Graph decompositions and their algorithms

Édouard Bonnet

ENS Lyon, LIP

May 5th, 2022, Demi-Journée du Pôle Calcul
Trees

Example of Min Weighted Dominating Set

Idea: keep 3 lightest dominating sets of each subtree (rooted at $u$) one containing $u$, one not containing $u$, and one disregarding $u$. 
Trees make most NP-hard problems easy

Example of Min Weighted Dominating Set

Idea: keep 3 lightest dominating sets of each subtree (rooted at \( u \)) one containing \( u \), one not containing \( u \), and one disregarding \( u \)
Trees make most NP-hard problems easy

Example of $\text{Min Weighted Dominating Set}$

Idea: keep 3 lightest dominating sets of each subtree (rooted at $u$) one containing $u$, one not containing $u$, and one disregarding $u$
Trees make most NP-hard problems easy

Example of Min Weighted Dominating Set

Idea: keep 3 lightest dominating sets of each subtree (rooted at \( u \)) one containing \( u \), one not containing \( u \), and one disregarding \( u \)
Trees make most NP-hard problems easy

Example of \textbf{Min Weighted Dominating Set}

Idea: keep 3 lightest dominating sets of each subtree (rooted at $u$) one containing $u$, one not containing $u$, and one disregarding $u$
Trees make most NP-hard problems easy

Example of Min Weighted Dominating Set

Idea: keep 3 lightest dominating sets of each subtree (rooted at $u$) one containing $u$, one not containing $u$, and one disregarding $u$
Trees make most NP-hard problems easy

Example of \textit{Min Weighted Dominating Set}

\begin{itemize}
  \item Idea: keep 3 lightest dominating sets of each subtree (rooted at $u$)
  \item one containing $u$, one not containing $u$, and one disregarding $u$
\end{itemize}
Trees make most NP-hard problems easy

Example of \textbf{Min Weighted Dominating Set}

Idea: keep 3 lightest dominating sets of each subtree (rooted at $u$) one containing $u$, one not containing $u$, and one disregarding $u$
Tree decomposition
Tree decomposition

Cover by bags mapping to a tree s.t. each vertex lies in a subtree
Tree decomposition: solving **Max Independent Set**

For each trace in each bag, keep a best solution in what is below.
Tree decomposition: solving \textbf{Max Independent Set}

For each trace in each bag, keep a best solution in what is below
Tree decomposition: solving **Max Independent Set**

For each trace in each bag, keep a best solution in what is below.
Tree decomposition: solving **Max Independent Set**

For each trace in each bag, keep a best solution in what is below.
Tree decomposition: solving **Max Independent Set**

For each trace in each bag, keep a best solution in what is below.
Tree decomposition: solving **Max Independent Set**

For each trace in each bag, keep a best solution in what is below
Tree decomposition: solving \textbf{Max Independent Set}

For each trace in each bag, keep a best solution in what is below.
Tree decomposition: solving **Max Independent Set**

For each trace in each bag, keep a best solution in what is below.
Tree decomposition: solving **Max Independent Set**

For each trace in each bag, keep a best solution in what is below
Tree decomposition: solving Max Independent Set

For each trace in each bag, keep a best solution in what is below.
Treewidth

Minimum largest bag size over all tree decompositions minus 1

- rediscovered several times in the 70’s and 80’s...
- made central by *Graph Minors* and algorithmic graph theory
- previous slide: $2^{O(tw)} n$ time with $n$ bags
Treewidth

Minimum largest bag size over all tree decompositions minus 1

- rediscovered several times in the 70’s and 80’s...
- made central by *Graph Minors* and algorithmic graph theory
- previous slide: $2^{O(tw)} n$ time with $n$ bags

Computing a tree decomposition?
Treewidth

Minimum largest bag size over all tree decompositions minus 1

- rediscovered several times in the 70’s and 80’s...
- made central by *Graph Minors* and algorithmic graph theory
- previous slide: $2^{O(tw)}n$ time with $n$ bags

Computing a tree decomposition? NP-hard but various algorithms

- width $2^{tw + 1}$ in $2^{O(tw)}n$
- width $5^{tw + 4}$ in $2^{6.76tw}n \log n$
- width $tw$ in $2^{O(tw^3)}n$
- width $tw\sqrt{\log tw}$ in $n^{O(1)}$
- width $tw$ in $1.74^n$
\(2^{O(\sqrt{n})}\) time algorithms on planar graphs via Lipton-Tarjan

Planar graphs have treewidth \(O(\sqrt{n})\)
$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan

Equivalently $O(\sqrt{n})$ balanced separators, i.e., sides of size $\leq 2n/3$
$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan

**Max Independent Set, 3-Coloring, Hamiltonian Path...**
$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan

Max Independent Set, 3-Coloring, Hamiltonian Path...
$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan

Max Independent Set, 3-Coloring, Hamiltonian Path...
$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan

Max Independent Set, 3-Coloring, Hamiltonian Path...

$T(n) \leq 2^{O(\sqrt{n})} T(2n/3) \leq \ldots \leq 2^{O(\sqrt{n}) \sum_i \sqrt{2/3}^i} = 2^{O(\sqrt{n})}$
$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan

Max Independent Set, 3-Coloring, Hamiltonian Path...

Even polyspace!
$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan

Max Independent Set, 3-Coloring, Hamiltonian Path...
$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan

Max Independent Set, 3-Coloring, Hamiltonian Path...

solve the extension List 3-Coloring
$2^{O(\sqrt{n})}$ time algorithms on planar graphs via Lipton-Tarjan

Max Independent Set, 3-Coloring, Hamiltonian Path...

solve the extension List 3-Coloring
Decomposition of dense graphs?

Graphs with small treewidth have linearly many edges

What about simple dense graphs?

clique

biclique
Decomposition of dense graphs?

Graphs with small treewidth have linearly many edges

What about simple dense graphs?

