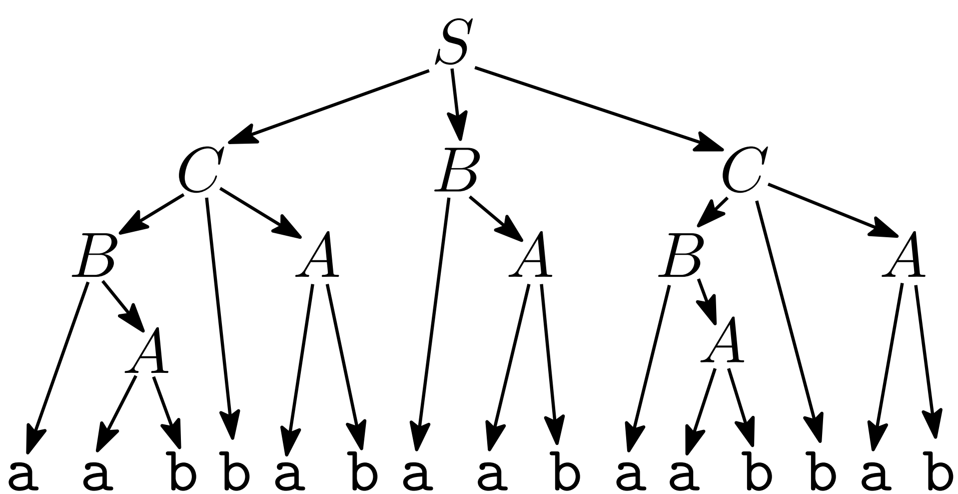
Basic Idea: Represent a data object $D$ by a context-free grammar $G$ that derives only $D$, i.e., $L(G) = \{D\}$.

## Data Domain: Strings

Data: `a a b b a b a a b a a b b a b`

Compression:

$$S \to C\,B\,C, \quad B \to \mathtt{a}\,A$$
$$C \to B\,\mathtt{b}\,A, \quad A \to \mathtt{a}\,\mathtt{b}$$



# Straight-Line Programs (SLPs)

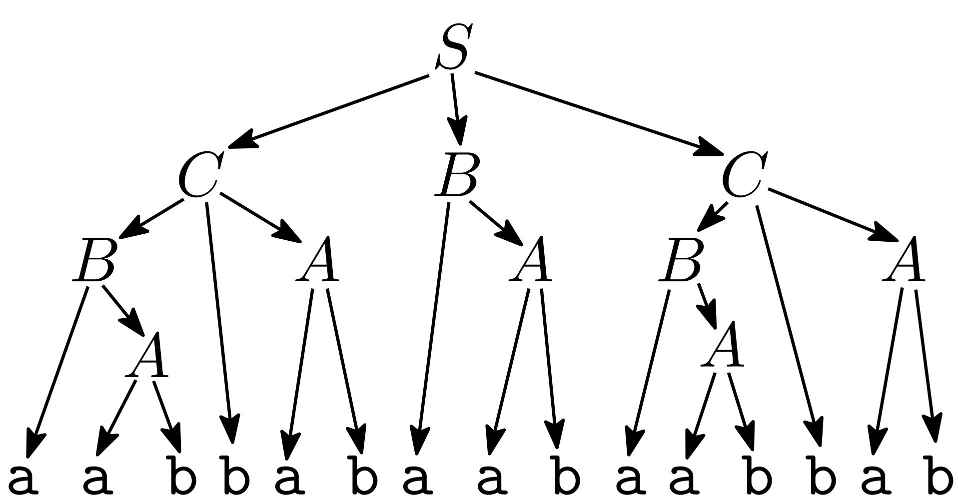
Basic Idea: Represent a data object $D$ by a context-free grammar $G$ that derives only $D$, i.e., $L(G) = \{D\}$.

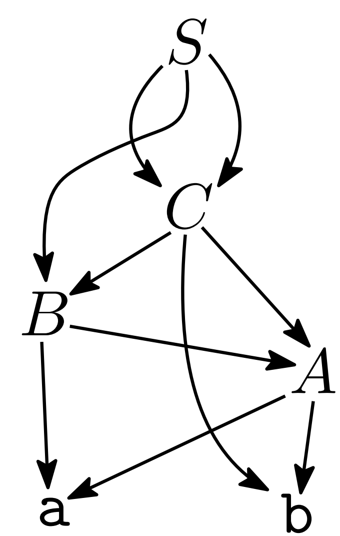
## Data Domain: Strings

Data: $\texttt{a a b b a b a a b a a b b a b}$

Compression: $S \to C\,B\,C$, $\quad B \to \texttt{a}\,A$

$C \to B\,\texttt{b}\,A$, $\quad A \to \texttt{a}\,\texttt{b}$



# String-SLPs - State of the Art

String-SLPs are very popular and have several good properties.

- exponential compression in the best case.
- tightly related to dictionary based compression, in particular LZ77 and LZ78.
- there are many fast compression algorithms that achieve excellent compression in practice.
- there are many fast algorithms that work directly on SLP-compressed strings.

# String-SLPs - State of the Art

String-SLPs are very popular and have several good properties.

- exponential compression in the best case.
- tightly related to dictionary based compression, in particular LZ77 and LZ78.
- there are many fast compression algorithms that achieve excellent compression in practice.
- there are many fast algorithms that work directly on SLP-compressed strings.

New angle and challenge: Extend known techniques ...

- ... to the query evaluation perspective.
- ... to the enumeration perspective.

# Document Spanners Over SLP-Compressed Texts

Document spanners:

Interpret regular expressions (or automata) as queries

# Document Spanners Over SLP-Compressed Texts

Document spanners:
Interpret regular expressions (or automata) as queries

$$q = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \ \mathtt{a}^+ \ \mathtt{c}^* \ \mathtt{b}^+ \ \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*$$

# Document Spanners Over SLP-Compressed Texts

Document spanners:
Interpret regular expressions (or automata) as queries

$$q = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \underbrace{\mathtt{a}^+}_{x} \mathtt{c}^* \underbrace{\mathtt{b}^+}_{y} \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*$$

# Document Spanners Over SLP-Compressed Texts

Document spanners:
Interpret regular expressions (or automata) as queries

$$q = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \underbrace{\mathtt{a}^+}_{x} \mathtt{c}^* \underbrace{\mathtt{b}^+}_{y} \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*$$

$q(\mathtt{aaacbcaccbbca}) =$

| $x$ | $y$ |
|---|---|
| $(1,3)$ | $(5,5)$ |
| $(2,3)$ | $(5,5)$ |
| $(3,3)$ | $(5,5)$ |
| $(7,7)$ | $(10,11)$ |
| $(7,7)$ | $(10,10)$ |

# Document Spanners Over SLP-Compressed Texts

Document spanners:
Interpret regular expressions (or automata) as queries

$$q = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \underbrace{\mathtt{a}^+}_{x} \mathtt{c}^* \underbrace{\mathtt{b}^+}_{y} \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*$$

$$q(\mathtt{aaacbcaccbbca}) =$$

| $x$ | $y$ |
|---|---|
| $(1,3)$ | $(5,5)$ |
| $(2,3)$ | $(5,5)$ |
| $(3,3)$ | $(5,5)$ |
| $(7,7)$ | $(10,11)$ |
| $(7,7)$ | $(10,10)$ |

# Document Spanners Over SLP-Compressed Texts

Document spanners:
Interpret regular expressions (or automata) as queries

$$q = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \underbrace{\mathtt{a}^+}_{x} \mathtt{c}^* \underbrace{\mathtt{b}^+}_{y} \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*$$

$$q(\mathtt{aaacbcaccbbca}) =$$

| $x$ | $y$ |
|---|---|
| $(1, 3)$ | $(5, 5)$ |
| $(2, 3)$ | $(5, 5)$ |
| $(3, 3)$ | $(5, 5)$ |
| $(7, 7)$ | $(10, 11)$ |
| $(7, 7)$ | $(10, 10)$ |

# Document Spanners Over SLP-Compressed Texts

**Theorem:**

Given a document spanner $q$,
a string D,
after preprocessing $O(|D|)$, we can enumerate $q(D)$ with constant delay.

[Bagan 2006]
[Amarilli et al. 2021]
[Florenzano et al. 2020]

# Document Spanners Over SLP-Compressed Texts

**Theorem:**

Given a document spanner $q$,
a string D,
after preprocessing $O(|D|)$, we can enumerate $q(D)$ with constant delay.

[Bagan 2006]
[Amarilli et al. 2021]
[Florenzano et al. 2020]

**Theorem:**

Given a document spanner $q$,
a string-SLP S that compresses a string D,
after preprocessing $O(|S|)$, we can enumerate $q(D)$ with constant delay.

[Schmid, Schweikardt, PODS 2021, PODS 2022]

# Document Spanners Over SLP-Compressed Texts

**Theorem:**

Given a document spanner $q$,
a string D,
after preprocessing $O(|D|)$, we can enumerate $q(D)$ with constant delay.

[Bagan 2006]
[Amarilli et al. 2021]
[Florenzano et al. 2020]

**Theorem:**

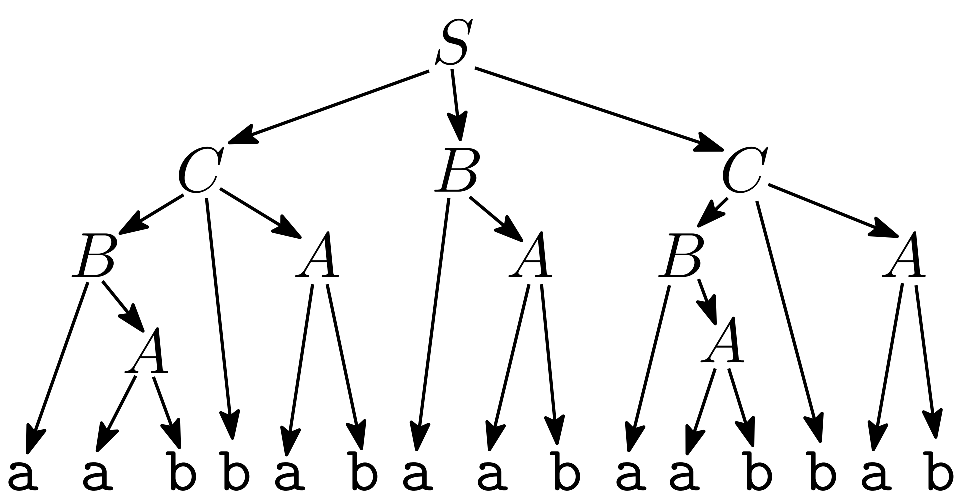
Given a document spanner $q$,
a string-SLP S that compresses a string D,
after preprocessing $O(|S|)$, we can enumerate $q(D)$ with constant delay.

