CMSO-transducing tree-like graph decompositions

R. Campbell, B. Guillon, M.M. Kanté, E.J. Kim, N. Köhler

March 6, 2025
STACS'25

How to handle graphs?

Answer: decompose them, recursively.

How to handle graphs?

Answer: decompose them, recursively

- deal with simple graphs;
- combine the results.

How to handle graphs?

Answer: decompose them, recursively

- deal with simple graphs;
- combine the results.

How to decompose graphs?

How to handle graphs?

Answer: decompose them, recursively

- deal with simple graphs;
- combine the results.

How to decompose graphs?

- tree decomposition
- clique decomposition
- modular decomposition
- split decomposition
- . . .

How to handle graphs?

Answer: decompose them, recursively

- deal with simple graphs;
- combine the results.

How to decompose graphs?

- tree decomposition
- clique decomposition
- modular decomposition
- split decomposition
- ...

Each time, the decomposition is a tree.

How to handle graphs?

Answer: decompose them, recursively

- deal with simple graphs;
- combine the results.

How to decompose graphs?

- tree decomposition
- clique decomposition
- modular decomposition
- split decomposition
- ...

Each time, the decomposition is a tree.

In many cases, the tree has vertices as leaves, and labeled inner nodes.

How to handle graphs?

Answer: decompose them, recursively

- deal with simple graphs;
- combine the results.

How to decompose graphs?

- tree decomposition
- clique decomposition
- modular decomposition
- split decomposition
- ...

Each time, the decomposition is a tree.

In many cases, the tree has vertices as leaves, and labeled inner nodes.

What does such tree-like decomposition allow?

- investigation of combinatorial properties;
- characterize classes of graphs;
- algorithm design (*e.g.*, dynamic programming).

What does such tree-like decomposition allow?

- investigation of combinatorial properties;
- characterize classes of graphs;
- algorithm design (*e.g.*, dynamic programming).

General framework

On graphs of bounded treewidth,

- **Thm (Courcelle 90)** every CMSO-definable property can be recognized by a tree automaton over the tree decomposition.
- **Thm (Bodlaender 93)** tree decomposition of bounded width can be obtained in linear-time.
- **Corollary** CMSO model checking can be done in linear time.

What does such tree-like decomposition allow?

- investigation of combinatorial properties;
- characterize classes of graphs;
- algorithm design (*e.g.*, dynamic programming).

MSO with counting predicates
$C_p(X) \equiv$ "X has size 0 modulo p"

General framework

On graphs of bounded treewidth,

- **Thm (Courcelle 90)** every CMSO-definable property can be recognized by a tree automaton over the tree decomposition.
- **Thm (Bodlaender 93)** tree decomposition of bounded width can be obtained in linear-time.
- **Corollary** CMSO model checking can be done in linear time.

What does such tree-like decomposition allow?

- investigation of combinatorial properties;
- characterize classes of graphs;
- algorithm design (*e.g.*, dynamic programming).

MSO with counting predicates
$C_p(X) \equiv$ "X has size 0 modulo p"

General framework

On graphs of bounded treewidth,

- **Thm (Courcelle 90)** every CMSO-definable property can be recognized by a tree automaton over the tree decomposition.
- **Thm (Bodlaender 93)** tree decomposition of bounded width can be obtained in linear-time.
- **Corollary** CMSO model checking can be done in linear time.

Converse result?

Courcelle's conjecture

question: On graphs of bounded treewidth
is every recognizable property CMSO-definable?

Converse result?

Courcelle's conjecture

question: On graphs of bounded treewidth
is every recognizable property CMSO-definable?

main difficulty: the input is the graph, and not the tree decomposition

Converse result?

Courcelle's conjecture

question: On graphs of bounded treewidth
is every recognizable property CMSO-definable?

main difficulty: the input is the graph, and not the tree decomposition

strategy (Courcelle 91):

1. obtain the decomposition **within CMSO**;
2. encode the recognizability using CMSO-formulas.

Converse result?

Courcelle's conjecture

question: On graphs of bounded treewidth
is every recognizable property CMSO-definable?

main difficulty: the input is the graph, and not the tree decomposition

strategy (Courcelle 91):

1. obtain the decomposition **within CMSO**;
2. encode the recognizability using CMSO-formulas.

by mean of CMSO-transductions

Converse result.

~~Courcelle's conjecture~~ Bojańczyk&Pilipczuk Theorem

question: On graphs of bounded treewidth
is every recognizable property CMSO-definable?

main difficulty: the input is the graph, and not the tree decomposition

strategy (Courcelle 91):

1. obtain the decomposition **within CMSO**;
2. encode the recognizability using CMSO-formulas.

by mean of CMSO-transductions

Theorem (Bojańczyk&Pilipczuk 16)

On graphs of bounded treewidth, recognizability = CMSO*-definability.*

key ingredient: a CMSO-transduction producing the decomposition

Converse result.