- clique
- biclique

- cliquewidth defined in the 90’s
- allows faster algorithms but hard to compute itself
- rankwidth [Oum, Seymour ’05] “equivalent” and approximable

We will see another equivalent definition via contraction sequences
A single vertex is a cograph,
Cographs

as well as the union of two cographs,
Cographs

and the complete join of two cographs.
Many NP-hard problems are polytime solvable on cographs
Cographs

For instance the independence number $\alpha(G)$ is polytime
Cographs

\[ G_1 \cup \alpha(G_1) + \alpha(G_2) \]

In case of a disjoint union: combine the solutions
Cographs

In case of a complete join: pick the larger one

\[ \alpha(G_1) + \alpha(G_2) \]

\[ \max\{\alpha(G_1), \alpha(G_2)\} \]
Cographs

In case of a complete join: pick the larger one
Another co-graph definition

Every induced subgraph has two twins
Another cograph definition

Every induced subgraph has two twins

Is there another algorithmic scheme based on this definition?
Another cograph definition

Every induced subgraph has two twins

We store in each vertex its inner max independent set
Another cograph definition

Every induced subgraph has two twins

We can find a pair of false/true twins
Another cograph definition

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Every induced subgraph has two twins

\[ \rightarrow \ldots \rightarrow \]

Sum them if they are false twins

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Another cograph definition

Every induced subgraph has two twins

Max them if they are true twins
Trigraphs

Three outcomes between a pair of vertices: edge, or non-edge, or red edge
Trigraphs

Three outcomes between a pair of vertices: edge, or non-edge, or red edge

Red graph: trigraph minus its black edges
Contractions in trigraphs

Identification of two non-necessarily adjacent vertices
Contractions in trigraphs

Identification of two non-necessarily adjacent vertices
Contractions in trigraphs

edges to $N(u) \triangle N(v)$ turn red, for $N(u) \cap N(v)$ red is absorbing
A contraction sequence of $G$:
Sequence of trigraphs $G = G_n, G_{n-1}, \ldots, G_2, G_1$ such that $G_i$ is obtained by performing one contraction in $G_{i+1}$. 
A contraction sequence of $G$:  
Sequence of trigraphs $G = G_n, G_{n-1}, \ldots, G_2, G_1$ such that $G_i$ is obtained by performing one contraction in $G_{i+1}$.

partition viewpoint:  $G_i \leftrightarrow (G, \mathcal{P}_i)$, vertex $\leftrightarrow$ part  
$G\langle S \rangle = G[\bigcup \text{vertices of } G \text{ contracted into a vertex of } S]$
A contraction sequence of $G$:
Sequence of trigraphs $G = G_n, G_{n-1}, \ldots, G_2, G_1$ such that $G_i$ is obtained by performing one contraction in $G_{i+1}$.

partition viewpoint: $G_i \leftrightarrow (G, \mathcal{P}_i)$, vertex $\leftrightarrow$ part
$G\langle S \rangle = G[\bigcup$ vertices of $G$ contracted into a vertex of $S]$
A contraction sequence of $G$:
Sequence of trigraphs $G = G_n, G_{n-1}, \ldots, G_2, G_1$ such that $G_i$ is obtained by performing one contraction in $G_{i+1}$.

partition viewpoint: $G_i \leftrightarrow (G, P_i)$, vertex $\leftrightarrow$ part $G\langle S \rangle = G[\bigcup \text{vertices of } G \text{ contracted into a vertex of } S]$
A contraction sequence of $G$:
Sequence of trigraphs $G = G_n, G_{n-1}, \ldots, G_2, G_1$ such that $G_i$ is obtained by performing one contraction in $G_{i+1}$.

partition viewpoint: $G_i \leftrightarrow (G, \mathcal{P}_i)$, vertex $\leftrightarrow$ part $G\langle S \rangle = G[\bigcup$ vertices of $G$ contracted into a vertex of $S]$
A contraction sequence of $G$:
Sequence of trigraphs $G = G_n, G_{n-1}, \ldots, G_2, G_1$ such that $G_i$ is obtained by performing one contraction in $G_{i+1}$.

partition viewpoint: $G_i \leftrightarrow (G, \mathcal{P}_i)$, vertex $\leftrightarrow$ part $G\langle S \rangle = G[\bigcup \text{vertices of } G \text{ contracted into a vertex of } S]$
A contraction sequence of $G$:
Sequence of trigraphs $G = G_n, G_{n-1}, \ldots, G_2, G_1$ such that $G_i$ is obtained by performing one contraction in $G_{i+1}$.

partition viewpoint: $G_i \leftrightarrow (G, \mathcal{P}_i)$, vertex $\leftrightarrow$ part $G\langle S \rangle = G[\bigcup \text{vertices of } G \text{ contracted into a vertex of } S]$
Reduced parameters

A graph class has bounded reduced $X$ if all its members admit a contraction sequence whose red graphs have bounded $X$.
Reduced parameters

A graph class has bounded reduced $X$ if all its members admit a contraction sequence whose red graphs have bounded $X$.

<table>
<thead>
<tr>
<th>red graphs have bounded ...</th>
<th>characterize bounded ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>degree</td>
<td><strong>twin-width</strong></td>
</tr>
<tr>
<td>component size</td>
<td><strong>cliquewidth</strong> (sparse: treewidth)</td>
</tr>
<tr>
<td>number of edges*</td>
<td>linear cliquewidth (sparse: pathwidth)</td>
</tr>
<tr>
<td>outdegree</td>
<td>(oriented) twin-width</td>
</tr>
<tr>
<td>degree + treewidth</td>
<td>?</td>
</tr>
<tr>
<td>cutwidth</td>
<td>?</td>
</tr>
<tr>
<td>bandwidth</td>
<td>?</td>
</tr>
</tbody>
</table>
Different conditions imposed in the sequence of red graphs

bd degree: defines bd twin-width

bd outdegree: defines bd oriented twin-width

bd component: redefines bd cliquewidth

bd #edges: redefines bd linear cliquewidth
Bd boolean-width \Rightarrow bd component twin-width

Bd boolean-width: binary tree layout s.t. every edge cut in the tree induces a bipartition with bd \# distinct neighborhoods
There is a subtree on $\ell \in [d + 1, 2d]$ leaves, where $d$ bounds the number of single-vertex neighborhoods in a bipartition.
Bd boolean-width $\Rightarrow$ bd component twin-width

Two vertices have the same neighborhood outside of this subtree
Bd boolean-width \Rightarrow bd component twin-width

Contracting them preserves the upper bound at $2d$ on the size of red connected components
Component twin-width and boolean-width are tied

Theorem (B., Kim, Reinald, Thomassé '22)

A class has bounded component twin-width iff it has bounded boolean-width/cliquewidth/rank-width.