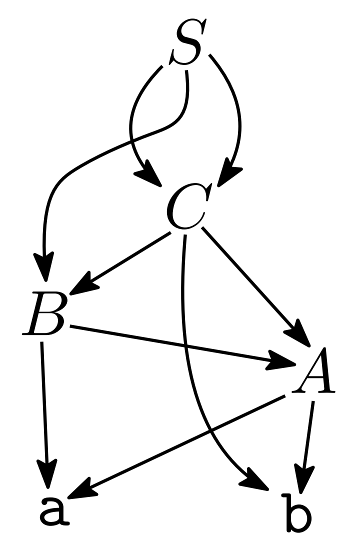
[Schmid, Schweikardt, PODS 2021, PODS 2022]

## Data Domain: Trees and Forests

# Straight-Line Programs for Trees

Recall string-SLPs:
$$S \to C\,B\,C, \quad B \to \mathtt{a}\,A$$
$$C \to B\,\mathtt{b}\,A, \quad A \to \mathtt{a}\,\mathtt{b}$$

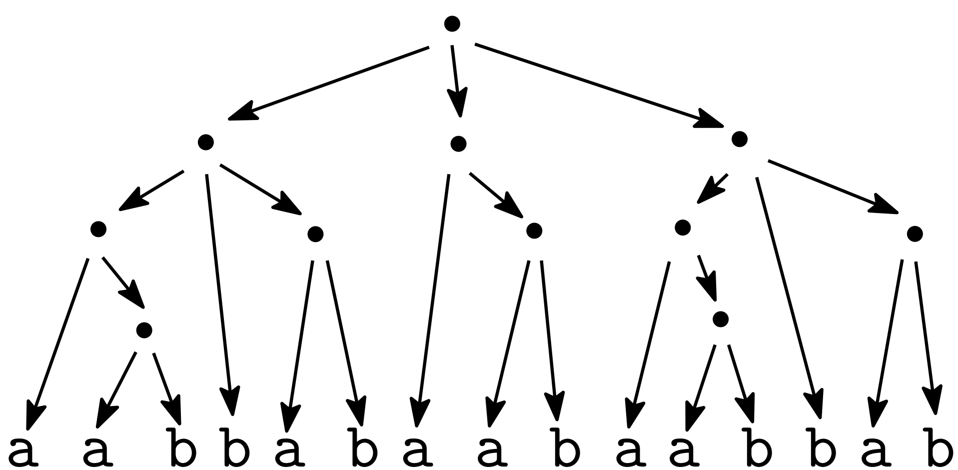


# Straight-Line Programs for Trees

Recall string-SLPs:

$$S \to C\,B\,C, \quad B \to \mathtt{a}\,A$$
$$C \to B\,\mathtt{b}\,A, \quad A \to \mathtt{a}\,\mathtt{b}$$

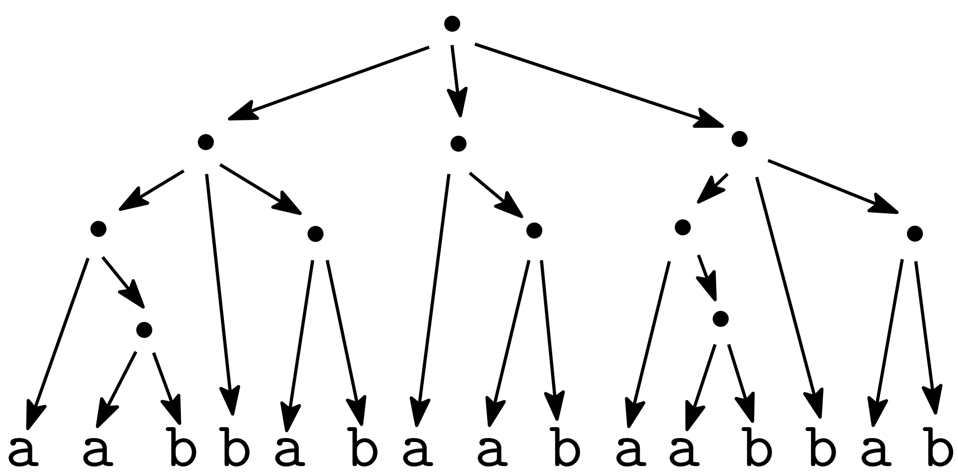



Expression over the monoid $(\Sigma^*, \cdot)$

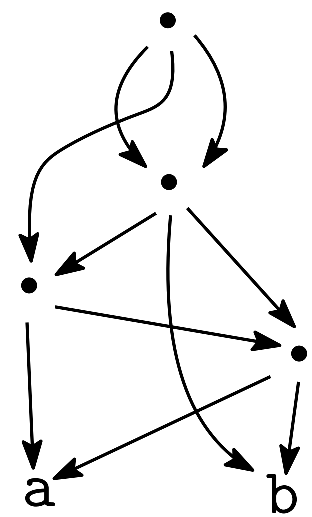
# Straight-Line Programs for Trees

Recall string-SLPs:

$$S \to C\,B\,C, \quad B \to \mathtt{a}\,A$$
$$C \to B\,\mathtt{b}\,A, \quad A \to \mathtt{a}\,\mathtt{b}$$




Expression over the monoid $(\Sigma^*, \cdot)$




Circuit over the monoid $(\Sigma^*, \cdot)$

# Straight-Line Programs for Trees

[Gascón et al., ToCS 2020]

Tree-SLPs: Instead of $(\Sigma^*, \cdot)$
use an algebra for labelled trees.

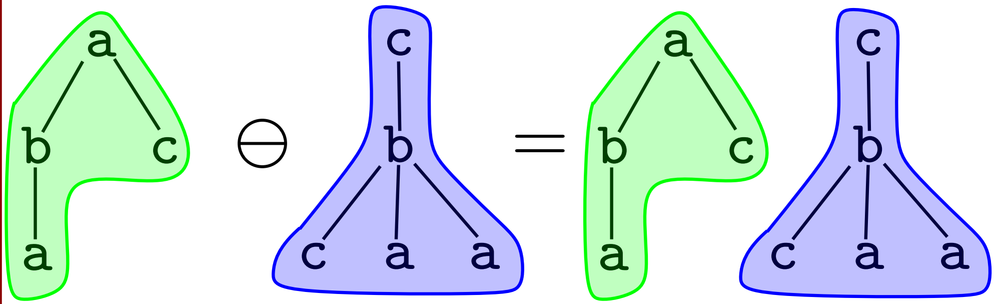
# Straight-Line Programs for Trees

[Gascón et al., ToCS 2020]

Tree-SLPs: Instead of $(\Sigma^*, \cdot)$
use an algebra for labelled trees.

Forest algebra: [Bojańczyk, Walukiewicz, Logic and Automata 2008]

Horizontal concatenation $\ominus$



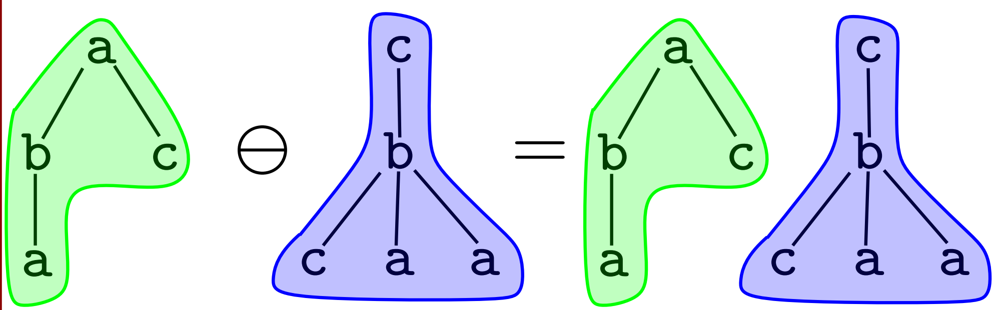
# Straight-Line Programs for Trees

[Gascón et al., ToCS 2020]

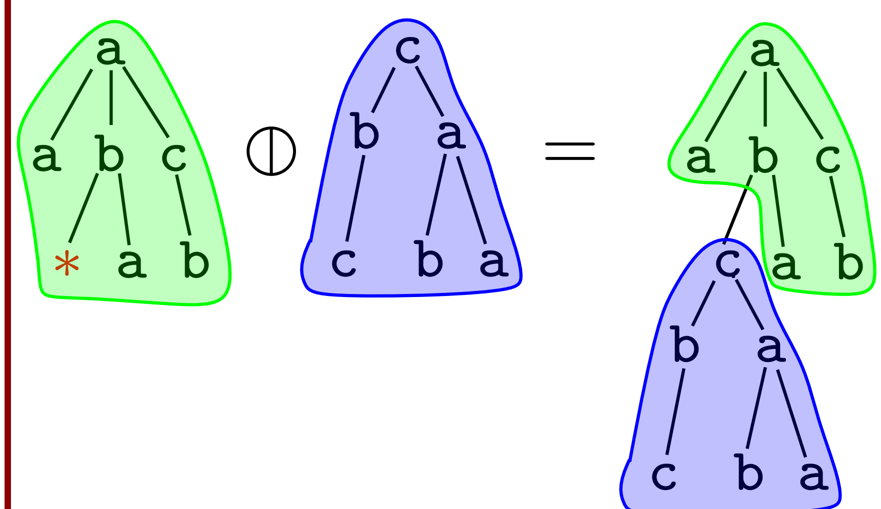
Tree-SLPs: Instead of $(\Sigma^*, \cdot)$
use an algebra for labelled trees.

