FO-Query Enumeration over SLP-Compressed Structures of Bounded Degree

Markus Lohrey
University of Siegen
Germany

Sebastian Maneth
University of Bremen
Germany

Markus L. Schmid
Humboldt University Berlin
Germany

MFCS 2025

FO-Query Evaluation

First order formula: $\psi(\underbrace{x_1, x_2, \ldots, x_k}_{\text{free variables}})$

Relational structure: $\mathcal{U} = (\underbrace{U}_{\text{universe}}, \underbrace{R_1, \ldots, R_s}_{\text{relations}}, \underbrace{c_1, \ldots, c_t}_{\text{constant}})$.

Result set: $\psi(\mathcal{U}) = \{(a_1, \ldots, a_k) \in U^k \mid \mathcal{U} \models \psi(a_1, \ldots, a_k)\}$.

FO-Query Evaluation

First order formula: $\psi(\underbrace{x_1, x_2, \ldots, x_k}_{\text{free variables}})$

Relational structure: $\mathcal{U} = (\underbrace{U}_{\text{universe}}, \underbrace{R_1, \ldots, R_s}_{\text{relations}}, \underbrace{c_1, \ldots, c_t}_{\text{constant}})$.

Result set: $\psi(\mathcal{U}) = \{(a_1, \ldots, a_k) \in U^k \mid \mathcal{U} \models \psi(a_1, \ldots, a_k)\}$.

$\rightsquigarrow$ Common abstraction of "SQL-queries over relational databases".

FO-Query Evaluation

First order formula: $\psi(\underbrace{x_1, x_2, \ldots, x_k}_{\text{free variables}})$

Relational structure: $\mathcal{U} = (\underbrace{U}_{\text{universe}}, \underbrace{R_1, \ldots, R_s}_{\text{relations}}, \underbrace{c_1, \ldots, c_t}_{\text{constant}})$.

Result set: $\psi(\mathcal{U}) = \{(a_1, \ldots, a_k) \in U^k \mid \mathcal{U} \models \psi(a_1, \ldots, a_k)\}$.

$\leadsto$ Common abstraction of "SQL-queries over relational databases".

Data Complexity

We consider ψ as constant, and measure complexity only in $|\mathcal{U}|$.

FO-Query Evaluation

First order formula: $\psi(\underbrace{x_1, x_2, \ldots, x_k}_{\text{free variables}})$

Relational structure: $\mathcal{U} = (\underbrace{U}_{\text{universe}}, \underbrace{R_1, \ldots, R_s}_{\text{relations}}, \underbrace{c_1, \ldots, c_t}_{\text{constant}})$.

Result set: $\psi(\mathcal{U}) = \{(a_1, \ldots, a_k) \in U^k \mid \mathcal{U} \models \psi(a_1, \ldots, a_k)\}$.

$\rightsquigarrow$ Common abstraction of "SQL-queries over relational databases".

Data Complexity

We consider ψ as constant, and measure complexity only in $|\mathcal{U}|$.

For simplicity, let's talk about directed graphs instead of structures.

Everything said for graphs also holds for general structures!

FO-Query Evaluation

First order formula: $\psi(\underbrace{x_1, x_2, \ldots, x_k}_{\text{free variables}})$

Directed graph: $\mathcal{G} = (V, E)$.

Result set: $\psi(\mathcal{G}) = \{(a_1, \ldots, a_k) \in V^k \mid \mathcal{G} \models \psi(a_1, \ldots, a_k)\}$.

$\rightsquigarrow$ Common abstraction of "SQL-queries over relational databases".

Data Complexity

We consider ψ as constant, and measure complexity only in $|\mathcal{U}|$.

For simplicity, let's talk about directed graphs instead of structures.

Everything said for graphs also holds for general structures!

A More Practical Perspective: Enumeration

$$\psi(x_1, \ldots, x_k) \quad + \quad \mathcal{G}$$

$$\Big\downarrow \text{ Preprocessing}$$

$$\mathsf{D}_{\psi,\mathcal{G}}$$

$$\Big\downarrow \text{ Enumeration}$$

$$\psi(\mathcal{G}) = \; (a, b, a, c, \ldots)$$

$$(d, b, d, a, \ldots)$$

$$(b, d, c, c, \ldots)$$

$$\vdots$$

A More Practical Perspective: Enumeration

$$\psi(x_1, \ldots, x_k) \quad + \quad \mathcal{G}$$

$\downarrow$ Preprocessing

$$\mathsf{D}_{\psi,\mathcal{G}}$$

$\downarrow$ Enumeration

$$\psi(\mathcal{G}) = (a, b, a, c, \ldots)$$

delay

$$(d, b, d, a, \ldots)$$

delay

$$(b, d, c, c, \ldots)$$

$$\vdots$$

A More Practical Perspective: Enumeration

$$\psi(x_1, \ldots, x_k) \quad + \quad \mathcal{G}$$

↓ Preprocessing $O(|\mathcal{G}|)$ (linear in data complexity)

$$\mathsf{D}_{\psi,\mathcal{G}}$$

↓ Enumeration

$$\psi(\mathcal{G}) = \ (a, b, a, c, \ldots)$$

delay $O(1)$ (constant in data complexity)

$$(d, b, d, a, \ldots)$$

delay $O(1)$ (constant in data complexity)

$$(b, d, c, c, \ldots)$$

$$\vdots$$

FO-Query Evaluation Over Degree Bounded Structures

Theorem — Durand, Grandjean, ACM TOCL 2007; Kazana, Segoufin, LMCS 2011

Let d be a constant. FO-queries over graphs of degree at most d can be enumerated with linear preprocessing and constant delay.

FO-Query Evaluation Over Degree Bounded Structures

Theorem Durand, Grandjean, ACM TOCL 2007
Kazana, Segoufin, LMCS 2011

Let d be a constant. FO-queries over graphs of degree at most d can be enumerated with linear preprocessing and constant delay.

Without the degree bound this is most likely not possible!

FO-Query Evaluation Over Degree Bounded Structures

Theorem — Durand, Grandjean, ACM TOCL 2007; Kazana, Segoufin, LMCS 2011

Let d be a constant. FO-queries over graphs of degree at most d can be enumerated with linear preprocessing and constant delay.

Without the degree bound this is most likely not possible!

Other approaches for linear preprocessing and constant delay:

Restrict the class of structures
(e.g. low degree, bounded expansion, nowhere dense)

Restrict the class of queries
(e.g. acyclic conjunctive queries)

Algorithmics on Compressed Data

General idea

Find algorithms that solve problems directly on compressed instances (without decompressing the instance).

Enumeration in the Compressed Setting

$\psi(x_1, \ldots, x_k)$ + **compression** D of $\mathcal{G}$

↓ Preprocessing $O(|D|)$ (linear in data complexity ... ... but wrt. the **compressed** size)

$\mathsf{D}_{\psi,D}$

↓ Enumeration

$\psi(\mathcal{G}) = (a, b, a, c, \ldots)$

delay $O(1)$ (constant in data complexity)

$(d, b, d, a, \ldots)$

delay $O(1)$ (constant in data complexity)

$(b, d, c, c, \ldots)$

$\vdots$

Straight-Line Programs (SLPs)

Main Idea

Compress input $\mathcal{I}$ as a context-free grammar G that describes exactly this object, i. e., $L(G) = \{\mathcal{I}\}$ (or $\mathrm{val}(G) = \mathcal{I}$ for short).

Straight-Line Programs (SLPs)

Main Idea

Compress input $\mathcal{I}$ as a context-free grammar G that describes exactly this object, i. e., $L(G) = \{\mathcal{I}\}$ (or $\text{val}(G) = \mathcal{I}$ for short).

SLP-framework is very well-established for strings:

Straight-Line Programs (SLPs)

Main Idea

Compress input $\mathcal{I}$ as a context-free grammar G that describes exactly this object, i. e., $L(G) = \{\mathcal{I}\}$ (or $\mathrm{val}(G) = \mathcal{I}$ for short).

SLP-framework is very well-established for strings:

$$w = \mathtt{a\,a\,b\,b\,a\,b\,a\,a\,b\,a\,a\,b\,b\,a\,b}$$

Straight-Line Programs (SLPs)

Main Idea

Compress input $\mathcal{I}$ as a context-free grammar G that describes exactly this object, i. e., $L(G) = \{\mathcal{I}\}$ (or $\mathsf{val}(G) = \mathcal{I}$ for short).

SLP-framework is very well-established for strings:

$$w = \mathtt{a\,a\,b\,b\,a\,b\,a\,a\,b\,a\,a\,b\,b\,a\,b}$$

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

Straight-Line Programs (SLPs)

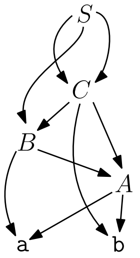
Main Idea

Compress input $\mathcal{I}$ as a context-free grammar G that describes exactly this object, i. e., $L(G) = \{\mathcal{I}\}$ (or $\mathrm{val}(G) = \mathcal{I}$ for short).

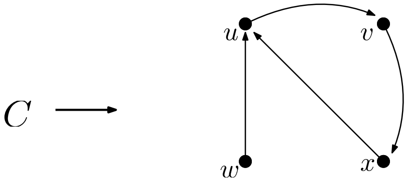
SLP-framework is very well-established for strings:

$$w = \mathtt{a\,a\,b\,b\,a\,b\,a\,a\,b\,a\,a\,b\,b\,a\,b}$$

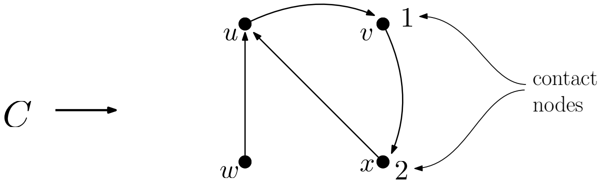
$$S \to C\,B\,C, \qquad B \to \mathtt{a}\,A$$
$$C \to B\,\mathtt{b}\,A, \qquad A \to \mathtt{a}\,\mathtt{b}$$