Proof.
We just saw one direction.
Theorem (B., Kim, Reinald, Thomassé '22)

A class has bounded component twin-width iff it has bounded boolean-width/cliquewidth/rank-width.

Proof.
We just saw one direction.
Conversely, build the binary tree layout based on the contractions. When red components merge, their subtree gets a same parent. □
Component twin-width and boolean-width are tied

Theorem (B., Kim, Reinald, Thomassé ’22)
A class has bounded component twin-width iff it has bounded boolean-width/cliquewidth/rank-width.

Proof.
We just saw one direction.
Conversely, build the binary tree layout based on the contractions. When red components merge, their subtree gets a same parent. □

Theorem (B., Kim, Reinald, Thomassé ’22)
A class has bounded total twin-width iff it has bounded linear boolean-width/cliquewidth/rank-width.
Is it easier to design algorithms via this characterization?

Solve 3-Coloring on a graph $G$ with a contraction sequence s.t.
all red graphs have components of size at most $d$
Is it easier to design algorithms via this characterization?

Solve 3-**Coloring** on a graph $G$ with a contraction sequence s.t.
all red graphs have components of size at most $d$

For every red component $C$ keep every profile
$V(C) \rightarrow 2^{\{1,2,3\}} \setminus \{\emptyset\}$ realizable by a proper 3-coloring of $G\langle C \rangle$
Is it easier to design algorithms via this characterization?

Solve 3-Coloring on a graph $G$ with a contraction sequence s.t.
all red graphs have components of size at most $d$

For every red component $C$ keep every profile
$V(C) \rightarrow 2^{\{1,2,3\} \setminus \{\emptyset\}}$ realizable by a proper 3-coloring of $G\langle C\rangle$
Is it easier to design algorithms via this characterization?

Solve 3-Coloring on a graph $G$ with a contraction sequence s.t.
all red graphs have components of size at most $d$

For every red component $C$ keep every profile $V(C) \rightarrow 2\{1,2,3\} \setminus \{\emptyset\}$ realizable by a proper 3-coloring of $G\langle C \rangle$
Is it easier to design algorithms via this characterization?

Solve 3-Coloring on a graph $G$ with a contraction sequence s.t. all red graphs have components of size at most $d$

Some tuples of the at most $d + 1$ profiles corresponding to merging red components are compatible
Is it easier to design algorithms via this characterization?

Solve **3-Coloring** on a graph $G$ with a contraction sequence s.t.
all red graphs have components of size at most $d$

Some tuples of the at most $d + 1$ profiles corresponding to merging red components are compatible
Is it easier to design algorithms via this characterization?

Solve **3-Coloring** on a graph $G$ with a contraction sequence s.t. all red graphs have components of size at most $d$

Some tuples of the at most $d + 1$ profiles corresponding to merging red components are incompatible
Is it easier to design algorithms via this characterization?

Solve 3-Coloring on a graph $G$ with a contraction sequence s.t. all red graphs have components of size at most $d$

Initialization: time $3n$
Update: time $7^d d^2$
Total: time $7^d d^2 n$
End: still a profile on the single vertex containing the whole graph?
Graph FO/MSO Model Checking  
Parameter: $|\varphi|$

Input: A graph $G$ and a first-order/monadic second-order sentence $\varphi \in FO/MSO(\{E\})$

Question: $G \models \varphi$?
Graph FO/MSO Model Checking

Parameter: $|\varphi|$

Input: A graph $G$ and a first-order/monadic second-order sentence $\varphi \in FO/MSO(\{E\})$

Question: $G \models \varphi$?

Example:

$$\varphi = \exists x_1 \exists x_2 \cdots \exists x_k \forall x \bigvee_{1 \leq i \leq k} x = x_i \bigvee_{1 \leq i \leq k} E(x, x_i) \lor E(x_i, x)$$

$G \models \varphi$? $\iff$
Formulas, sentences, and model checking

**Graph FO/MSO Model Checking**  
**Parameter:** $|\varphi|$  
**Input:** A graph $G$ and a first-order/monadic second-order sentence $\varphi \in FO/MSO(\{E\})$  
**Question:** $G \models \varphi$?

Example:

$$\varphi = \exists x_1 \exists x_2 \cdots \exists x_k \forall x \left( \bigvee_{1 \leq i \leq k} x = x_i \bigvee \bigvee_{1 \leq i \leq k} E(x, x_i) \lor E(x_i, x) \right)$$

$G \models \varphi \iff k$-Dominating Set
Formulas, sentences, and model checking

**Graph FO/MSO Model Checking**

**Parameter:** $|\varphi|$

**Input:** A graph $G$ and a first-order/monadic second-order sentence $\varphi \in FO/MSO(\{E\})$

**Question:** $G \models \varphi$?

**Example:**

$$\varphi = \exists x_1 \exists x_2 \cdots \exists x_k \bigwedge_{1 \leq i < j \leq k} \neg (x_i = x_j) \land \neg E(x_i, x_j) \land \neg E(x_j, x_i)$$

$G \models \varphi$? $\iff$
Formulas, sentences, and model checking

**Graph FO/MSO Model Checking**

**Parameter:** $|\varphi|$

**Input:** A graph $G$ and a first-order/monadic second-order sentence $\varphi \in FO/MSO(\{E\})$

**Question:** $G \models \varphi$?

Example:

$$\varphi = \exists x_1 \exists x_2 \cdots \exists x_k \bigwedge_{1 \leq i < j \leq k} \neg (x_i = x_j) \land \neg E(x_i, x_j) \land \neg E(x_j, x_i)$$