Forest algebra: [Bojańczyk, Walukiewicz, Logic and Automata 2008]

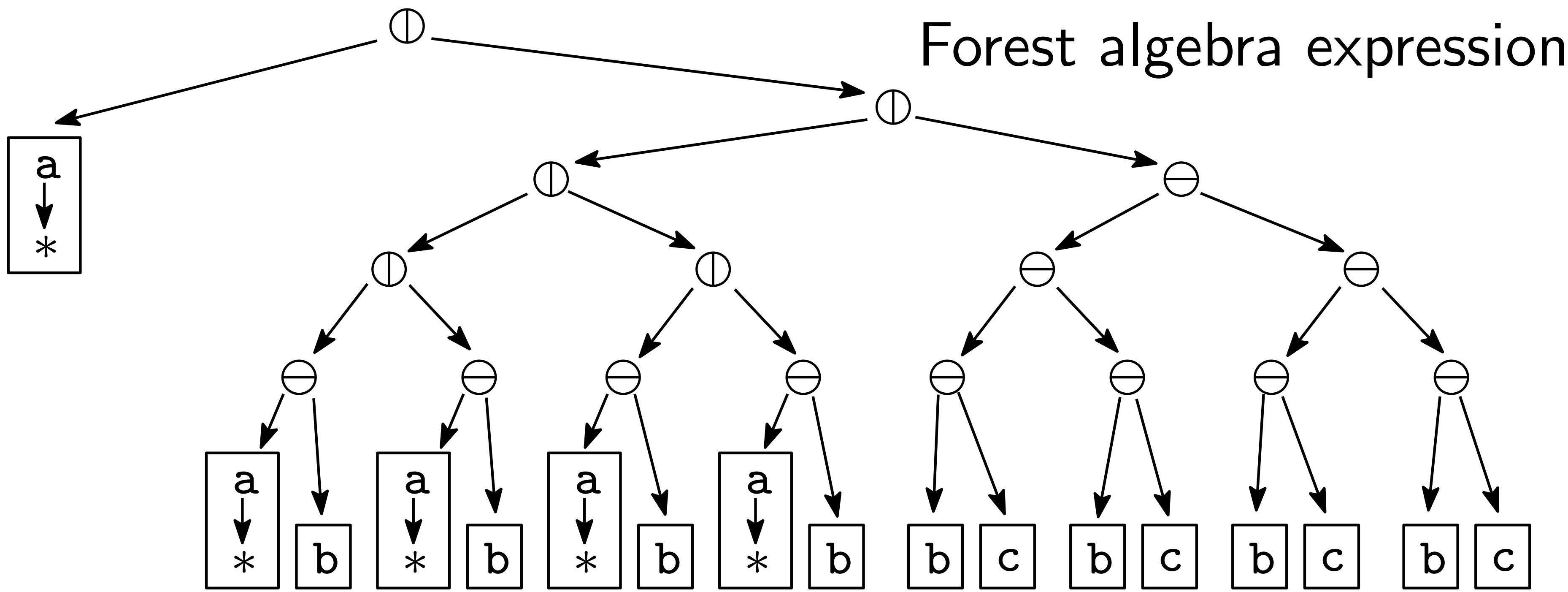
Horizontal concatenation $\ominus$



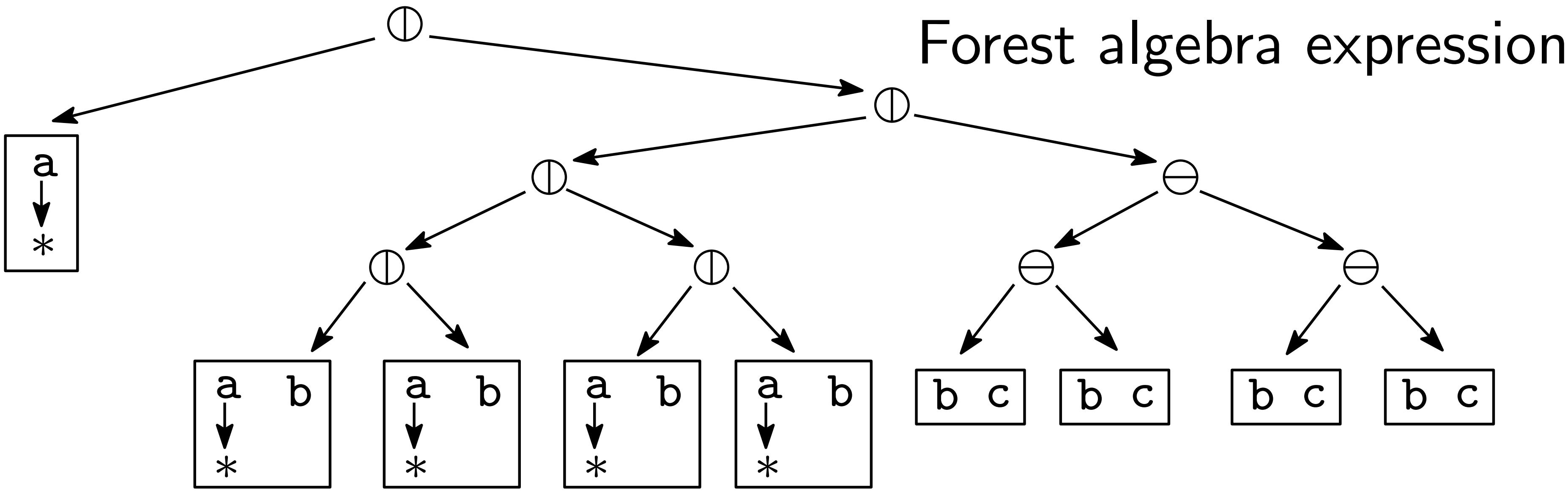
Vertical concatenation $\obar$



# Straight-Line Programs for Trees

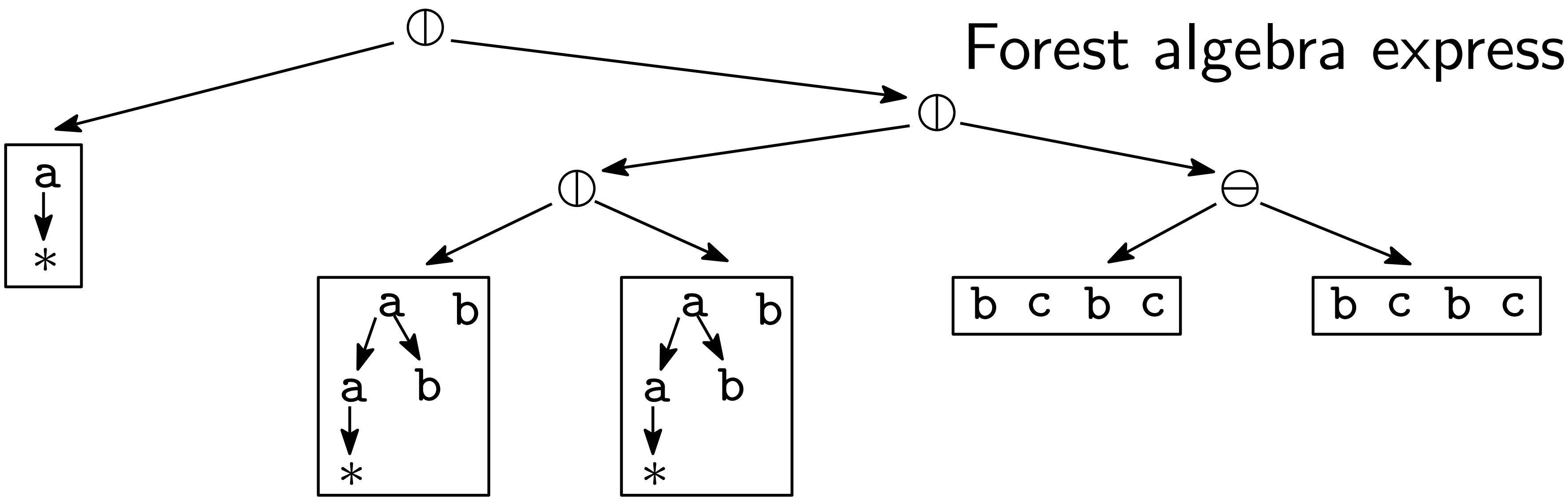


# Straight-Line Programs for Trees

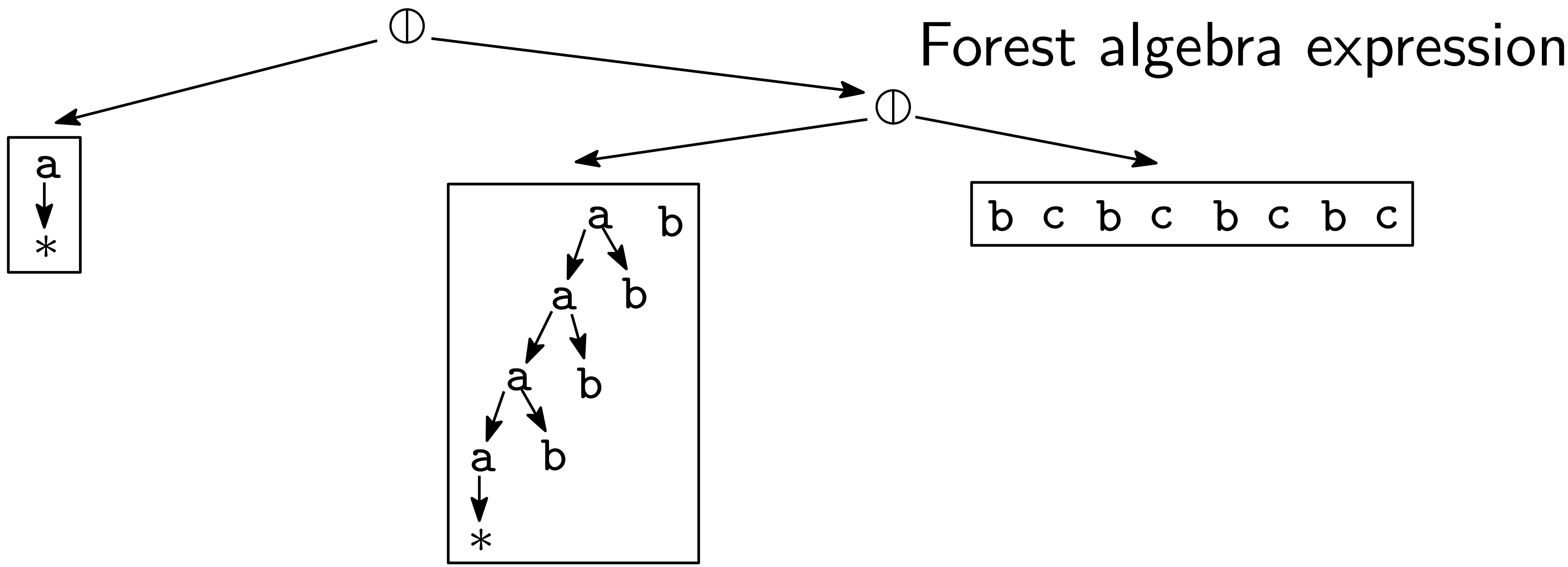


# Straight-Line Programs for Trees

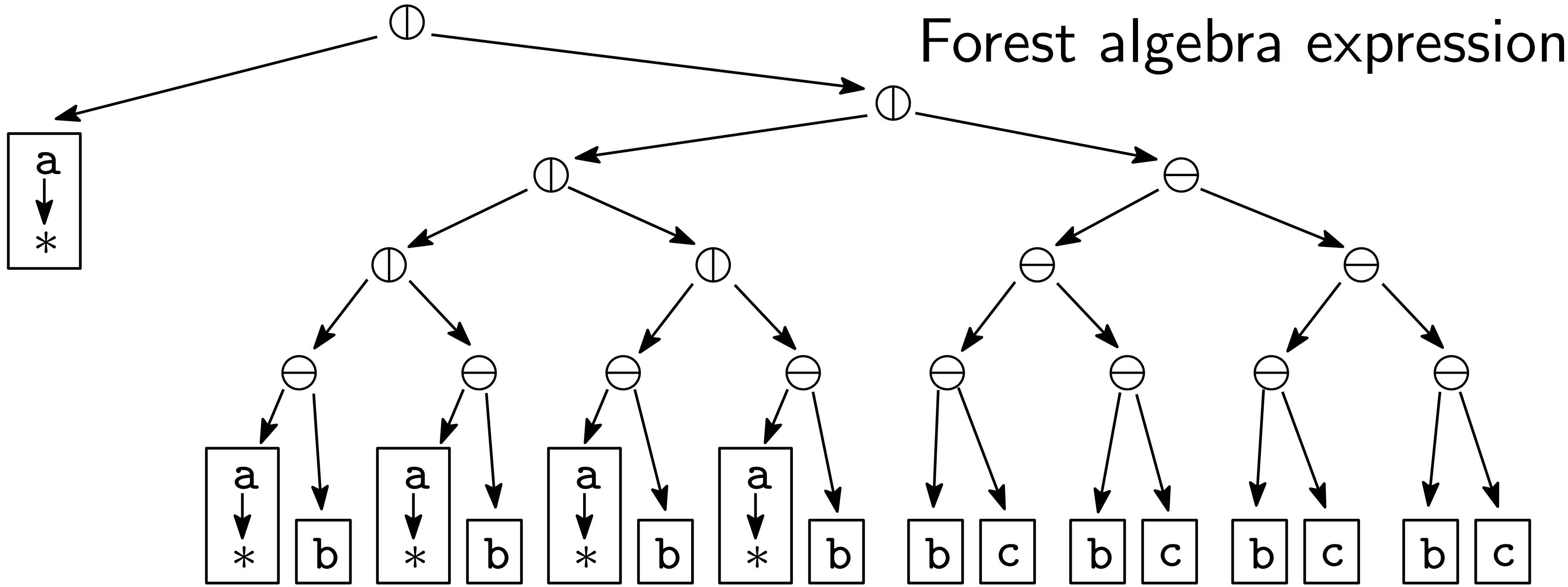
Forest algebra expression



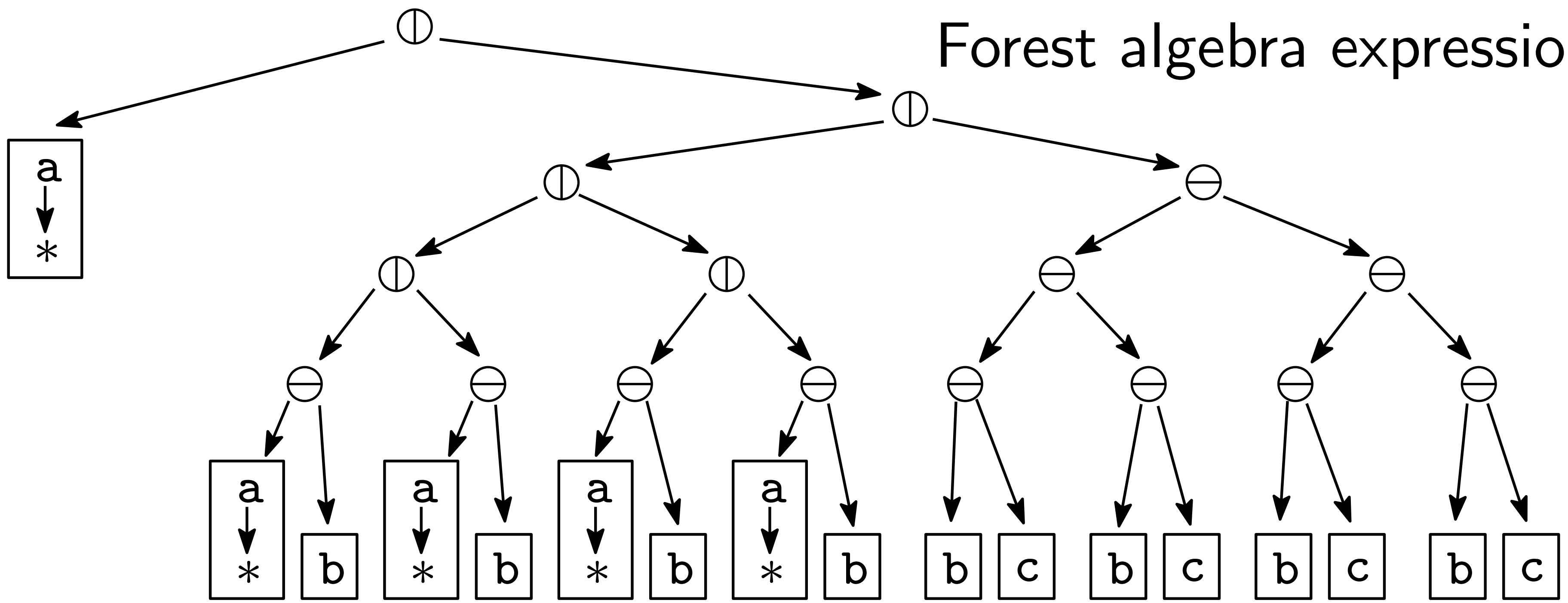
# Straight-Line Programs for Trees



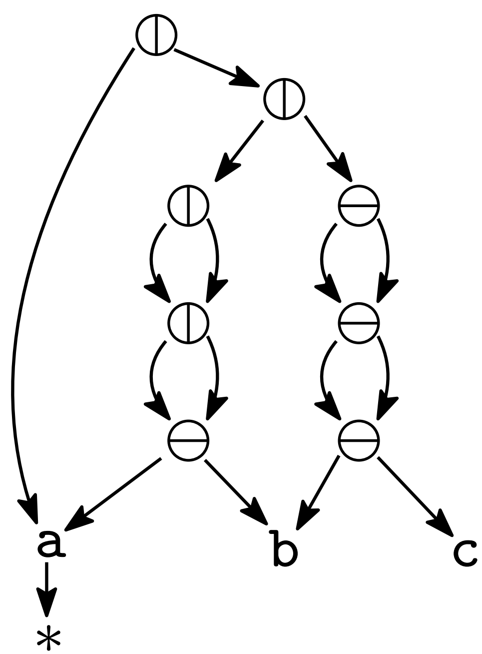
# Straight-Line Programs for Trees



# Straight-Line Programs for Trees




Forest algebra expression




Forest SLP

# MSO Enumeration Over SLP-Compressed Forests

Queries in monadic second order logic (MSO-queries)

Like first order logic, but also with quantifications over second order variables (i.e., set variables).

# MSO Enumeration Over SLP-Compressed Forests

Queries in monadic second order logic (MSO-queries)

Like first order logic, but also with quantifications over second order variables (i.e., set variables).

Example: Query all $(x, Z)$ from a tree such that $x$ is a red node, $Z$ is the set of $x$'s children and all nodes of $Z$ are blue.

$$q(x, Z) = \mathsf{red}(x) \wedge (\forall y \in Z : \mathsf{blue}(y)) \wedge (\forall y : y \in Z \iff \mathsf{child}(x, y))$$

# MSO Enumeration Over SLP-Compressed Forests

Theorem:

Given an MSO query $q(X_1, \ldots, X_m)$,
a node-labelled unranked forest F,