~~Courcelle's conjecture~~ Bojańczyk&Pilipczuk Theorem

question: On graphs of bounded treewidth
is every recognizable property CMSO-definable?

main difficulty: the input is the graph, and not the tree decomposition

strategy (Courcelle 91):

1. obtain the decomposition **within CMSO**;
2. encode the recognizability using CMSO-formulas.

by mean of CMSO-transductions

Theorem (Bojańczyk&Pilipczuk 16)

On graphs of bounded treewidth, recognizability = CMSO*-definability.*

key ingredient: a CMSO-transduction producing the decomposition

CMSO-Transductions

Σ and Γ: two relationnal signatures. *E.g.*,

- **Σ:** the graph vocabulary $\{\text{edge}\}$ — $\text{edge}(x, y) \equiv$ "x–y is an edge"
- **Γ:** the tree-dec vocabulary $\{\text{parent}, \text{bag}\}$ — $\text{parent}(x, y)$; $\text{bag}(v, x)$

CMSO-Transductions

Σ and Γ: two relationnal signatures. *E.g.*,

- **Σ:** the graph vocabulary $\{\text{edge}\}$ — $\text{edge}(x, y) \equiv$ "x–y is an edge"
- **Γ:** the tree-dec vocabulary $\{\text{parent}, \text{bag}\}$ — $\text{parent}(x, y)$; $\text{bag}(v, x)$

Definition (CMSO-transduction)

A Σ-to-Γ CMSO-transductions is a composition of:

domain restriction: filter the input structures (partial identity);
universe restriction: subset the universe;
interpretation: interpret each relation name from Γ in Σ;

using CMSO-formulas.

CMSO-Transductions

Σ and Γ: two relationnal signatures. *E.g.*,

Σ: the graph vocabulary $\{\text{edge}\}$ — $\text{edge}(x, y) \equiv$ "x–y is an edge"

Γ: the tree-dec vocabulary $\{\text{parent}, \text{bag}\}$ — $\text{parent}(x, y)$; $\text{bag}(v, x)$

Definition (CMSO-transduction)

A Σ-to-Γ CMSO-transductions is a composition of:

copying: make k copies of the universe;
domain restriction: filter the input structures (partial identity);
universe restriction: subset the universe;
interpretation: interpret each relation name from Γ in Σ;

using CMSO-formulas.

CMSO-Transductions

Σ and Γ: two relationnal signatures. *E.g.*,

- **Σ:** the graph vocabulary $\{\text{edge}\}$ — $\text{edge}(x, y) \equiv$ "x–y is an edge"
- **Γ:** the tree-dec vocabulary $\{\text{parent}, \text{bag}\}$ — $\text{parent}(x, y)$; $\text{bag}(v, x)$

Definition (CMSO-transduction)

A Σ-to-Γ CMSO-transductions is a composition of:

colouring: guess a finite family of unary relations;

copying: make k copies of the universe;

domain restriction: filter the input structures (partial identity);

universe restriction: subset the universe;

interpretation: interpret each relation name from Γ in Σ;

using CMSO-formulas.

CMSO-Transductions

Σ and Γ: two relationnal signatures. *E.g.*,

Σ: the graph vocabulary $\{\text{edge}\}$ — $\text{edge}(x, y) \equiv$ "x–y is an edge"

Γ: the tree-dec vocabulary $\{\text{parent}, \text{bag}\}$ — $\text{parent}(x, y)$; $\text{bag}(v, x)$

Definition (CMSO-transduction)

A Σ-to-Γ CMSO-transductions is a composition of:

colouring: guess a finite family of unary relations;

copying: make k copies of the universe;

domain restriction: filter the input structures (partial identity);

universe restriction: subset the universe;

interpretation: interpret each relation name from Γ in Σ;

using CMSO-formulas.

Theorem (Backward Translation Theorem)

$\tau^{-1}(Y)$ *is* CMSO-*definable* *for* τ, *a* Σ-*to*-Γ CMSO-*transduction* *and* Y, *a* CMSO-*definable set of* Γ-*structures.*

For which classes of graphs, definability = recognizability?

Which decompositions can be CMSO-transduced?

CMSO-definability = recognizability for graphs of:

- **Thm (Rabin 69)** treewidth 1 (*i.e.*, trees, even infinite)
- **Thm (Bojańczyk&Pilipczuk 16)** bounded treewidth
- **Thm (Bojańczyk,Grohe&Pilipczuk 20)** bounded linear cliquewidth

For which classes of graphs, definability = recognizability?

Which decompositions can be CMSO-transduced?

CMSO-definability = recognizability for graphs of:

- Thm (Rabin 69) treewidth 1 (*i.e.*, trees, even infinite)
- Thm (Bojańczyk&Pilipczuk 16) bounded treewidth
- Thm (Bojańczyk,Grohe&Pilipczuk 20) bounded linear cliquewidth

and, if using **order-invariant** CMSO (more expressive than CMSO):

- Thm (Courcelle 96) on cographs (= cliquewidth ≤ 2)

For which classes of graphs, definability = recognizability?

Which decompositions can be CMSO-transduced?

CMSO-definability = recognizability for graphs of:

- **Thm (Rabin 69)** treewidth 1 (*i.e.*, trees, even infinite)
- **Thm (Bojańczyk&Pilipczuk 16)** bounded treewidth
- **Thm (Bojańczyk,Grohe&Pilipczuk 20)** bounded linear cliquewidth

and, if using **order-invariant** CMSO (more expressive than CMSO):

- **Thm (Courcelle 96)** on cographs (= cliquewidth ≤ 2)

One of the consequences of our results

- **Thm** CMSO-definability = recognizability on cographs.

Our results

Theorems

There are CMSO-transductions from graphs to:

- modular decompositions;
    - cotrees (in case of cographs);
- split decompositions;
- bi-join decompositions.

For both undirected and directed graphs.

Our results

Theorems

There are CMSO-transductions from graphs to:

- modular decompositions;
  - cotrees (in case of cographs);
- split decompositions;
- bi-join decompositions.

For both undirected and directed graphs.

All these results use the following key ingredient:

Theorem

There is a CMSO-transduction from **laminar** set systems to **laminar** trees.