$G \models \varphi \iff k$-Independent Set
**Graph FO/MSO Model Checking**

**Parameter:** $|\varphi|$

**Input:** A graph $G$ and a first-order/monadic second-order sentence $\varphi \in FO/MSO(\{E\})$

**Question:** $G \models \varphi$?

Example:

$$\varphi = \exists X_1 \exists X_2 \exists X_3 (\forall x \bigvee_{1 \leq i \leq 3} X_i(x)) \land \forall x \forall y \bigwedge_{1 \leq i \leq 3} (X_i(x) \land X_i(y) \rightarrow \neg E(x, y))$$

$G \models \varphi \iff \quad$
Formulas, sentences, and model checking

Graph FO/MSO Model Checking

Parameter: $|\varphi|$

Input: A graph $G$ and a first-order/monadic second-order sentence $\varphi \in FO/MSO(\{E\})$

Question: $G \models \varphi$?

Example:

$\varphi = \exists X_1 \exists X_2 \exists X_3 (\forall x \bigvee_{1 \leq i \leq 3} X_i(x)) \land \forall x \forall y \bigwedge_{1 \leq i \leq 3} (X_i(x) \land X_i(y) \rightarrow \neg E(x, y))$

$G \models \varphi \iff 3$-COLORING
Courcelle’s theorems

We will reprove with contraction sequences:

**Theorem (Courcelle, Makowsky, Rotics '00)**

*MSO model checking can be solved in time* \( f(|\varphi|, d) \cdot |V(G)| \) *given a witness that the clique-width/component twin-width of the input* \( G \) *is at most* \( d \).

**generalizes**

**Theorem (Courcelle '90)**

*MSO model checking can be solved in time* \( f(|\varphi|, t) \cdot |V(G)| \) *on graphs* \( G \) *of treewidth at most* \( t \).
Courcelle’s theorems

We will reprove with contraction sequences:

**Theorem (Courcelle, Makowsky, Rotics ’00)**

*MSO model checking can be solved in time $f(|\varphi|, d) \cdot |V(G)|$ given a witness that the clique-width/component twin-width of the input $G$ is at most $d$.*

generalizes

**Theorem (Courcelle ’90)**

*MSO model checking can be solved in time $f(|\varphi|, t) \cdot |V(G)|$ on graphs $G$ of treewidth at most $t$.*

Instead of maintaining all the possible profiles of 3-colorings, maintain all the sentences of quantifier depth $\leq q$ satisfied by a red component!
Rank-$k$ $m$-types

Sets of non-equivalent formulas/sentences of quantifier rank at most $k$ satisfied by a fixed structure:

$$tp^L_k(\mathcal{A}, \bar{a} \in A^m) = \{\varphi(\bar{x}) \in L[k] : \mathcal{A} \models \varphi(\bar{a})\},$$

$$tp^L_k(\mathcal{A}) = \{\varphi \in L[k] : \mathcal{A} \models \varphi\}.$$
**Rank-\(k\) \(m\)-types**

Sets of non-equivalent formulas/sentences of quantifier rank at most \(k\) satisfied by a fixed structure:

\[
\text{tp}_k^L(\mathcal{A}, \bar{a} \in A^m) = \{ \varphi(\bar{x}) \in L[k] : \mathcal{A} \models \varphi(\bar{a}) \},
\]

\[
\text{tp}_k^L(\mathcal{A}) = \{ \varphi \in L[k] : \mathcal{A} \models \varphi \}.
\]

**Theorem (folklore)**

*For \(L \in \{FO, MSO\}\), the number of rank-\(k\) \(m\)-types is bounded by a function of \(k\) and \(m\) only.*

**Proof.**

"\(L[k + 1]\) are Boolean combinations of \(\exists x L[k]\)." \(\square\)
Rank-$k$ $m$-types

Sets of non-equivalent formulas/sentences of quantifier rank at most $k$ satisfied by a fixed structure:

$$tp^L_k(\mathcal{A}, \bar{a} \in A^m) = \{ \varphi(\bar{x}) \in \mathcal{L}[k] : \mathcal{A} \models \varphi(\bar{a}) \},$$

$$tp^L_k(\mathcal{A}) = \{ \varphi \in \mathcal{L}[k] : \mathcal{A} \models \varphi \}.$$

Theorem (folklore)

For $\mathcal{L} \in \{FO, MSO\}$, the number of rank-$k$ $m$-types is bounded by a function of $k$ and $m$ only.

Proof.

“$\mathcal{L}[k + 1]$ are Boolean combinations of $\exists x \mathcal{L}[k]$.”

□

Rank-$k$ types partition the graphs into $g(k)$ classes.
Efficient Model Checking = quickly finding the class of the input.
FO Ehrenfeucht-Fraissé game

2-player game on two \( \sigma \)-structures \( \mathcal{A}, \mathcal{B} \) (for us, colored graphs)
At each round, Spoiler picks a structure ($B$) and a vertex therein.
FO Ehrenfeucht-Fraissé game

$A$ \[ a_1, \ldots, a_k \] \[ \vdots \] \[ B \]

Duplicator answers with a vertex in the other structure
After $q$ rounds, Duplicator wishes that $a_i \mapsto b_i$ is an isomorphism between $\mathcal{A}[a_1, \ldots, a_k]$ and $\mathcal{B}[b_1, \ldots, b_k]$.
After $q$ rounds, Duplicator wishes that $a_i \mapsto b_i$ is an isomorphism between $\mathcal{A}[a_1, \ldots, a_k]$ and $\mathcal{B}[b_1, \ldots, b_k]$.
FO Ehrenfeucht-Fraissé game

When no longer possible, Spoiler wins
FO Ehrenfeucht-Fraisse game

When no longer possible, Spoiler wins
If Duplicator can survive $k$ rounds, we write $\mathcal{A} \equiv^\text{FO}_k \mathcal{B}$

Here $\mathcal{A} \equiv^\text{FO}_2 \mathcal{B}$ and $\mathcal{A} \not\equiv^\text{FO}_3 \mathcal{B}$
MSO Ehrenfeucht-Fraissé game

$A$ and $B$ are two graphs. The game is the same, but Spoiler can now play set moves.