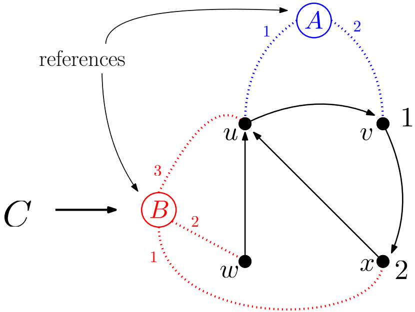
SLPs for Graphs



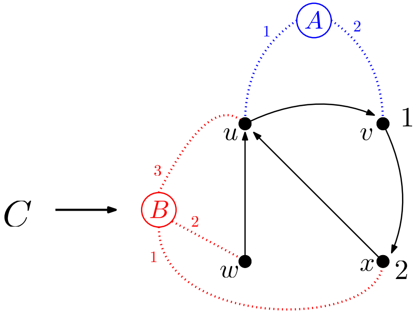
SLPs for Graphs

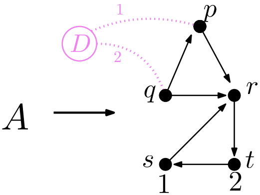


SLPs for Graphs

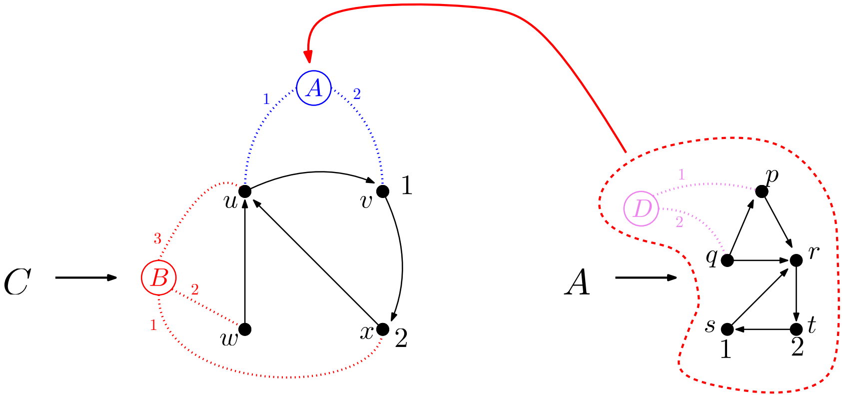


SLPs for Graphs

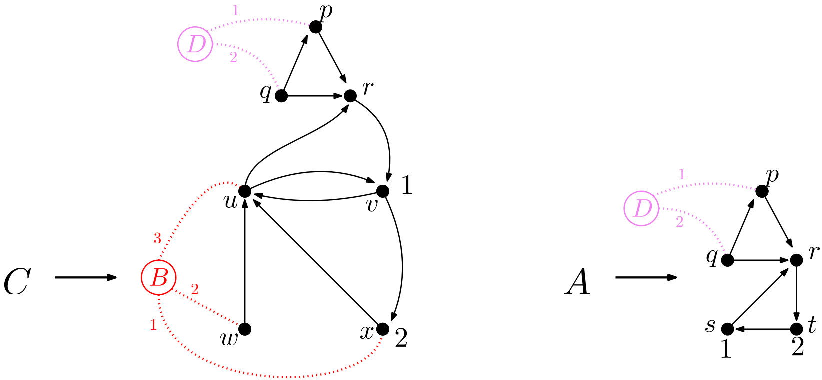




SLPs for Graphs

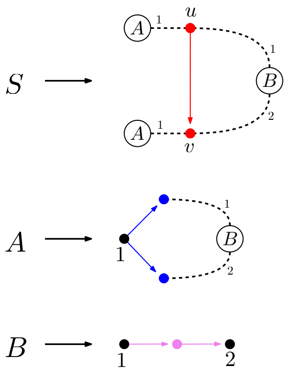


SLPs for Graphs



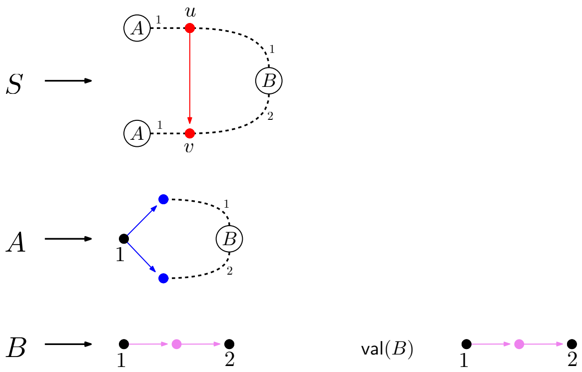
SLPs for Graphs

Graph SLP D



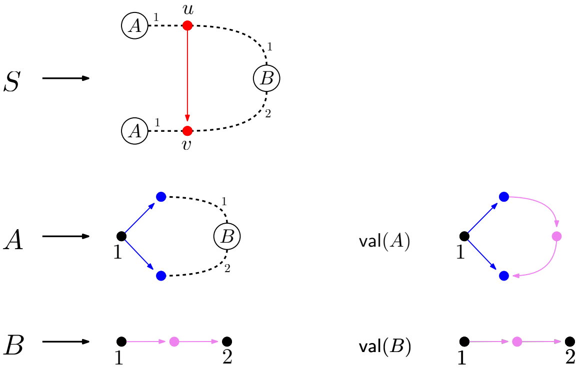
SLPs for Graphs

Graph SLP D



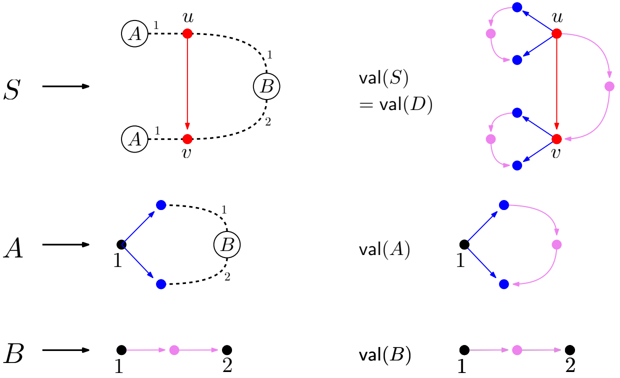
SLPs for Graphs

Graph SLP D



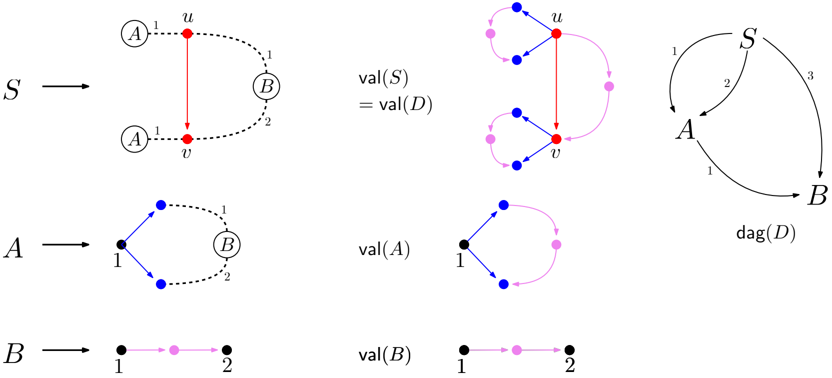
SLPs for Graphs

Graph SLP D



SLPs for Graphs

Graph SLP D



What we want ...

$\psi(x_1, \ldots, x_k)$ + graph SLP D such that $\mathsf{eval}(D)$ has degree at most d

Preprocessing $O(|D|)$ (linear in data complexity ... ... but wrt. the **compressed** size)

$\mathsf{D}_{\psi,D}$

Enumeration

$\psi(\mathsf{eval}(D)) = (a, b, a, c, \ldots)$

delay $O(1)$ (constant in data complexity)

$(d, b, d, a, \ldots)$

delay $O(1)$ (constant in data complexity)

$(b, d, c, c, \ldots)$

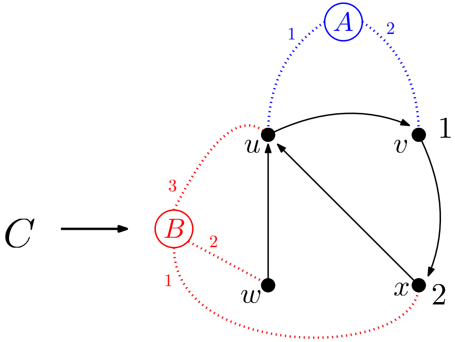
$\vdots$

. . . and the problem with that.

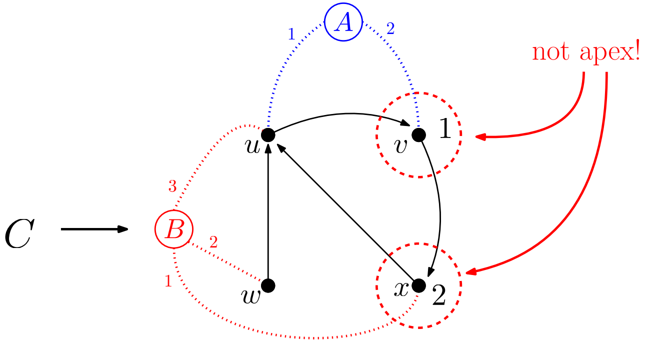
Theorem — Lohrey, JCSS 2012

- There is a fixed FO-formula for which model checking for SLP-compressed graphs is “intractable”.
- Model checking for SLP-compressed graphs is in NL for every fixed FO-formula, if the SLP has the **apex property**.

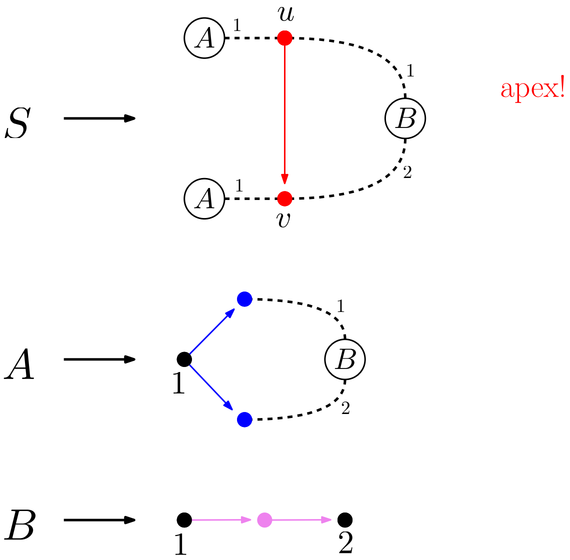
The Apex Property



The Apex Property



The Apex Property



Our Main Result

$\psi(x_1, \ldots, x_k)$ + **apex** graph SLP D such that $\mathsf{eval}(D)$ has degree at most d

↓ Preprocessing $O(|D|)$ (linear in data complexity … … but wrt. the **compressed** size)

$\mathsf{D}_{\psi,D}$

↓ Enumeration

$\psi(\mathsf{eval}(D)) = (a, b, a, c, \ldots)$

delay $O(1)$ (constant in data complexity)

$(d, b, d, a, \ldots)$

delay $O(1)$ (constant in data complexity)

$(b, d, c, c, \ldots)$

$\vdots$

Related Results

Similar results in the recent literature:

- MSO-query enumeration on SLP-compressed strings
  [S., Schweikardt, PODS 2021/2022]
  [Munoz, Riveros, ICDT 2023]
- MSO-query enumeration on SLP-compressed trees
  [Lohrey, S., PODS 2024]

Proof Roadmap

Step 1: Reduction to a simpler enumeration problem.