Our results

Theorems

There are CMSO-transductions from graphs to:

- modular decompositions;
    - cotrees (in case of cographs);
- split decompositions;
- bi-join decompositions.

For both undirected and directed graphs.

All these results use the following key ingredient:

Theorem

There is a CMSO-transduction from **laminar** set systems to **laminar** trees.

Our results

Theorems

There are CMSO-transductions from graphs to:

- modular decompositions;
  - cotrees (in case of cographs);
- split decompositions;
- bi-join decompositions.

For both undirected and directed graphs.

All these results use the following key ingredient:

Theorem

There is a CMSO-transduction from **laminar** set systems to **laminar** trees.

Modular decompositon

Def: (modules) $M \subseteq V(G)$, such that

for each $v \notin M$ either uv is an edge for all $u \in M$, or uv is not an edge for all $u \in M$.

Modular decompositon

Def: (modules) $M \subseteq V(G)$, such that

for each $v \notin M$ either uv is an edge for all $u \in M$, or uv is not an edge for all $u \in M$.

"Vertices in M are not distinguished from outside of M."

Modular decompositon

Def: (modules) $M \subseteq V(G)$, such that

for each $v \notin M$ — either uv is an edge for all $u \in M$, or uv is not an edge for all $u \in M$.

"Vertices in M are not distinguished from outside of M."

Basic properties:

- two disjoint modules are either fully adjacent or fully non-adjacent.
- if two modules intersect, their union is a module.

Modular decompositon

Def: (modules) $M \subseteq V(G)$, such that

for each $v \notin M$ either uv is an edge for all $u \in M$, or uv is not an edge for all $u \in M$.

"Vertices in M are not distinguished from outside of M."

Basic properties:

- two disjoint modules are either fully adjacent or fully non-adjacent.
- if two modules intersect, their union is a module.

Def: (strong modules) a module, not overlapping another one.

Modular decompositon

Def: (modules) $M \subseteq V(G)$, such that

for each $v \notin M$ either uv is an edge for all $u \in M$, or uv is not an edge for all $u \in M$.

"Vertices in M are not distinguished from outside of M."

Basic properties:

- two disjoint modules are either fully adjacent or fully non-adjacent.
- if two modules intersect, their union is a module.

Def: (strong modules) a module, not overlapping (either disjoint or related by inclusion) another one.

Modular decompositon

Def: (modules) $M \subseteq V(G)$, such that

for each $v \notin M$ either uv is an edge for all $u \in M$, or uv is not an edge for all $u \in M$.

"Vertices in M are not distinguished from outside of M."

Basic properties:

- two disjoint modules are either fully adjacent or fully non-adjacent.
- if two modules intersect, their union is a module.

Def: (strong modules) a module, not overlapping another one,

(overlapping: either disjoint or related by inclusion)

in particular, $V(G)$ and $\{v\}$ for each v, are strong modules.

Modular decompositon

Def: (modules) $M \subseteq V(G)$, such that

for each $v \notin M$ either uv is an edge for all $u \in M$, or uv is not an edge for all $u \in M$.

"Vertices in M are not distinguished from outside of M."

Basic properties:

- two disjoint modules are either fully adjacent or fully non-adjacent.
- if two modules intersect, their union is a module.

Def: (strong modules) a module, not overlapping another one, (overlapping: either disjoint or related by inclusion)

in particular, $V(G)$ and $\{v\}$ for each v, are strong modules.

The inclusion relation on strong modules defines a tree, called **laminar tree**.

Modular decompositon

Def: (modules) $M \subseteq V(G)$, such that

for each $v \notin M$ either uv is an edge for all $u \in M$, or uv is not an edge for all $u \in M$.

"Vertices in M are not distinguished from outside of M."

Basic properties:

- two disjoint modules are either fully adjacent or fully non-adjacent.
- if two modules intersect, their union is a module.

Def: (strong modules) a module, not overlapping another one, (overlapping: either disjoint or related by inclusion)

in particular, $V(G)$ and $\{v\}$ for each v, are strong modules.

The inclusion relation on strong modules defines a tree, called **laminar tree**.

The tree underlying the modular decomposition is this laminar tree.

Modular decompositon

Def: (modules) $M \subseteq V(G)$, such that

for each $v \notin M$ either uv is an edge for all $u \in M$, or uv is not an edge for all $u \in M$.

"Vertices in M are not distinguished from outside of M."

Basic properties:

- two disjoint modules are either fully adjacent or fully non-adjacent.
- if two modules intersect, their union is a module.

Def: (strong modules) a module, not overlapping (either disjoint or related by inclusion) another one,

in particular, $V(G)$ and $\{v\}$ for each v, are strong modules.

The inclusion relation on strong modules defines a tree, called **laminar tree**.

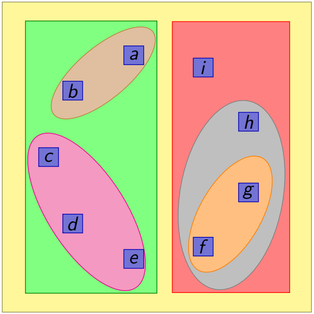
The tree underlying the modular decomposition is this laminar tree.

Laminar tree

- **Def (set system):** a family of subsets $\mathcal{S}$ of a ground set U.
- **Def (laminar set system):** when no two members of $\mathcal{S}$ overlap.