Same game but Spoiler can now play set moves.
MSO Ehrenfeucht-Fraïssé game

Same game but Spoiler can now play set moves
MSO Ehrenfeucht-Fraissé game

To which Duplicator answers a set in the other structure
Again we write $\mathcal{A} \equiv^k_{\text{MSO}} \mathcal{B}$ if Duplicator can survive $k$ rounds.
Theorem (Ehrenfeucht-Fraissé)

For every $\sigma$-structures $\mathcal{A}, \mathcal{B}$ and logic $\mathcal{L} \in \{FO, MSO\}$,

$$\mathcal{A} \equiv^L_k \mathcal{B} \text{ if and only if } tp^L_k(\mathcal{A}) = tp^L_k(\mathcal{B}).$$
Theorem (Ehrenfeucht-Fraissé)  
For every $\sigma$-structures $A$, $B$ and logic $L \in \{FO, MSO\}$, 

$$A \equiv^L_k B \text{ if and only if } tp^L_k(A) = tp^L_k(B).$$

Proof. 
Induction on $k$. 

$(\Rightarrow)$ $L[k+1]$ formulas are Boolean combinations of $\exists x \varphi$ or $\exists X \varphi$ where $\varphi \in L[k]$. Use the answer of Duplicator to $x = a$ or $X = A$. 
Theorem (Ehrenfeucht-Fraissé)

For every \( \sigma \)-structures \( \mathcal{A}, \mathcal{B} \) and logic \( \mathcal{L} \in \{ \text{FO}, \text{MSO} \} \),

\[
\mathcal{A} \equiv^\mathcal{L}_k \mathcal{B} \text{ if and only if } \text{tp}^\mathcal{L}_k(\mathcal{A}) = \text{tp}^\mathcal{L}_k(\mathcal{B}).
\]

Proof.

Induction on \( k \).

\((\Rightarrow)\) \( \mathcal{L}[k+1] \) formulas are Boolean combinations of \( \exists x \varphi \) or \( \exists X \varphi \) where \( \varphi \in \mathcal{L}[k] \). Use the answer of Duplicator to \( x = a \) or \( X = A \).

\((\Leftarrow)\) If \( \text{tp}^\mathcal{L}_{k+1}(\mathcal{A}) = \text{tp}^\mathcal{L}_{k+1}(\mathcal{B}) \), then the type \( \text{tp}^\mathcal{L}_k(\mathcal{A}, a) \) is equal to some \( \text{tp}^\mathcal{L}_k(\mathcal{B}, b) \). Move \( a \) can be answered by playing \( b \). \( \square \)
MSO model checking for component twin-width \( d \)

**Partitioned sentences:** sentences on \((E, U_1, \ldots, U_d)\)-structures, interpreted as a graph vertex partitioned in \( d \) parts

Maintain for every red component \( C \) of every trigraph \( G_i \)

\[
\text{tp}^\text{MSO}_k(G, \mathcal{P}_i, C) = \{ \varphi \in \text{MSO}_{E,U_1,\ldots,U_d}(k) : (G(C), \mathcal{P}_i(C)) \models \varphi \}.
\]
MSO model checking for component twin-width $d$

**Partitioned sentences:** sentences on $(E, U_1, \ldots, U_d)$-structures, interpreted as a graph vertex partitioned in $d$ parts

Maintain for every red component $C$ of every trigraph $G_i$

$$tp_k^{\text{MSO}}(G, \mathcal{P}_i, C) = \{ \varphi \in \text{MSO}_{E,U_1,\ldots,U_d}(k) : (G\langle C \rangle, \mathcal{P}_i\langle C \rangle) \models \varphi \}.$$
MSO model checking for component twin-width $d$

**Partitioned sentences:** sentences on $(E, U_1, \ldots, U_d)$-structures, interpreted as a graph vertex partitioned in $d$ parts

Maintain for every red component $C$ of every trigraph $G_i$

$$tp^\text{MSO}_k(G, P_i, C) = \{\varphi \in \text{MSO}_{E,U_1,...,U_d}(k) : (G\langle C \rangle, P_i\langle C \rangle) \models \varphi\}.$$
MSO model checking for component twin-width \(d\)

**Partitioned sentences:** sentences on \((E, U_1, \ldots, U_d)\)-structures, interpreted as a graph vertex partitioned in \(d\) parts

Maintain for every red component \(C\) of every trigraph \(G_i\)

\[
\text{tp}^\text{MSO}_k(G, P_i, C) = \{ \varphi \in \text{MSO}_{E, U_1, \ldots, U_d}(k) : (G(C), P_i(C)) \models \varphi \}.
\]

\(C\) arises from \(C_1, \ldots, C_{d'}\): \(\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)\)
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

Duplicator combines her strategies in the red components
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

If Spoiler plays a vertex in the component of type $\tau_1$, 

---

If Spoiler plays a vertex in the component of type $\tau_1$, 

---
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

Duplicator answers the corresponding winning move
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

Same in the component of type $\tau_2$
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

Same in the component of type $\tau_2$
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

Same in the component of type $\tau_2$
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

Same in the component of type $\tau_2$
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

and so on
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

and so on
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

and so on
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

and so on
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

and so on
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

If Spoiler plays a set, Duplicator looks at the intersection with $C_1$, 

---
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

If Spoiler plays a set, Duplicator looks at the intersection with $C_1$, 
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

calls her winning strategy in $C_1'$
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

same for the other components
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game.

same for the other components
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

same for the other components
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

and plays the union
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

that fully defines the winning strategy of Duplicator
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

that fully defines the winning strategy of Duplicator
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game

that fully defines the winning strategy of Duplicator
Showing $\tau = F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ via MSO EF game
Turning it into a uniform algorithm

Reminder:

▶ #non-equivalent partitioned sentences of rank $k$: $f(d, k)$
▶ #rank-$k$ partitioned types bounded by $g(d, k) = 2^{f(d, k)}$