Step 2: Solve this problem in the **uncompressed** setting.

Step 3: Extension to the SLP-compressed setting.

Proof Roadmap

Step 1: Reduction to a simpler enumeration problem.

Standard application of Gaifman-locality of FO-logic

Step 2: Solve this problem in the **uncompressed** setting.

Step 3: Extension to the SLP-compressed setting.

Proof Roadmap

Step 1: Reduction to a simpler enumeration problem.

Standard application of Gaifman-locality of FO-logic

Step 2: Solve this problem in the **uncompressed** setting.

Kazana, Segoufin, LMCS 2011

Step 3: Extension to the SLP-compressed setting.

Proof Roadmap

Step 1: Reduction to a simpler enumeration problem.

Standard application of Gaifman-locality of FO-logic

Step 2: Solve this problem in the **uncompressed** setting.

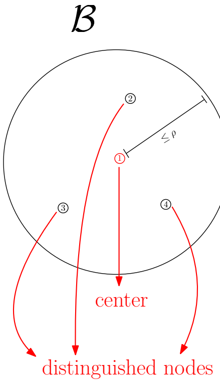
Kazana, Segoufin, LMCS 2011

Step 3: Extension to the SLP-compressed setting.

Our contribution

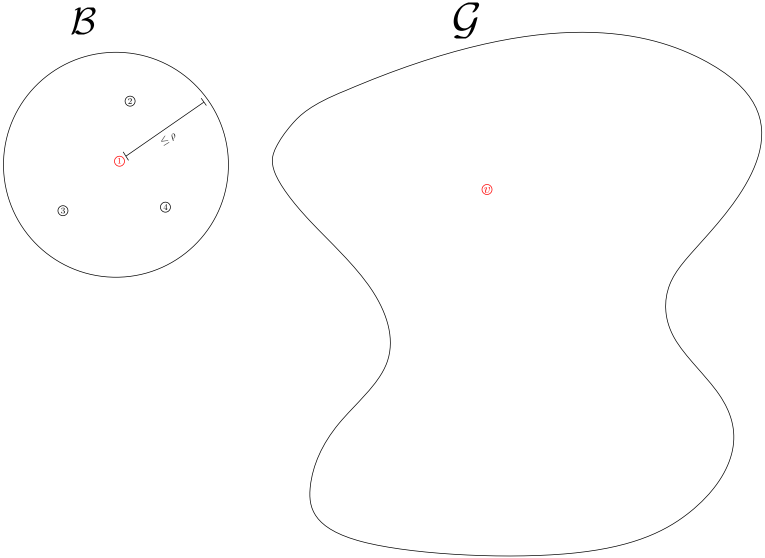
Step 1 – Reduction (ρ-Neighbourhood Types)

$r, \rho \in \mathbb{N}$ with $r \leq \rho$ are some fixed constants



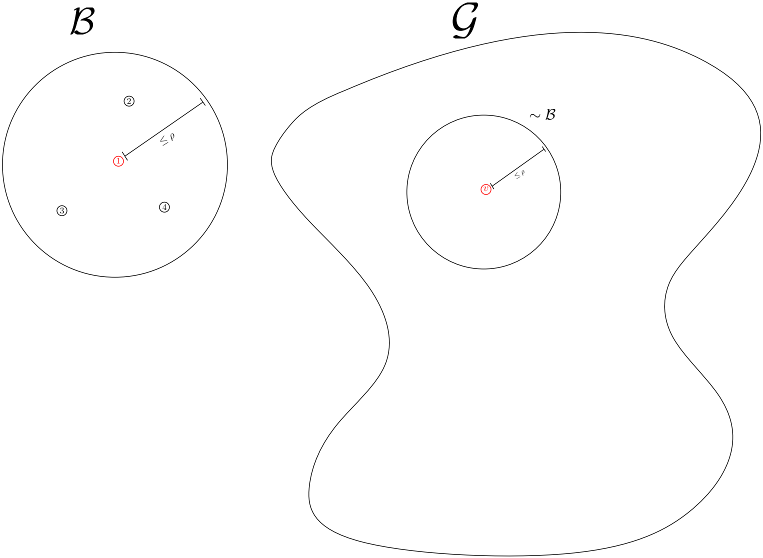
Step 1 – Reduction (ρ-Neighbourhood Types)

$r, \rho \in \mathbb{N}$ with $r \leq \rho$ are some fixed constants



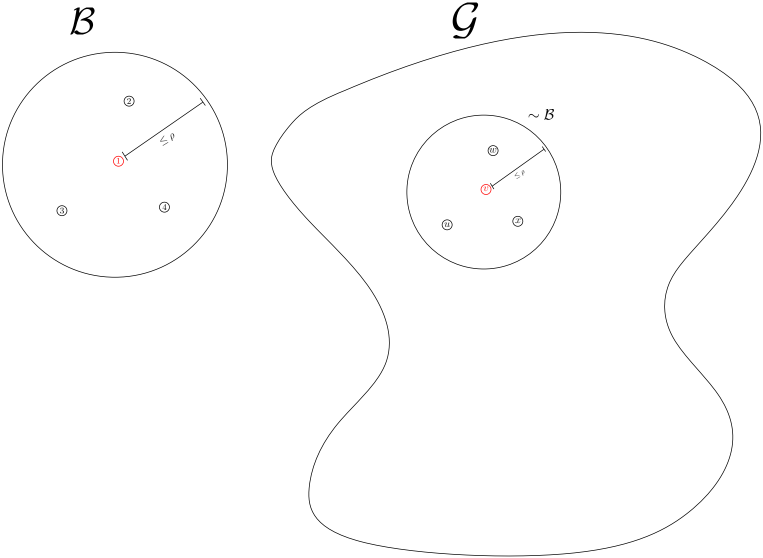
Step 1 – Reduction (ρ-Neighbourhood Types)

$r, \rho \in \mathbb{N}$ with $r \leq \rho$ are some fixed constants



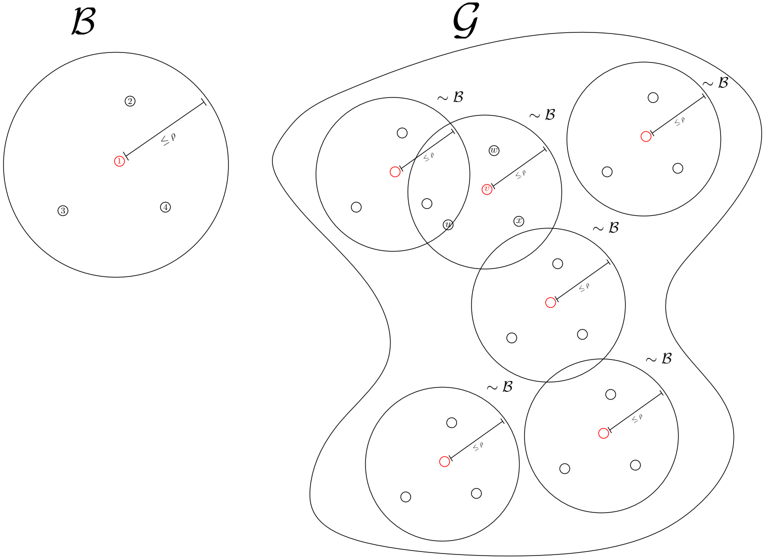
Step 1 – Reduction (ρ-Neighbourhood Types)

$r, \rho \in \mathbb{N}$ with $r \leq \rho$ are some fixed constants



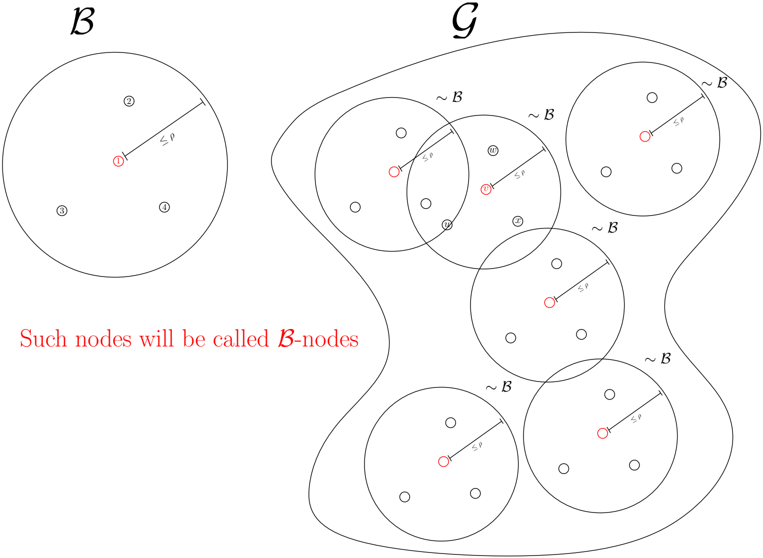
Step 1 – Reduction (ρ-Neighbourhood Types)

$r, \rho \in \mathbb{N}$ with $r \leq \rho$ are some fixed constants



Step 1 – Reduction (ρ-Neighbourhood Types)

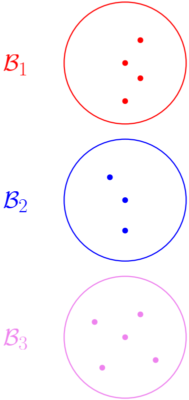
$r, \rho \in \mathbb{N}$ with $r \leq \rho$ are some fixed constants

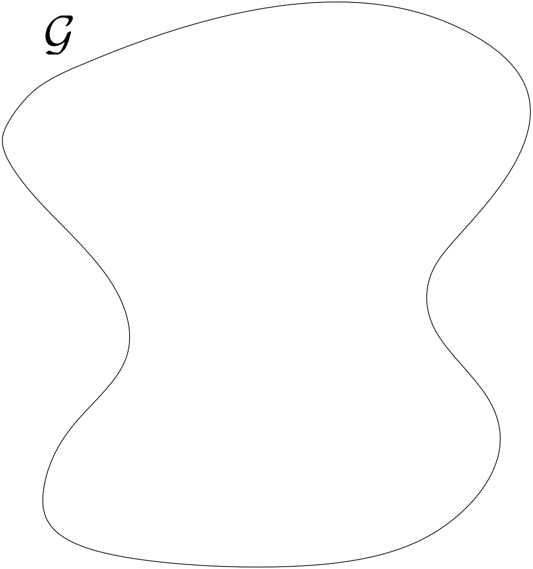


Such nodes will be called $\mathcal{B}$-nodes

Step 1 – Reduction (Admissible Tuples)