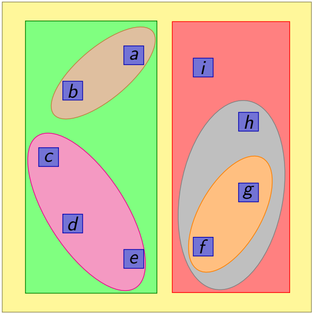
Laminar tree

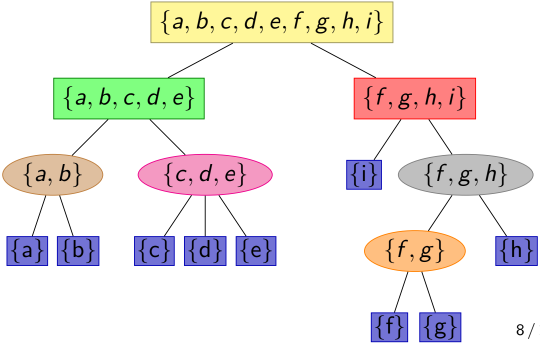
- Def (set system): a family of subsets $\mathcal{S}$ of a ground set U.
- Def (laminar set system): when no two members of $\mathcal{S}$ overlap.



Laminar tree

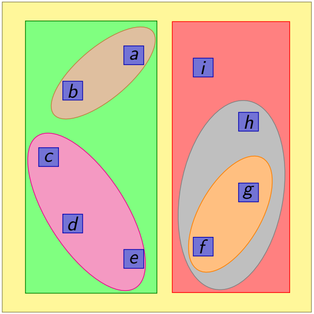
- **Def (set system):** a family of subsets $\mathcal{S}$ of a ground set U.
- **Def (laminar set system):** when no two members of $\mathcal{S}$ overlap.
- **Def (laminar tree):** tree with node $\mathcal{S}$ and $\supseteq$ as ancestor relation

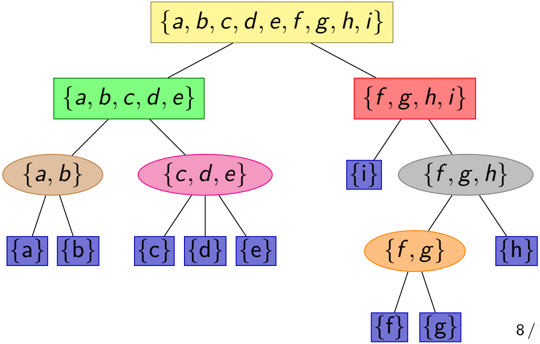




Laminar tree

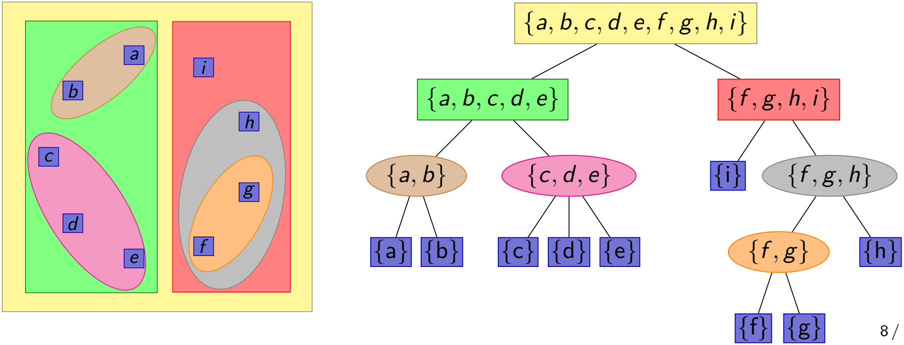
- Def (set system): a family of subsets $\mathcal{S}$ of a ground set U.
- Def (laminar set system): when no two members of $\mathcal{S}$ overlap,
  $$U \in \mathcal{S},\ \emptyset \notin \mathcal{S}$$
- Def (laminar tree): tree with node $\mathcal{S}$ and $\supseteq$ as ancestor relation with U as root





Laminar tree

- **Def (set system):** a family of subsets $\mathcal{S}$ of a ground set U.
- **Def (laminar set system):** when no two members of $\mathcal{S}$ overlap, $U \in \mathcal{S}$, $\emptyset \notin \mathcal{S}$, and $\{v\} \in \mathcal{S}$ for $v \in U$.
- **Def (laminar tree):** tree with node $\mathcal{S}$ and $\supseteq$ as ancestor relation with U as root, $\{v\}$ for $v \in U$ as leaves.



Key construction

Theorem

There is a non-deterministic C_2 MSO-transduction that:
inputs a laminar set system $(U, \mathcal{S})$;
outputs the laminar tree T with U as set of leaves.

Key construction

Theorem

counting modulo 2

There is a non-deterministic C_2 MSO-transduction that:
inputs a laminar set system $(U, \mathcal{S})$;
outputs the laminar tree T with U as set of leaves.

Key construction

counting modulo 2

Theorem

There is a non-deterministic C_2 MSO-transduction that:
inputs a laminar set system $(U, \mathcal{S})$;
outputs the laminar tree T with U as set of leaves.

Main difficulty:
The CMSO-transduction needs to represent each member of $\mathcal{S}$ by one (copy of a) vertex.

Key construction

Theorem

counting modulo 2

There is a non-deterministic C_2 MSO-transduction that:
inputs a laminar set system $(U, \mathcal{S})$;
outputs the laminar tree T with U as set of leaves.

Main difficulty:
The CMSO-transduction needs to represent each member of $\mathcal{S}$ by one (copy of a) vertex.

Proof idea:

1. guess two subsets of vertices L and R;
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

Key construction

Theorem

counting modulo 2

There is a non-deterministic C_2MSO-transduction that:
inputs a laminar set system $(U, \mathcal{S})$;
outputs the laminar tree T with U as set of leaves.