For each newly observed type $\tau$,

▶ keep a representative $(H, P)_\tau$ on at most $(d + 1)g(d, k)$ vertices
▶ determine the 0, 1-vector of satisfied sentences on $(H, P)_\tau$
▶ record the value of $F(\tau_1, \ldots, \tau_{d'}, B, X, Y)$ for future uses
Turning it into a uniform algorithm

Reminder:

▶ \#non-equivalent partitioned sentences of rank \( k \): \( f(d, k) \)
▶ \#rank-\( k \) partitioned types bounded by \( g(d, k) = 2^{f(d,k)} \)

For each newly observed type \( \tau \),

▶ keep a representative \((H, \mathcal{P})\) on at most \((d + 1)g(d, k)\) vertices
▶ determine the 0, 1-vector of satisfied sentences on \((H, \mathcal{P})\)
▶ record the value of \( F(\tau_1, \ldots, \tau_{d'}, B, X, Y) \) for future uses

To decide \( G \models \varphi \), look at position \( \varphi \) in the 0, 1-vector of tp\(_k^{\text{MSO}}\)(\( G \))
Twin-width is more general than the classic widths
Twin-width is more general than the classic widths
Twin-width is more general than the classic widths
Twin-width is more general than the classic widths
Twin-width is more general than the classic widths
Twin-width is more general than the classic widths
Twin-width is more general than the classic widths

4-sequence for planar grids, but unbounded cliquewidth
$\geq 2 \log n)$-subdivisions have twin-width at most 4
\((\geq 2 \log n)\)-subdivisions have twin-width at most 4

Add a red full binary tree whose leaves are the vertex set
$(\geq 2 \log n)$-subdivisions have twin-width at most 4

Take any subdivided edge
(≥ 2 \log n)-subdivisions have twin-width at most 4

Shorten it to the length of the path in the red tree
$(\geq 2 \log n)$-subdivisions have twin-width at most 4

Zip the subdivided edge in the tree
\((\geq 2 \log n)\)-subdivisions have twin-width at most 4

Zip the subdivided edge in the tree
(≥ 2 log n)-subdivisions have twin-width at most 4

Zip the subdivided edge in the tree
\((\geq 2 \log n)\)-subdivisions have twin-width at most 4

Zip the subdivided edge in the tree
$(\geq 2 \log n)$-subdivisions have twin-width at most 4

Zip the subdivided edge in the tree
\(\geq 2 \log n\)-subdivisions have twin-width at most 4

Zip the subdivided edge in the tree
\((\geq 2 \log n)\)-subdivisions have twin-width at most 4

Zip the subdivided edge in the tree
$(\geq 2 \log n)$-subdivisions have twin-width at most 4

Move to the next subdivided edge also of unbounded cliquewidth
Theorem
The following classes have bounded twin-width, and $O(1)$-sequences can be computed in polynomial time.

- Bounded rank-width, and even, boolean-width graphs,
- every hereditary proper subclass of permutation graphs,
- posets of bounded antichain size (seen as digraphs),
- unit interval graphs,
- $K_t$-minor free graphs,
- map graphs,
- subgraphs of $d$-dimensional grids,
- $K_t$-free unit $d$-dimensional ball graphs,
- $\Omega(\log n)$-subdivisions of all the $n$-vertex graphs,
- cubic expanders defined by iterative random 2-lifts from $K_4$,
- strong products of two bounded twin-width classes, one with bounded degree, etc.
Theorem
The following classes have bounded twin-width, and $O(1)$-sequences can be computed in polynomial time.

- Bounded rank-width, and even, boolean-width graphs,
- every hereditary proper subclass of permutation graphs,
- posets of bounded antichain size (seen as digraphs),
- unit interval graphs,
- $K_t$-minor free graphs,
- map graphs,
- subgraphs of $d$-dimensional grids,
- $K_t$-free unit $d$-dimensional ball graphs,
- $\Omega(\log n)$-subdivisions of all the $n$-vertex graphs,
- cubic expanders defined by iterative random 2-lifts from $K_4$,
- strong products of two bounded twin-width classes, one with bounded degree, etc.

Can we solve problems faster, given an $O(1)$-sequence?
**$k$-Independent Set** given a $d = O(1)$-sequence

**$d$-sequence:** $G = G_n, G_{n-1}, \ldots, G_2, G_1 = K_1$

**Algorithm:** For every connected subset $D$ of size at most $k$ of the red graph of every $G_i$, store in $T[D, i]$ one largest independent set in $G(D)$ intersecting every vertex of $D$. 

**Running time:** $d^2 k n^2$ red connected subgraphs, actually only $d^2 k n = 2^{O(d)(k)n}$ updates

**How to compute** $T[D, i]$ from all the $T[D', i] + 1$?
**k-Independent Set** given a $d = O(1)$-sequence

$d$-sequence: $G = G_n, G_{n-1}, \ldots, G_2, G_1 = K_1$

Algorithm: For every connected subset $D$ of size at most $k$ of the red graph of every $G_i$, store in $T[D, i]$ one largest independent set in $G(D)$ intersecting every vertex of $D$.

Initialization: $T[\{v\}, n] = \{v\}$

End: $T[\{V(G)\}, 1] = IS$ of size at least $k$ or largest IS in $G$

Running time: $d^{2k} n^2$ red connected subgraphs, actually only $d^{2k} n = 2^{O_d(k)} n$ updates
**k-Independent Set** given a \(d = O(1)\)-sequence

\(d\)-sequence: \(G = G_n, G_{n-1}, \ldots, G_2, G_1 = K_1\)

Algorithm: **For every connected subset** \(D\) **of size at most** \(k\) **of the red graph of every** \(G_i\), **store in** \(T[D, i]\) **one largest independent set in** \(G\langle D\rangle\) **intersecting every vertex of** \(D\).