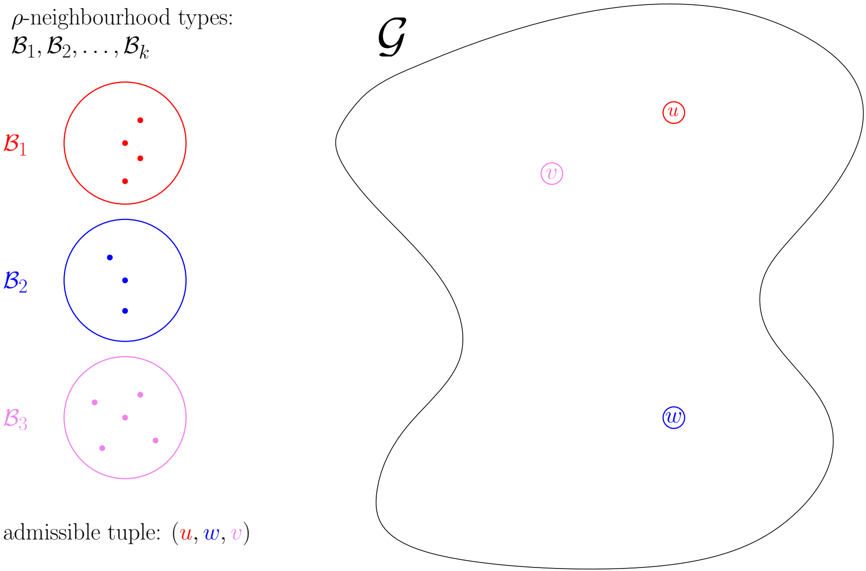
ρ-neighbourhood types:
$\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k$





Step 1 – Reduction (Admissible Tuples)

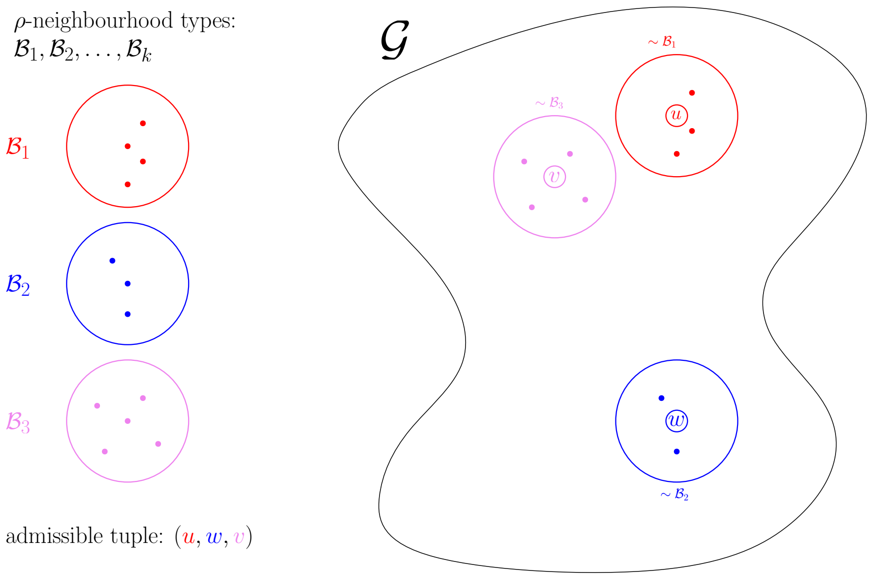
ρ-neighbourhood types:
$\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k$



admissible tuple: (u, w, v)

Step 1 – Reduction (Admissible Tuples)

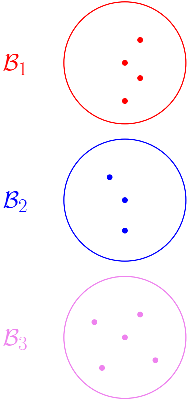
ρ-neighbourhood types:
$\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k$

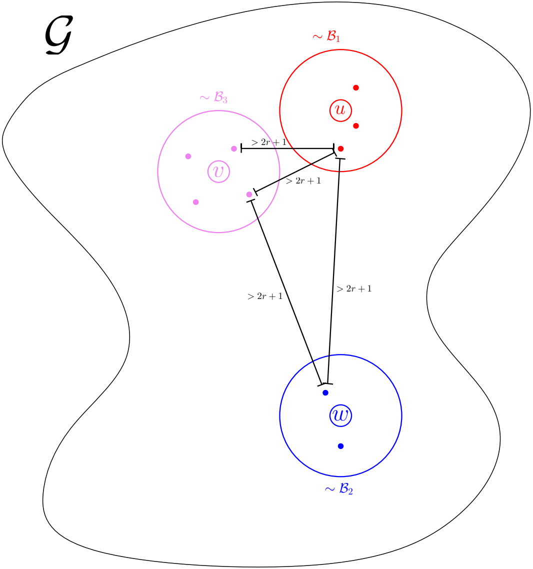


admissible tuple: (u, w, v)

Step 1 – Reduction (Admissible Tuples)

ρ-neighbourhood types:
$\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k$

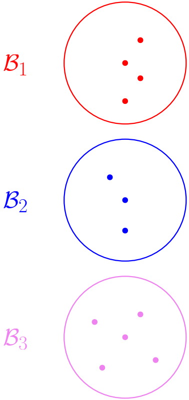


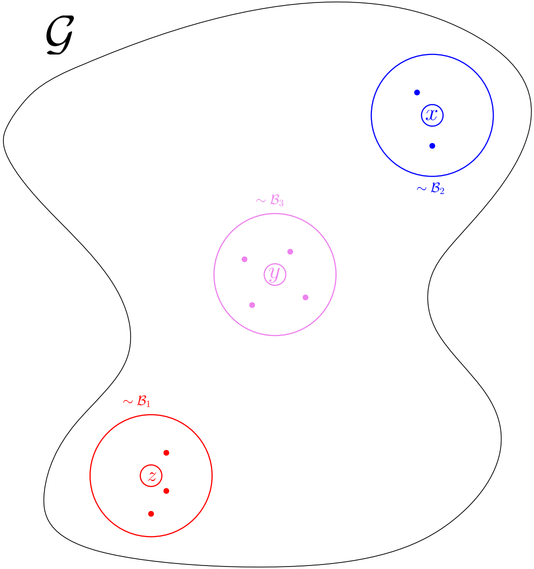


admissible tuple: (u, w, v)

Step 1 – Reduction (Admissible Tuples)

ρ-neighbourhood types:
$\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k$

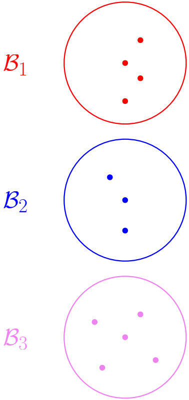




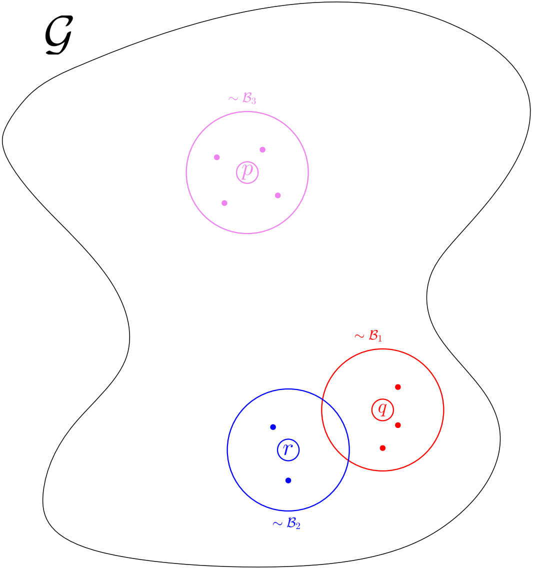
admissible tuple: (z, x, y)

Step 1 – Reduction (Admissible Tuples)

ρ-neighbourhood types:
$\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k$

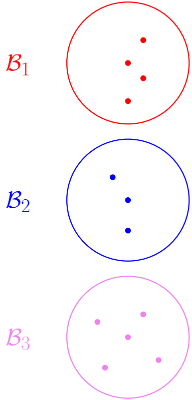


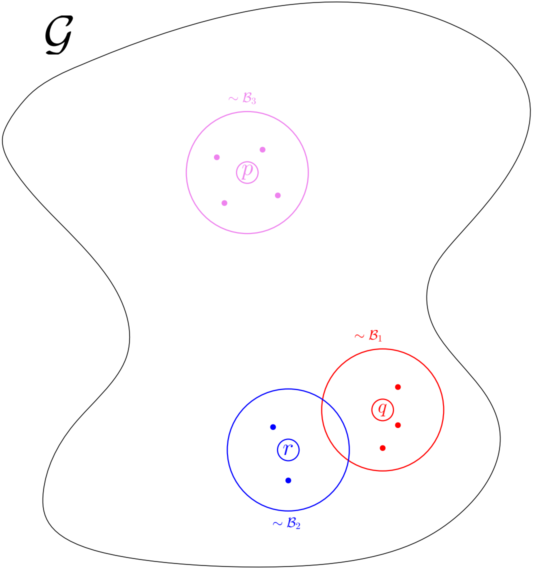
admissible tuple: (q, r, p)



Step 1 – Reduction (Admissible Tuples)

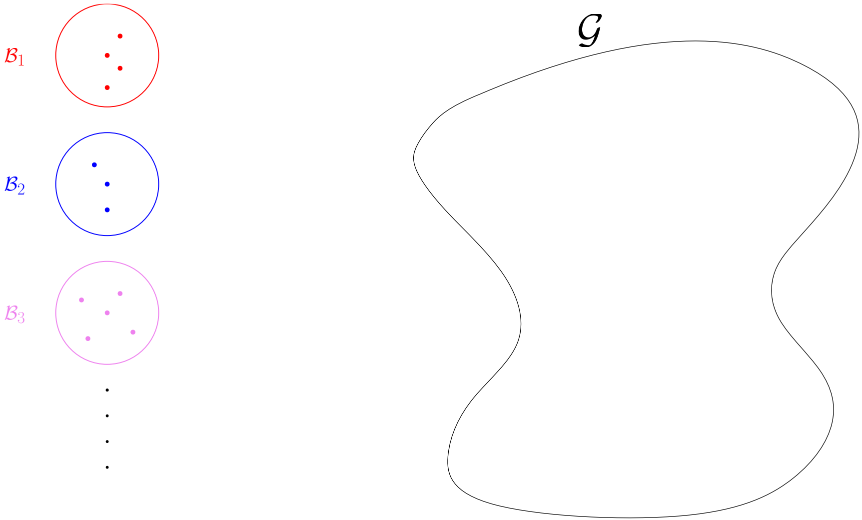
ρ-neighbourhood types:
$\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_k$





Task: After linear preprocessing, enumerate admissible tuples with constant delay.