Main difficulty:
The CMSO-transduction needs to represent each member of $\mathcal{S}$ by one (copy of a) vertex.

Proof idea:

1. guess two subsets of vertices L and R;
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

smallest $X \in \mathcal{S}$ including $\{\ell, r\}$

Key construction

counting modulo 2

Theorem

There is a non-deterministic C_2MSO-transduction that:
inputs a laminar set system $(U, \mathcal{S})$;
outputs the laminar tree T with U as set of leaves.

Main difficulty:
The CMSO-transduction needs to represent each member of $\mathcal{S}$ by one (copy of a) vertex.

Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

smallest $X \in \mathcal{S}$ including $\{\ell, r\}$

Key construction

counting modulo 2

Theorem

There is a non-deterministic C_2MSO-transduction that:
inputs a laminar set system $(U, \mathcal{S})$;
outputs the laminar tree T with U as set of leaves.

Main difficulty:
The CMSO-transduction needs to represent each member of $\mathcal{S}$ by one (copy of a) vertex.

Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$,

Key construction

Theorem

counting modulo 2

There is a non-deterministic C_2MSO-transduction that:
inputs a laminar set system $(U, \mathcal{S})$;
outputs the laminar tree T with U as set of leaves.

Main difficulty:
The CMSO-transduction needs to represent each member of $\mathcal{S}$ by one (copy of a) vertex.

Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$,
for each member $Z \subsetneq X$ of $\mathcal{S}$, $Z \cap \{\ell, r\} = \emptyset \iff |Z \cap (L \cup R)|$ is even.

Key construction

Theorem

counting modulo 2

There is a non-deterministic C_2MSO-transduction that:
inputs a laminar set system $(U, \mathcal{S})$;
outputs the laminar tree T with U as set of leaves.

Main difficulty:
The CMSO-transduction needs to represent each member of $\mathcal{S}$ by one (copy of a) vertex.

Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$,
for each member $Z \subsetneq X$ of $\mathcal{S}$, $Z \cap \{\ell, r\} = \emptyset \iff |Z \cap (L \cup R)|$ is even.

Key construction

counting modulo 2

Theorem

There is a non-deterministic C_2MSO-transduction that:
inputs a laminar set system $(U, \mathcal{S})$;
outputs the laminar tree T with U as set of leaves.

Main difficulty:
The CMSO-transduction needs to represent each member of $\mathcal{S}$ by one (copy of a) vertex.

Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$,
for each member $Z \subsetneq X$ of $\mathcal{S}$, $Z \cap \{\ell, r\} = \emptyset \iff |Z \cap (L \cup R)|$ is even.

Z is a descendant of X

Key construction

Theorem

There is a non-deterministic C_2MSO-transduction that:
inputs a laminar set system $(U, \mathcal{S})$;
outputs the laminar tree T with U as set of leaves.

counting modulo 2

Main difficulty:
The CMSO-transduction needs to represent each member of $\mathcal{S}$ by one (copy of a) vertex.

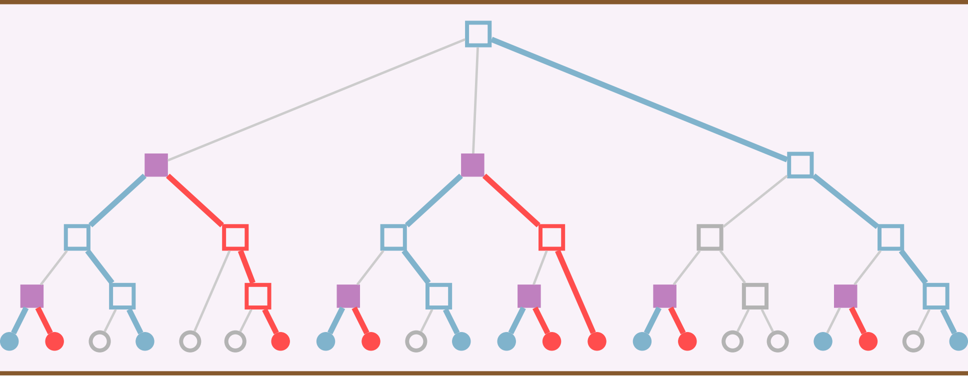
Proof idea:

1. guess two subsets of vertices L and R;
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

on correct guesses

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$,
for each member $Z \subsetneq X$ of $\mathcal{S}$, $Z \cap \{\ell, r\} = \emptyset \iff |Z \cap (L \cup R)|$ is even.

Z is a descendant of X



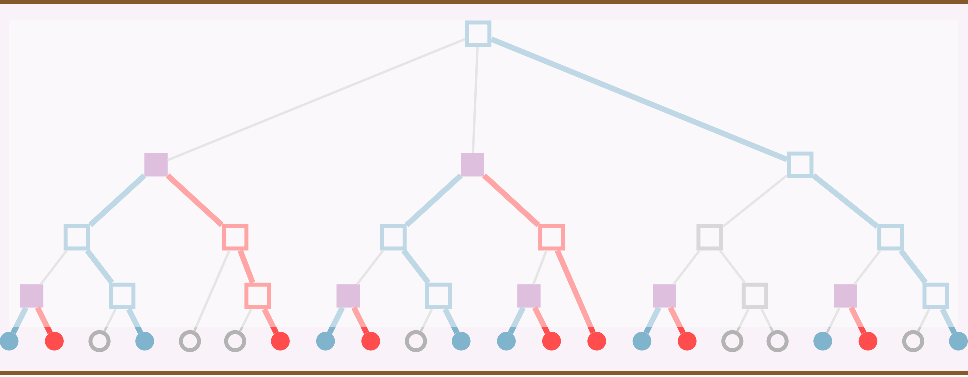
Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$,

for each member $Z \subsetneq X$ of $\mathcal{S}$, $Z \cap \{\ell, r\} = \emptyset \iff |Z \cap (L \cup R)|$ is even.