after preprocessing $O(|F|)$, we can enumerate $q(F)$ with output linear delay.

[Bagan, CSL 2006]

[Kazana, Segoufin ACM ToCL 2013]

# MSO Enumeration Over SLP-Compressed Forests

**Theorem:**

Given an MSO query $q(X_1, \ldots, X_m)$,
a node-labelled unranked forest F,
after preprocessing $O(|F|)$, we can enumerate $q(F)$ with output linear delay.

[Bagan, CSL 2006]

[Kazana, Segoufin ACM ToCL 2013]

**Theorem:**

Given an MSO query $q(X_1, \ldots, X_m)$,
a forest-SLP S that compresses a node-labelled unranked forest F,
after preprocessing $O(|S|)$, we can enumerate $q(F)$ with output linear delay.

[Lohrey, Schmid, PODS 2024]

# MSO Enumeration Over SLP-Compressed Forests

**Theorem:**

Given an MSO query $q(X_1, \ldots, X_m)$,
a node-labelled unranked forest F,
after preprocessing $O(|F|)$, we can enumerate $q(F)$ with output linear delay.

[Bagan, CSL 2006]

[Kazana, Segoufin ACM ToCL 2013]

**Theorem:**

Given an MSO query $q(X_1, \ldots, X_m)$,
a forest-SLP S that compresses
a node-labelled unranked forest F,
after preprocessing $O(|S|)$, we can enumerate $q(F)$ with output linear delay.

[Lohrey, Schmid, PODS 2024]

# Data Domain: Relational Data

# Straight-Line Programs for Relational Structures

Relational structure: $(\mathcal{U}, R_1, R_2, \ldots, R_m)$

finite universe

relations over $\mathcal{U}$

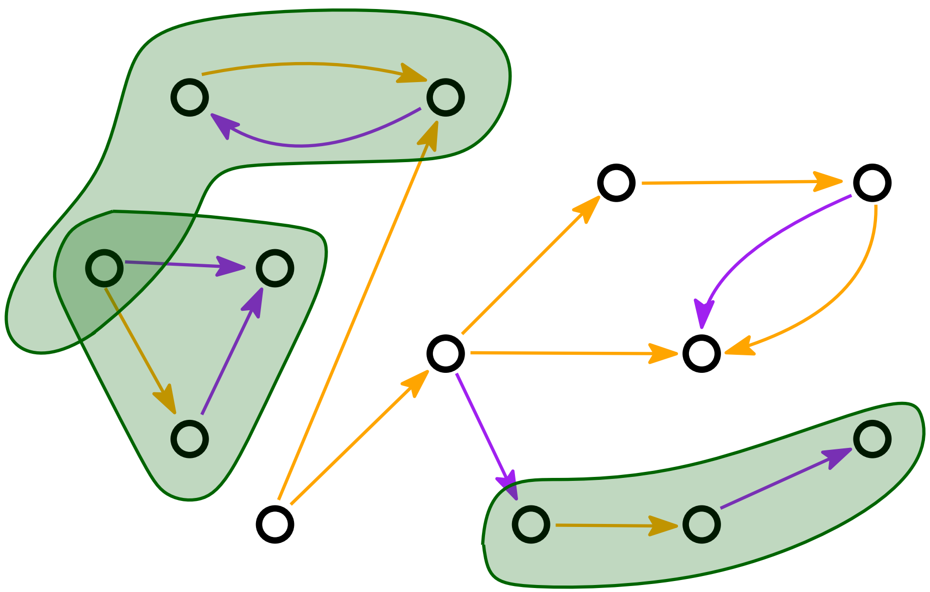
# Straight-Line Programs for Relational Structures

Relational structure: $(\mathcal{U}, R_1, R_2, \ldots, R_m)$

finite universe

relations over $\mathcal{U}$

Example:



$\mathcal{U}$ has $13$ distinct elements.