Step 2 – Uncompressed Setting (Algorithm)



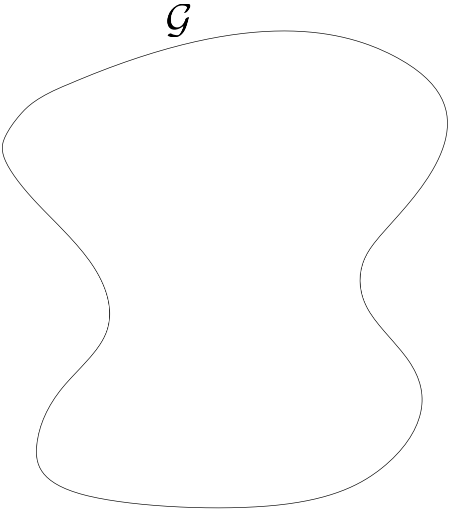
Step 2 – Uncompressed Setting (Algorithm)

$L_{\mathcal{B}_1}$ u v w $\cdots$

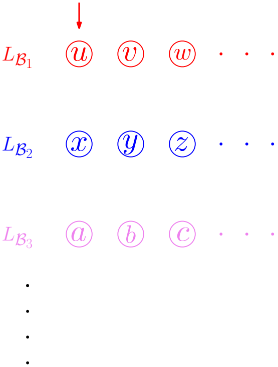
$L_{\mathcal{B}_2}$ x y z $\cdots$

$L_{\mathcal{B}_3}$ a b c $\cdots$

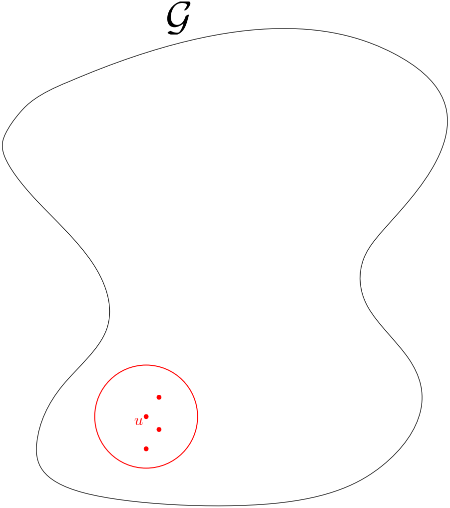
$\vdots$



Step 2 – Uncompressed Setting (Algorithm)



output tuple: $(u, \ldots)$



Step 2 – Uncompressed Setting (Algorithm)

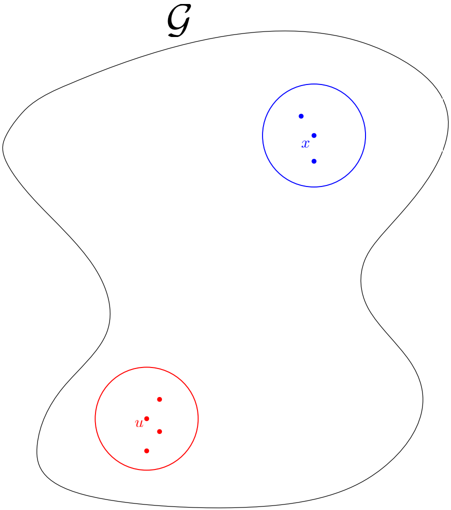
$L_{\mathcal{B}_1}$ ↓ⓤ ⓥ ⓦ $\cdots$

$L_{\mathcal{B}_2}$ ↓ⓧ ⓨ ⓩ $\cdots$

$L_{\mathcal{B}_3}$ ⓐ ⓑ ⓒ $\cdots$

$\vdots$

output tuple: $(u, x, \ldots)$



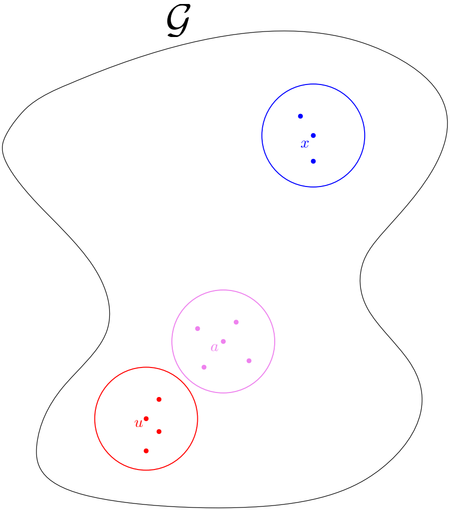
Step 2 – Uncompressed Setting (Algorithm)

$L_{\mathcal{B}_1}$ u v w $\cdots$

$L_{\mathcal{B}_2}$ x y z $\cdots$

$L_{\mathcal{B}_3}$ a b c $\cdots$

$\vdots$



output tuple: $(u, x, a, \ldots)$

Step 2 – Uncompressed Setting (Algorithm)

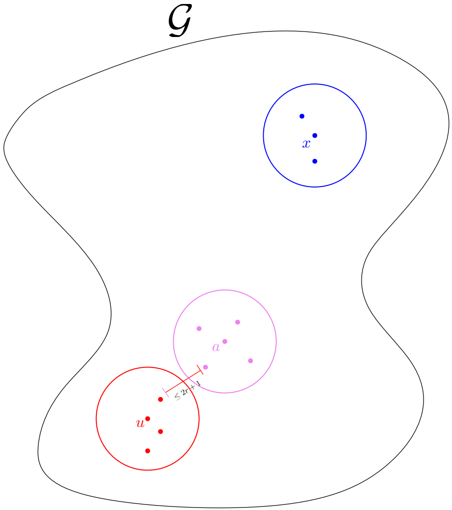
$L_{\mathcal{B}_1}$ u v w $\cdots$

$L_{\mathcal{B}_2}$ x y z $\cdots$

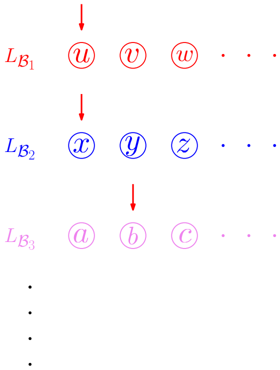
$L_{\mathcal{B}_3}$ a b c $\cdots$

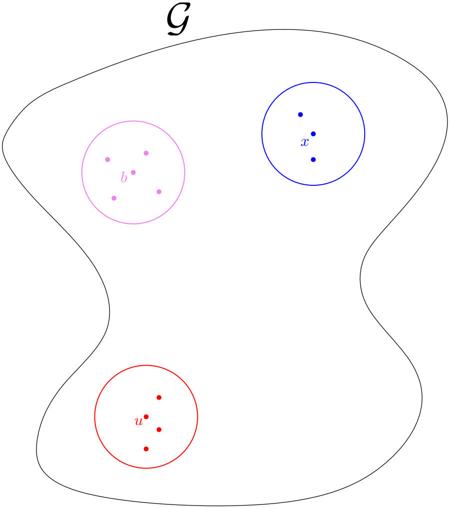
$\vdots$

output tuple: $(u, x, a, \ldots)$



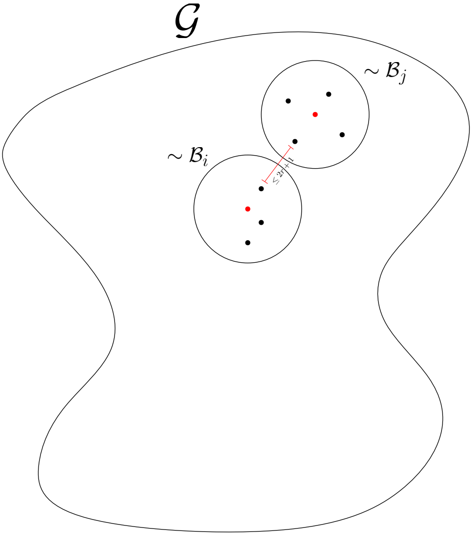
Step 2 – Uncompressed Setting (Algorithm)



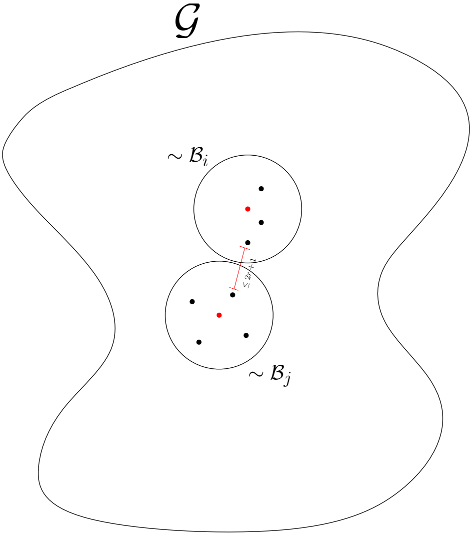


output tuple: $(u, x, b, \ldots)$

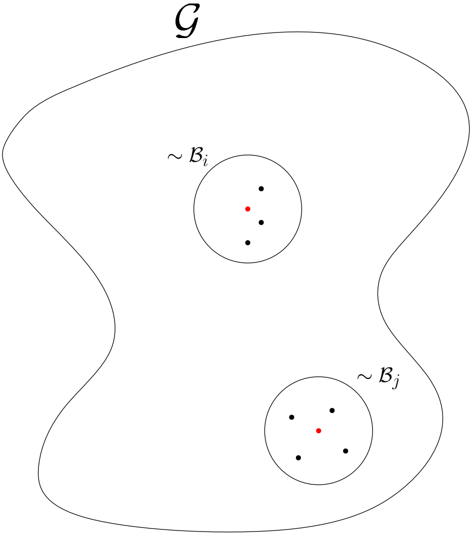
Step 2 – Uncompressed Setting (Delay)



Step 2 – Uncompressed Setting (Delay)



Step 2 – Uncompressed Setting (Delay)



Step 3 – Compressed Setting

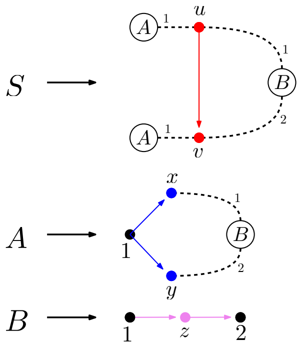
Three main challenges:

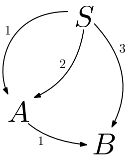
1. How do we represent nodes from the structures?

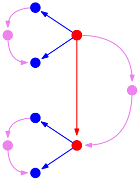
2. How do we represent ρ-neighbourhoods of elements?