Z is a descendant of X

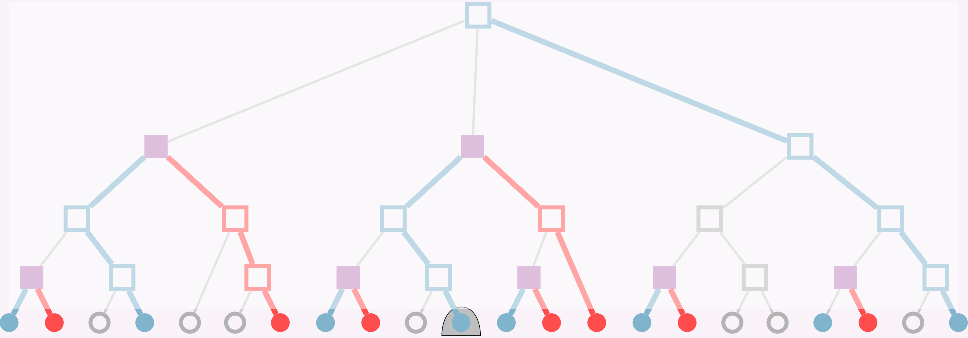


Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$, for each member $Z \subsetneq X$ of $\mathcal{S}$, $Z \cap \{\ell, r\} = \emptyset \iff |Z \cap (L \cup R)|$ is even.

Z is a descendant of X



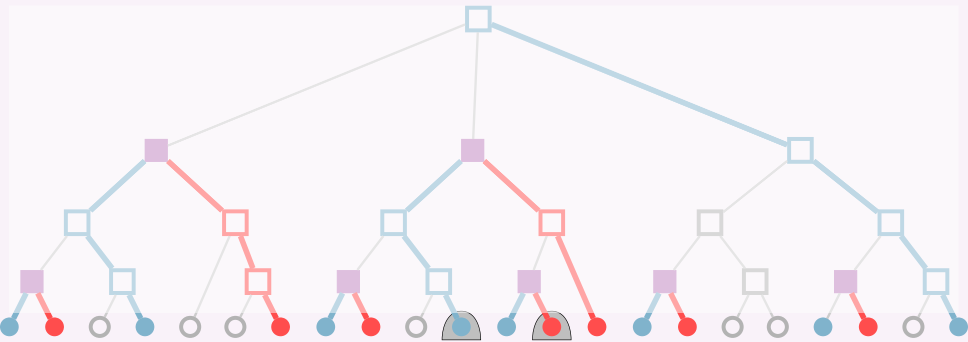
Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$,

for each member $Z \subsetneq X$ of $\mathcal{S}$, $Z \cap \{\ell, r\} = \emptyset \iff |Z \cap (L \cup R)|$ is even.

Z is a descendant of X



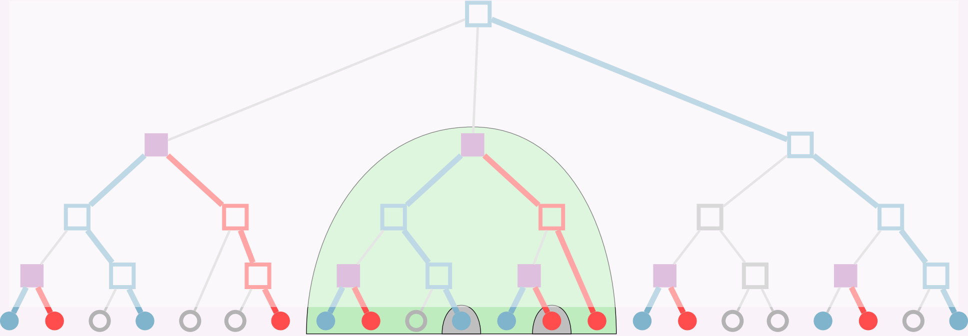
Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$,

for each member $Z \subsetneq X$ of $\mathcal{S}$, $Z \cap \{\ell, r\} = \emptyset \iff |Z \cap (L \cup R)|$ is even.

Z is a descendant of X

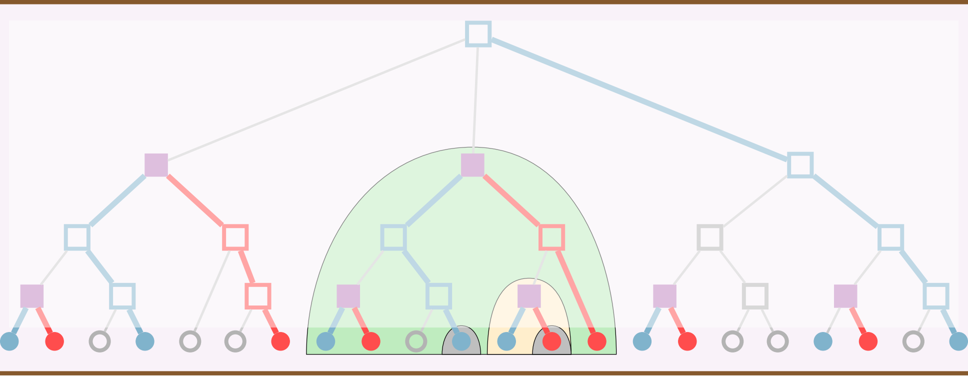


Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$, for each member $Z \subsetneq X$ of $\mathcal{S}$, $Z \cap \{\ell, r\} = \emptyset \iff |Z \cap (L \cup R)|$ is even.