$R_1$ has arity $2$.

$R_2$ has arity $2$.

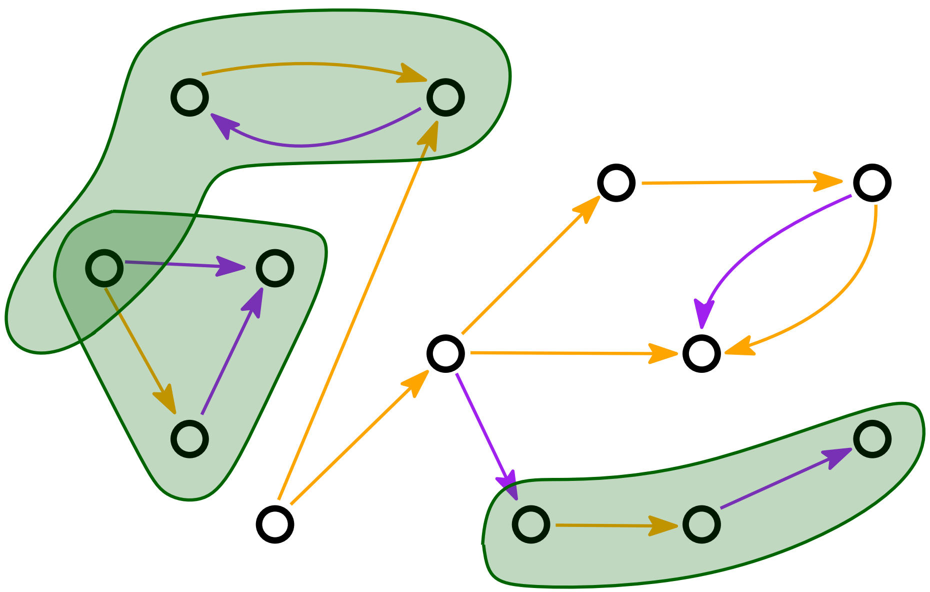
$R_3$ has arity $3$.

# Straight-Line Programs for Relational Structures

Relational structure: $(\mathcal{U}, R_1, R_2, \ldots, R_m)$

finite universe    relations over $\mathcal{U}$

Example:



$\mathcal{U}$ has $13$ distinct elements.

$R_1$ has arity $2$.

$R_2$ has arity $2$.

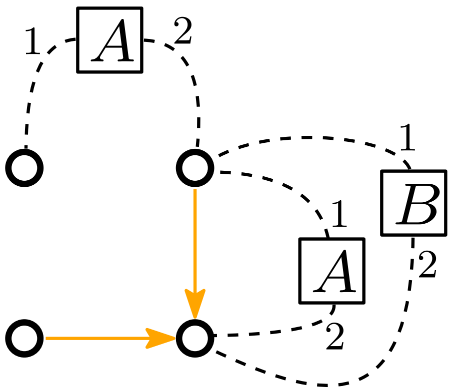
$R_3$ has arity $3$.

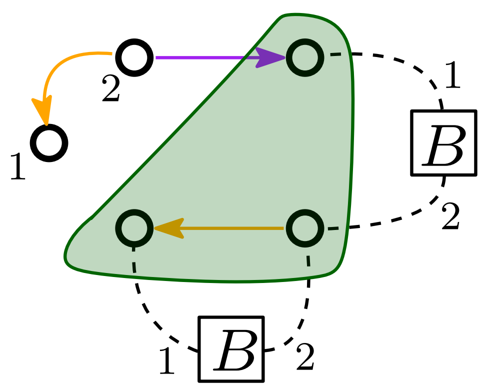
Relational straight-line programs: SLPs based on hyper-edge replacement grammars.

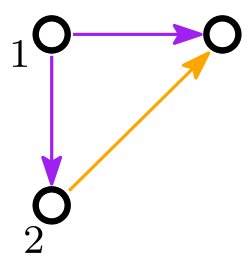
[Lengauer, Wanke, SICOMP 1988]

# Straight-Line Programs for Relational Structures

Relational-SLP with $3$ rules:

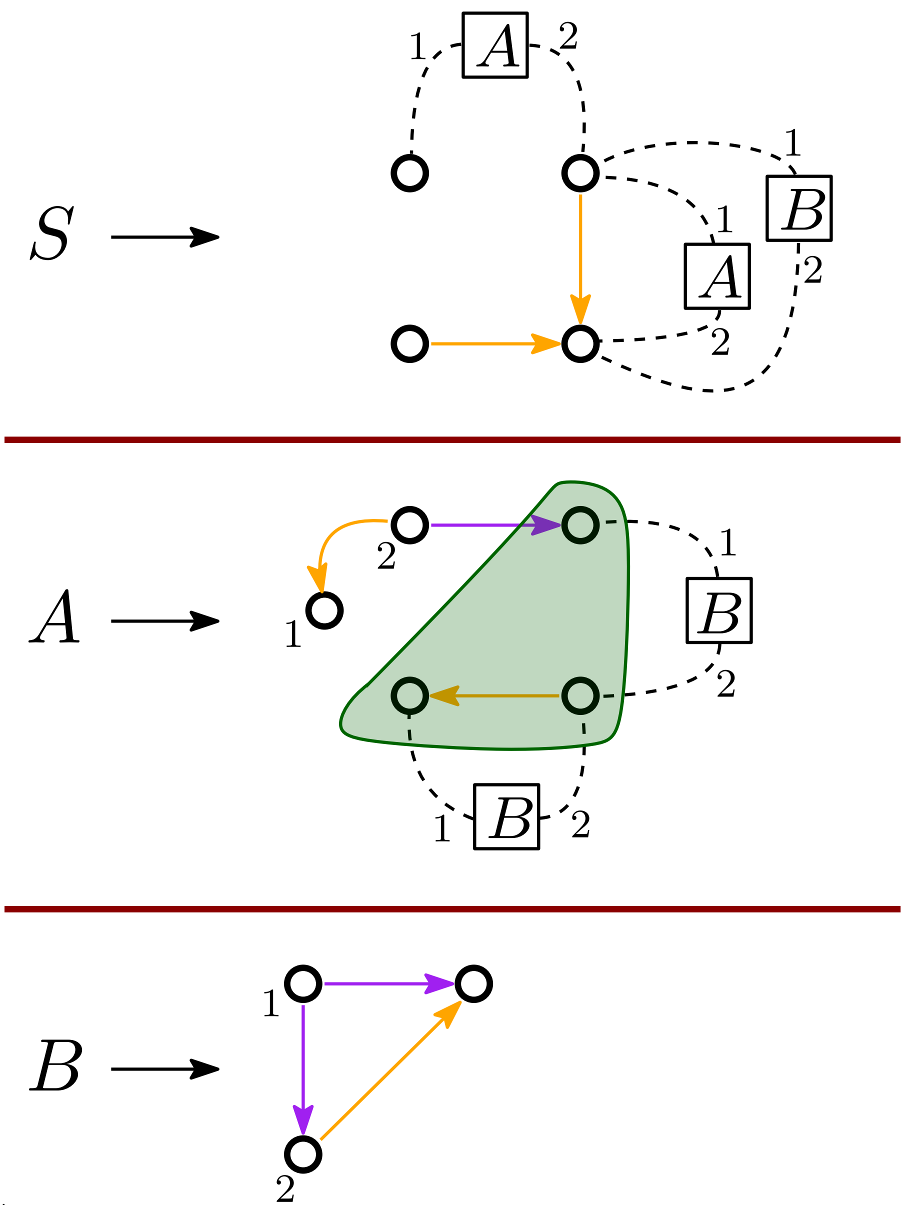
$S \longrightarrow$ 


$A \longrightarrow$ 

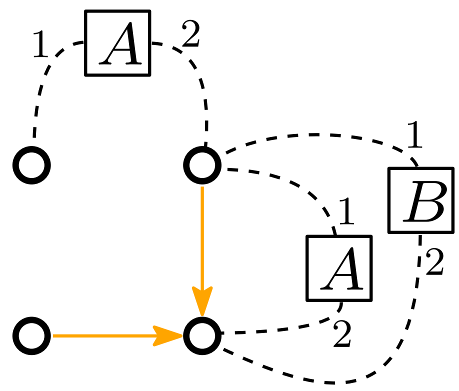

$B \longrightarrow$ 

# Straight-Line Programs for Relational Structures

Relational-SLP with $3$ rules:



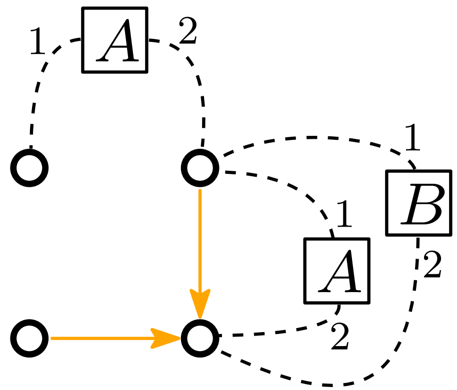

Derivation:



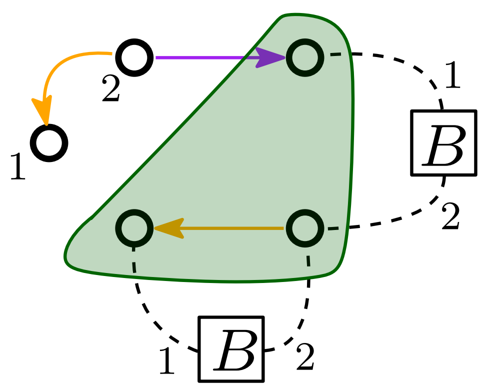
# Straight-Line Programs for Relational Structures