3. How do we enumerate admissible tuples?

Step 3 – Compressed Setting (Node Representation)

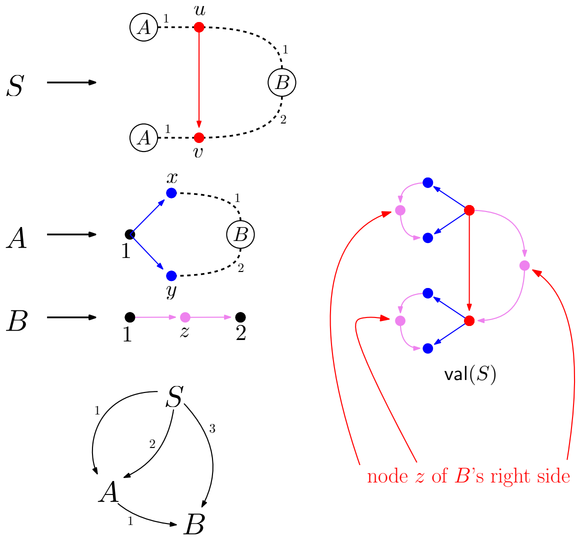




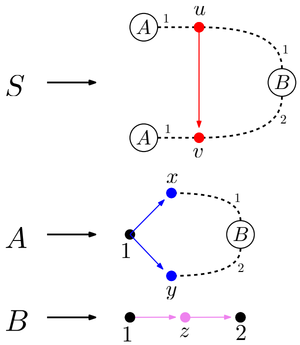


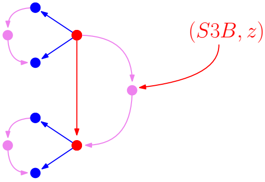
val(S)

Step 3 – Compressed Setting (Node Representation)

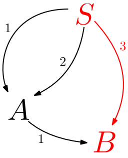


Step 3 – Compressed Setting (Node Representation)

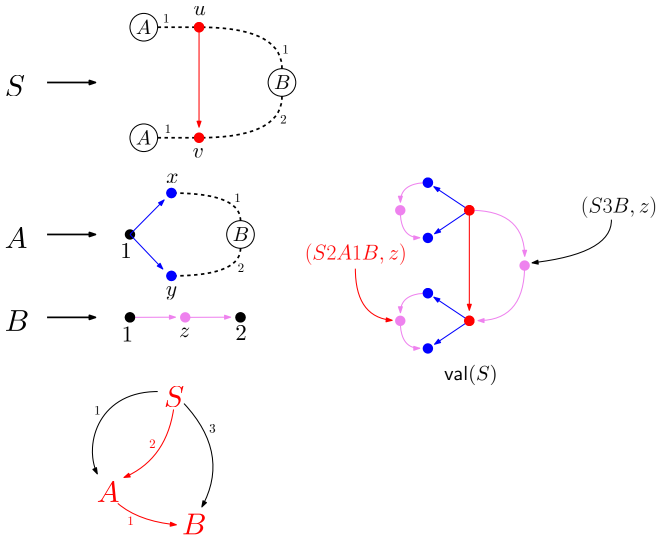




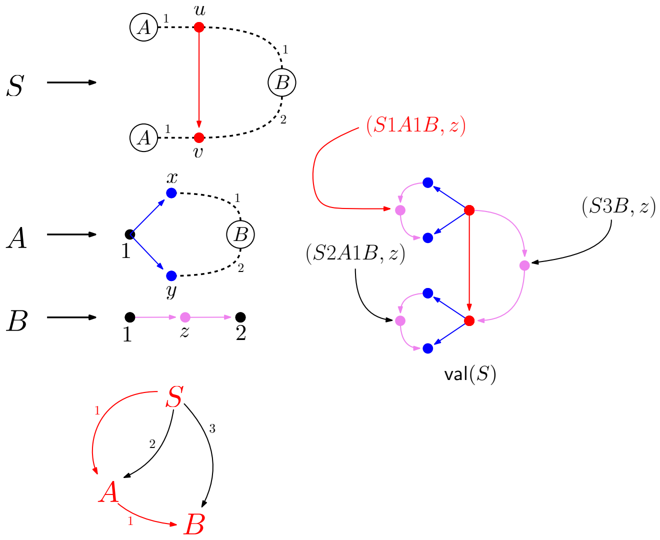
val(S)



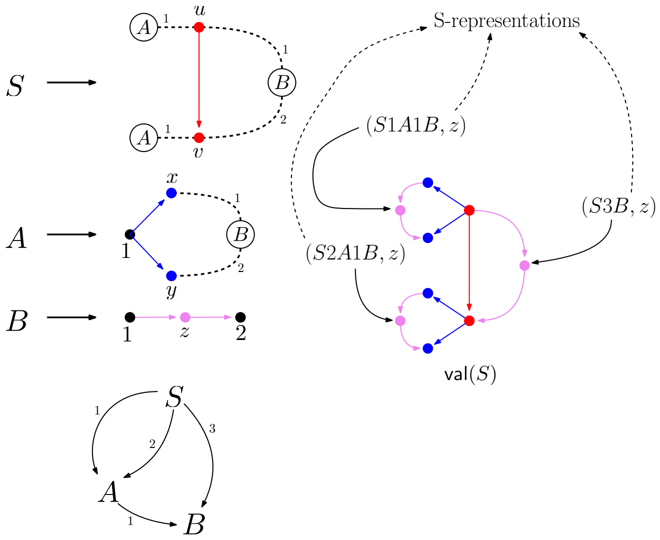
Step 3 – Compressed Setting (Node Representation)



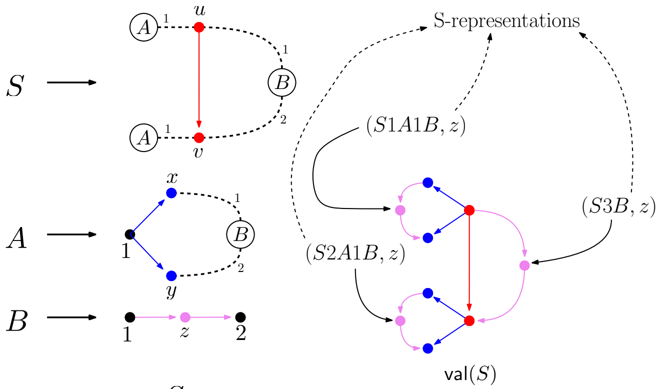
Step 3 – Compressed Setting (Node Representation)

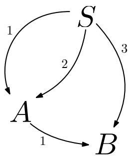


Step 3 – Compressed Setting (Node Representation)



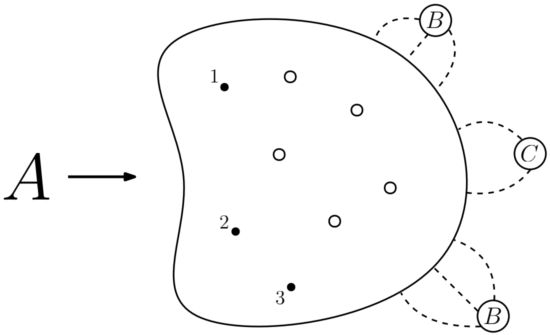
Step 3 – Compressed Setting (Node Representation)



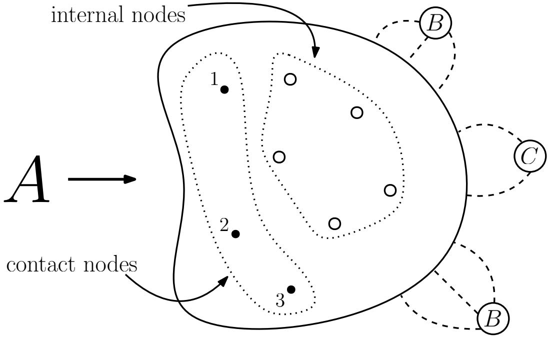


General problem:
S-representations have size $\Theta(|D|)$
(due to the path component)

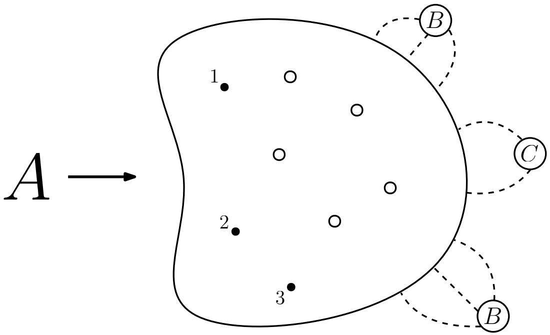
Step 3 – Compressed Setting (Expansions)



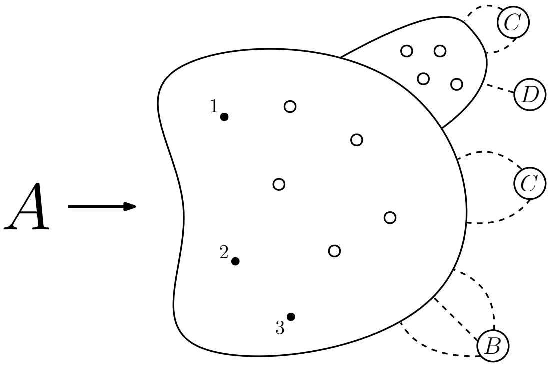
Step 3 – Compressed Setting (Expansions)



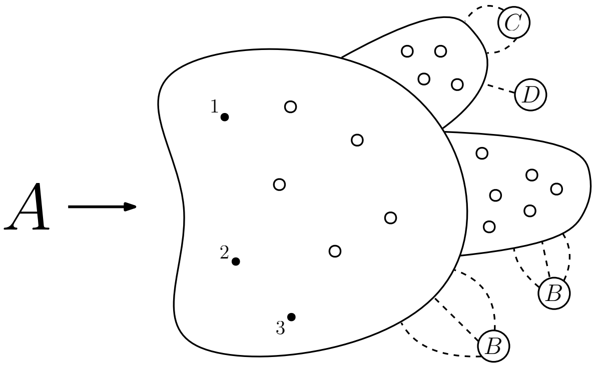
Step 3 – Compressed Setting (Expansions)



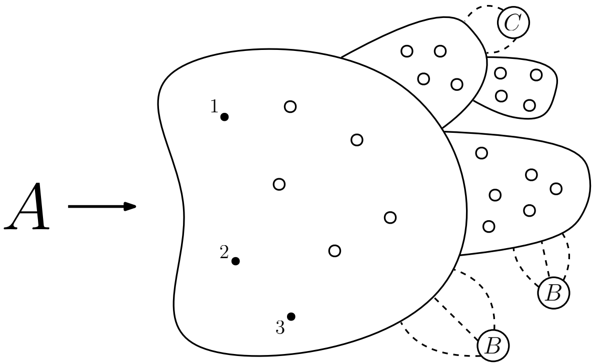
Step 3 – Compressed Setting (Expansions)



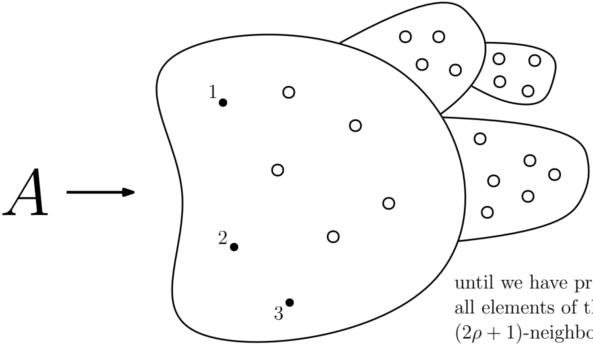
Step 3 – Compressed Setting (Expansions)



Step 3 – Compressed Setting (Expansions)



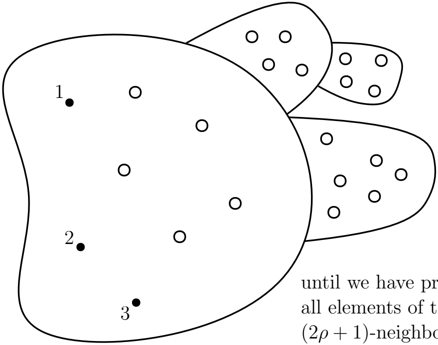
Step 3 – Compressed Setting (Expansions)



until we have produced all elements of the $(2\rho + 1)$-neighbourhood of the internal nodes.