Z is a descendant of X

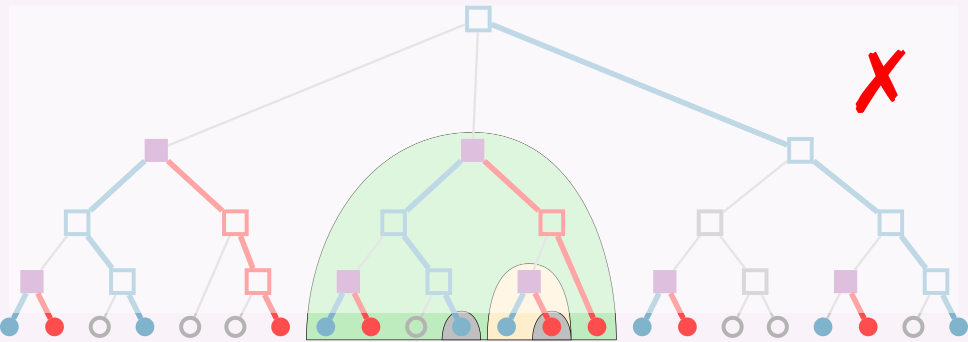


Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$,
for each member $Z \subsetneq X$ of $\mathcal{S}$, $Z \cap \{\ell, r\} = \emptyset \iff |Z \cap (L \cup R)|$ is even.

Z is a descendant of X



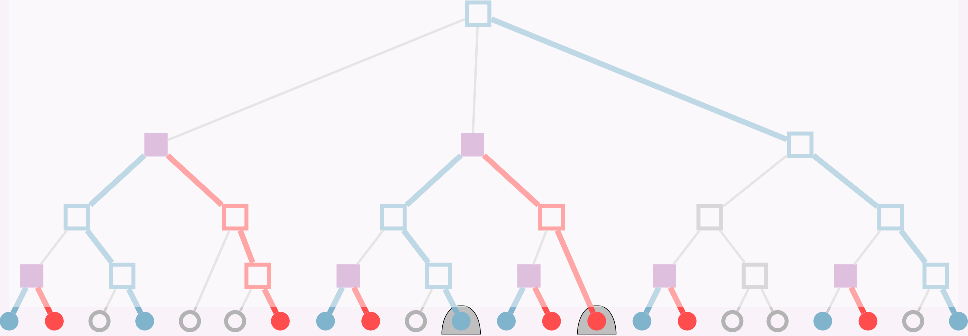
Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$,

for each member $Z \subsetneq X$ of $\mathcal{S}$, $Z \cap \{\ell, r\} = \emptyset \iff |Z \cap (L \cup R)|$ is even.

Z is a descendant of X

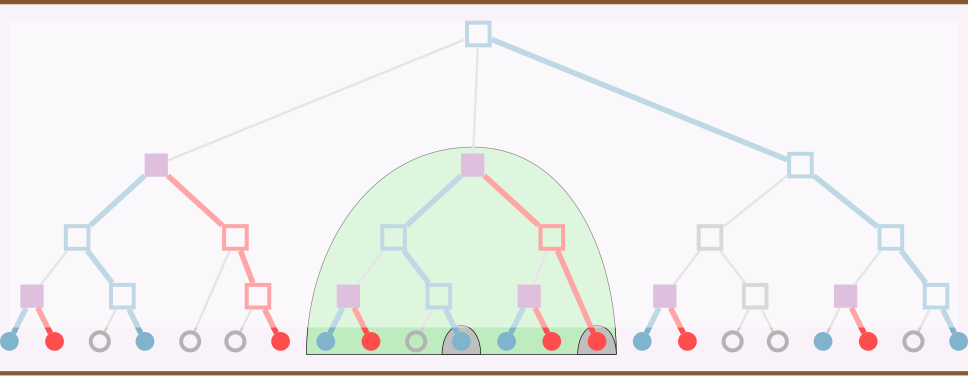


Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$, for each member $Z \subsetneq X$ of $\mathcal{S}$, $Z \cap \{\ell, r\} = \emptyset \iff |Z \cap (L \cup R)|$ is even.

Z is a descendant of X



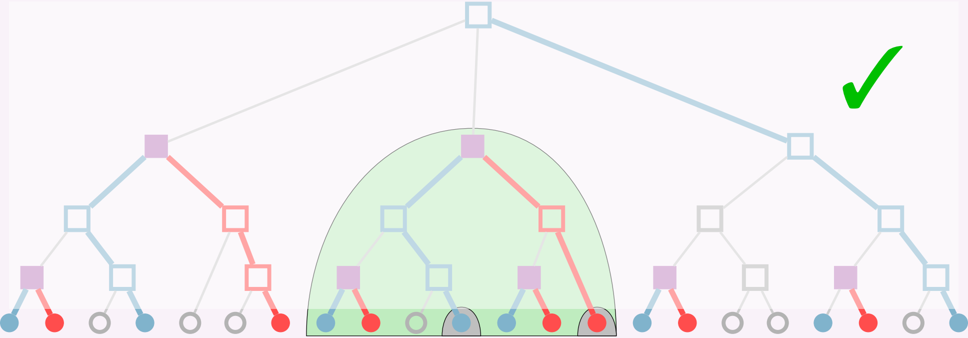
Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$,

for each member $Z \subsetneq X$ of $\mathcal{S}$, $Z \cap \{\ell, r\} = \emptyset \iff |Z \cap (L \cup R)|$ is even.