## Relational-SLP with $3$ rules:

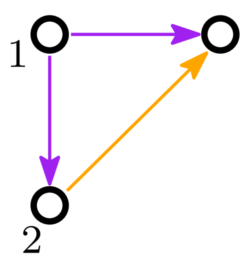
$S \longrightarrow$



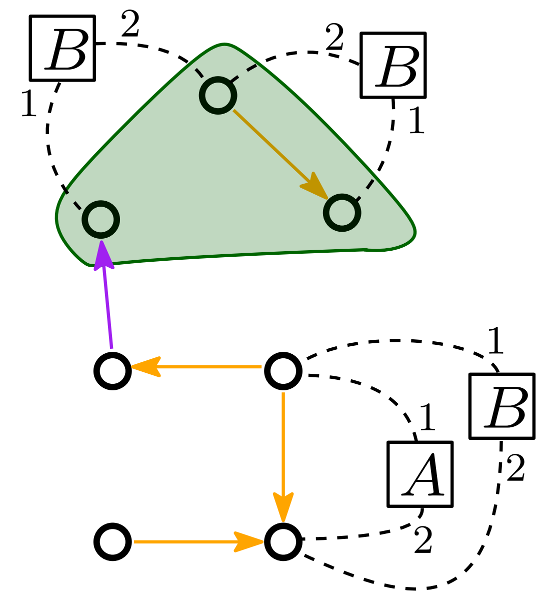

$A \longrightarrow$




$B \longrightarrow$



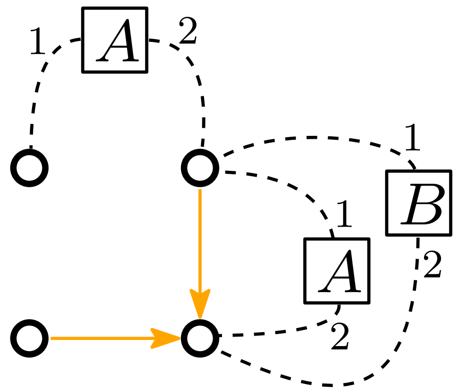

## Derivation:



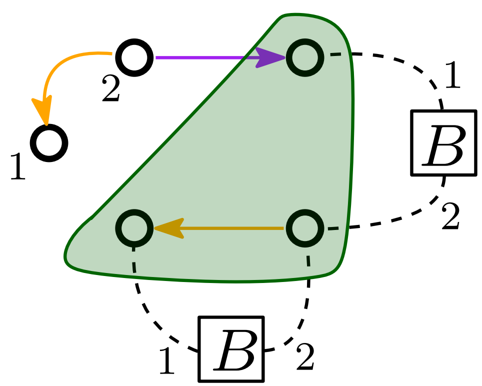
# Straight-Line Programs for Relational Structures

## Relational-SLP with $3$ rules:

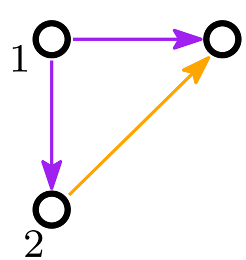
$S \longrightarrow$



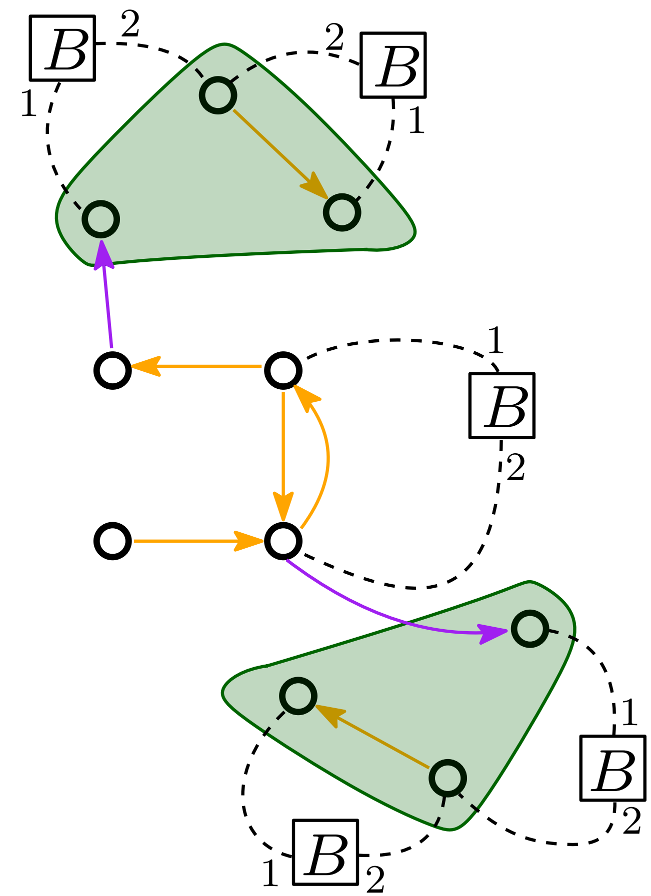

$A \longrightarrow$




$B \longrightarrow$

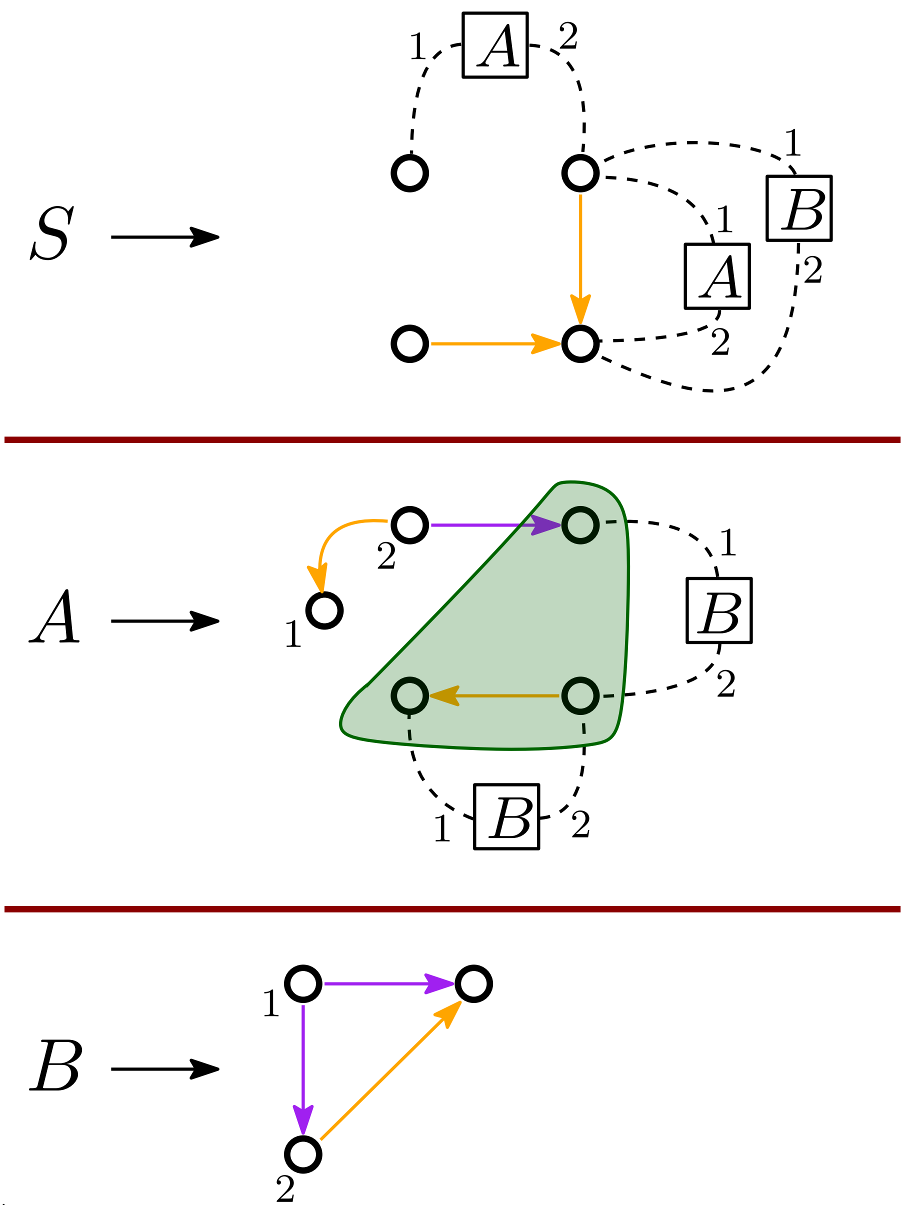



## Derivation:

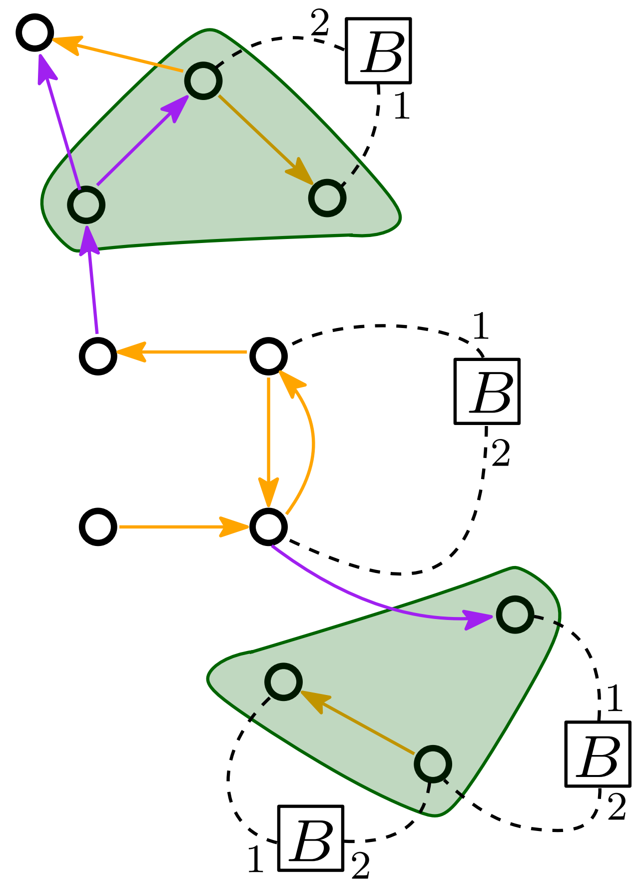


# Straight-Line Programs for Relational Structures

## Relational-SLP with $3$ rules:



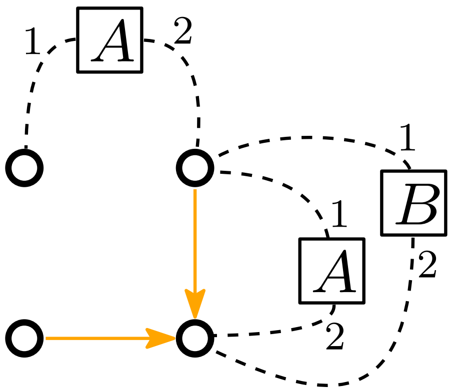

## Derivation:



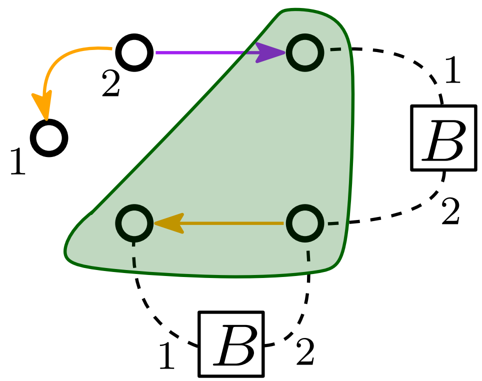
# Straight-Line Programs for Relational Structures