Initialization: \(T[\{v\}, n] = \{v\}\)

End: \(T[\{V(G)\}, 1] = \text{IS of size at least } k\) **or largest IS in** \(G\)

Running time: \(d^{2k}n^2\) red connected subgraphs, actually only \(d^{2k}n = 2^{O_d(k)}n\) updates

**How to compute** \(T[D, i]\) **from all the** \(T[D', i + 1]\)?
**$k$-Independent Set: Update of partial solutions**

Best partial solution inhabiting $\bullet$?
$k$-Independent Set: Update of partial solutions

$G_{i+1}$

$G_i$

3 unions of $\leq d + 2$ red connected subgraphs to consider in $G_{i+1}$ with $u$, or $v$, or both
The previous algorithm generalizes to:

**Theorem (B., Kim, Thomassé, Watrigant ’20)**

*FO model checking can be solved in time $f(|\varphi|, d) \cdot |V(G)|$ on graphs $G$ given with a $d$-sequence.*
FO model checking on graphs of bounded twin-width

The previous algorithm generalizes to:

Theorem (B., Kim, Thomassé, Watrigant ’20)

*FO model checking can be solved in time* \( f(|\varphi|, d) \cdot |V(G)| \) *on graphs G given with a d-sequence.*

Add **Gaifman’s locality of FO** to our MSO model checking algorithm
The previous algorithm generalizes to:

Theorem (B., Kim, Thomassé, Watrigant ’20)

*FO model checking can be solved in time* $f(|\varphi|, d) \cdot |V(G)|$ *on graphs G given with a d-sequence.*

Add **Gaifman’s locality of FO** to our MSO model checking algorithm

**Thank you for your attention!**
Local tuple of parts

$(P_1, P_2, \ldots, P_q)$ is local around $P_1$ if...
\((P_1, P_2, \ldots, P_q)\) is local around \(P_1\) if...
\(P_2\) is at distance at most \(2^{k-2}\) from \(\{P_1\}\) in \((G, P_i)\)
Local tuple of parts

$$(P_1, P_2, \ldots, P_q)$$ is local around $P_1$ if ...

$P_2$ is at distance at most $2^{k-2}$ from \{P_1\} in $(G, P_i)$
Local tuple of parts

\((P_1, P_2, \ldots, P_q)\) is local around \(P_1\) if...

\(P_3\) is at distance at most \(2^{k-3}\) from \(\{P_1, P_2\}\) in \((G, \mathcal{P}_i)\)
Local tuple of parts

\((P_1, P_2, \ldots, P_q)\) is local around \(P_1\) if...

\(P_3\) is at distance at most \(2^{k-3}\) from \(\{P_1, P_2\}\) in \((G, \mathcal{P}_i)\)
$(P_1, P_2, \ldots, P_q)$ is local around $P_1$ if...

$P_4$ is at distance at most $2^{k-4}$ from $\{P_1, P_2, P_3\}$ in $(G, \mathcal{P}_i)$
$(P_1, P_2, \ldots, P_q)$ is local around $P_1$ if...

$P_4$ is at distance at most $2^{k-4}$ from $\{P_1, P_2, P_3\}$ in $(G, \mathcal{P}_i)$
Local tuple of parts

($(P_1, P_2, \ldots, P_q)$ is local around $P_1$ if...

$P_q$ is at distance at most $2^{k-q}$ from $\{P_1, \ldots, P_{q-1}\}$ in $(G, \mathcal{P}_i)$
$(P_1, P_2, \ldots, P_q)$ is local around $P_1$ if...

$P_q$ is at distance at most $2^{k-q}$ from \{P_1, \ldots, P_{q-1}\} in (G, \mathcal{P}_i)$
$(P_1, P_2, \ldots, P_q)$ is local around $P_1$ if...

$P_q$ is at distance at most $2^{k-q}$ from $\{P_1, \ldots, P_{q-1}\}$ in $(G, \mathcal{P}_i)$
Partitioned local sentences and types

A prenex sentence is *partitioned local around* \( X \) in \((G, \mathcal{P}_i)\) if of the form

\[
Qx_1 \in X \ Qx_2 \in P_2 \ldots \ Qx_k \in P_k \ \psi(x_1, \ldots, x_k)
\]

with

- \( \psi \) is quantifier-free, and
- \((X, P_2, \ldots, P_k)\) local around \( X \) in \((G, \mathcal{P}_i)\).
Partitioned local sentences and types

A prenex sentence is partitioned local around $X$ in $(G, \mathcal{P}_i)$ if of the form $Qx_1 \in X \ Qx_2 \in P_2 \ldots \ Qx_k \in P_k \ \psi(x_1, \ldots, x_k)$ with

- $\psi$ is quantifier-free, and
- $(X, P_2, \ldots, P_k)$ local around $X$ in $(G, \mathcal{P}_i)$.

And the corresponding types:

$$\text{ltp}^\text{FO}_k(G, \mathcal{P}_i, X) = \{\varphi : \text{qr}(\varphi) \leq k, \varphi \text{ is partitioned local around } X \text{ in } (G, \mathcal{P}_i), (G, \mathcal{P}_i) \models \varphi\}.$$
Partitioned local sentences/types in $(G, \mathcal{P}_n)$ and $(G, \mathcal{P}_1)$

**Initialization of the dynamic programming**

In $(G, \mathcal{P}_n = \{\{v\} : v \in V(G)\})$: for every $v \in V(G)$,

$Q_{x_1} \in \{v\} \quad Q_{x_2} \in \{v\} \quad \ldots \quad Q_{x_k} \in \{v\} \quad \psi \equiv \psi(v, v, \ldots, v)$

Partitioned local types are easy to compute in $(G, \mathcal{P}_n)$
Partitioned local sentences/types in \((G, \mathcal{P}_n)\) and \((G, \mathcal{P}_1)\)

Initialization of the dynamic programming

In \((G, \mathcal{P}_n = \{ \{v\} : v \in V(G)\})\): for every \(v \in V(G)\),

\[ Qx_1 \in \{v\}, Qx_2 \in \{v\}, \ldots, Qx_k \in \{v\} \]

\[ \psi \equiv \psi(v, v, \ldots, v) \]

Partitioned local types are easy to compute in \((G, \mathcal{P}_n)\)