Step 3 – Compressed Setting (Expansions)

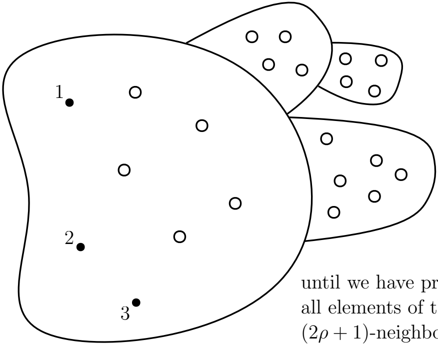
Expansion of A ($\mathcal{E}(A)$):



until we have produced all elements of the $(2\rho + 1)$-neighbourhood of the internal nodes.

Step 3 – Compressed Setting (Expansions)

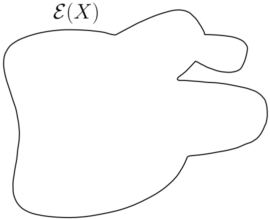
Expansion of A ($\mathcal{E}(A)$):



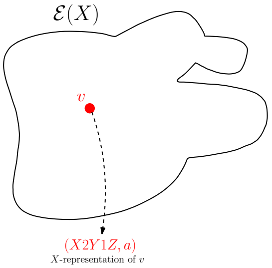
until we have produced all elements of the $(2\rho + 1)$-neighbourhood of the internal nodes.

We can compute all expansions in a preprocessing in linear time $O(|D|)$.

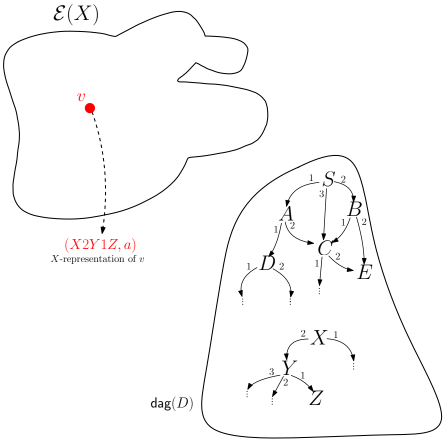
Step 3 – Compressed Setting (Embedding)



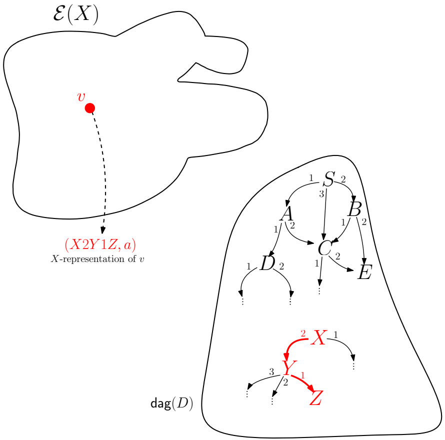
Step 3 – Compressed Setting (Embedding)



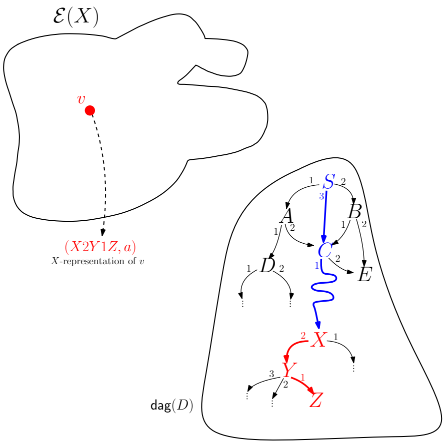
Step 3 – Compressed Setting (Embedding)



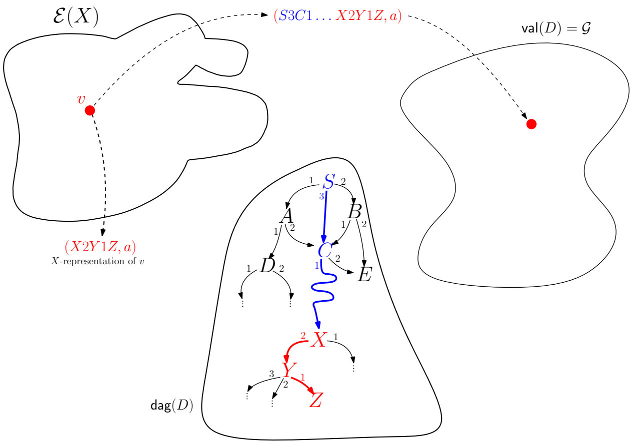
Step 3 – Compressed Setting (Embedding)



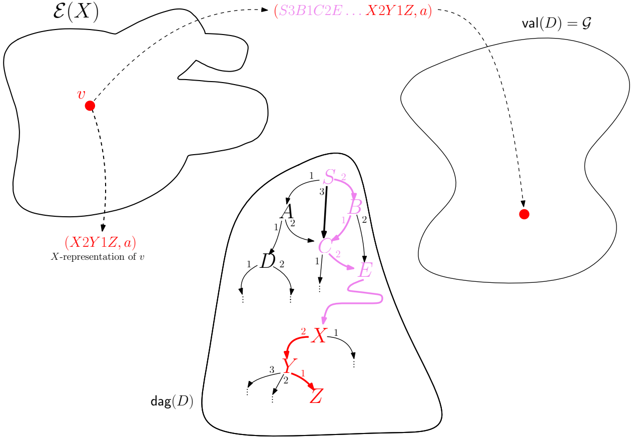
Step 3 – Compressed Setting (Embedding)



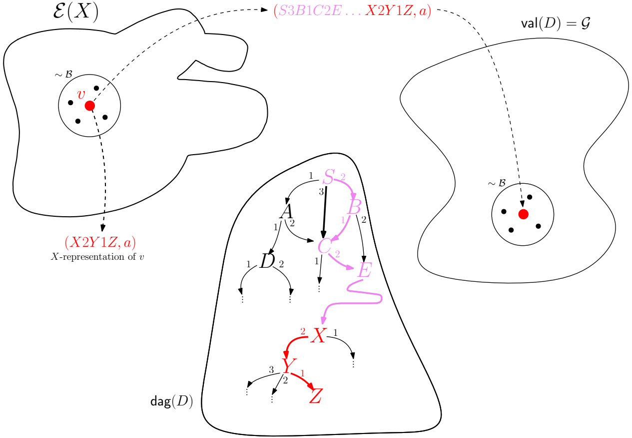
Step 3 – Compressed Setting (Embedding)



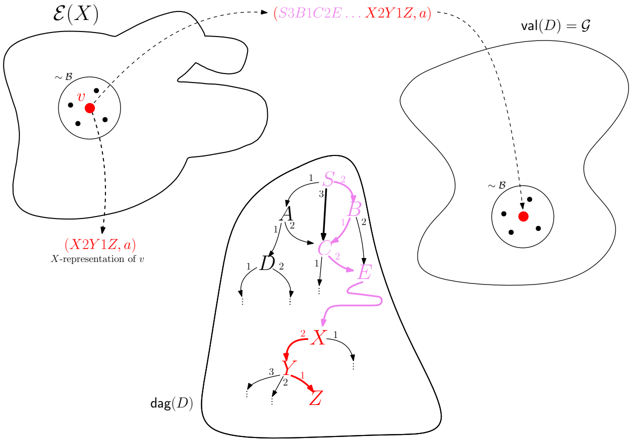
Step 3 – Compressed Setting (Embedding)



Step 3 – Compressed Setting (Embedding)

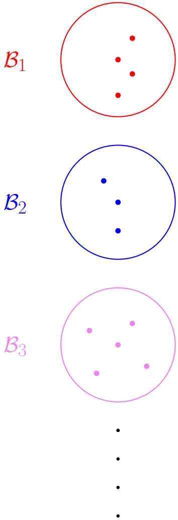


Step 3 – Compressed Setting (Embedding)



$\cup_{X \in N}\{S\text{-to-}X\text{-paths}\} \cdot \{\text{type-}\mathcal{B} \text{ nodes in } \mathcal{E}(X)\}. \xleftrightarrow{\text{bijection}} \{\text{type-}\mathcal{B} \text{ nodes in } \mathsf{val}(D)\}$

Step 3 – Compressed Setting (Enumerating $\mathcal{B}$-Nodes)



Step 3 – Compressed Setting (Enumerating $\mathcal{B}$-Nodes)

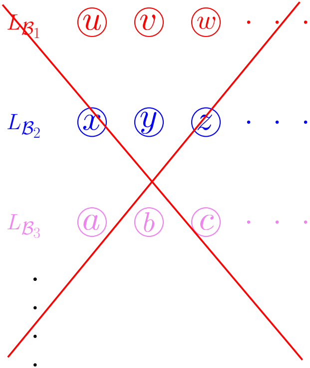
$L_{\mathcal{B}_1}$ ⓤ ⓥ ⓦ $\cdots$

$L_{\mathcal{B}_2}$ ⓧ ⓨ ⓩ $\cdots$

$L_{\mathcal{B}_3}$ ⓐ ⓑ ⓒ $\cdots$

$\vdots$

Step 3 – Compressed Setting (Enumerating $\mathcal{B}$-Nodes)