## Relational-SLP with $3$ rules:

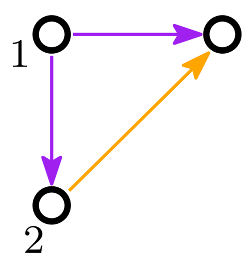
$S \longrightarrow$



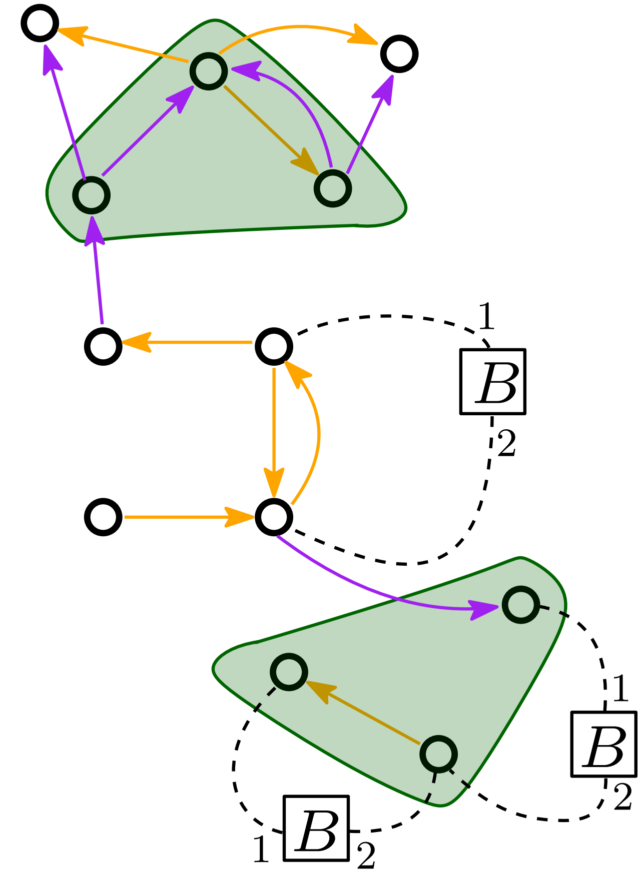

$A \longrightarrow$




$B \longrightarrow$



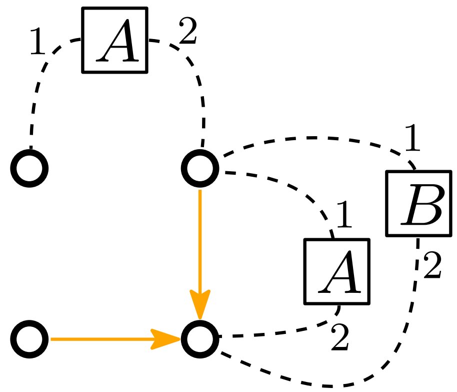

## Derivation:



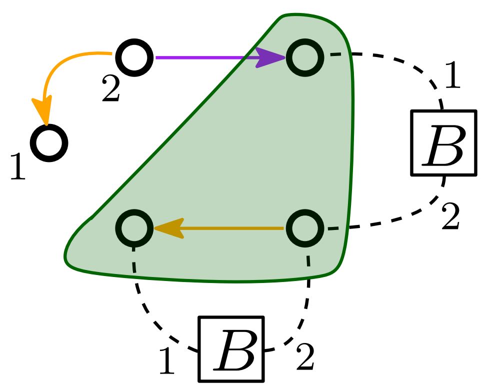
# Straight-Line Programs for Relational Structures

## Relational-SLP with $3$ rules:

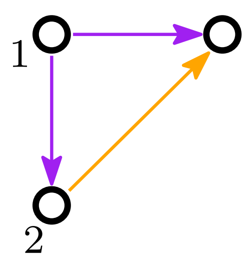
$S \longrightarrow$



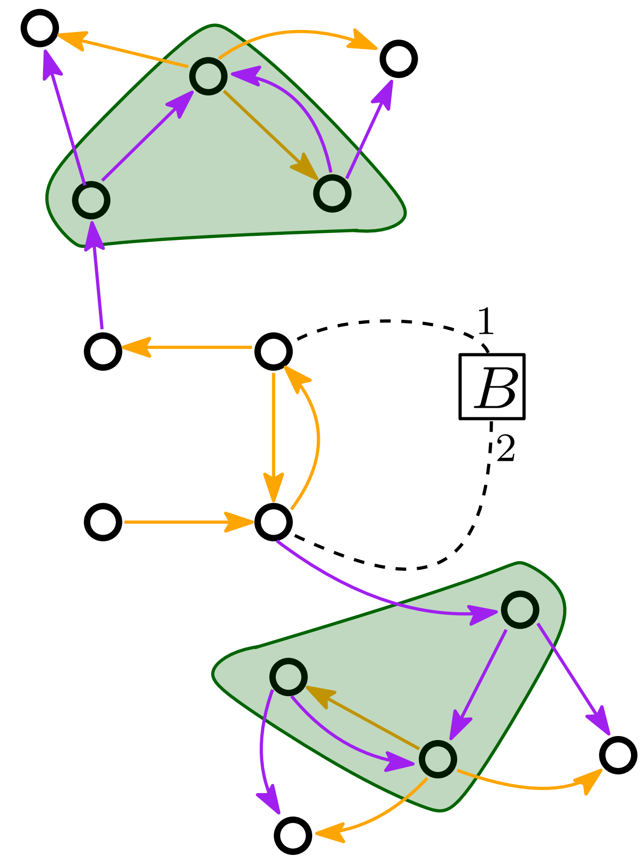

$A \longrightarrow$




$B \longrightarrow$



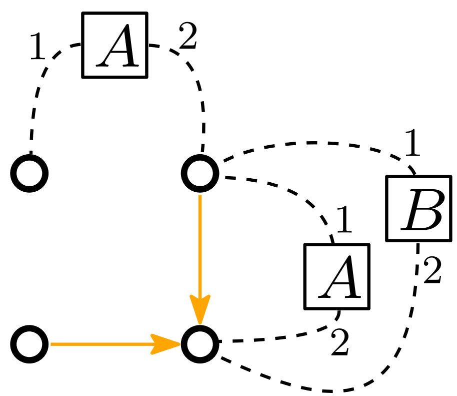

## Derivation:



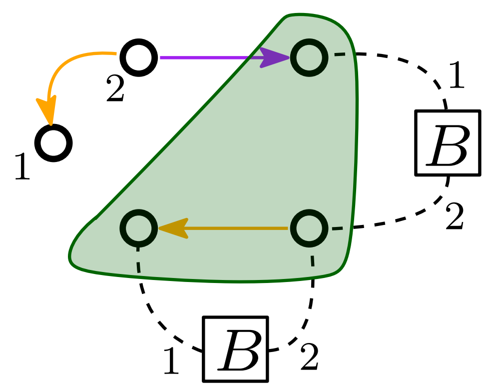
# Straight-Line Programs for Relational Structures

Relational-SLP with $3$ rules:

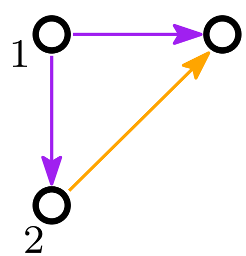
$S \longrightarrow$



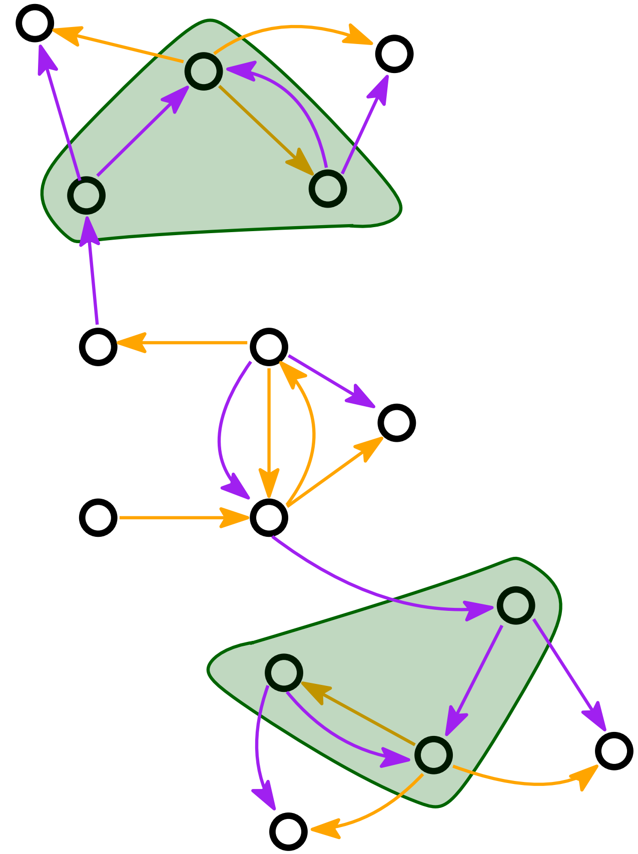

$A \longrightarrow$




$B \longrightarrow$




Derivation:



# FO Enumeration Over SLP-Compressed Relational Structures

Theorem:
Given an FO query $q(x_1, x_2, \ldots, x_m)$, a relational structure $D$ of bounded degree,
after preprocessing $O(|D|)$, we can enumerate $q(D)$ with constant delay.