Output of the dynamic programming

In \((G, \mathcal{P}_1 = \{V(G)\})\):

\[ Qx_1 \in V(G), Qx_2 \in V(G), \ldots, Qx_k \in V(G) \]

\[ \psi \equiv \text{classic sentences} \]

The partitioned local type in \((G, \mathcal{P}_1)\) coincides with the type of \(G\)
Partitioned local types give the partitioned types

\[ \text{Isom. } f : \mathcal{P}_i \rightarrow \mathcal{P}'_i \text{ with } \text{ltp}^\text{FO}_k(G, \mathcal{P}_i, X) = \text{ltp}^\text{FO}_k(G', \mathcal{P}'_i, f(X)) \]

\[(G, \mathcal{P}_i)\]

\[(G', \mathcal{P}'_i)\]

Local strategies win the global game
Partitioned local types give the partitioned types

\[ \text{Isom. } f : \mathcal{P}_i \rightarrow \mathcal{P}_i' \text{ with } \text{ltp}^{\text{FO}}_k(G, \mathcal{P}_i, X) = \text{ltp}^{\text{FO}}_k(G', \mathcal{P}_i', f(X)) \]

\[ (G, \mathcal{P}_i) \]

\[ (G', \mathcal{P}_i') \]

Say, Spoiler plays in $P_1$
Partitioned local types give the partitioned types

\[ \text{Isom. } f : P_i \rightarrow P_i' \text{ with } \text{ltp}^\text{FO}_k (G, P_i, X) = \text{ltp}^\text{FO}_k (G', P_i', f(X)) \]

\((G, P_i)\)

\((G', P_i')\)

Duplicator answers in \(f(P_1)\) following the local game around \(P_1\)
Partitioned local types give the partitioned types

Isom. \( f : P_i \rightarrow P'_i \) with \( \text{ltp}^{\text{FO}}_k (G, P_i, X) = \text{ltp}^{\text{FO}}_k (G', P'_i, f(X)) \)

\[ (G, P_i) \]

\[ (G', P'_i) \]

Now when Spoiler plays close to \( P_1 \) or \( f(P_1) \)
Partitioned local types give the partitioned types

\[ \text{Isom. } f : \mathcal{P}_i \to \mathcal{P}'_i \text{ with } \text{ltp}^{\text{FO}}_k (G, \mathcal{P}_i, X) = \text{ltp}^{\text{FO}}_k (G', \mathcal{P}'_i, f(X)) \]

\[(G, \mathcal{P}_i) \]

\[\cdots\]

\[(G', \mathcal{P}'_i) \]

Duplicator follows the winning local strategy
Partitioned local types give the partitioned types

\[ \text{Isom. } f : \mathcal{P}_i \rightarrow \mathcal{P}'_i \text{ with } \text{ltp}^{\text{FO}}_k (G, \mathcal{P}_i, X) = \text{ltp}^{\text{FO}}_k (G', \mathcal{P}'_i, f(X)) \]

\( (G, \mathcal{P}_i) \)

\( (G', \mathcal{P}'_i) \)

Duplicator follows the winning local strategy
Partitioned local types give the partitioned types

Isom. $f : P_i \rightarrow P'_i$ with $\text{ltp}^k_{\text{FO}}(G, P_i, X) = \text{ltp}^k_{\text{FO}}(G', P'_i, f(X))$

If Spoiler plays too far
Partitioned local types give the partitioned types

\[ \text{Isom. } f : \mathcal{P}_i \rightarrow \mathcal{P}'_i \text{ with } \text{ltp}^k_{\text{FO}}(G, \mathcal{P}_i, X) = \text{ltp}^k_{\text{FO}}(G', \mathcal{P}'_i, f(X)) \]

\((G, \mathcal{P}_i)\)

\(f\)

\((G', \mathcal{P}'_i)\)

Duplicator starts a new local game around that new part
Partitioned local types give the partitioned types

Isom. \( f : \mathcal{P}_i \rightarrow \mathcal{P}'_i \) with

\[
\text{ltp}^\text{FO}_k (G, \mathcal{P}_i, X) = \text{ltp}^\text{FO}_k (G', \mathcal{P}'_i, f(X))
\]

\((G, \mathcal{P}_i)\)

\((G', \mathcal{P}'_i)\)

Duplicator starts a new local game around that new part
Concluding as in the MSO model checking algorithm

\[(G, \mathcal{P}_{i+1}) \simto (G, \mathcal{P}_i) : X \text{ and } Y \text{ are merged in } Z\]

Partitioned local types around \(P\)

- only needs an update if \(P\) is at distance at most \(2^{k-1}\) from \(Z\)
Concluding as in the MSO model checking algorithm

\[(G, \mathcal{P}_{i+1}) \rightsquigarrow (G, \mathcal{P}_i) : X \text{ and } Y \text{ are merged in } Z\]

Partitioned local types around \(P\)
- only needs an update if \(P\) is at distance at most \(2^{k-1}\) from \(Z\)
- update only involves parts at distance at most \(2^{k-1}\) from \(P\)
Concluding as in the MSO model checking algorithm

\[(G, \mathcal{P}_{i+1}) \sim (G, \mathcal{P}_i) : X \text{ and } Y \text{ are merged in } Z\]

Partitioned local types around \( P \)

- only needs an update if \( P \) is at distance at most \( 2^{k-1} \) from \( Z \)
- update only involves parts at distance at most \( 2^{k-1} \) from \( P \)
- hence at most \( d^{2^k} \) parts: conclude like MSO model checking
Concluding as in the MSO model checking algorithm

\[(G, P_{i+1}) \leadsto (G, P_i) : X \text{ and } Y \text{ are merged in } Z\]

Partitioned local types around \( P \)

- only needs an update if \( P \) is at distance at most \( 2^{k-1} \) from \( Z \)
- update only involves parts at distance at most \( 2^{k-1} \) from \( P \)
- hence at most \( d^{2^k} \) parts: conclude like MSO model checking

Each contraction: \( O_{d,k}(1) = O(d^{2^k}) \) updates in \( O_{d,k}(1) = f(d, k) \)
Total time: \( O_{d,k}(n) \)