Z is a descendant of X

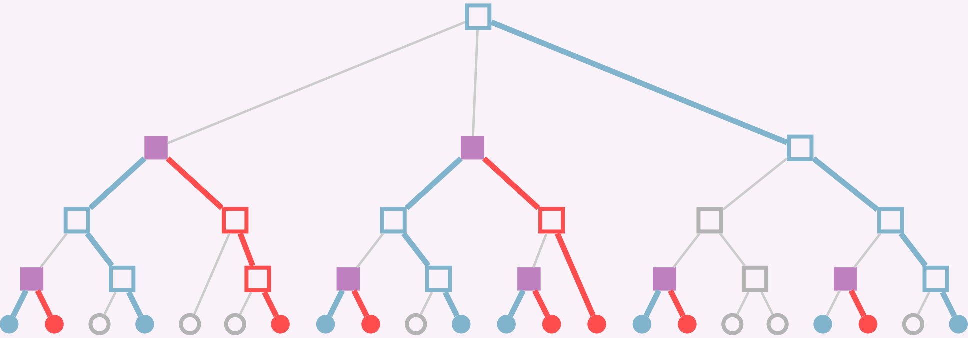


Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$, for each member $Z \subsetneq X$ of $\mathcal{S}$, $Z \cap \{\ell, r\} = \emptyset \iff |Z \cap (L \cup R)|$ is even.

Z is a descendant of X



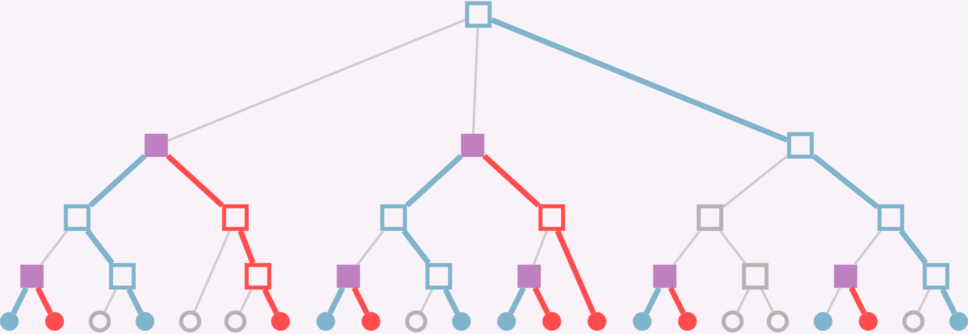
Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$, for each member $Z \subsetneq X$ of $\mathcal{S}$, $Z \cap \{\ell, r\} = \emptyset \iff |Z \cap (L \cup R)|$ is even.

Z is a descendant of X

one (L, R, μ) is not sufficient



Proof idea:

1. guess two subsets of vertices L and R; on correct guesses
2. bijectively map L to R, by some CMSO-definable function μ;
3. say that $\ell \in L$ represent the least common ancestor of ℓ and $r = \mu(\ell)$.

- $\mu(\ell) = r$ if, considering the smallest $X \in \mathcal{S}$ including $\{\ell, r\}$, for each member $Z \subsetneq X$ of $\mathcal{S}$, $Z \cap \{\ell, r\} = \emptyset \iff |Z \cap (L \cup R)|$ is even.

Z is a descendant of X

one (L, R, μ) is not sufficient, but four are.

Conclusion

Theorem

There are CMSO-transductions from set systems:

- to laminar trees (if laminar);

and from (both directed and undirected) graphs:

- to modular decompositions;
  - to cotrees (if cograph);
- to split decompositions;
- to bi-join decompositions.

Conclusion

Theorem

There are CMSO-transductions from set systems:

- to laminar trees (if laminar);
- to (weakly)-partitive trees (if (weakly-)partitive);

and from (both directed and undirected) graphs:

- to modular decompositions;
  - to cotrees (if cograph);
- to split decompositions;
- to bi-join decompositions.

Conclusion

Theorem

There are CMSO-transductions from set systems:

- to laminar trees (if laminar);
- to (weakly)-partitive trees (if (weakly-)partitive);
- to (weakly)-bipartitive trees (if (weakly-)bipartitive);

and from (both directed and undirected) graphs:

- to modular decompositions;
  - to cotrees (if cograph);
- to split decompositions;
- to bi-join decompositions.

Conclusion

Theorem

There are CMSO-transductions from set systems:

- to laminar trees (if laminar);
- to (weakly)-partitive trees (if (weakly-)partitive);
- to (weakly)-bipartitive trees (if (weakly-)bipartitive);

and from (both directed and undirected) graphs:

- to modular decompositions;
  - to cotrees (if cograph);
- to split decompositions;
- to bi-join decompositions.

Open directions:

- which decompositions of which structures?
- branch-decomposition and clique-decomposition.
- is counting necessary?

Conclusion

Theorem

There are CMSO-transductions from set systems:

- to laminar trees (if laminar);
- to (weakly)-partitive trees (if (weakly-)partitive);
- to (weakly)-bipartitive trees (if (weakly-)bipartitive);

and from (both directed and undirected) graphs:

- to modular decompositions;
  - to cotrees (if cograph);
- to split decompositions;
- to bi-join decompositions.

Open directions:

- which decompositions of which structures?
- branch-decomposition and clique-decomposition.
- is counting necessary?

Thank you for your attention.