Step 3 – Compressed Setting (Enumerating $\mathcal{B}$-Nodes)

$$\mathsf{Enum}(\mathcal{B}_1): \quad \underbrace{(p_1, x_1) \overbrace{\qquad}^{O(1)} (p_2, x_2) \overbrace{\qquad}^{O(1)} (p_3, x_3)}_{S\text{-representations of } \mathcal{B}_1\text{-nodes}} \ \ldots$$

$$\mathsf{Enum}(\mathcal{B}_2): \quad \underbrace{(q_1, y_1) \overbrace{\qquad}^{O(1)} (q_2, y_2) \overbrace{\qquad}^{O(1)} (q_3, y_3)}_{S\text{-representations of } \mathcal{B}_2\text{-nodes}} \ \ldots$$

$$\mathsf{Enum}(\mathcal{B}_3): \quad \underbrace{(r_1, z_1) \overbrace{\qquad}^{O(1)} (r_2, z_2) \overbrace{\qquad}^{O(1)} (r_3, z_3)}_{S\text{-representations of } \mathcal{B}_3\text{-nodes}} \ \ldots$$

$\vdots$

Step 3 – Compressed Setting (Enumerating $\mathcal{B}$-Nodes)

Enum($\mathcal{B}$)

Preprocessing:
for every non-terminal X,
compute all $\mathcal{B}$-nodes in $\mathcal{E}(X)$
(in their X-representations)

(p_1, x_1)

(p_2, x_2)

(p_3, x_3)

$\vdots$

Step 3 – Compressed Setting (Enumerating $\mathcal{B}$-Nodes)

Enum($\mathcal{B}$)

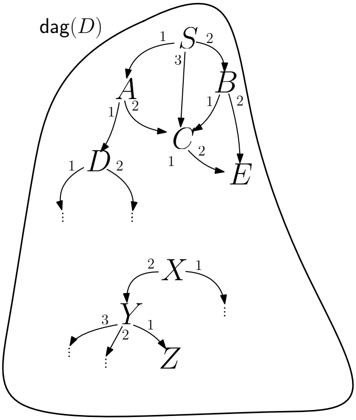
Preprocessing:
for every non-terminal X,
compute all $\mathcal{B}$-nodes in $\mathcal{E}(X)$
(in their X-representations)

(p_1, x_1)

(p_2, x_2)

(p_3, x_3)

⋮



Step 3 – Compressed Setting (Enumerating $\mathcal{B}$-Nodes)

Enum($\mathcal{B}$)

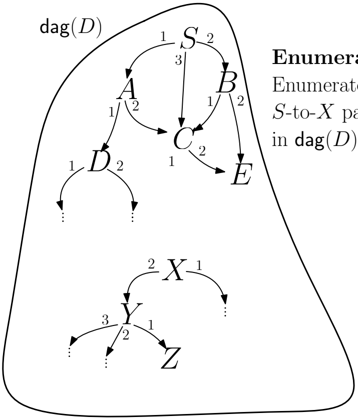
Preprocessing:
for every non-terminal X,
compute all $\mathcal{B}$-nodes in $\mathcal{E}(X)$
(in their X-representations)

(p_1, x_1)

(p_2, x_2)

(p_3, x_3)

$\vdots$



Enumeration:
Enumerate all
S-to-X paths
in dag(D) . . .

Step 3 – Compressed Setting (Enumerating $\mathcal{B}$-Nodes)

Enum($\mathcal{B}$)

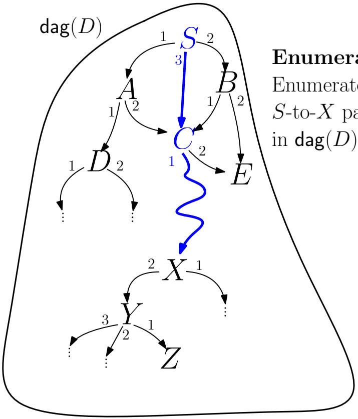
Preprocessing:
for every non-terminal X,
compute all $\mathcal{B}$-nodes in $\mathcal{E}(X)$
(in their X-representations)

(p_1, x_1)

(p_2, x_2)

(p_3, x_3)

⋮



Enumeration:
Enumerate all
S-to-X paths
in dag(D) ...

Step 3 – Compressed Setting (Enumerating $\mathcal{B}$-Nodes)

Enum($\mathcal{B}$)

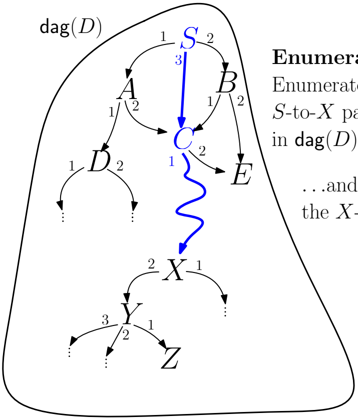
Preprocessing:
for every non-terminal X,
compute all $\mathcal{B}$-nodes in $\mathcal{E}(X)$
(in their X-representations)

$(s \cdot p_1, x_1)$

$(s \cdot p_2, x_2)$

$(s \cdot p_3, x_3)$

⋮



Enumeration:
Enumerate all
S-to-X paths
in dag(D) ...

...and prepend to
the X-representations

Step 3 – Compressed Setting (Enumerating $\mathcal{B}$-Nodes)

Enum($\mathcal{B}$)

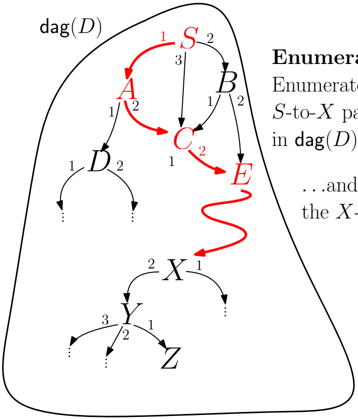
Preprocessing:
for every non-terminal X,
compute all $\mathcal{B}$-nodes in $\mathcal{E}(X)$
(in their X-representations)

$(s \cdot p_1, x_1)$ $(t \cdot p_1, x_1)$

$(s \cdot p_2, x_2)$ $(t \cdot p_2, x_2)$

$(s \cdot p_3, x_3)$ $(t \cdot p_3, x_3)$

$\vdots$



Enumeration:
Enumerate all
S-to-X paths
in dag(D) ...

...and prepend to
the X-representations

Step 3 – Compressed Setting (Enumerating $\mathcal{B}$-Nodes)

Enum($\mathcal{B}$)

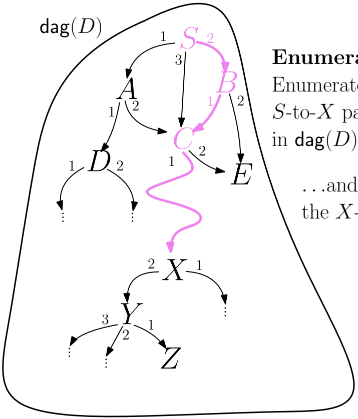
Preprocessing:
for every non-terminal X,
compute all $\mathcal{B}$-nodes in $\mathcal{E}(X)$
(in their X-representations)

$(s \cdot p_1, x_1)$ $(t \cdot p_1, x_1)$ $(q \cdot p_1, x_1)$

$(s \cdot p_2, x_2)$ $(t \cdot p_2, x_2)$ $(q \cdot p_2, x_2)$

$(s \cdot p_3, x_3)$ $(t \cdot p_3, x_3)$ $(q \cdot p_3, x_3)$

⋮



Enumeration:
Enumerate all
S-to-X paths
in dag(D) ...

...and prepend to
the X-representations

Step 3 – Compressed Setting (Enumerating $\mathcal{B}$-Nodes)

Enum($\mathcal{B}$)

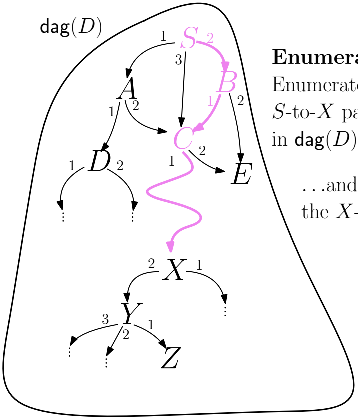
Preprocessing:
for every non-terminal X, compute all $\mathcal{B}$-nodes in $\mathcal{E}(X)$ (in their X-representations)

$(s \cdot p_1, x_1)$ $(t \cdot p_1, x_1)$ $(q \cdot p_1, x_1)$

$(s \cdot p_2, x_2)$ $(t \cdot p_2, x_2)$ $(q \cdot p_2, x_2)$

$(s \cdot p_3, x_3)$ $(t \cdot p_3, x_3)$ $(q \cdot p_3, x_3)$

⋮



Enumeration:
Enumerate all S-to-X paths in dag(D) ...

...and prepend to the X-representations

Problem: the paths $s \cdot p_1, s \cdot p_2, \ldots$ are too large!

Step 3 – Compressed Setting (Enumerating $\mathcal{B}$-Nodes)

Enum($\mathcal{B}$)

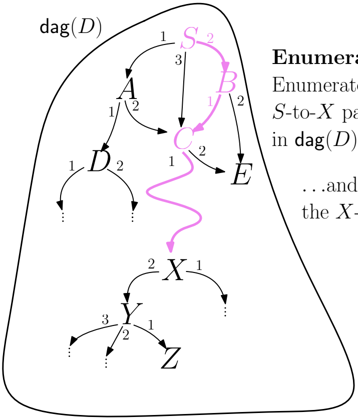
Preprocessing:
for every non-terminal X,
compute all $\mathcal{B}$-nodes in $\mathcal{E}(X)$
(in their X-representations)

$(23, x_1)$ $(4, x_2)$ $(253, x_1)$

$(5, x_2)$ $(23, x_2)$ $(8, x_2)$

$(17, x_3)$ $(35, x_3)$ $(13, x_3)$

$\vdots$



Enumeration:
Enumerate all
S-to-X paths
in dag(D) ...

...and prepend to
the X-representations

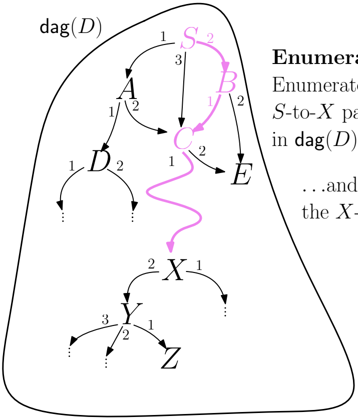
Solution: represent each path by its number
in the lexicographical ordering of all paths starting in S.

Step 3 – Compressed Setting (Enumerating $\mathcal{B}$-Nodes)

Enum($\mathcal{B}$)

Preprocessing:
for every non-terminal X,
compute all $\mathcal{B}$-nodes in $\mathcal{E}(X)$
(in their X-representations)

$(23, x_1)$	$(4, x_2)$	$(253, x_1)$
$(5, x_2)$	$(23, x_2)$	$(8, x_2)$
$(17, x_3)$	$(35, x_3)$	$(13, x_3)$
⋮		



Enumeration:
Enumerate all
S-to-X paths
in dag(D) . . .

. . .and prepend to
the X-representations

Solution: represent each path by its number
in the lexicographical ordering of all paths starting in S.

The End – Thank you very much for your attention!