[Durand, Grandjean, ACM ToCL 2007]

[Kazana, Segoufin LMCS 2011]

## FO Enumeration Over SLP-Compressed Relational Structures

**Theorem:**
Given an FO query $q(x_1, x_2, \ldots, x_m)$,
a relational structure $D$ of bounded degree,
after preprocessing $O(|D|)$, we can enumerate $q(D)$ with constant delay.

[Durand, Grandjean, ACM ToCL 2007]

[Kazana, Segoufin LMCS 2011]

**Theorem:**
Given an FO query $q(x_1, x_2, \ldots, x_m)$,
a relational-SLP S that compresses a relational structure $D$ of bounded degree,
after preprocessing $O(|S|)$, we can enumerate $q(D)$ with constant delay.

[Maneth, Lohrey, Schmid, MFCS 2025]

## FO Enumeration Over SLP-Compressed Relational Structures

**Theorem:**
Given an FO query $q(x_1, x_2, \ldots, x_m)$,
a relational structure $D$ of bounded degree,
after preprocessing $O(|D|)$, we can enumerate $q(D)$ with constant delay.

[Durand, Grandjean, ACM ToCL 2007]

[Kazana, Segoufin LMCS 2011]

**Theorem:**
Given an FO query $q(x_1, x_2, \ldots, x_m)$,
a relational-SLP S that compresses a relational structure $D$ of bounded degree,
after preprocessing $O(|S|)$, we can enumerate $q(D)$ with constant delay.

[Maneth, Lohrey, Schmid, MFCS 2025]

# Techniques

# Techniques

We apply concepts and tools from:

Formal languages (grammars, tree automata)

Logics (FO/MSO logic, Gaifman locality)

Algorithmics (enumeration algorithms, graph algorithms)

# Techniques

We apply concepts and tools from:

Formal languages (grammars, tree automata)

Logics (FO/MSO logic, Gaifman locality)

Algorithmics (enumeration algorithms, graph algorithms)

Path enumeration techniques for DAGs are crucial:

# Techniques

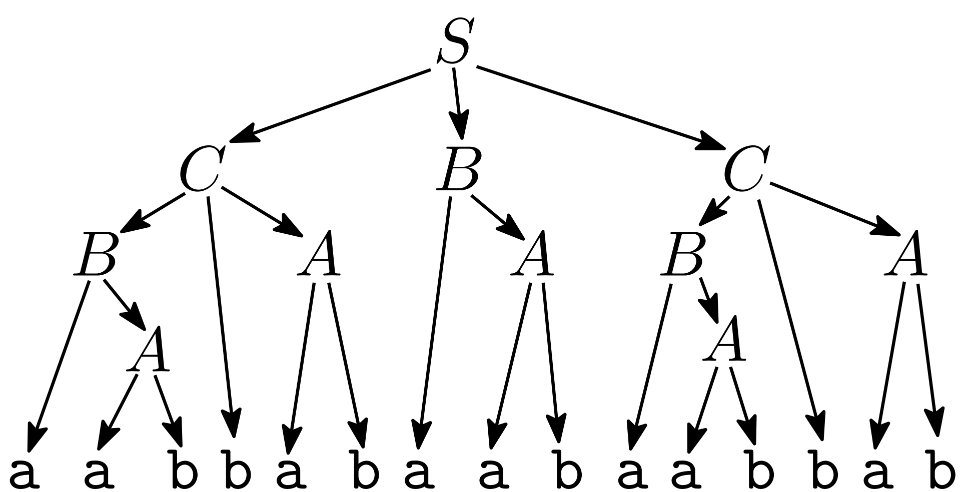
We apply concepts and tools from:

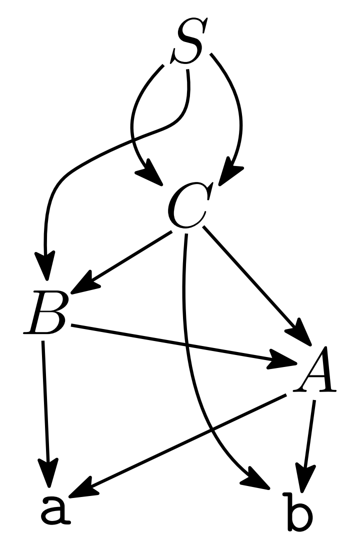
Formal languages (grammars, tree automata)

Logics (FO/MSO logic, Gaifman locality)

Algorithmics (enumeration algorithms, graph algorithms)

Path enumeration techniques for DAGs are crucial:



# Techniques

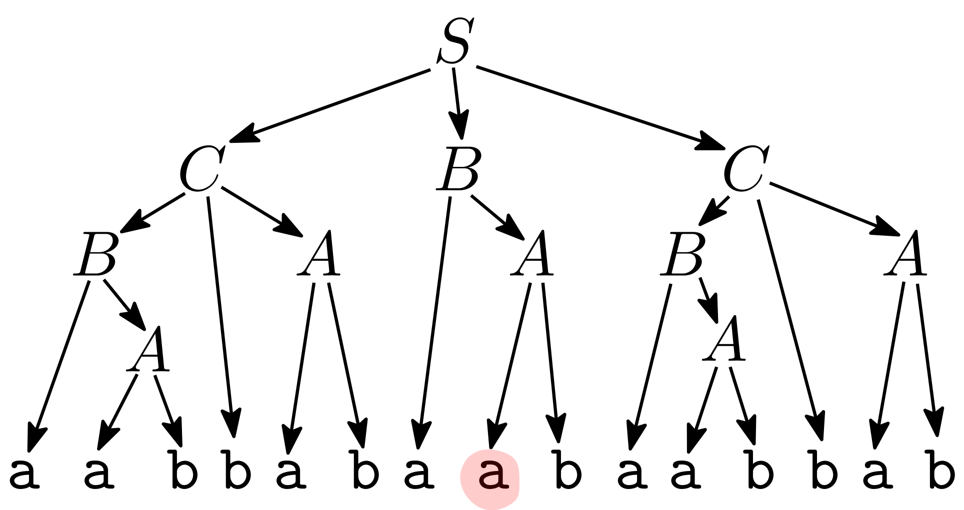
We apply concepts and tools from:

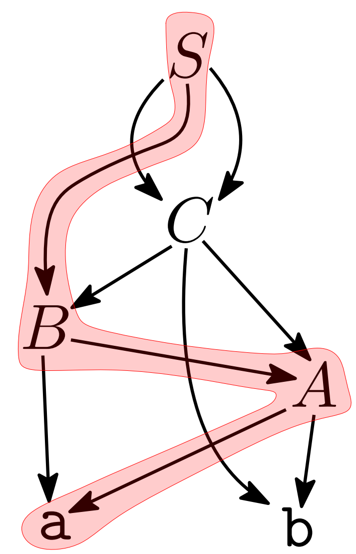
Formal languages (grammars, tree automata)

Logics (FO/MSO logic, Gaifman locality)

Algorithmics (enumeration algorithms, graph algorithms)

Path enumeration techniques for DAGs are crucial:



# Techniques

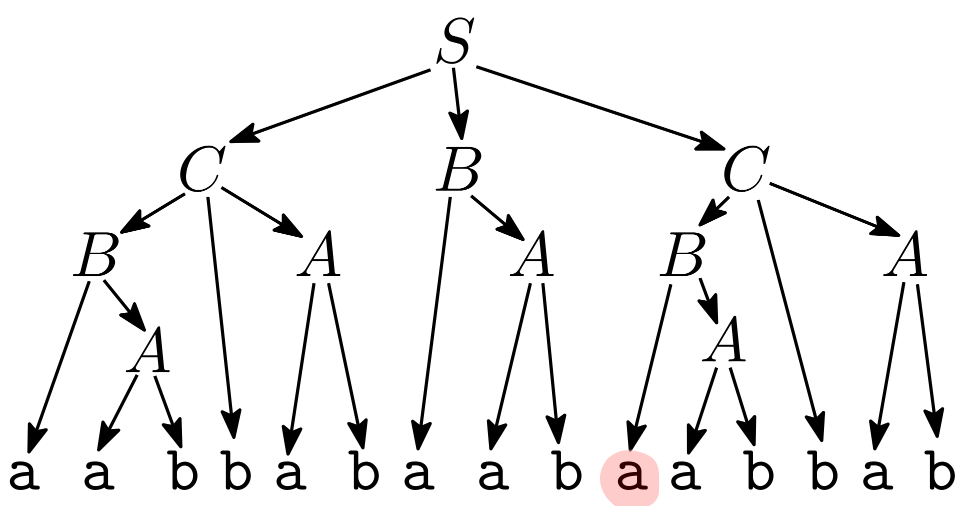
We apply concepts and tools from:

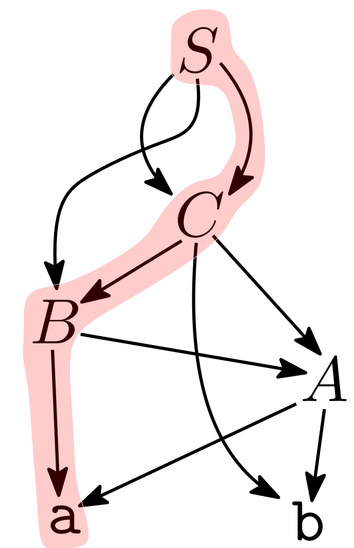
Formal languages (grammars, tree automata)

Logics (FO/MSO logic, Gaifman locality)

Algorithmics (enumeration algorithms, graph algorithms)

Path enumeration techniques for DAGs are crucial:



# Conclusions

What we learned: State-of-the-art query enumeration algorithms can be extended to the SLP-compressed setting!

# Conclusions

What we learned: State-of-the-art query enumeration algorithms can be extended to the SLP-compressed setting!

## Tasks for the future:

More such results, e.g., conjunctive queries (join expressions).
SLP-compression algorithms for relational structures.
Prototype implementation + experimental evaluation.

# Conclusions

What we learned: State-of-the-art query enumeration algorithms can be extended to the SLP-compressed setting!

## Tasks for the future:

More such results, e.g., conjunctive queries (join expressions).
SLP-compression algorithms for relational structures.
Prototype implementation + experimental evaluation.

## Other research topics I am interested in:

Database theory: information extraction, graph databases.
String algorithms and problems in formal languages.
Parameterised complexity.

# Conclusions

What we learned: State-of-the-art query enumeration algorithms can be extended to the SLP-compressed setting!

## Tasks for the future:

More such results, e.g., conjunctive queries (join expressions).
SLP-compression algorithms for relational structures.
Prototype implementation + experimental evaluation.

## Other research topics I am interested in:

Database theory: information extraction, graph databases.
String algorithms and problems in formal languages.
Parameterised complexity.

Thank you very much for your attention!